Automated Network Planning for Industrial Ethernet Networks

Holger Machens

Department of Computing, Cork Institute of Technology, Cork, Ireland.

Follow this and additional works at: https://sword.cit.ie/allthe

Part of the Computer-Aided Engineering and Design Commons, and the Computer Sciences Commons

Recommended Citation


This Master Thesis is brought to you for free and open access by the Dissertations and Theses at SWORD - South West Open Research Deposit. It has been accepted for inclusion in Theses by an authorized administrator of SWORD - South West Open Research Deposit. For more information, please contact sword@cit.ie.
Automated Network Planning for Industrial Ethernet Networks

H. Machene
Master of Science

2009
Automated Network Planning for Industrial Ethernet Networks

Holger Machens
Master of Science
Department of Computing

Supervisors:
Dr. Jeanne Stynes
Dr. Reinhold Kröger

Submitted to Cork Institute of Technology, May 2009
Automated Network Planning for Industrial Ethernet Networks

Holger Machens
Cork Institute of Technology

Abstract

Engineers usually use computer aided methods to ease the increasingly complex planning task of automation plants. One such computer aided method for automated planning of Industrial Ethernet networks under given network QoS requirements is presented in this thesis. The method is based on partitioning the given set of network nodes according to specified communication relationships between the nodes, thereby creating subsets of nodes which are joined using given networking algorithms. Several networking algorithms may be applicable for each subset of nodes and any complete solution is a combination of several partial solutions, one for each subset of nodes. Enumeration through all possible complete solutions is realized by a best-first search over all combinations of evaluated partial solutions.

An implementation using the Industrial Ethernet protocol PROFINET RT is presented: a prototype for a network planning tool is developed based on the Network Design Framework (NDF). The application allowed a successful evaluation of the network planning method and the NDF as a framework for network planning tools. A parallel variant of the best-first search is also explored to provide acceptable response times for the network planning tool. This variant works on one complete solution while considering multiple partial solutions in parallel based on the subsets of nodes from the partitioning process, and provides an alternative to the approach of Kumar et al, who proposed working on multiple complete solutions for the given problem in parallel.
## Contents

1 Introduction .......................................................... 1

2 Background .......................................................... 5

2.1 Industrial Ethernet ............................................... 5

2.1.1 Modbus TCP .................................................. 7

2.1.2 EtherNet/IP .................................................... 8

2.1.3 EtherCAT ....................................................... 11

2.1.4 ETHERNET Powerlink ..................................... 13

2.1.5 Foundation Fieldbus High Speed Ethernet .......... 15

2.1.6 PROFINET ..................................................... 17

2.2 Network Planning ............................................... 24

2.2.1 Problems and Algorithms of Network Planning ..... 25

2.2.2 Cluster Analysis ............................................. 27

2.3 The Network Design Framework ............................ 29

2.3.1 Overview ....................................................... 29

2.3.2 Data Models ................................................. 32

2.3.3 Search Process ............................................. 38

2.3.4 Utilities ......................................................... 43

2.3.5 Available Modules ......................................... 44
2.4 Concurrent Systems .................................................. 45
  2.4.1 Models for Concurrent Computation ......................... 47
  2.4.2 Design Patterns for Concurrent Systems ................... 50
  2.4.3 Evaluation of Parallelisation ................................. 55

3 Parallelisation of the NDF Search Process ............................ 58
  3.1 Analysis .................................................................. 58
    3.1.1 Parallel Construction of Multiple Solutions .............. 59
    3.1.2 Parallel Construction of Single Solutions ................. 61
    3.1.3 Selection of a Parallelisation Approach ................ 66
    3.1.4 Speedup Estimation ......................................... 68
    3.1.5 Design Rules .................................................. 71
  3.2 Design ..................................................................... 72
    3.2.1 Overview ........................................................ 73
    3.2.2 Task Scheduling ................................................ 74
    3.2.3 Task Execution ............................................... 79
    3.2.4 Backtracking Control ....................................... 81
    3.2.5 Messaging ...................................................... 85
  3.3 Implementation ..................................................... 87
    3.3.1 Changes to the NDF ........................................ 87
    3.3.2 NDF Enhancements: Implementation Details ............ 88
    3.3.3 Implementation Effort ...................................... 89

4 Application in an Industrial Ethernet ............................... 92
  4.1 Selection of the Application Environment ....................... 92
  4.2 Analysis .............................................................. 94
List of Figures

2.1 Logical view on message exchange between function blocks .......... 6
2.2 OSI layers in CIP ................................................. 8
2.3 CIP application model .............................................. 9
2.4 EtherCat strings .................................................... 12
2.5 Topology according to the IAONA Guide .......................... 13
2.6 Traffic levels in Powerlink ......................................... 14
2.7 HSE protocol stack ............................................... 16
2.8 PROFINET-based automation plant ............................... 18
2.9 Ethernet frame extended by the VLAN tag ....................... 21
2.10 IRT time-slice method ........................................... 22
2.11 Active Control Connection Object ............................... 22
2.12 Cyclic IO data exchange ......................................... 23
2.13 Acyclic event forwarding ........................................ 23
2.14 A minimum spanning tree ....................................... 25
2.15 A maximum spanning tree ...................................... 26
2.16 A steiner tree ....................................................... 26
2.17 Partitioning-based clustering .................................... 27
2.18 Dendrogram ......................................................... 28
2.19 Density-based clustering ......................................... 28
4.4 Differences of topologies: Line topology for scenario 1 ........................................ 100
4.5 Differences of topologies: Tree topology for scenario 1 ........................................ 100
4.6 Differences of topologies: Linking scenario 2 ....................................................... 101
4.7 Differences of topologies: Line topology for scenario 2 ........................................ 101
4.8 Differences of topologies: Tree topology for scenario 2 ........................................ 101
4.9 Interconnection problem in tree topologies .............................................................. 103
4.10 Ratio between local and global optimas .................................................................. 103
4.11 Example for a case with more than two devices ...................................................... 105
4.12 Switch positioning in line topologies .................................................................... 105
4.13 Non-optimum solution using grid aligned cabling .................................................. 105
4.14 Interconnection problem in line topologies .............................................................. 106
4.15 UML class diagram: Area partitioning modules ...................................................... 118
4.16 UML class diagram: Distance-based partitioning module ...................................... 119
4.17 UML class diagram: Switch inserting production module MDInserter ................. 121
4.18 UML class diagram: Production module ComposingProducer .................................. 122
4.19 UML class diagram: Validation module AnalyticValidation ................................... 122
4.20 UML class diagram: Export module CSVExport .................................................... 123
4.21 UML class diagram: Export module PartsListReport ........................................... 124
4.22 UML class diagram: Export module QoSReport ..................................................... 125
5.1 Example of a density function for a line of devices .................................................. 130
5.2 Example of a line topology ....................................................................................... 131
5.3 Speedup with 10 devices per area ............................................................................. 137
5.4 Speedup with 20 devices per area ............................................................................. 138
5.5 Speedup with 50 devices per area ............................................................................. 138

IX
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.6 Speedup with 10 devices per area and minimum control overhead</td>
<td>142</td>
</tr>
<tr>
<td>5.7 Speedup with 20 devices per area and minimum control overhead</td>
<td>142</td>
</tr>
<tr>
<td>5.8 Speedup with 50 devices per area and minimum control overhead</td>
<td>143</td>
</tr>
<tr>
<td>5.9 Chart with speedup in case of backtracking</td>
<td>146</td>
</tr>
<tr>
<td>5.10 Cumulative histogram on cost ratio between (8 nodes)</td>
<td>151</td>
</tr>
<tr>
<td>5.11 Cumulative histogram on cost ratio (12 nodes)</td>
<td>151</td>
</tr>
<tr>
<td>5.12 Optimum to generated costs (12 nodes)</td>
<td>152</td>
</tr>
<tr>
<td>5.13 Ratio of optimum to generated latency (12 nodes)</td>
<td>153</td>
</tr>
<tr>
<td>5.14 Latency in solutions for 150 nodes (line.gr)</td>
<td>155</td>
</tr>
<tr>
<td>5.15 Processing time for 50 to 5000 nodes without restrictions</td>
<td>160</td>
</tr>
<tr>
<td>5.16 Processing time for 50 to 5000 nodes with restrictions</td>
<td>161</td>
</tr>
<tr>
<td>B.1 Networks with nodes of equal degrees</td>
<td>B-2</td>
</tr>
<tr>
<td>C.1 Decomposition according to Data Flow rules</td>
<td>C-2</td>
</tr>
<tr>
<td>C.2 Decomposition of a functional program</td>
<td>C-3</td>
</tr>
<tr>
<td>C.3 Elimination of redundancy in a functional program</td>
<td>C-3</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Scope of backtracking conditions .............................................................. 63
3.2 Speedup after backtracking ........................................................................ 70
3.3 Lines of code for parallel search algorithm .............................................. 90
3.4 Lines of code for parallel search algorithm (continued) ....................... 91
4.1 Overview: Industrial Ethernet environments ........................................... 93
4.2 Model mapping: Plant description .......................................................... 117
4.3 Model mapping: Material List ................................................................. 117
4.4 Model mapping: Units for requirements .................................................. 117
4.5 List of values to be exported ................................................................. 124
4.6 Lines of code for partitioning modules .................................................... 126
4.7 Lines of code for production modules ..................................................... 127
4.8 Lines of code for partitioning modules .................................................... 127
4.9 Lines of code for export modules ........................................................... 127
5.1 Hardware .................................................................................................. 134
5.2 Problem sizes .......................................................................................... 135
5.3 Devices per area ....................................................................................... 135
5.4 Devices per area ....................................................................................... 136
5.5 Control overhead in the worst case ......................................................... 140
Listings

B.1 Enumeration of edge combinations ............................................. B-4
B.2 Enumeration of degree combinations ......................................... B-5
C.1 Imperativ programming language example .................................. C-1
C.2 Single assignment convention example ..................................... C-1
C.3 Decomposition Example ............................................................. C-2
C.4 Example for a sequence of function call statements .................. C-3
Chapter 1

Introduction

The engineering of industrial automation systems is a complex and time-consuming task. Complex units of automation devices are distributed and are frequently based on multilevel network architectures such as fieldbuses and higher-level Ethernet-based industrial networks. In this architecture the boundaries between traditional fieldbuses and higher networks become increasingly blurred. While the engineering of industrial fieldbus structures is well under control, the design of Ethernet-based networks for automation purposes is still problematic when a large number of end points is given and in particular when quality-of-service criteria such as end-to-end delays and usage limits have to be met.

The Industrial Ethernet market is constantly growing. In January 2008 the ARC Advisory Group in its study about "Industrial Ethernet-Based Device Networks" [Gro08] projected a 27.5% growth in the market during the next five years. While just 1 million Industrial Ethernet nodes were delivered in the year 2007, the research institute concludes that the amount will increase up to 3 million by 2012. This data reflects the increasing relevance of Ethernet in the automation domain and the growing size and complexity of Ethernet-based automation systems.

To handle the increasingly complex tasks in automation plants planning, engineers use tools in the computer aided methods domain (e.g. Computer Aided Design (CAD), Computer Aided Engineering (CAE), etc.), which traditionally operate on single-user workstations. But an appropriate support for the planning task in Industrial Ethernet
networks with a specified expected size still does not exist in this tool chain.

The Laboratory of Distributed Systems of Wiesbaden University of Applied Sciences (FHW), in cooperation with Siemens AG, a well-known global player in the market of plant automation, developed a generic method and framework for automated network planning, the Network Design Framework ([MK06b], [MK06a]) which is extensible by modules that cover specific rules for different network types or domains. The author of this thesis is one of the two main researchers in this project.

The network planning problem addressed here is known to be NP complete [San03] and therefore it is inevitable that heuristics must be used to find a solution. The Network Design Framework (NDF) supports the partition of a given set of end points into smaller subsets, to reduce the complexity, as one method to apply heuristics. For every subset of end points a set of modules, with specialized algorithms employing heuristics for a specific network type or domain, can propose possible solutions (networks) with a value for its estimated quality. Based on this quality value and the set of proposals, the network planning tool realises a best-first search over all subsets to find a suitable low priced solution under the given quality-of-service criteria.

The development of the overall tool framework led to a first prototypical implementation of the framework and simple example modules demonstrating the principles and basic functionality. But the simple examples did not give an adequate proof that the framework and its solving strategy is applicable in real-world Industrial Ethernet environments. Therefore a goal of this thesis is to develop a network planning tool for a specific Industrial Ethernet environment and thereby analyse the adequacy of the framework for such applications. Further evaluation will also address the quality of the generated solutions, the performance and the end user usability, and software engineering specific properties such as flexibility and extensibility of the framework.

Bad response time from the network planning tool is always a possible disadvantage for end users. But increasing the size of networks inevitably results in heavier processor load during network generation, even when partitioning is employed. First experiences with the example modules already showed this performance problem in some circumstances. Current developments in the hardware industry indicate increas-
ing availability of multi-core and multi-processor systems in the workstation domain. A utilisation of these two or four core architectures already available today can be a possible solution for this problem. Therefore, another goal of this thesis is the development of a parallelisation strategy to increase the processing resources of the framework in order to reduce calculation times based on a shared memory approach suitable for those architectures.

Summarising, the thesis addresses the following topics:

- The parallelisation of the network planning tool.
- The application of the approach on a representative Industrial Ethernet protocol.
- The evaluation of the thesis results, i.e. the parallel search algorithm, the network planning tool and additionally the underlying planning approach of the NDF and its adequacy as framework for such applications.

After this introduction, the thesis starts with a chapter providing the foundation for the two main tasks of the thesis. It gives a broad overview of Industrial Ethernet networks, introduces the usual problems encountered in designing networks and presents available algorithms to resolve them. A detailed description of the NDF follows. Finally, methods for the modeling, development and evaluation of concurrent systems which can be used for the formulation of a parallelisation strategy are presented.

Chapter three concentrates on the parallelisation of the search process of the Network Design Framework. The parallelisation strategies introduced in the first chapter will be analysed according to their applicability to the NDF and its specific search strategy. The result of the analysis will lead to the design of the parallelisation extensions, also described in this chapter.

Chapter four discusses the application of the framework in an industrial Ethernet environment. The chapter starts with an analysis of the industrial Ethernet environments introduced in the first chapter, according to their common attributes. Based on this analysis one of them is chosen for application on the framework and used to identify
the rules to be realized by the extensions to the framework. The design and implementation are also described in this chapter.

Chapter five presents the evaluation and its results. It starts with an analysis of the evaluation tasks, followed by a description of the tools used to perform the evaluations before the results are discussed. The evaluation considers the quality, performance and usability of the developed network planning tool, the speedup of the parallelised algorithm and the NDF according to its capabilities as a framework for network design algorithms.

The thesis ends with a conclusion that summarises the work and its results and on this basis, suggests approaches for further research projects.
Chapter 2

Background

This chapter introduces the background material necessary to understand the research topic. Section 2.1 provides an overview of industrial Ethernet networks, the target environment for our network planning tool. The fundamental algorithms of network planning and network design are presented in Section 2.2, leading to a discussion in Section 2.3 of the Network Design Framework, the basis of the network planning tool. The last section provides an overview on concurrent systems and the parallelisation techniques needed in the design of the parallelisation extensions to the Network Design Framework.

2.1 Industrial Ethernet

Fieldbus protocols are the predecessors of Industrial Ethernet protocols. The structure of an Industrial Ethernet-based application is mostly related to a paradigm, standardized in the standard IEC 61131 [Lew01], which was developed for fieldbus systems at that time. This standard specifies the software on automation devices as abstract Function Blocks offering a set of input/output values that are updated cyclically via the network (see Figure 2.1. Each function block is connected to a master, where all values come together, establishing a common process image. In a process cycle the master receives the output values from the connected slaves and delivers the appropriate values to their input slots. The master is usually the component that controlled the
whole process because he has the complete view of the process image. The master is in most cases a *Programmable Logic Controller (PLC)*, a programmable automation device. The cyclic character of communication is the traditional way automation devices communicate with each other via a fieldbus.

![Diagram of message exchange](image)

**Figure 2.1: Logical view on message exchange between function blocks**

Determinism is an important property for communication in automation systems. The fieldbus systems are distinguished from Ethernet-based systems by their deterministic and reliable behaviour, which can be ascribed to their very controlled communication and their cyclic processing manner. Ethernet is a less deterministic medium as it is based on the media access control protocol CSMA/CD, and it is affected by the queuing behaviour of switches. The first disadvantage is avoided in most industrial Ethernet environments by the use of completely switched full duplex networks. This means that every station is connected to a switch via a full duplex twisted pair cable; Collisions are impossible in these networks because one station is always exclusively connected to exactly one switched Ethernet segment. The second disadvantage, the queuing behaviour of switches, is tackled in different ways as can be seen in the next section, but the absence of the determinism of fieldbus systems remains.

Real-time systems are generally classified as *hard real-time* or *soft real-time* systems. Hard real-time systems always deliver messages at the specified time and every deviation is handled as an error. Soft real-time systems deliver the messages mostly at the
right time, but slight deviations in defined ranges are allowed. Soft real-time often use best effort delivery of messages. Best effort means that the network services provide no quality of service (QoS) guarantees about the delivery of messages, their delivery time, the transfer rate or any other property in communication or even if they are delivered anytime. Most industrial Ethernet protocols belong to the family of soft real-time protocols.

The next sub-sections give a broad overview of the most well-known modern industrial Ethernet communication environments.

2.1.1 Modbus TCP

Modbus [MI04] is an application-level communication protocol developed by Modicon (www.modicon.com) for their automation devices.

The protocol provides message-based master/slave communication based on a request/response paradigm. The protocol was originally developed for serial lines (like RS232) with binary and ASCII encoded messages.

There are different transport media/protocol adaptions for Modbus:

- **Modbus RTU** (Remote Terminal Unit) for serial lines with a binary protocol.
- **Modbus ASCII** for serial lines with ASCII encoding.
- **Modbus Plus** for high-speed token passing networks.
- **Modbus TCP** is a newer adaption on TCP/IP which is proposed to become an IEC standard.

Modbus is mainly oriented to IEC 61131 but other models are supported too. Every device has a set of functions that can be called. The function call results in a response or an exception message - these are the only message types needed.

The protocol has no QoS support and has no real-time abilities. Nevertheless there exist extensions for Modbus to use the Real Time Publish/Subscribe technique [Gro05b], specified by the OMG.
Messages consist of a target address, the id of the function to be called and a data portion. The length of a Modbus frame is limited to 256 bytes due to the first transport media used, namely the serial lines network RS485.

### 2.1.2 EtherNet/IP

EtherNet/IP was specified by a cooperation of the two organisations of users and vendors of ControlNet (ControlNet International, www.controlnet.org) and DeviceNet (Open DeviceNet Vendor Assoc. (ODVA), www.odva.org). EtherNet/IP includes a *Control and Information Protocol (CIP, see [IntOla]),* that defines an application and communication model for automation systems and the EtherNet/IP adaption of the CIP to Ethernet (see [Int01b]). *ControlNet* and *DeviceNet* are two alternative bindings to Ethernet that can be used under CIP (see Figure 2.2).

![Figure 2.2: OSI layers in CIP](image)

The CIP is a peer-to-peer “object”-oriented communication protocol for automation systems independent of the physical layer and the data link layer. The CIP application model corresponds in terminology and semantics to the application model of the
OMG UML (see www.uml.org for more information). It contains the following model entities (see Figure 2.3):

- **Class**: A meta model of application objects, also called *Object Class*.

- **Object**: An instance of a class.

- **Service**: Services provided by an object or class. In UML terminology the service is called method (or static method in case of a class).

- **Attribute**: Attributes of objects or classes.

- **Behaviour**: Runtime behaviour of objects that is described in e.g. activity diagrams and implemented in code.

![Figure 2.3: CIP application model](image)

The application model includes a defined view of a network topology. This describes the following entities of a network:

- **System**: A system is a collection of (network) domains.

- **Domain**: A domain contains a collection of networks.

- **Network**: A network is a collection of Subnets. In a network each node identifier (e.g. Ethernet MAC ID) is guaranteed to be unique.

- **Subnet**: A subnet may contain multiple segments connected by bridges (e.g. a switched Ethernet Subnet). On all segments the nodes use the same network protocols.
• **Segment**: The physical network segment with all nodes connected to one segment of the transmission media (e.g. Ethernet network segments).

• **Node**: A network node, host for a set of objects with one physical network adapter.

In addition, the CIP application may have *Clusters*, which define logical groups of nodes. A Cluster is independent from the network structure and can therefore span all formerly described topology entities up to domains. Clusters are defined according to the relationship of the nodes contained in the Cluster. Three types of relationships are described in the specification:

• Master/Slave Point-to-Point Communication

• Multicast Master/Slave Communication

• Peer-to-Peer Communication

Communication models in CIP are distinguished by connection types. Connections can be established between two or more devices. CIP defines two connection types:

• **I/O Connections** for unidirectional message transfer between a producer as sender and a consumer as receiver of attribute state information from the producer. I/O connections communicate, attribute values. Multiple I/O values for one connection are assembled by *Assembly Objects* in a block.

• **Explicit Messaging Connections** are request/reply-oriented service calls - in other words: remote method invocations. These method calls can be either
  
  – **synchronous**, with a reply sent by the application after service completion or
  
  – **asynchronous**, with a reply immediately sent by the communication layer after receiving and validating the request.

CIP networks are characterised by the cyclic transmission of messages on both connection types. Message transfer can generally be triggered in three different ways:
• **Cyclic:** The transmission is triggered by a so called *Transmission Trigger Timer* in a cyclic way. If no new data is available the producer sends an alive message.

• **Change-Of-State:** The transmission is triggered at change of state of I/O values.

• **Application Object Triggered:** The application decides when to transmit.

In all three cases the consumer can be configured to expect a transmission in a predefined amount of time realising a watchdog functionality driven by a *Watchdog Timer*.

Besides the watch dog support there is no other QoS assurance that provides in-time delivery of messages. If the protocol is comparatively lightweight, EtherNet/IP is used for soft real-time applications only; typical I/O cycle time is about 10ms. Nevertheless it has proved to be applicable in the delivery of time-critical data (cp. [QHGJ06]).

### 2.1.3 EtherCAT

The information provided in this section is based on the Technical Introduction provided by the EtherCAT Technology Group (see [Gro05a]) and an article in the trade journal “Electronic” (see [JB03]).

The EtherCAT protocol was originally developed by Beckhoff (Verl, Germany) and is today maintained by the EtherCAT Technology Group, an organisation of EtherCAT system vendors and users.

EtherCAT sits on top of OSI layer 2 and realises the network, transport and session layer. EtherCAT uses full duplex Ethernet hardware to establish a low-level bus-like communication with high throughput and low latency. There are two types of devices in an EtherCAT system:

• **EtherCAT Masters:** Ordinary PCs or any PC-like hardware with one or more Ethernet adapters.

• **EtherCAT Slaves:** Special EtherCAT devices.

EtherCAT slaves are strung together in lines (see Figure 2.4). Such lines are connected to the master at one end. On the Ethernet layer, the string of slaves is viewed as
one Ethernet device with the first slave in the string acting as a proxy. Also, the MAC address of the first slave is used to address the string of slaves on the Ethernet layer.

Any Ethernet frame, sent to the string of slaves, is seen and processed by every slave. The slaves read and write bits out of and into the bit stream of the passing Ethernet frame. The last slave sends the frame back to the master on the second channel of the full duplex Ethernet cable. Therefore a string can logically be viewed as a ring, with the advantage that collisions are impossible and the maximum bandwidth of the hardware is available to communication.

EtherCAT operates like a distributed shared memory system. The distributed process image consists of physical memory located on the EtherCAT slaves (maximum 64 KB), mapped in a 4 GB sized logical address space, managed by the master. The only operations that are communicated via EtherCAT and processed by the hardware of the slaves are read and write operations on this physical memory in the slaves. The mapping between logical and physical memory addresses is projected on the master and distributed to the slaves during the initialisation phase.

An Ethernet frame can carry one or more EtherCAT telegrams. Every EtherCAT telegram transports a read, write or read/write command for a number of slaves. Multicast and Broadcast are also supported as well as slave-to-slave communication (implicitly). But EtherCat does not perform well for event-based systems, where the priorisation of events is of concern (cp. [CVZ08]).
2.1.4 ETHERNET Powerlink

ETHERNET Powerlink is an Industrial Ethernet protocol introduced by the company B&R (www.br-automation.com, Austria) in 2001 and is now under the management of the ETHERNET Powerlink Standardization Group (EPSG).

The overall topology of an ETHERNET-Powerlink-based system follows the planning and installation guideline provided by the IAONA [ea03], a consortium of well-known companies on the automation sector like Rockwell and Hirschmann in cooperation with a group of the Fraunhofer Research Institute.

![Topo.png](image)

Figure 2.5: Topology according to the IAONA Guide

Several Machine Distributors (MD) are placed in the field level to provide connectivity between different machines or subsystems of a machine (see Figure 2.5). In the level above, these subnets are interconnected with Floor Distributors (FDs) and Building Distributors and so on. It is a common hierarchical network topology.

Powerlink distinguishes between real-time level and non-real-time level in the network (see Figure 2.6). On the real-time level, Powerlink provides hard real-time capabilities.

The real-time capability is restricted to a real-time domain of one Ethernet collision.

\(^1\)http://www.ethernet-powerlink.org
domain (Ethernet segment). All automation devices in a collision domain are connected to the same segment, listening to all messages.

The protocol is based on a Time Division Multiple Access (TDMA) paradigm: The communication on the network is divided into a so called isochronous phase (real-time) and an asynchronous phase (non-real-time). In asynchronous phase, any Ethernet-based protocol can be used. In isochronous phase, every device gets its own time-slice for exclusive access to the Ethernet segment. This provides cycle-times for isochronous communication at 100μs and a jitter in the range of a microsecond. Due to the time-slicing technique the protocol will support cyclic data exchange but has problems with acyclic occurring traffic (e.g. alarms). This issue is for example addressed in [SV07].

The time-slicing of the traffic in one Ethernet segment is controlled (scheduled) by a so called Management Node (MN, also called Bus Scheduler). It also controls which device is allowed to send at a specific time in the isochronous phase. Therefore no collisions can occur on the segment in this phase.

Scheduling is based on synchronisation of the devices involved by a broadcast frame from the MN at startup. After the startup phase, the MN calls every node to send its real-time data and manages the network media. The data to be sent is defined in the development stage, considering the boundary of the time-slots. Messages are always
broadcasted to every node in the segment. During the asynchronous phase any node can send any data without collision prevention on the media.

Every object in a Powerlink environment is described and registered in a central *Object Dictionary*. Every function or parameter of an object can be addressed with a 24 bit reference. This reference can be used to get peer-to-peer access on such elements in the asynchronous phase. The asynchronous phase also allows the exchange of IP-protocol-based messages.

### 2.1.5 Foundation Fieldbus High Speed Ethernet

The Fieldbus Foundation, a standards proposal group of several companies in the automation domain, developed two communication protocols for automation plants:

- **H1**, a fieldbus protocol with 31.25 kbit/s transfer rate and
- **High Speed Ethernet (HSE)**, an Industrial Ethernet protocol.

Both protocols provide the same application level protocol, i.e. they are based on the same application model.

The application model uses the function block concept as described in Section 2.1. Any component in the application is represented by a block. The specification distinguishes between three types of blocks:

- **Resource Block**: This is a block offering management information about a device, for example for identification and version control. Therefore only one Resource Block exists per device.

- **Function Block**: This is an ordinary function block which realises the application control behaviour.

- **Transducer Block**: Transducer Blocks are used to read and write the input and output lines to sensors and actors, thereby allowing device independent implementation of the function blocks similar to device drivers in usual operating systems.
Execution of programs and communication between function blocks in H1 is controlled by system management supported by a time synchronisation protocol. In HSE, the communication occurs unscheduled.

The communication in both protocols is based on the Fieldbus Access Sublayer (FAS) and the Fieldbus Message Specification on OSI layer 7 (see Figure 2.7). The Abstract Syntax Notation 1 (ASN.1) is used on the presentation layer.

The FMS offers a set of services for the interaction between function blocks:

- **Context Management Services**: Context Management Services are used to establish or release so called Virtual Communication Relationships, which define the communication behaviour for a certain relationship between output and input slots of function blocks.

- **Object Dictionary Services**: The Object Dictionary is a registry for input or output variables of devices (so-called objects). The dictionary provides the mechanism for the management of Object Descriptions.

- **Variable Access Services**: This kind of services provide functions for accessing or changing (publishing) of variables.

- **Event Service**: The Event Service supports the reporting of events and management of event processing.

- **Upload/Download Services**: This service group enables uploading and downloading of data (usually code or configurations) from or to the target.
• **Program Invocation Services**: The Program Invocation Services provide remote program execution and control facilities.

The service functions are mapped to a configurable set of Virtual Communication Relationships (VCRs) defining the communication behaviour for a given relation:

• **Client/Server**: This is for request/response relationships, for example for user requests via HMI.

• **Report Distribution**: This VCR is for the sending of events (reports) to a defined group of receivers. It is used for example in the Event Service.

• **Publisher/Subscriber**: This VCR is for one-to-many communication of output variables, with the difference that existing data from earlier messages is overwritten in the receiver. This VCR is used for example in the Variable Access Service.

### 2.1.6 PROFINET

This section explains the structure of PROFINET-based automation plants according to the PROFINet\(^2\) Installation Guideline (see [Ele02]) and describes the PROFINET protocol based on the System Description [Ele06] provided by the PROFIBUS User Organisation (PNO, German abbreviation for PROFIBUS Nutzerorganisation). Some information in this section is based on oral information from PROFINET experts of Siemens.

PROFINET is an industrial Ethernet protocol derived from the communication protocol for the PROFIBUS fieldbus. It is specified and maintained by the PNO.

**PROFINET plants**

In PROFINET-based automation plants (see Figure 2.8) PROFINET enabled automation devices, so called *IO Devices* with fixed functionality (see elements \(D\) and \(D_s\)

\(^2\)There are different notations for PROFINET: PROFINET, PROFINet and Profinet. In this work the notation PROFINET is used generally but in this case the title of the guideline is cited.
in the figure) and IO Controllers on any kind of programmable components like Programmable Logic Controllers (PLCs), are connected to each other via Ethernet network components (switches and cables). Some of the IO Devices are equipped with an internal switch that allows direct interconnection between these components (see $D_s$ in the figure). At several places in the plant, operation mode switches ($OMS$, not to be mixed up with network switches) or emergency switches (not in the figure) are installed, which allow switching between different operation modes of device groups: devices can be switched off in an emergency case or the process can be switched to another alternative group of devices. So called IO Supervisors (Human Machine Interfaces, HMI), in most cases standard industrial PCs with specific monitoring software (like WinCC) connected to the Ethernet network, are used to manage and monitor groups of devices close to the HMI node.

**Logical Areas of Automation Plants**

In PROFINET-based automation plants (and in most other plants too) several types of logical areas can be defined for planning and maintenance purposes. Typical types of areas are:

- **PLC Area**: A group of devices connected to one PLC.

- **Operation Mode Area**: A group of devices controlled by an operation mode switch.

- **Emergency Area**: A group of devices connected to an emergency switch, that is used to turn off all devices in case of an emergency event.
• **HMI Area:** A group of devices to be monitored and managed by the same HMI.

These areas also play an important role in the planning of automation plants. For example they often mark the boundaries of functional subsystems or subnets where special conditions rule.

**PROFINET Networks**

In general, the PROFINET protocols do not impose any special requirements on the network components or the network topology. A PROFINET-ready network can consist of common Ethernet switches and cables and the topology can be of star, line or mixed structure. But aggressive environmental influences in a production plant such as extreme temperatures and acid liquids often place an extra demand on the components. The PNO therefore defines different types of network components:

1. those components that must be enclosed in safe environments like cabinets and
2. those components that are prepared for the aggressive environment in the production plant and can be positioned outside such enclosures.

The PNO also specifies detailed robustness requirements that must be met by components positioned in the open.

The PROFINET Installation Guideline also gives suggestions for network topology. Every switch in the network induces a specific amount of latency based on its internal forwarding procedure. For example switches uses port-specific queues to store outgoing Ethernet frames if the port is busy and switches have a fixed processing time to decode the target address of an incoming frame and map it to the right output port. Therefore, the guideline recommends planning to start by establishing subnets for the device communication in one PLC Area and go on connecting them on a higher level. This reduces the number of switches between two communication partners and as a result, the latencies induced by switches. Another result is a more hierarchical structure for the automation network although in practice line structures are always preferred to reduce the cabling costs.
Switches are classified into three usage classes by the PNO:

- **Machine Distributor (MD):** Switches located at the field level near the machines.

- **Field Distributor (FD):** Switches connecting subnets established by Machine Distributors (also called Floor Distributors in other domains like Powerlink and office networking).

- **Building Distributor (BD):** Also known from office networks, Building Distributors provide a connection to the backbone and usually collect all connections in a building.

It is often possible to deduce the different properties of a switch based on its role. For example, a Machine Distributor has just a few ports and needs to be more robust. A Field Distributor has more ports than a Machine Distributor, has lower costs per port and usually has a lower internal latency but a higher queueing probability for outgoing frames because the bigger number of ports results in potentially more incoming frames. A Building Distributor usually has an uplink, i.e., it has a port with higher bandwidth which in most cases means a higher price.

**PROFINET Protocols**

PROFINET communication is based on the so called PROFINET Component Based Architecture (CBA) defined by the PNO [CBA04]. The CBA defines the logical view of a PROFINET network as a set of interacting Automation Objects, so called AUTOs, which include IO Devices, IO Controller and IO Supervisors. Part of this architecture is an abstraction layer on top of the underlying communication protocols, namely:

- **Microsoft DCOM:** DCOM is a middleware for distributed object-oriented applications and was the first communication protocol used in PROFINET. The middleware supports synchronous remote method calls and asynchronous event notification. The communication protocol is based on DCE RPC (see [GRO97]),
which is in turn based on TCP/IP or UDP/IP for multicasts. DCOM is not thought to be used for real-time applications. It has a comparatively high latency in the range of a millisecond due to protocol stack processing and a high jitter, around ten milliseconds (see [CBA04]).

- **Real-Time (RT):** The protocol was formerly known as *Soft Real Time (SRT).* It is developed for soft real-time applications. RT is directly based on the Ethernet layer (OSI layer 2) and its real-time properties are achieved by using priorities for the Ethernet frames according to the IEEE 802.1p standard for *Virtual Local Area Networks (VLANs).* The VLAN standard allows the establishment of several virtual Ethernet networks on the same physical network structure. This is achieved by an extension to the Ethernet frame of four bytes with four fields (see Figure 2.9): A protocol type identifier which declares the frame as a VLAN-frame, one field to distinguish the byte order (Canonical Format Identifier, CFI), a field with a VLAN ID to identify the virtual LAN the frame will be transferred on and a field with a priority for the frame. This priority (0-7) determines the processing order of the frames in VLAN-enabled network components like switches, network adapters and also the network services in the operating systems involved. In switches, for example, the incoming frames are queued at the outgoing ports in the corresponding priority queue.

![Figure 2.9: Ethernet frame extended by the VLAN tag](image)

- **Isochronous Real-Time (IRT):** This protocol uses a TDMA method to separate real-time and non real-time traffic on the network (see also Figure 2.10).
Both types of traffic get their own fixed time-slice and therefore guaranteed bitrate. The time-slicing is achieved by applying special network components (e.g., switches and automation devices) to provide time synchronisation and define time-slices. The high precision of IRT allows cycle times of 1 millisecond at minimum with a jitter of 1µs for sets of up to 100 automation devices. This is very close to the competing EtherCat protocol which is the fastest real-time Industrial Ethernet protocol (cp. [Pry08]).

![IRT time-slice method](image)

Figure 2.10: IRT time-slice method

The protocol abstraction layer is established through the **Active Control Connection Object** (ACCO, see Figure 2.11) that is located on every device. It forwards the messages between the different communication channels and the AUTOs above.

![Active Control Connection Object](image)

Figure 2.11: Active Control Connection Object

Based on a provider/consumer model the ACCO supports two different communication models:

- **Cyclic IO data exchange**: This communication model is supported by all protocols. A provider offers a set of outputs that can be defined as inputs for a consumer. The local ACCO is triggered by event notifications generated by a timer with a predefined cycle-time. It fetches the output values and forwards them over a channel to the ACCO of the consumer device. The remote ACCO delivers the data to the input slots of the consumer AUTO (see also Figure 2.12).
• **Acyclic event forwarding**: This model is currently not supported for RT and IRT. The provider notifies the local ACCO when an event subscribed to by a consumer occurs. The local ACCO forwards the event to the remote ACCO on the consumer device and the remote ACCO delivers the event to the consumer AUTO (see also Figure 2.13).

![Diagram showing cyclic IO data exchange](image)

**Figure 2.12: Cyclic IO data exchange**

The CBA defines different *Quality of Service Types (QoS Types)* for particular combinations of protocol and communication model used on a communication channel between two AUTOs. An additional *QoS Value* provides the cycle-time to be used on the channel, which can be one of 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000 milliseconds.

In addition to the PROFINET protocols, a number of sporadic acyclic network traffic generated by configuration and monitoring tools can occur in PROFINET networks. Every PROFINET device supports OLE for Process Control (OPC) for management purposes. Other protocols such as Simple Network Management Protocol (SNMP, see [Pos82]) or HTTP and FTP can also be used for those purposes. Furthermore protocols

---

3 [www.opcfoundation.org](http://www.opcfoundation.org)
such as SOAP can be used for a vertical integration into business processes, e.g., for integration into an Enterprise Resource Management (ERP, see [Gro98]) system and so on.

2.2 Network Planning

This section provides the background for the network planning (also termed network design) related parts of this thesis. It introduces the algorithms and related problems in network planning needed to understand the processing principles of the Network Design Framework (NFD), explained in the next section, and to realize the domain specific network planning tool on top of it.

The network planning task consists of spanning a network infrastructure on a set of nodes under given constraints such as maximum traffic load for edges or maximum cost for the whole network. Application areas include interconnecting integrated circuits, planning of road networks, water conduit, electric cabling and so on. The most well known application of network planning in the telecommunication area is to optimize traffic routing through a backbone or some kind of wide area network, i.e. routing through an existing physical network topology (cp. [FCC03], [BF77] or [KS99]). But in this work, the focus is on network planning for Ethernet networks, especially for Industrial Ethernet as described in Section 2.1.

The task of network planning for Industrial Ethernet is to span a minimal cost network for a set of network endpoints given the traffic load induced by the endpoint devices and the set of materials to be used in the network.

Examples of network planning problems in this form include: Find the shortest path between all nodes considering the cost and/or the capacity of the edges and/or the nodes; Insert switches into a network, using the number of ports available in a switch expressed as the maximum degree of a node, etc. Obviously, the network planning problem is very complex and many example problems are known to be NP-hard (see e.g. [Rav93]).
Network Planning uses algorithms from a number of diverse disciplines, e.g., Graph Theory, Numerical Analysis, Artificial Intelligence. The next sections describe the network planning problems and algorithms relevant to this thesis, including Clustering, which is employed in the NDF to reduce the complexity of the network planning problem. A good overview on Network Design problems and algorithms is given in [CR98].

2.2.1 Problems and Algorithms of Network Planning

Two graph structures are often employed in network planning, namely Spanning trees and Steiner trees.

**Spanning Trees** A tree is a connected graph without cycles. Given a connected undirected graph $G = (V, E)$, a spanning tree of $G$ is a subgraph $S = (V, E_s)$ of $G$ that is a tree and connects all the vertices $v \in V$ with edges $E_s \subseteq E$. In network design, a network node is represented as a vertex in a graph and vertices are joined by an edge when the corresponding network nodes are physically connected. Spanning trees are a very important construct in network design, taken from graph theory.

![Minimum spanning tree](image)

*Figure 2.14: A minimum spanning tree*

In a connected weighted graph (also called weighted network), a Minimum Spanning Tree (MST, see Figure 2.14) is a Spanning Tree with minimal total weight while a Maximum Spanning Tree (see Figure 2.15) is one with a maximal total weight.
The Minimum Spanning Tree of a weighted (usually undirected) network can be calculated using Kruskal’s Algorithm with a complexity of $O(|E| \log |E|)$ or $O(|E| \log |V|)$ depending on the chosen implementation (see [Kru56]) or Prim’s Algorithm, with a running time between $O(|E| \log |V|)$ and $O(|V|^{2})$ (see [Pri57]). The Maximum Spanning Tree can be calculated using the same algorithms by inverting the costs of the edges.

Steiner Trees A Steiner Tree (see Figure 2.16) is a spanning tree $S = (N, K)$ in a graph $G = (V, E)$ with all terminal nodes $T \subseteq V$ in $N$ ($T \supseteq N$). This means that a Steiner Tree is the spanning tree that connects all terminal nodes via a subset of the mediating nodes in the graph. The Steiner Tree Problem was defined by the mathematician Jakob Steiner in the first half of the 19th century and is known to be NP-hard. The best known algorithm to solve the Steiner tree problem is the Dreyfus-Wagner algorithm (see [DW72]) with a complexity of $O(3^k)$, where $k$ is number of terminal nodes. But there are also several approximation algorithms for it (see e.g.
[Rav93] or [RZ00] with a complexity of $O(mn^2)$ an approximation ratio of about 1.55).

### 2.2.2 Cluster Analysis

Cluster Analysis is a discipline in Data Mining. Clustering is used to identify clusters in a set of points (data objects) described by multidimensional vectors, which represent a set of properties for every point. A cluster is a set of points that are quite similar to each other and different to the points in the other clusters. Similarity between two points is based on an abstract measure called distance and calculated by an appropriate distance function $\delta(v_1, v_2)$ for any two nodes. The distance is defined by the difference between the two vectors, which can be of any kind and combination, e.g., spatial distance or a difference of temperature, time, noise volume, communication intensity and so on. Therefore clustering is a useful method to identify groups of nodes in a network, especially if the set of properties is diverse.

According to [HK04] there are five different categories of clustering methods:

1. Partitioning-based methods: Partitioning methods divide a given set of points into a set of predefined clusters (see Figure 2.17). These methods often start with a given assignment of points to clusters; this is then optimized by moving the points into the better suiting cluster based on a given distance function. A well known partitioning-based clustering method is k-Means Clustering (see [McQ67]) with a computational complexity of $O(nkt)$ where $n$ is the total number of objects, $k$ is the number of clusters and $t$ is the number of iterations. Usually $t$ and $k$ are chosen to be less than $n$.

2. Hierarchical methods: Hierarchical methods use algorithms like the minimal spanning tree to calculate a tree representation of possible clusters, which is called dendrogram. A node in the tree represents a cluster covering all points beyond it and any horizontally cut through the tree represents a possible partition of the whole set of points into clusters. The single points are the leaves at the bottom of the dendrogram representing the smallest clusters possible. At the top
of a dendrogram the root represents the biggest possible cluster, i.e. the whole point set. A well known hierarchical method is Ward Clustering ([War63]) with a running time of $O(n^3)$.

3. Density-based methods: The particular assumption of density-based methods is the presence of a noisy background with clusters in the foreground. These methods differentiate between those points that are dense enough to build a cluster with a minimum number of points and those points that are sparsely distributed over the background and not assigned to any cluster.

4. Grid-based methods: These methods use grid structures to partition the given set of points according to the different components/properties. The only task is to scale the grid to find a suitable partition of the point set. The advantage of a grid-
based method is their low processing complexity (usually $O(n)$). \textbf{ST}atistical \textbf{IN}formation \textbf{G}rid (STING) [HK04] is an example grid-based method.

![Grid-based clustering](image)

Figure 2.20: \textit{Grid-based clustering}

5. Model-based methods: Model-based methods assign points to clusters of given dimensions. The clusters are predefined in a statistical model and not dynamically detected.

2.3 \textbf{The Network Design Framework}

This section describes the Network Design Framework (NDF) developed by the Laboratory of Distributed Systems of the Wiesbaden University of Applied Sciences.

2.3.1 \textbf{Overview}

The Network Design Framework is a framework to realize network planning tools. The NDF is characterized by three main features:

- a set of data models to work on,

- an extension mechanism to plug in modules customizing the network planning process

- and a process control module, that schedules and controls the planning process.
The goal of the overall planning process is to transform the problem instance, formally described as the set of communicating application components and their interaction, and the requirements for the network to be designed into an appropriate solution. The problem instance consists of the following parts:

- geometry information about communication end points,
- information about communication behaviour and requirements for the network namely:
  - end-to-end delay
  - maximum network load
  - and maximum acceptable price
- and information about the network component types to be used to establish the network.

The solution consists of:

- the set of cables
- couple components like switches
- and the network properties with information about induced network load and end-to-end latency for all defined communication relations.

The planning process is divided into separate phases:

- An initialization phase, to read the problem instance from any source like file system or database into the memory of the tool.
- A search phase, where the problem is successively transformed into a solution. The search process consists of partitioning (decomposing) the problem into smaller problem parts with subsets of the given nodes, producing partial solutions for problem parts and validating the produced partial solutions. The validation is used to trigger backtracking of transformations in case of requirement
violations. The search phase repeats these tasks until the problem is completely transformed to a solution or if it determines that it can't generate a valid network for this problem. The search process will be explained in more detail later in this section.

- A finalisation phase, where the solution will mainly be exported to arbitrary data sinks like file systems, databases or even a remote client. The solution, stored in the data model, will be transformed in any output format like XML, comma separated value lists or Excel sheets. In this phase detailed analysis tasks of the network properties by simulation tools can also be triggered or reports can be generated on the generated solution.

![Figure 2.21: NDF architecture overview](image)

Figure 2.21 depicts an architectural overview of the Network Design Framework. The architecture is conceptually divided into five subsystems corresponding to the five main phases during the generation of a network structure:

- **Initialise**: This subsystem provides *Initialisation Modules* for the initialisation phase to read the problem instance and store it in the internal data models.
• **Search:** This subsystem provides a set of so called *Intelligent Modules* for the different tasks in the search phase:

  - *Partitioning Modules* to partition the problem or already partitioned parts of the problem into smaller problem partitions.
  - *Production Modules* to produce partial solutions using different network generation algorithms.
  - *Validation Modules* to validate the partial solutions produced by the production modules.

• **Finalise:** This subsystem has a set of *Export Modules* to support the various tasks in the finalisation phase.

• **Process Control:** This subsystem consists of the *Core* and the *Process Control* modules. The *Core* manages and loads the modules according to a given configuration file and controls the steps through the main process phases of initialisation, searching and finalisation. The *Process Controller*, controlling the search phase, realises the search using the transformations (partitioning and producing) provided by the Search Modules.

• **Visualisation:** This subsystem covers user interfaces for monitoring and command purposes that are realized by User Interface Modules (*UI Modules*).

The next sections describe the data models, currently available modules and the search process in more detail.

### 2.3.2 Data Models

This section gives an overview of the data models available in the Network Design Framework.
2.3.2.1 Problem Model

A problem can be partitioned by the search process, therefore problems can be hierarchically organized as a tree. Figure 2.22 shows an overview of the problem model elements. This data model covers mainly the network devices (Node) and logical interconnections of devices (InteractionLink). Additionally, cabinets (Rack) and Areas in the plant can be defined. All these elements have an attribute with an id, given by the user to identify references on this element later on in the exported solution.

Every network device (automation components and network components) is considered an abstract node. The only information needed for the calculation of a network is their position, i.e. the 3D-coordinates $x$, $y$ and $z$, and the transfer rate supported by their network adapter. Also given by the user is the node type, e.g. “SPS”, “Soft-SPS”, “Sensor” etc. This type attribute has no internal meaning and is used for user interface purposes only. Some automation devices have an embedded network switch to allow direct connection to other devices. The properties of this embedded switch are declared in a type description in the material model. The node itself has just a reference on it (internalCoupleComponent).

Their interconnection is represented by logical links (see InteractionLink in Figure 2.24) between them. Logical links define the traffic between nodes, therefore they are directed and assigned to a protocol (TCP or SRT in the figure) used on the link.

![UML Diagram: Problem Overview](image1)

![UML Diagram: Node](image2)
The two ends of a link are assigned to the nodes in the role of a server and a client (they can likewise represent roles like master and slave, for example). The roles define the direction of the link and implicitly the traffic induced by the communication defined for the link. Depending on the protocol several parameters needed to calculate the actual traffic induced by the logical link are assigned to the link. In the example here, the length of the packets (data) to be sent and the frequency of sending (cycle) are specified for SRT. The values of the parameters can be fixed (staticValue) or assigned to stochastic functions (e.g. Pareto) to define respectively static or stochastic communication behaviour for a link.

Nodes can be assigned to functional areas in the plant like areas for emergency stops or groups of nodes belonging to a subsystem and so on. Areas can be classified by a type attribute. The area type can be used by the search modules to perform special tasks on this group of nodes during the search process.

The last element type, the rack, is used to define cabinets in the automation plant (fixed points in the plant) where a given amount of network components (capacity) can be placed in a safe area. Just like the nodes, the location of a cabinet can be defined by 3D-coordinates (x, y and z).
2.3.2.2 Material Model

This model is used to describe the network component types available to build the network (see Figure 2.25). It covers so called couple component types such as switches, hubs and repeaters and cable types.

All network components have similar attributes, e.g., the name as identifier, the maximum transfer rate supported by the component, the kind of environment where the component can be installed and a price (cost) for the component. Therefore all components are derived from a common base type NetworkComponent as shown in Figure 2.26. A CableType has just one attribute more: the maximum length maxLength technically allowed for a cable.

CoupleComponent is another derived class of NetworkComponent. This class has a defined number (numPorts) of ports (Port) with varying transfer rate and a (category) attribute to indicate the distributor category it belongs to, e.g., Field Distributors or Building Distributors. The defined CoupleComponent types are SwitchType, HubType for hubs and RepeaterType. There are other additional CoupleComponent subtypes such as routers.

For more detailed traffic analysis, a number of parameters are available for SwitchTypes to describe their behaviour. For example during a simulation a switch needs parameters such as the operationMode (store-and-forward or cut-through), the number of frame queues, their size (queueSize) and priority rules handled in the switch, the internal processing time (processingTime) the type of internal memory management (bufferType) continuous or block-oriented, and the memory size (bufferSize) available for frames.
2.3.2.3 Requirements Model

The requirements demanded of any solution are defined by the requirements model (see Figure 2.27). They can be seen as the primary objectives of network generation. Secondary objectives such as the geometric structure of the network and the way to lay the cables are dependent on the production rules available in the tool and are therefore part of the tool configuration.

The requirements include the quality of service (QoS) requirements, i.e. end-to-end-latency \( \text{maxEndToEnd} \) and traffic load \( \text{maxLoad} \) for the network as a whole.
(NetworkRequirements) and for single logical links in detail (LinkRequirements). Furthermore, the maximum costs (maxTotalCost) for the whole network and the maximum calculation time (maxProcessingTime) for the search process will be defined for the network to be generated.

The network requirements carry the list of link requirements. Every link requirement is associated with a logical link referenced by the attribute (link).

### 2.3.2.4 Network Configuration Model

![UML Diagram: Network configuration model](image_url)

This model is part of the tool output. The network configuration model (see Figure 2.28) describes the physical network generated between the given nodes defined in the problem model. Such a description consists of the cables (PhysicalLink) and couple components (CoupleNode) that establish the network. Cables and couple components are instances of the types defined in the material model, which is indicated by a reference (type) on them. A cable is laid along the path described by a list of points (Position). Couple components have an identifier (id) and a position given in 3D-coordinates x, y and z similar to the nodes in the problem model. The identifier is used to identify the nodes that the two ends of a physical link (left and right) are connected to. The physical link also has attributes to specify the actual length.
(length), the type of repeater (repeaterType) used to increase the reach of the cable to the given length and the number of repeaters (numRepeater) used.

### 2.3.2.5 Network Properties Model

![UML Diagram: Network properties model](image)

In addition to the network configuration, the network properties also model the results of the primary objectives defined in the requirements model as properties of the network (see Figure 2.29). Some of the network properties are:

- For the whole network the maximum end-to-end delay (maxEndToEnd) found on all logical links and the maximum network load (maxLoad) found on all cables of the network, the overall costs (maxCost) and a flag indicating if the generated configuration is proved to be (valid).

- For every single logical link (see sequence of LogicalLinkVerdicts) the end-to-end latency (maxEndToEnd) and maximum network load (maxLoad) reached on the cables.

- Additionally, for every physical link (see sequence of PhysicalLinkVerdicts), the maximum load (maxLoad) determined for the cable.

The properties model is directly derived from the requirements model because both have the same elements (see Figure 2.27).
2.3.3 Search Process

This section explains the search process provided by the Network Design Framework. Various approaches to solve several isolated problems in network planning under different, sometimes mutually exclusive preconditions were described in Section 2.2. To date, there is no algorithm that is flexible enough to merge different approaches or aspects of the different algorithms to generate an optimal solution of passable complexity. The high complexity of algorithms is a common problem in the domain of artificial search and is often solved by applying heuristics.

The Network Design Framework addressed the complexity problem by using two simple principles that are driven by heuristics. The first one uses the given boundaries of subsystems in the network to create logical partitions of the nodes to be connected. This principle is based on the idea that the given subsystems include more communication inside than outside the subsystem. The second principle uses a set of so called production rules (by Production Modules, named after the production systems [KN87]) that are able to generate the network for given sub-problems under well-defined preconditions by applying certain heuristics. The other benefit of this principle is the flexibility of the algorithm to be customised for different needs, i.e. constructional, safety or reliability requirements and so on.

The search process is managed by the Process Controller. It uses the partition, production and validation operations provided by the loaded modules to realise a depth-first search with evaluation function based on heuristics and backtracking. This type of graph search method is also known as a greedy algorithm (see [Lud02]) which is also a member of the class of best-first search algorithms. A fourth operation called merge is provided by the Process Controller itself to merge a given sub-problem with its parent problem.

The four operations in detail are:

- **Partitioning**: Partitioning is applied on a problem (a set of network nodes) to partition it into smaller problem parts (subsets of the former set of nodes). These
problem parts are logical *sub-problems* of the former problem, seen as *parent problem*. If solutions are found for the sub-problems, so-called *partial solutions*, the parent problem can be solved by connecting the partial solutions.

- **Production**: Production is applied on a problem (set of nodes) to solve it by producing a solution consisting of an interconnecting network between the nodes consisting of couple nodes and cables. In case a problem has a parent problem, a proxy for the network is placed into the parent problem. After solving all child problems of a parent problem, the resulting parent problem can be solved by generating a network which connects the inserted proxies of all sub-problems. The restriction of the production on the set of nodes in a given sub-problem is also a precondition for the application of best first search.

- **Merging**: Merging is applied on a problem to move all nodes back into the parent problem. It is used for special cases, for example, if a partition rated a sub-problem with one node only. Merging can reverse partitioning, therefore cycles in the search process are possible. Prevention of cycles can be very complicated. The NDF supports only a simple prevention mechanism based on a flag, indicating whether the sub-problems of a parent problem have changed since its last partition. More generic algorithms might be possible but they consume more processing time because they rely on matching the current processing sequence with previous sequences in the whole processing path. Because there is only one way to merge nodes into the parent node, this task is provided by the process controller itself and not by further other modules.

- **Validation**: Validation is applied on partial solutions built by production modules. The validation ensures that the requirements, defined in the requirements model, are met by the proposed solution, otherwise it will be discarded.

The partitioning, production and merging operations are used to transform a problem. The validation operation only validates a problem transformation carried out by a production operation. Validation can thus be seen as a post-production operation.
Figure 2.30: Problem partition example

Figure 2.31: Problem tree example
As already stated, problems can be partitioned (decomposed) into sub-problems (see example in Figure 2.30), which results in a tree structure of sub-problems as shown in the AND/OR Graph (see [Nil82]) in Figure 2.31, where the children of a node (sub-problem) must be solved first before the node itself can be solved. All problems in this problem tree have to be gradually solved by the Process Controller, beginning at the leaves walking up to the root problem. The order in solving the sub-problems of a parent problem depends on the scheduling strategy of the Process Controller. The Process Controller currently available in the Network Design Framework creates a sequential schedule for the sub-problems.

![Legend:](Legend.png)

Figure 2.32: NDF search tree example

The combination of the sequence of subproblems and the alternative transformations for each of the subproblems establishes the search tree (see 2.32 for an example) which must be differentiated from the problem tree above to understand the search process. The nodes represent states of the system that will be reached if a given transformation, represented by the edges, is applied to a problem. Every system state is associated with the problem, which is ready to be processed in the given state. Therefore a state represents a problem. The leaves in the tree are the possible solutions for the root problem, i.e., they are the networks that might be generated if they are valid according to the given requirements. Because partitioning produces new sub-problems to be added to the set of unsolved problems, this tree is not static.

The Process Controller must decide which transformation operation to apply. In NDF, every Production Module makes a proposal about the next operation to be carried out. A Production Module can propose every one of the three transformation operations. For every proposal the quality of the solution is estimated by the Production Module.
and expressed in a value (this can, e.g., be the price of the sub-network) that represents the evaluation function in the best first search. The result of this process is a set of alternative operation proposals for the problem, sorted according to their estimated quality.

Now the Process Controller can choose the best proposal, execute it and remove it from the set of proposals. If the chosen operation is a production the Process Controller validates the solution by applying the loaded Validation Module to detect any requirement violation. If the solution has been marked as invalid the backtracking procedure is initiated by the Process Controller.

Backtracking in the search tree is carried out by removing the solution for a problem. The problem itself is moved back to the set of problems to be solved. This gives the search process a new chance to find a solution for the given problem, choosing the next proposal out of the set of proposals made for this problem. If no proposal is left the backtracking proceeds with the previously solved problem.

The search process enumerates all possible combinations of partial solutions proposed by the given set of production modules unless a total solution was found earlier.

After execution of the chosen operation the Process Controller proceeds with the next problem in the same way until all problems are solved, in other words, until the root problem is solved.

The search process, with the transformation operations and backtracking steps described above, establishes the sequential depth-first search on a tree spanned by problems and their proposed transformations. Problem tree (see Figure 2.31) and search tree (see Figure 2.32) are not the same. This is an important difference to keep in mind in the development of new Process Controllers.

2.3.4 Utilities

A number of utilities are available to the modules.

- **Cable Installer**: A Cable Installer calculates a cable route according to different rules:

43
- Laying the cable diagonally through the three dimensional space (unrealistic in reality but often used for fast approximations).

- Laying the cable in parallel to the coordinate axes of the coordinate plane used.

- Laying the cable along a predefined grid.

- Laying the cable along a predefined grid via the floor or the ceiling of a room.

• **Distance Functions**: Distance Functions calculate the distance between two abstract nodes, meaning nodes or couple nodes, according to different rules. The first class of rules is derived from the set of cable rules, which directly influences the length of a cable to be laid. The second class of rules is based on an abstract distance deduced from the communication intensity between two nodes. The communication intensity is calculated from the network load induced by the logical links between the nodes on a virtual physical network structure. A higher load on the link results in a closer relationship between the involved nodes and thus in a shorter abstract distance calculated for these two nodes. This communication-related distance can be combined with the geometric distance functions above to new combined distance functions regarding both aspects.

• **Center Functions**: Center Functions calculate the center for a set of abstract nodes according to a given distance function. Using a communication related distance function for the center function, tightly coupled nodes can be found in the plant, which is, for example, useful for the partitioning task if no subsystem boundaries are given.

### 2.3.5 Available Modules

The Network Design Framework comes with a few example modules which are described in this section.
XML Initialiser Module

This is an Initialiser Module to read the input (see Figure 2.21: material, problem, requirements) from XML documents. There are XML schemata available for all data models of the input documents. The input processing is triggered by the Process Controller and executes according to the following sequence:

1. Read material list
2. Read problem
3. Read requirements

It is important to strictly follow this sequence to correctly establish the references between the entities of the different data models.

K-Means Cluster Partitioning Module

This Partitioning Module partitions according to the k-means clustering algorithm described in Section 2.2.2.

Ward Cluster Partitioning Module

This Partitioning Module partitions according to Ward’s algorithm (see Section 2.2.2).

Production Module Example

This module is a simple example of a production module. It tries to connect all nodes in a given problem to one couple node and moves the couple node as proxy of this sub-network into the parent problem. If the number of nodes in a problem is too big, the module proposes a partition step.

XML Export

This module is for XML export. It writes the output given in the network configuration and network properties model into one XML document each.
2.4 Concurrent Systems

This section introduces the concepts for the design and evaluation of concurrent systems with influences from the domain of parallel computing. Concurrent Computing focuses on the existence of multiple threads of execution (for example a set of individual programs). The central issue is conflict-free sharing of resources between multiple threads of execution computing at the same time but the physical execution can be parallel or pseudo-parallel with interleaved execution of the programs on a single processor (see [Leo00]). Parallel Computing, on the other hand, is the simultaneous execution of the same program, divided into a set of tightly coupled sub tasks running on hardware, which provide real parallelism like multiprocessor architectures (see [ea01]).

The transformation of a sequential system into a system that solves multiple tasks at the same time (in parallel) is called parallelisation.

The first prerequisite for the parallelisation of a sequential system is the identification of potentially concurrent sub-tasks. According to Lamport’s happened-before relationship between events in a concurrent system (see [Lam78]), two tasks are concurrent if none of the tasks needs to happen before the other - in other words, if no path in the directed graph exists between them. This definition sounds trivial but it provides a helpful terminology to identify and describe problems in the development of parallel systems. This relationship between tasks of a concurrent system is also often termed precedence graph, i.e. nodes represent the tasks and a directed arc the precedence relation between the tasks.

Parallelisation can be achieved by two methods (see [Fre94]):

- **Implicit Parallelisation:** The program is automatically parallelised e.g. by a compiler often found in the domain of parallel computing. A lot of programming languages supports the implicit parallelisation by their special syntax.

- **Explicit Parallelisation:** The partitioning into a set of tasks is designed during the development of the program.
Both methods are oriented to common models and patterns for parallelisation of systems.

The success of a parallelisation depends on the right match between the high level design of the parallelised application and the runtime environment. Today there are several combinations of hardware and operating system utilities (threads, processes, shared memory, message passing) up to application servers and distributed virtual machines used for concurrent computing. All of them have different influences on parallel execution of tasks, which has to be considered in the design and the evaluation of a parallel application.

The following sections provide the necessary background for parallelisation. They start with an overview of basic historical design and computational models as well as design patterns for concurrent and parallel systems known today, and conclude with a presentation of the methods and criteria that can be used to evaluate a parallelised version of the NDF.

2.4.1 Models for Concurrent Computation

Parallelisation of a system is a very complex and error prone task. In the 70s and 80s many computational models and appropriate tools were developed to ease the design and implementation of parallel systems and to prevent common errors. Even though the tools and languages in this area are not up-to-date and the required platforms are no longer available, the techniques provided by the underlying models for analysing and identifying the independent tasks of a system remain valid. The domain of models for concurrent computation is very large and cannot be reflected in this work completely. Therefore this section focuses on the most influential concepts.

2.4.1.1 Data Flow

The Data Flow model (see [Ack88]) was a major research topic in the 1970s and 80s. Several compilers and programming languages were developed in this area such as the
vector dialects of Fortran and Pascal for multiprocessors. Data Flow parallelism works on instruction level and on higher levels of a software architecture.

As its name suggests, parallelism in the Data Flow model is data-flow-oriented. A Data Flow system contains transformation operations for data portions connected in a graph.

A basic rule in the design of a Data Flow system is to consider the locality of effects in a transformation. This means a transformation must not interfere with data portions other than the data portion it is actually working on. This is also called freedom of side effects and leads to the single assignment convention for variables in the system. The single assignment convention is the most fundamental rule for Data Flow systems. All other rules or properties of Data Flow systems or languages depend on this convention, which has a strong influence on the programming structures in these systems. Because a variable is assigned only once, it is more an alias for a constant value and the assignment statement is more a declaration of the constant. Therefore, such languages are also called declarative languages. Also every procedure must be stateless otherwise side effects would occur. If a system follows the given rules it can be decomposed according to the transformation instructions and ordered by the data flow (see Appendix C.1 for an example).

As a result of the transformation, the system behaves conceptually in the following way. The required context data "flows" from one processing location in the source code (e.g., a method) to the next. If a transformation of the context data occurs in a location, a new data object is created and forwarded to the next location. Finally, every processing statement waits until all its prerequisites are available (created). This procedure guarantees that concurrency problems never occur in such systems and it provides a very high level of parallelism.

2.4.1.2 Functional Programming

Functional Programming work began with the publication of the first functional programming language Lambda Calculus [Bar88] by Alonzo Church in the early 1930s,
long before the existence of computer systems with enough power to support languages such as this. The fundamental concepts of Functional Programming (see [Hud89]) are closely related to the Data Flow model. In fact, some of the programming languages for data flow systems are functional languages.

Lambda Calculus is a very complicated language and was never intended to serve as a programming language. Examples of functional languages used today include APL, Lisp, Haskell and XSLT.

The underlying model of computation for functional languages is the function\(^4\). A function can take functions as arguments and return functions as well. Such functions are also called higher-order functions. Functions are not allowed to have side effects such as changing global variables, having a state or return values that depend on more than the arguments of the function (for example, terminal input or input from random number generators). A program is said to be pure functional if it consists only of functions without side effects. Some functional languages use a construct called monad to define operations with side effects that must be executed in a specific order.

The absence of side effects makes such functions thread-safe and allows parallel execution. Therefore call-by-future evaluation and call-by-need evaluation are possible. The only processing order given by the program structure in a pure functional program is established by the arguments that are results of function calls.

### 2.4.1.3 Actor Model

The Actor Model was introduced by Carl Hewitt and Henri Baker in the year 1977 [HB77]. It describes configurations of systems consisting of independent entities, called Actors (see Figure 2.33), communicating with each other using asynchronous messaging. The messages sent by an Actor are called messengers, which are themselves actors too. Receiving a messenger is called an event. The actor receiving an event is called the target of the event. An actor is reactive: only the events can activate the actor's behaviour. The behaviour in response to an event can include the creation

\(^4\)For an example of a functional program see Appendix C.2.
of a finite number of new actors and the sending of a finite number of messengers to other actors or itself.

![Actions of an actor](image)

Figure 2.33: Actions of an actor

The events of an actor are partially ordered: Events can be the results of other events (predecessors) in that an actor sends new messengers to other actors. Such events have a causal relationship. So called fork events can result in a set of successor events. Such successor events do not have to be ordered and can therefore be processed concurrently. The counterpart of a fork event is the join event, which has a set of predecessor events. In case of a join the activation of the event is deferred until all predecessor events occurred.

A special type of actor is the Cell. Cells are persistent and carry a value. The value of a cell can be updated and requested by other actors. The cell therefore represents a shared variable.

### 2.4.2 Design Patterns for Concurrent Systems

Besides the attempt, as described in the last section, to develop computational models based on specific programming languages and in most cases for a specific hardware platform, many design patterns were established from practical experiences over time. In some cases the patterns were based on the computational models. Some of these design patterns are for general use in concurrent computations while others are merges or slightly different variants. The most influential patterns will be described in the following sections.
2.4.2.1 Monitor Object

The Monitor Object Pattern (see [ea01]) is probably the most commonly used design pattern in concurrent systems. It is based on the well-known monitor mechanism for synchronisation of concurrent access on shared resources first published by Per Brinch Hansen in 1970 [Han70].

The Monitor Object Pattern describes four roles (see Figure 2.34):

- **Monitor Object**: The object that encapsulates shared resources.
- **Synchronized Methods**: Special methods of the Monitor Object which synchronise (serialise) the access of concurrent threads on the protected resource.
- **Monitor Lock**: A lock mechanism to control the access to the resource in the Synchronized Method.
- **Monitor Conditions**: Predicates that determine whether a thread in a Synchronized Method must wait or whether it can proceed.

![Figure 2.34: Class diagram: Object Monitor pattern](image)

2.4.2.2 Active Object

The Active Object Pattern is also known as Concurrent Object Pattern and has a strong relationship to the Actor Model (see Section 2.4.1.3).

When modelling object-oriented systems, the system is commonly viewed as consisting of Passive Objects. Any "active" computational thread is independent of the object model and works on the set of given objects. In an Active Model the objects
themselves are computational units: Active Objects that interact with each other. The communication between Active Objects is decoupled by event passing.

An Active Object consists of five components (see Figure 2.35):

- **Proxy**: This component is a kind of Facade and provides abstraction from the event communication mechanism to an Active Object. In other words: this component is the Proxy for an Active Object in its role as Servant.

- **Method Request**: The event or basis of a specialised event which is sent to the Active Object by the Proxy.

- **Activation List**: A list of all the Method Requests created by the Proxy which are ready to be delivered.

- **Scheduler**: This component runs in the thread of the Active Object. It observes the Activation List and decides which Method Request to process next. It creates a specific order, which can be simply the arrival order or an order based on particular properties of the Method Requests. The Method Requests can therefore provide a guard method that gives "green light" if the request is ready to be processed.

- **Future**: If a Method Request has a return value, it is given to the caller in a Future Object, which provides asynchronous delivery of the value. It encapsulates the locked storage for the result value, which will be delivered in the future, after the method is processed. It is unlocked and the caller can read it when the result is delivered.

A variant of this pattern is the Master/Slave Pattern (see [ea00]) where a Master component (an Active Object) schedules and delivers tasks to Slave components (also Active Objects) based on the divide and conquer principle.

### 2.4.2.3 Leader/Followers

The Leader/Followers Pattern is the pattern to realise Thread Pools (see [ea01]). Several event sources (streams such as sockets or pipes) provide events to be processed.
Each event is delivered to one thread in the Thread Pool, which executes the appropriate functionality on objects of the environmental system.

In this pattern, four roles exist (see Figure 2.36):

- **Handles:** Handles define the event sources the Thread Pool listens to. Handles are combined in a Handle Set.

- **Event Handler:** This is the interface for the objects with encapsulating threads for the processing of the incoming events.

- **Concrete Event Handler:** Concrete Event Handlers implement the concrete processing methods for the events of the given Handle Set.

- **Thread Pool:** Handle Sets and Event Handlers are put together in a Thread Pool.

The set of Event Handlers can be viewed as a queue: One of the Event Handlers is the Leader and the others are the Followers. The Leader is responsible for processing an event that arrives in the Thread Pool. But before he starts processing the event he first makes the next Event Handler the new Leader.

The conceptual difference to the Active Object or Master/Slave pattern is, that the Event Handler does not implement the functionality of a specific task. It is just the activity carrier and provides the Hook methods. The functionality for a given event is implemented elsewhere.
2.4.2.4 Thread-Local Storage

This pattern is also called Thread-Specific Storage and is based on a mechanism that is provided by some runtime environments: The **Thread-Local Storage** is a storage, which is attached to a thread and is only visible to this thread. The mechanism therefore prevents concurrent access at its root. This mechanism is only applicable to a few problems, but it is a powerful solution in this case because there is no interference with other threads and therefore the maximum speedup can be reached.

The Thread-Local Storage Pattern consists of six roles:

- **Thread-Specific Object**: This is the storage attached to exactly one thread.

- **Key**: Assigned to every Thread-Specific Object is a unique Key for identification.

- **Key Factory**: The Keys are created by a Key Factory.

- **Thread-Specific Object Set**: This is the Set of Thread-Specific Objects attached to a thread. It provides a method to set and get the Objects identified by keys.

- **Thread-Specific Object Proxy**: This is a proxy for the access to the Thread-Specific Objects. It is used by exactly one thread accessing its Thread-Specific Objects and carries an identifier for the thread which is used to identify the Thread-Specific Object Set.
• Application Thread: Application Thread is the Role of a thread (client) using the Thread-Specific Object feature.

![Class diagram: Thread-Local Storage pattern](image)

Figure 2.37: Class diagram: Thread-Local Storage pattern

This model is different to the others in that it prevents concurrent access whereas others deal with concurrent access in various ways.

### 2.4.3 Evaluation of Parallelisation

Factors influencing the beneficial effects of parallelisation include:

- The number of available processors: We need multiple processing units to realise parallel computing. Increasing the number of processing units results in an increased computing potential.

- The size of available memory: An often used mechanism to prevent concurrency problems is to copy the input data of a task such that every sub-task has exclusive access and no time-consuming synchronization between them is needed. In this case, the size of the available memory plays a decisive role.

- Granularity of locks: The parallelism of a task is closely related to the size of its critical sections. The bigger the critical sections are, the less possible it is to
execute tasks in parallel because other tasks using the same data need to wait longer for the lock to be free.

- Underlying communication infrastructure: In distributed systems, the communication infrastructure has direct impact on the computing time of a sub-task. The more data is communicated resulting in more traffic and the less bandwidth and higher latency is provided by the infrastructure the more overhead must be calculated for the sub-task. Also latency is bound to a physically theoretical maximum due to the maximum propagation time at light speed on the transmission medium.

- Communication protocols: Another obvious influence is the size and number of messages sent to communicate data. These parameters are related to the communication protocols. A more implicit influence is the additional serialising and deserialising effort to map data to a message and vice versa which sometimes heavily decreases the efficiency of a parallel system.

- Process scheduling: Another important impact is produced by the scheduling of processes. For every process switch on a processor, the operating system needs to save the current state of the processor registers of the running process, load the registers for the new process and possibly switch to another address space, which invalidates internal caches. In some cases internal optimisation structures such as pipelines are also affected. This impact is directly dependent on the number of running tasks.

An important aspect to consider when evaluating a parallel system is its scalability. This is because many of the parameters listed above are unknown at development time and depend on the computing architecture the system will run on later, e.g., the quality of utilisation of variable resources such as processors and memory by the parallel system.

The benefit of a parallelisation can be expressed by a factor named speedup \( S(p) \), which is defined as the quotient of the time for the sequential processing of a given task
on one processor \((T_{(1)})\) and the processing time in the parallel system with \(p\) processors \((T_{(p)})\).

\[
S_{(p)} = \frac{T_{(1)}}{T_{(p)}}
\]  

(2.1)

The **ideal speedup** is achieved if \(\frac{T_{(1)}}{T_{(p)}} = p\) is true, i.e. a metric for the utilisation of the processors is given by the *efficiency* \(E\) which is defined as

\[
E_{(p)} = \frac{1}{S_{(p)}}
\]  

(2.2)

High efficiency is also an indicator of good scalability in a parallel system.

The most well-known formula to estimate the speedup of parallel systems is from Gene Amdahl ([Amd67]) who also published the formulas for speedup and efficiency above. He considered a part \(a\) of the task that must be executed sequentially, a parallelised part \(\frac{1-a}{p}\) and a last part \(o_{(p)}\) for communication overhead between \(p\) independent processors.

\[
S_{(p)} = \frac{1}{a + o_{(p)} + \frac{1-a}{p}}
\]  

(2.3)

This formula is known as Amdahl’s Law and it shows that the speedup can never be greater than \(\frac{1}{a}\) even if the parallelised part and the communication overhead converge to zero.

\[
S_{(p)} \leq \frac{1}{a}
\]  

(2.4)
Chapter 3

Parallelisation of the NDF Search Process

This chapter contains the analysis and design of the procedure and components required for the parallelisation of the search process.

3.1 Analysis

The NDF search process is a best-first search algorithm. A generic parallelisation of best-first search is published by Kumar et al. in 1988 [KRR88]. This approach follows in parallel multiple independent paths in the search tree and constructs different candidate solutions at the same time (see Figure 3.1).

![Figure 3.1: Kumar's parallel best-first search](image-url)
Another approach, proposed here, is to parallelise the construction of the single solution that will be reached on the currently active path in the search tree (see Figure 3.2).

Figure 3.2: Parallel construction of a single solution

These two approaches to parallelise the NDF search process will be discussed and analysed in this section and a suitable design will be presented to fit the extensions into the framework.

3.1.1 Parallel Construction of Multiple Solutions

Kumar et al's [KRR88] idea is to follow in parallel multiple paths of the search tree (see Figure 3.1), trying to get the best-first search solution as fast as possible. The method is as follows: Any alternative edge leading to a next state in the search tree is potentially parallelisable. The state just needs to be duplicated and allocated to another process or thread. A very simple way to do this is to fork the whole process in every state and put each process on one of the separate paths.

Usually, the edges to be processed next (also called open edges) are ordered by the estimated quality of their potential solution and the best edges are distributed to the available processors. When backtracking occurs, the edge being processed is discarded and the processor is allocated to the next unprocessed edge.

To find the solution of a greedy search algorithm as needed for the NDF, the edges must be ordered by their depth in the search tree and their estimated value according to the objective function (the rating result of the production modules). The process
stops when the solution is found on the most promising path, which is identical to the solution of a greedy search.

Figure 3.3 shows a possible processing scenario for four processors (P1 to P4) using this algorithm. A coloured line represents an edge processed by one of the four available processors.

An advantage of this approach is that every processor is working absolutely independently so concurrency problems cannot occur. The big disadvantage of this approach is the cost in forking the search process.

There are two basic ways to fork a state:

- **Copy the whole state**: This is the most performance-consuming option. Copying the whole state of the network design tool means copying the whole plant description with all nodes in it, all currently available partial solutions, the state of the problem tree, the tool configuration and so on.

- **Copy the path to the current state**: Instead of copying the whole state, only the path through the search tree to the current state is copied. But to reconstruct the path, the tool will need to reconfigure and reinitialise the plant description at every fork.

Both variants produce nearly the same unacceptable overhead for the network generation tool. But modifications to the fork strategy are possible, reducing the number of forks. One strategy is to fork only at the highest level in the search tree and process the lower levels according to the original single-task backtracking method. A possible resulting distribution of the processors is illustrated in Figure 3.4.
This kind of fork control greatly reduces the amount of copy operations but does tend to produce unbalanced parallelism, as shown in the figure. Here the processor P1 must process two times more edges than the processor P2.

The best way to reduce the number of forks is to only fork a state on demand. This means that any processor that becomes idle requests a new open edge. Another processor can respond to the request by offering an open edge in its part of the search tree to the requesting processor.

A possible processing scenario is depicted in Figure 3.5. This approach produces the lowest number of fork costs compared to other fork strategies and the processing load is well balanced between the processors.

This approach is also well suited to distributed systems because the processing of independent paths in the search tree requires no interaction between the processors, except for the states where a fork is requested or a solution is produced and must be compared to other solutions.

3.1.2 Parallel Construction of Single Solutions

This parallelisation approach is based on the idea that independent areas in the network could be generated in parallel. Independent areas refer to sub-problems produced by
the partitioning step in the search process. Figure 3.6 shows an example problem tree with the root problem at the top.

Figure 3.6: Problem tree example

A single path in the search tree leading to the best-first solution may look like the one in Figure 3.7. Each sub-problem in the problem tree is transformed on the path to the solution. A (sub)problem can occur more than once on the path, but in different states and for a problem in a specific state. Several optional transformations may exist.

Only those transformations that work on independent sub-problems in the problem tree are parallelisable. For example: A partition of a problem can occur only if no sub-problems with subsets of the problem exist. Using a more generic view one could say: There are dependencies between the transformation operations that must be considered in a parallel execution of the search process.

The dependencies between the transformations result in a parallel execution schedule as depicted in Figure 3.8. Operations on independent paths in the schedule can be processed in parallel; others need to be deferred until their predecessors are solved. This schedule grows dynamically with every new partitioning operation and shrinks
with every backtracking of a partitioning operation.

The current state depicted by the fat lined arrows in the schedule reflects the same state depicted by the fat lines in the search tree in Figure 3.7. It demonstrates that this algorithm works concurrently on several levels in the search tree and on the same path to a single solution.

<table>
<thead>
<tr>
<th>Influence scope</th>
<th>Parameter</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Local</strong></td>
<td>Network load</td>
<td>Network load is the maximum network load detected on any cable in the network. Its appearance in a backtracking condition therefore affects one partial solution only (local).</td>
</tr>
<tr>
<td><strong>Global</strong></td>
<td>Overall cost</td>
<td>The overall cost of the network is influenced by any solution produced in the network and is therefore a parameter with global influences.</td>
</tr>
<tr>
<td></td>
<td>End-to-end latency</td>
<td>The end-to-end latency is determined from a logical link between two communication end nodes. A backtracking condition caused by the end-to-end latency restriction always affects two nodes, which can be situated in different parts of the network. Therefore it also has global influences.</td>
</tr>
</tbody>
</table>

Table 3.1: Scope of backtracking conditions

Backtracking must be considered in the concurrent construction of a single solution. Because some of the parameters in the objective function are global parameters, a backtracking condition can occur at several points in the search tree at the same time. Backtracking occurs for two different basic reasons:

- **Backtracking due to unresolved requirements of the objective function for the search process.**

- **Backtracking because a dead-end is reached.** This happens when all candidate solutions have been tried and rejected in previous steps in the search process or when there is no solution for the problem under consideration.
Table 3.1 lists the scope of influence of each property of a partial solution, which is part of the objective function. Two of the properties may also affect the properties of other partial solutions, e.g., overall costs. If the search process determines that the overall costs for the network are exceeded in one partial solution, the same result will occur in all other partial solutions produced concurrently. Similarly, if the latency exceeds the predefined maximum in one partial solution, this will also occur in the partial solution containing the other communication end-point of the affected logical link.

The global influences of a backtracking condition demand an intelligent backtracking control. For example, with uncontrolled backtracking, the partial solutions at both ends will be discarded in the end-to-end latency requirement violation described above, even though backtracking just one of the partial solutions involved may suffice.

Another bad side effect of uncontrolled backtracking is that some paths in the search tree may be excluded from processing since every backtracking step changes the path through the search tree. For example, Figure 3.9 shows (1) a simple backtracking condition with two nodes involved in the search tree and (2) the resulting path with uncontrolled backtracking when both partial solutions are discarded. Both successors in the backtracking condition were produced by the transformation operation $f()$ and replaced using transformation operation $g()$. As shown in Figure 3.10, solutions using the paths through the states (3) $C_I$ and $E_{fg}$ or (4) $C_g$ and $E_{gf}$ will be excluded from consideration by this method in the remaining search process.

![Figure 3.9: A possible backtracking condition](image)

To ensure that no paths in the search tree are excluded from processing backtracking must start at the bottom-most edge in the path through the search tree, as in sequential
When backtracking a transformation operation, the previous state of the network must be restored. Due to the locality of most changes this is performed by a simple backup of the previous state of the sub-problem. But the validation needs to attach state information such as the current maximum load and latency in a given state of the search process to the logical links in the network. And these logical links are part of multiple sub-problems during the search process. To manage the recovery of the states of the logical links, an incremental backup of their states must be associated with those sub-problems whose transformation resulted in the changes. There is no alternative way to do this because backtracking always applies to transformation operations (which are indirectly represented in the schedule by a sub-problem affected by the transformation) and not to logical links.

The incremental backup demands backtracking in an order exactly reverse to the former order of validations. To achieve this validation can occur in sequential order. To maintain an acceptable amount of parallel computation, the transformation operations can be decoupled from the validation of the generated partial solutions. By this method, the transformation operations can be processed as soon as they are available in the schedule and the validation occurs downstream as a sequential process as soon as the next partial solution in the search tree is generated.

Unlike the parallel construction of multiple solutions, where each processor has its own data space, this approach works on the same data base concurrently. Most of the transformation operations are working on independent areas in the problem instance.
but the merge operation access the same data concurrently. Here the parent problem, where the sub problems get merged in, is the shared data. Merging also happens when a production module moves a proxy node into the parent problem.

In a distributed version of the parallel network planning framework, the shared global data affected would have to be centrally managed in some kind of master. In such a scenario every access to the global data results in communication, and the problems and solutions must be transferred between the master and its slaves to delegate problems and collect the results.

3.1.3 Selection of a Parallelisation Approach

Two possible approaches on the parallelisation of the search process were presented in the previous sections. In this section these are compared to determine the one better fitting into the framework and more promising for end user purposes.

Figure 3.11: Search tree with a possible situation in search process

Concurrent construction of multiple solutions performs well in scenarios where there is a lot of backtracking. But when little or no backtracking occurs, a lot of processing time is wasted following paths through the search tree which are later discarded. The second approach, concurrent construction of single solutions, optimises both cases implicitly because it has a finer granularity and works concurrently in one path. The search tree in Figure 3.11 shows a case, where the solution is found on one of the lower rated paths at the right. In parallel construction of multiple paths in the search tree this solution is found very early, because all the edges beginning in the root problem are
processed in parallel but the algorithm cannot decide if this solution is the best until it has iterated all possible paths in the sub-tree to the left of the solution tree. Therefore the number of edges that must be processed before the solution is found and verified against some other possible solutions in the left part of the tree equals the number of edges that must be processed using parallel construction of multiple paths or using the single threaded search algorithm. How fast the solution will be found and verified in this case simply depends on the number of processing units. But in a case where the solution is found on the first path (the left-most path) the parallel construction of one path will find the solution up to \( n \) times faster, where \( n \) is the number of available processing units.

The disadvantage of this single path approach is the amount of sequential processing required due to the dependencies between the transformation operations in the execution schedule. But the speedup, compared to the multiple solutions approach, is still high because the multiple solutions approach always works with only one processor on one solution and therefore needs the same time as a sequential processing of the first solution without backtracking. The NDF is mainly designed to optimize towards the first solution and uses backtracking in exceptional cases only. Because it is more promising for the best case where the solution is found in the first try the single path parallelism approach is selected for the parallelisation of the NDF.

The theoretical realisations of the distributed versions of parallel search are described briefly in the sections on the approaches. But they will not be considered any further, because the shared memory based approach is sufficient to evaluate the concept, and distributed systems are currently not the main application domain for the tools based on the NDF. In most cases, those tools will run on a single standard PC (shared memory systems) at most utilizing two or four independent processing units by hyper-threading, dual- or quad-core architectures.
3.1.4 Speedup Estimation

As described in Section 2.4.3, the speedup is defined as the quotient of time for the sequential processing of a given task on one processor \( T_{(1)} \) and the processing time in the parallel system with \( p \) processors \( T_{(p)} \).

\[
S_{(p)} = \frac{T_{(1)}}{T_{(p)}} \tag{3.1}
\]

The transformation operations contributed very little to the sequential part of the search process and is therefore ignored. The greatest processing effort performed in parallel by the search process is used in the two main transformation operations partitioning and production.

A possible worst case execution schedule for parallel execution is given in Figure 3.12 with 22 transformation operations, say units of work, to be processed. Any partitioning operation in the search process creates just two new sub-problems in this example. The number of production operations to be executed in parallel in one path is therefore just two. In processes with a higher partitioning rate, the number of available production operations to be executed will be higher.

![Execution schedule with 22 UoW](image)

Figure 3.12: Execution schedule with 22 UoW

This worst case execution schedule should give a good estimate of the parallelisation speedup. To simplify the comparison, the processing time for a unit of work is considered to be 1. For real world applications an equal partition of workload based on units
of work can be considered, making this still a realistic approximation.

<table>
<thead>
<tr>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>P1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>9</td>
<td>13</td>
<td>17</td>
<td>P2</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>10</td>
<td>14</td>
<td>18</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>7</td>
<td>11</td>
<td>15</td>
</tr>
</tbody>
</table>

Figure 3.13: Processor utilisation

In this model, a system with two, three or four processors could be utilized as depicted in Figure 3.13. According to Equation 3.1 the speedup could be calculated based on the utilization achieved by manual scheduling. The resulting values are depicted in the graph in Figure 3.14 for worst case execution schedules with 4, 10, 22 and 46 units of work in systems with two, three or four processors.

![Graph showing speedup for two, three and four processors](image)

Figure 3.14: Speedup for two, three and four processors

As the graph shows, a significant speedup is already possible in this small worst case scenario. The speedup in real world application will be even higher due to the higher partitioning frequency and the larger number of work units associated with the greater number of nodes.

The effort of backtracking compared to transformation operations is very low and will
not be considered here. Transformation operations work on algorithms with higher processing complexity where backtracking always simply restores a backup with a complexity of at most $O(n)$.

But backtracking removes only parts of the schedule which means that after backtracking only that part of the schedule can be processed in parallel. This reduces possible parallelism.

<table>
<thead>
<tr>
<th>Node</th>
<th>Parallelism</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>$2p_1$</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>$2p_1 + 2 \cdot 2p_1 = 6p_1$</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>$14p_1$</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>$30p_1$</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>$58p_1 + 2p_2$</td>
<td>1.033</td>
</tr>
<tr>
<td>7</td>
<td>$112p_1 + 4p_2$</td>
<td>1.035</td>
</tr>
<tr>
<td>18</td>
<td>$222p_1 + 10p_2$</td>
<td>1.043</td>
</tr>
<tr>
<td>13</td>
<td>$442p_1 + 22p_2$</td>
<td>1.047</td>
</tr>
</tbody>
</table>

Table 3.2: Speedup after backtracking

Table 3.2 lists some speedup calculations for the last few nodes in the schedule (according to the search tree order). The speedup calculation is made under the assumption that for every task in the schedule, only two transformations are possible and that the valid solution is reached if the second choice for each node beyond a given node in the search tree is generated. The node analysed is listed in the first column and the calculated speedup is given in the last column of the table. In the central column, the amount of parallelisable tasks is given in a formula. In this formula non-parallelisable tasks are represented by $p_1$ and dual parallelisable tasks are represented by $p_2$. For the cases analysed, no tasks were identified as requiring to be processed with more than one other task in parallel, therefore no $p_3$ appears in the table. The parallelisation estimate is calculated based on the following considerations: Beginning at the first row in the table with the last node in the schedule, there are no tasks to be processed in parallel, because node 22 depends on the available solutions for all other nodes in the schedule. Therefore both possible solutions for node 22 must be processed single threaded after backtracking, which results in the formula $2 \cdot p_1$ and a speedup of 1. In the second row, the case is that all nodes up to node 21 will be backtracked.
Because node 22 depends on node 21 both nodes have to be processed sequentially (single threaded). Both solutions of node 21 must be combined with the two solutions for node 22. Therefore an effort of $2p_1$ occurs for node 21 and an overall effort of $2 \times 2p_1$ occurs for node 22, which results in $6p_1$ and so on.

This small table provides an insight into the processing effort involved when backtracking in a parallel process. Even when only a few nodes need to be completely enumerated in the search tree, the number of tasks to be processed single threaded rises very fast. In this simple example (8 steps back in the search tree) 442 tasks are to be processed single threaded. This is of course the worst case scenario for a schedule as mentioned above but the situation will be much worse if the number of choices per node is higher.

As already mentioned in the discussion on the different choices of parallelisation approaches for the search process, this work aims to optimise in the most usual case, where the fast solution with no or only a few backtracking steps is of greater interest than the case where a bigger part of the search tree must be enumerated. The latter demands a lot of processing time even with massive parallelisation, which will not be available anyway on low budget systems for end users in the next decade. Therefore the low speedup in the backtracking case is still acceptable because the approach of Kumar et al. provides no speedup for the case where the solution is found on the first path in the tree.

3.1.5 Design Rules

The Network Design Framework was designed to be easily extended by any application developer. Therefore commonly expected properties of a framework played a decisive role in its development. NDF is easily understandable and provides a simple usage, allowing the developer to focus on the development of network algorithms in the search modules rather than the integration of his algorithm into the framework.

Because concurrency can lead to deadlocks and inconsistent data, especially in the parallelisation of the search process, the models and design patterns for concurrent
computation presented in Section 2.4.1 were analysed to find a set of basic design rules that can be used in this case.

A basic principle in every one of the three computation models presented is to prevent concurrent work on global data. This is achieved by the following two simple rules:

- A thread always gets exclusive access to the data associated with the task to be performed.
- A thread never accesses global data.

The data flow model uses data portions flowing through the system. In functional programming these data portions are the parameters and return values of functions and in the actor model the data portions are called events.

This of course is not useful in all applications but it maps very well for the NDF, which uses this principle even when it is not multithreaded yet. The data needed (e.g. the sub-problem) for every task is provided via parameters of the method and the data is exclusively transformed by only one module in a state of the search process, which is quite similar to the data flow model. The only exception that violates this rule is in the global validation that is used to detect parameters that exceed the objective function.

This basic principle will be retained where possible in the enhancements to support the parallelisation of NDF to keep its usage simple for the application developer. If the rules are not applicable then some form of concurrency control such as the Monitor Pattern will be used.

### 3.2 Design

This section introduces the design of the NDF extensions to realise the parallelisation of the NDF search process based on the analysis in the previous sections.
3.2.1 Overview

As depicted in Figure 3.15, the design of the multithreaded process controller is subdivided into four major logical subsystems responsible for dedicated aspects in the process control:

![Logical Architecture Overview](Image)

- **Task Scheduling**: This subsystem is responsible for the management of the tasks in a schedule and determining which task must be executed when considering the relationships between tasks.

- **Task Execution**: This subsystem is responsible for the management of available execution threads and parallel execution of the search tasks.

- **Backtracking Control**: This subsystem is responsible for controlled backtracking: It keeps track of the validation order and performs backtracking to previous tasks.

- **Messaging**: To prevent concurrency problems, as mentioned in the design rules, the scheduling and execution of tasks is decoupled by the active object pattern and this subsystem provides an asynchronous local message exchange between those active objects.

Every logical subsystem in the figure is assigned a dedicated colour. The design of the different subsystems will be explained in the following sections and the colours will be used in the different diagrams to indicate the relationship of the components to certain subsystems.
3.2.2 Task Scheduling

Figure 3.16: UML class diagram: Overview

Figure 3.16 gives an overview of the main components for task scheduling and related components. The class MultiThreadController implements the interface of the base class ProcessController from the NDF and is the scheduling component for the parallelised search (referred to as scheduler in the remaining text). It schedules the tasks represented by objects of the type Task assembled in the search schedule, represented by the component Schedule.

Tasks may be in four different states:

- **Created**: The task has just been created but is not ready to be processed because of unresolved preconditions.

- **Ready**: The task is ready to be processed.

- **Active**: The task is currently being processed.

- **Completed**: The task is completed.
For fast lookups, ready or active tasks are collected in the ReadyList or ActiveList respectively. Execution of tasks is delegated to the Task Execution subsystem. Delegation of tasks is realised via asynchronous messaging, which decouples the scheduler from task execution. The termination of an execution is also notified via messaging to the scheduler.

As mentioned in the overview, backtracking and downstream validation is completely controlled by the BacktrackingControl. It is responsible for tracking the validation order of the tasks according to the search tree and for resolving upcoming backtracking conditions.

The logical execution order according to the search tree is based on two dependencies:

1. the defined relationship between predecessor and successor tasks and

2. the order of the set of successors of each partitioning task as given by the partitioning algorithm.

The second type of relationship is used to keep a defined logical sequential execution order of tasks through all search and backtracking steps but the order of these successors is arbitrary. Keeping the execution order through all search and backtracking steps prevents the loss of search paths as explained in Section 3.1.2.

This order is therefore used to sequence the tasks entering the ready state and later on the active state. To keep validation and backtracking order in sync with the execution order, the backtracking control maintains a stack of the validation. It therefore receives method calls from the scheduler to update the stack each time a task completes processing and is ready to be validated.

The sequence diagrams 3.17, 3.19 and 3.18 depict the task scheduling process.

Sequence diagram 3.17 depicts the delegation of a Task to the Task Execution. It starts when tasks are available to be executed and the Task Execution subsystem signals free threads of execution via Messaging. The method calls in the diagram are
Figure 3.17: UML sequence diagram: Task delegation

assigned to two different threads $a$ and $b$. The number for each call has a prefix ($a$ or $b$), which indicates the thread executing the call.

- a.1. The scheduler retrieves the next task to be executed from the ReadyList.

- a.2. The scheduler marks the retrieved task as active and moves it into the ActiveList.

- a.3. The task is delegated to the Task Execution subsystem by a message sent by the Messaging subsystem.

- b.1. The Task Execution receives the delegation message.

- b.2. The delegated task is executed by the Task Execution subsystem.

- a.4. During the execution, the scheduler waits for completion messages by calling the receive method of the Messaging subsystem.

- b.3./b.4. After execution, the Task Execution produces a completed message, sends this to the scheduler and waits for the next delegation message by calling the blocking receive method.
After completion the scheduler must manage the changes in the task schedule, depending on the transformation operation performed by the finished task.

Sequence diagram 3.18 depicts the procedure after a production transformation or a merge (which is very similar to a production in its effects on the task schedule).

Figure 3.18: UML sequence diagram: Finish after production/merge

- a.6./a.7. After the scheduler receives the message, which indicates a delegated task to be completed, the scheduler removes the task from the ActiveList.
- a.8. Before the scheduler does anything else, it reports the completed task to the BacktrackingControl.
- a.9. The scheduler updates the state of the completed task.
- a.9.1. If a task is a predecessor of another task, the scheduler notifies the other (successor) task of the state change.
- a.9.1.1 By changing the state of a predecessor the precondition of a successor may be fulfilled. The scheduler, as a listener of task state changes, gets notified about it.
- a.9.1.1.1 The notification of a task that has changed into state ready causes the scheduler to move it to the ReadyList.
Figure 3.19: UML sequence diagram: Finish after partition

Sequence diagram 3.19 depicts the finishing procedure after a partitioning transformation.

- a.6. After the scheduler receives the message notification of a completed task, it removes the task from the ActiveList as before.

- a.7. The scheduler again reports the completed task to the Backtracking-Control.

- a.8. The scheduler retrieves the partitions generated by the partitioning transformation and creates new tasks for each of them (see tasks 3.1, 3.2 and 3.3 in Figure 3.20) and one task to generate interconnections between them afterwards (see task 3.4 in Figure 3.20).

Figure 3.20: Extension of the task schedule after partitioning operation

- a.9. The scheduler updates the state of the completed task as before.
• a.9.1. The state update of the predecessor task is notified to the successors: the tasks for the partitions of the problem of the predecessor task.

• a.9.1.1 By changing the state of the predecessor the tasks for the new partitions as successors of the partitioning task automatically switches into the ready state. The scheduler, as a listener of task state changes, gets notified.

• a.9.1.1.1 The notification of a task that enters the ready state cause the scheduler to move it to the ReadyList.

3.2.3 Task Execution

Figure 3.21 provides an overview of the classes related to the task execution in the parallelised process control. Tasks are performed by Workers, which are active objects representing physically processing threads. The Workers are managed in a WorkerPool according to the Leader/Follower Pattern (see Section 2.4.2). The message exchange between Workers and scheduler is realised by the Messaging framework, which provides asynchronous message exchange.

Figure 3.21: UML class diagram: Task execution aspect
Sequence diagram 3.22 depicts the task execution procedure in progress:

![Sequence diagram 3.22: UML sequence diagram: Task execution](image)

- a.1. The scheduler sends a delegation message to the WorkerPool.

- b.1.1. The leading Worker blocking in the `receive()` call of the Worker-Pool now receives this message. The next Worker becomes leader.

- b.1. The task to be processed is retrieved from the message and returned to the now active Worker.

- b.2. The Worker starts executing the Task. The task itself knows what to do for the search process and makes use of the methods already available in the ProcessController which are also used by the single thread controller to perform the transformation operations.

- b.3. After completion, the Worker sends the result through the messaging framework to the scheduler.

- b.4. The Worker then waits for the next task to be processed.
b.4.1. The WorkerPool manages the queue of Workers and puts the Worker at the back of the waiting queue. If no other Worker is waiting, the new Worker immediately becomes the leader and calls the receive() method.

3.2.4 Backtracking Control

The separation of the backtracking control from scheduling means that the backtracking strategy can easily be replaced by another backtracking control component. Figure 3.23 gives an overview of all classes and methods related to the Backtracking Control subsystem in the parallel search process.

![UML class diagram: Backtracking Control](image)

According to Section 3.1.2 the BacktrackingControl must consider the order in which the generated solutions of the tasks can be validated and backtracked afterwards. For every task, it must determine the task’s position in the search tree and whether it is ready to be validated, i.e., whether all its predecessors have already been validated.

Therefore every Task maintains a counter of the predecessors that must be validated.
before it is ready for validation. This counter is updated by the Backtracking-Control each time a predecessor is validated.

Every Task is stored in the ProcessedSet (not shown in the figure) first, which collects all Tasks processed but not yet validated. When backtracking is triggered, the solutions of these Tasks must be discarded because they are on a deeper level in the search tree.

The validation stack maintains the order of validations to be done next. The top-most Task on the stack is always the next one to be validated if the solution is available. It is therefore initialised with the root problem in the beginning. When notified about a completed task, the backtracking control checks whether the Task is at the topmost position on the ValidationStack, i.e., whether it is ready to be validated according to its predecessors.

During validation the BacktrackingControl also analyses the function of the Task and updates the ValidationStack according to the transformation operation performed by the Task. In the case of a production or merge the validated task will be removed from the stack and the successor will be placed on the ValidationStack if all predecessors have been validated. In the case of a partition the Task is also removed from the stack and all the successors (generated sub-problems) are pushed onto the stack.

Every successful validated (accepted) Task is stored on the AcceptanceOrder stack. This helps in keeping track of the right backtracking order. Backtracking-Control also has access to Schedule and ReadyList to manage rescheduling of the Tasks during the backtracking procedure.

Backtracking control subdivides the handling of backtracking into two phases:

- Preparation phase: When a backtracking condition is detected, the backtracking preparation phase starts. In this phase scheduling is suspended and the scheduler awaits the termination of all currently active tasks. These tasks will be added to the processed set by the BacktrackingControl but will not be validated.
• Resolution phase: If no more tasks are active, the resolution of the backtracking condition is performed.

[Diagram: UML sequence diagram: Entering backtracking mode]

During the preparation phase the procedure given in sequence diagram 3.24 occurs:

• 1. **BacktrackingControl** is notified about every resolved Task.

• 1.1 Every completed Task is placed in the ProcessedSet first.

• 1.2. The BacktrackingControl analyses the validation order of tasks using a component called ValidationStack and the isValidationReady method of the Task.

• 1.3. If the Task is ready to be validated, the BacktrackingControl removes it from the ValidationStack and proceeds with the analysis of the Task (step 1.4).

• 1.4. The BacktrackingControl checks the verdict of the resolved task for a possible backtracking condition. If no verdict is available, the task is in a missing proposal condition.
• 1.5. The BacktrackingControl suspends the scheduling process. In suspended mode, the scheduler stops delegating Tasks to the Workers.

• 1.6. The Task involved is stored as invalid in the BacktrackingControl.

• 1.7 If the solution of the Task is valid, the Task is moved to the Acceptance-Order stack.

• 1.8 The ValidationStack is updated.

Figure 3.25: UML sequence diagram: Resolve backtracking condition

Afterwards the backtracking resolution phase starts, as depicted in sequence diagram 3.25:

• 1. The scheduler diagnoses the empty ActiveList and requests the BacktrackingControl to resolve the backtracking condition now.

• 1.1. The BacktrackingControl enters a loop to revoke all Tasks in the ProcessedSet first.
• 1.2. In this loop, it does a “smart” backtrack of these Tasks, because they have not been validated yet.

• 1.3. The BacktrackingControl analyses the invalid Task stored during the preparation phase. If the Task produced an invalid solution, it is selected for backtracking. If the Task is in a missing proposal state, the previous Task is taken from the AcceptanceOrder stack and selected for backtracking.

• 1.4. The selected Task is backtracked. All backtracking related operations specific to the Task are in the charge of the Task itself.

• 1.5. The Task is enqueued in the ReadyList and

• 1.6. its status is set to ready again. If this Task has no more proposals, the search process will return a MissingProposal condition to the next delegation of this Task.

• 2. After solving the backtracking condition the scheduler resumes scheduling the tasks. Thus it first iterates through the ReadList and utilises the Workers until all workers are busy again.

3.2.5 Messaging

![UML class diagram: Messaging](image)

Figure 3.26: UML class diagram: Messaging
The parallelisation needs a simple messaging framework for local asynchronous message exchanges. As depicted in Figure 3.26, it is based on just one base class, called MessageReceiver, which enables the derived classes to asynchronously receive Messages.

Senders need a reference on the MessageReceiver to enqueue a Message in the message queue of the receiver. The receiver can make a blocking call on receive to wait on an incoming message or use the polling mechanism supported by the peek method to retrieve messages in a non-blocking manner.

The message classes are derived from Message (see Figure 3.27). In this application, we have just a few messages:

- **MDelegateTask**: This message is sent by the scheduler to the WorkerPool to delegate a task to a Worker.

- **MTaskCompleted**: This message is sent by a Worker after completion of a Task.

- **MException**: This message is sent by a Worker if an exception occurred during its operation.

- **MShutdown**: This message is sent by the Scheduler to the Worker to request its shutdown.

![UML class diagram: Messages](image)

Figure 3.27: *UML class diagram: Messages*
3.3 Implementation

This section contains information about the implementation of the parallel algorithm.

The Network Design Framework is entirely written in C++ and developed for Windows and Linux on 32 bit architectures. To support both platforms NDF makes use of only POSIX compliant system calls and the boost libraries (see boost.org), especially for concurrency control and threading. On Linux systems boost is based on a Pthreads library with POSIX thread API. The GNU C library (C runtime library) since version 2.3.2 supports the Native Posix Thread Library (NPTL) as a Pthread implementation that maps each thread to its own kernel scheduling entity. Thus each thread is a regular kernel thread which is an important requirement for parallel computing (vs. concurrent computing).

The build environment is based on the GNU Make utility and the detailed system documentation is generated from tagged source code documentation with the documentation system tool Doxygen.

In the following sections the specific changes and enhancements to the NDF, the implementation-specific details of the parallelisation and an overview of the implementation effort are documented.

3.3.1 Changes to the NDF

Process Controller as Module

The first version of the NDF did not provide the possibility to choose between different process controllers. To introduce the Multi-Threaded Controller a module category called Process Controller is introduced, with its own Factory according to the templates given by the existing modules. The process controller can be defined in the configuration file and is instantiated as the first module right after reading the configuration during startup. All the generic methods of process controllers have been moved into a common base class called ProcessController. Now, the process controller base
class is responsible for generic tasks such as the management of listeners and forwarding of notifications with progress information. The SingleThreadedController and its data structures have been moved into its own separate directory in the build tree.

Logging Interface

Logging is very important for the analysis especially in concurrent systems. NDF provides a logging subsystem to optionally log information to a file and/or to the console. To provide concurrent access to the log and maintain consistent log lines (e.g. one line per thread) in the output the logging class has been redesigned and now uses a stream buffer with a thread local storage for each thread. Each time a thread writes to the stream, the output is stored in its thread local storage. If the thread causes a flush of the stream, e.g. by writing a carriage return or manually calling the flush method, the buffer of its local storage is written to the output (file and/or standard out). This way each line found in the log is written by one thread only.

3.3.2 NDF Enhancements: Implementation Details

The Worker Class

The class Worker makes use of the boost class thread to get a thread. The thread class requires a so called Functor class (see Functor pattern) as parameter to its constructor. This Functor will be called by the thread (e.g. Functor::operator()) after initialisation. During the initialization the Functor object is copied several times because some internal methods of the boost library passes the Functor by value instead of by reference. These temporary copies of the Functor are destroyed when the thread creation is finished. Because the Worker should start working immediately after initialisation, it initializes the thread within the constructor. And because the thread will be shutdown when the worker is destroyed it deletes the thread object in the destructor. To prevent a crash during destruction of the copies the member attribute for the thread is initialized with NULL before the constructor of the boost class thread is called.
and is tested to be not equal to NULL before its destruction in the destructor of the Worker.

Efficient Memory Management

Because the objects of the classes Task and Message are very often created and destroyed, the pool object mechanism of the NDF is used for memory management. This mechanism manages a pool of objects of the same type. Each time a pool object is freed it will be moved back into the pool and reused in case a new object of this type is needed again. Pool objects inherit from the base class PoolObject and implement the method reinitPoolObject, which is called by the mechanism each time a pool object is reused. A parametrised version of the factory template PoolFactory is used each time an object is needed. The factory decides whether a new object is created or an object of the pool is recycled.

3.3.3 Implementation Effort

The number of lines of code of the source files for the parallel search algorithm are listed in tables 3.3 and 3.4. The total number of lines is 2868 (see next page).
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcceptanceOrder.cpp</td>
<td>Maintains the order of accepted tasks</td>
<td>23</td>
</tr>
<tr>
<td>AcceptanceOrder.h</td>
<td>Header file of class AcceptanceOrder</td>
<td>20</td>
</tr>
<tr>
<td>ActivationSequence.cpp</td>
<td>Maintains the sequence of activated tasks</td>
<td>63</td>
</tr>
<tr>
<td>ActivationSequence.h</td>
<td>Header file of class ActivationSequence</td>
<td>30</td>
</tr>
<tr>
<td>ActiveList.cpp</td>
<td>Maintains the list of active tasks</td>
<td>50</td>
</tr>
<tr>
<td>ActiveList.h</td>
<td>Header file</td>
<td>27</td>
</tr>
<tr>
<td>BacktrackingControl.cpp</td>
<td>Implementation of the BacktrackingControl</td>
<td>257</td>
</tr>
<tr>
<td>BacktrackingControl.h</td>
<td>Header file of class BacktrackingControl</td>
<td>56</td>
</tr>
<tr>
<td>DotDumpService.cpp</td>
<td>Helper class to dump current schedule in dot format</td>
<td>155</td>
</tr>
<tr>
<td>DotDumpService.h</td>
<td>Header file of class DotDumpService</td>
<td>33</td>
</tr>
<tr>
<td>Message.cpp</td>
<td>Implementation of the message base class</td>
<td>21</td>
</tr>
<tr>
<td>Message.h</td>
<td>Header file of the Message base class</td>
<td>48</td>
</tr>
<tr>
<td>MessageFactory.cpp</td>
<td>Factory class for messages</td>
<td>63</td>
</tr>
<tr>
<td>MessageFactory.h</td>
<td>Header file of the class MessageFactory</td>
<td>37</td>
</tr>
<tr>
<td>MessageReceiver.cpp</td>
<td>Implementation of the methods for message receiving</td>
<td>53</td>
</tr>
<tr>
<td>MessageReceiver.h</td>
<td>Header file of class MessageReceiver</td>
<td>57</td>
</tr>
<tr>
<td>MessageTypes.h</td>
<td>Header file of message classes</td>
<td>64</td>
</tr>
<tr>
<td>MultiThreadController.cpp</td>
<td>Implementation of the class MultiThreadController</td>
<td>648</td>
</tr>
<tr>
<td>MultiThreadController.h</td>
<td>Header file of the class MultiThreadController</td>
<td>124</td>
</tr>
<tr>
<td>ProcessedSet.cpp</td>
<td>Set of already processed tasks</td>
<td>43</td>
</tr>
<tr>
<td>ProcessedSet.h</td>
<td>Header file of class ProcessedSet</td>
<td>26</td>
</tr>
<tr>
<td>ReadyList.cpp</td>
<td>Maintains task which are ready to be processed</td>
<td>55</td>
</tr>
<tr>
<td>ReadyList.h</td>
<td>Header file of the class ReadyList</td>
<td>28</td>
</tr>
<tr>
<td>Schedule.cpp</td>
<td>Maintains the task schedule</td>
<td>78</td>
</tr>
<tr>
<td>Schedule.h</td>
<td>Header file of the class Schedule</td>
<td>36</td>
</tr>
<tr>
<td>SearchTask.cpp</td>
<td>Class SearchTask maintains scheduling specific infor-</td>
<td>201</td>
</tr>
<tr>
<td></td>
<td>mation of a search task</td>
<td></td>
</tr>
<tr>
<td>SearchTask.h</td>
<td>Header file of the class SearchTask</td>
<td>119</td>
</tr>
<tr>
<td>SearchTaskList.cpp</td>
<td>List of search tasks</td>
<td>50</td>
</tr>
<tr>
<td>SearchTaskList.h</td>
<td>Header file of the base class SearchTaskList</td>
<td>27</td>
</tr>
<tr>
<td>SearchTaskListener.cpp</td>
<td>Implementation of the methods of SearchTaskListener</td>
<td>14</td>
</tr>
<tr>
<td>SearchTaskListener.h</td>
<td>Header file of the base class SearchTaskListener</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>to be continued on next page ...</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3: Lines of code for parallel search algorithm
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task.cpp</td>
<td>Implementation of the class Task</td>
<td>26</td>
</tr>
<tr>
<td>Task.h</td>
<td>Header file of the base class Task</td>
<td>36</td>
</tr>
<tr>
<td>ValidationStack.cpp</td>
<td>Implementation of the class ValidationStack</td>
<td>25</td>
</tr>
<tr>
<td>ValidationStack.h</td>
<td>Header file of the class ValidationStack</td>
<td>19</td>
</tr>
<tr>
<td>Worker.cpp</td>
<td>Implementation of the class Worker</td>
<td>73</td>
</tr>
<tr>
<td>Worker.h</td>
<td>Header file of the class Worker</td>
<td>45</td>
</tr>
<tr>
<td>WorkerPool.cpp</td>
<td>Implementation of the worker pool</td>
<td>74</td>
</tr>
<tr>
<td>WorkerPool.h</td>
<td>Header file of the class WorkerPool</td>
<td>39</td>
</tr>
<tr>
<td><strong>Total number of LoC</strong></td>
<td></td>
<td><strong>2868</strong></td>
</tr>
</tbody>
</table>

Table 3.4: Lines of code for parallel search algorithm (continued)
Application in an Industrial Ethernet

The NDF will be evaluated by applying it in an Industrial Ethernet environment. This chapter describes this task, beginning with the choice of Industrial Ethernet to use, followed by an analysis of its requirements that must be met by a network planning tool based on the NDF, the design of the tool and its implementation.

4.1 Selection of the Application Environment

Table 4.1 lists the characteristic features for network planning of the Industrial Ethernet protocols presented in Section 2.1: Protocols are listed in columns, features or properties in rows. A cell is marked with different symbols:

- + indicates that the property is available or supported by the Industrial Ethernet protocol.

- - indicates that the property is not available or supported by the respective Industrial Ethernet protocol.

- 0 indicates that the property or feature is not applicable for this protocol or no information on this property is available for the protocol.

The properties chosen are the following:
Table 4.1: Overview: Industrial Ethernet environments

- **Switched**: The precondition for the usage of the protocol is a fully switched network.

- **Collision free**: Collisions are prevented by any mechanism or condition such as arbitration or a fully switched, full duplex infrastructure or anything else.

- **Time slicing**: The protocol uses a time slicing approach.

- **Master/Slave-oriented**: The protocol is aligned to a Master/Slave communication model.

- **Peer-to-peer available**: The protocol supports peer-to-peer communication between devices.

- **Request/Response**: The protocol also supports request/response based message passing.

- **Lightweight protocol**: The protocol is lightweight, it produces minimal overhead and in some cases, the message size is limited to a tiny amount of bytes.

- **Hierarchical structure**: The protocol or the compliance with any guideline demands a hierarchical network structure.

- **IEC 61131 or likewise**: The protocol is based on an IEC 61131 standard compliant model.
• **Cyclic traffic:** The message passing is cyclic using the given protocol.

• **Sporadic traffic:** Sporadic traffic can occur according to the given protocol or additionally to it.

• **Soft real-time:** The protocol supports soft real-time communication.

• **Hard real-time:** The protocol supports hard real-time communication.

It is clear from the table that PROFINET is the only Industrial Ethernet that meets all the selected criteria, which also explains its dominant market position. One of the main forces in the PROFINET domain is Siemens AG, with whom the Laboratory of Distributed Systems of the University of Applied Sciences Wiesbaden has worked closely in the past and from whom advice and assistance is still available. PROFINET was also the environment used in the development of the NDF. Given all these advantages, it was easy to choose PROFINET as the application environment for this work.

PROFINET provides three types of communication protocols: DCOM, RT and IRT. IRT (hard real-time) and DCOM (not lightweight and no soft real-time support) are non-standard Industrial Ethernet protocols (see Table 4.1). They will be considered in the analysis since they are typical Industrial Ethernet variants but they will not be fully designed or implemented as their contribution to the evaluation process is negligible compared to the effort of a full development.

### 4.2 Analysis

Applying the Network Design Framework to an Industrial Ethernet environment is an essential task in the evaluation of the framework in this thesis. To evaluate the framework it is important to use the features provided by the framework, as described in the Section 2.3.

Network generation for PROFINET must consider the following requirements, which are derived from the description in Section 2.1 and from the experiences of Siemens experts:
• **Provide a mapping on the NDF data model:** The tool should provide a generic model to map a given PROFINET application on the NDF data model.

• **Support different topology types:** Most of the Industrial Ethernet variants, and especially PROFINET, are based on a hierarchical model, which implies a hierarchical or tree structured network topology. According to the PROFINet Installation Guideline [Ele02] all topology types like tree, star, line or ring topologies are allowed. The guideline also states that the best way to build a network is to start on the field layer, connecting master and slaves in a PLC area, and go on building the interconnection between the PLC areas on a higher level afterwards. This in turn implies a hierarchical tree topology. In practice, line topologies are also used very often because the cabling costs can be much lower than in tree topologies in specific circumstances, for example, in production plants that have long line structures as built from conveyors.

• **Support several cabling variants:** No specific general cabling rules exist in PROFINET, but some cabling variants are applied in practice which should be supported: Laying cables along a slot along a wall or via the ceiling and along walls. These cabling variants must be supported by the network planning tool.

• **Support different materials:** Many different network components are used in PROFINET as in other Industrial Ethernet environments. The components can differ in their robustness against environmental influences (acid, liquid, dust, etc.) or in their technical features (transfer rates supported or number and type of ports in switches). The network planning tool for PROFINET must support this diversity of materials.

• **Report network properties:** The user needs a way to easily evaluate the quality of a generated network and to get an overview of the required material for installation. This extract of information must be contained in the reports provided.

The sections that follow will address the problem domains identified by the above PROFINET network generation requirements and mapping to the NDF:
• Model Mapping: How to map the PROFINET network model on the NDF model

• Tree and Line Topologies: Analysis of cost and communication properties of the network topologies to be supported.

• The Influence of Partitioning: Analysis of the influences on the solutions by partitioning during the search process.

• Network generation strategies: Analysis of the ways to realise the network generation given the limitations of the framework and the algorithms that must be encapsulated in modules.

• Approaches for NDF Search Modules: Take a closer look at the Search Modules required to realise the proposed strategies.

• Reporting and Export: Find useful reports on the generated network to enable the user to quickly determine the properties of the generated network for his environment.

### 4.2.1 Model Mapping

Mapping an abstract PROFINET application model to the NDF data model consists of the following tasks:

- Mapping the application topology consisting of IO Devices, IO Controllers and IO Supervisor to node elements in the NDF model. All attributes of a node are also available from the PROFINET application. The component type (IO Device, IO Controller and IO Supervisor) can be mapped to the type attribute of the node.

- Mapping the network load information for the RT protocol to Logical Links between the nodes. The parameters of the cyclic message exchange can be mapped to an existing protocol representative for Logical Links called Soft Real Time, which takes the cycle time and the amount of data to be transmitted to calculate the network load.
Additionally the Rack element in the NDF model can be used to define switch cabinets. PROFINET uses usual Ethernet equipment, therefore it can be easily mapped to the NDF Material List model. The environmental requirements for a network component (e.g. to keep it “inside” a safe environment such as a cabinet), as defined in the Installation Guideline, can also be defined by an appropriate attribute.

4.2.2 Tree and Line Topologies

The network generation tool must support the generation of tree and line shaped network topologies. This section analyses the properties of these topologies and searches for any boundary conditions that must be considered by network generation algorithms for PROFINET. The topologies generated will differ in Quality of Service in end-to-end latency and network load, and in cost.

4.2.2.1 Cost differences

According to the experience reported by Siemens engineers, the costs for each port in the network are nearly the same, independent of the number of ports per switch. This means that the price per port is independent of the number of the switches inserted in the network.

In line topologies the same number of ports is needed but a different amount of cable is used, therefore the cost is different. This is proven by the following procedure to transform any tree topology into a line topology without changes in the number of ports used. The process is based only on one easy transformation operation as depicted in Figure 4.1.

A line topology consists of switches connected in a line which are connected to a number of devices. There is one switch distinguished from the others on each end of the line as only one port is needed to connect to the line of switches. Starting with a tree topology such as the one in state A in Figure 4.1 with \( m = 4 \) switches, where every switch has exactly \( n = 3 \) ports, we already have a basic structure for a line topology.
Figure 4.1: Transformation of tree to line topologies

consisting of the switches at the outer upper shape of the tree, namely s1, s2 and s4, where s2 and s4 are the two ends of the line.

To get a line topology, the set of switches under the given line of switches (in the example only the switch s3) need to be moved into the line. In detail:

1. The connection between the switch (s3) and its parent (s1) is cut and a connection to an underlying device (d3) is moved instead from the switch to its parent. Afterwards two ports of the switch that must be moved in the line are now free (see state B).

2. The switch is then inserted between two switches of the line. In the example, the switch s3 is moved between s1 and s4 and it ends up in a line topology (see state C).

The first step of the operation can be seen as a remove of the switch from the connection between the device d3 and the switch s1 and the second step is a move of the switch into the connection between two adjacent switches. Thus it is only a move of the switch from one connection to another and the number of ports used is the same in both topologies. The cost only differs in the amount of cable used to establish the different topologies.

The difference in the amount of cable used in tree and line topologies depends on the position of the devices to be connected. For a set of devices arranged in a line such as
those in Figure 4.2, the tree topology wastes more cable. Take devices x and y as an example: In the line topology the route between x and y runs via a direct connection between their switches. In the tree topology the routes of both devices move away from each other first until they meet in the root switch at the top.

Therefore with respect to costs, a line topology is always the better choice for a line of devices and this is directly related to the total amount of cable used in the network. Providing line topologies for lines of devices in the automation plant demands only a consideration of the total amount of cable used in the network and there is no need to find explicit geometric structures in the set of devices. The same simple rule is valid for all types of topologies. A generic algorithm that considers the total amount of cable used for the network can automatically generate tree and line topologies where needed.

4.2.2.2 Communication differences

The quality of a generated network with respect to its communication capabilities is measured by the end-to-end latency for logical communication links between pairs of devices and the network load. The end-to-end latency is mainly related to the number of switches between the two endpoints of a logical link because the propagation time on the Ethernet cable is negligible in comparison to the latency and queuing time in switches. Therefore the analysis in this section concerning end-to-end latency is based on counting the number of switches. The network load is estimated by the number of
The analysis starts with a look at different communication scenarios in line topologies. The scenario depicted in Figure 4.3 shows a line of devices (d0 - d5) and three logical communication links (L1 - L3) each between two neighbouring devices.

Figure 4.4: Line topology for scenario 1

This is a typical scenario for a physical line topology as depicted in Figure 4.4. Any device is connected to a small switch and the switches are connected to each other to build a line. The mapping of the logical links to physical links is depicted by the lines along the physical links and via the switches in the topology.

In the line topology shown any logical link passes two switches and there are at most two logical links via one physical link, which occurs between a switch and a device only.

Figure 4.5: Tree topology for scenario 1

The corresponding tree topology is depicted in Figure 4.5. Here, there is only one hop between d2 and d3, which is one less than in the line topology. But the mappings for L1 and L3 are much worse because they now have three hops which is one more for each of them. This shows that a tree topology is definitely worse for this scenario because the average number of hops for a logical link is increased.

Figure 4.6 shows another scenario. There are two logical links over a long distance (see L1 and L3). Looking at the solution in Figure 4.7 the problems arising for line...
topologies are clear. The average number of hops for a logical link and the overall network load dramatically increase. Logical links L1 and L3 run via four switches and the whole traffic concentrates between the switches s2 and s3 which will be a potential bottleneck.

For these scenarios a solution based on a tree topology similar to that in Figure 4.8 provides much better capabilities. The short distance traffic is separated from the long distance traffic. Therefore the average number of hops and the estimated network load is lower than in line topology.

The analysis shows that no single topology fits all structures of communication relationships between nodes in the network. Therefore any algorithm used during network generation must take the communication relations under consideration.

As stated in the PROFInet Installation Guideline, a good strategy is to start with separating the communication in PLC areas from long distance communication in the automation plant. This for example might be done using the first partitioning step in the search phase. The same principle should be applied on the partitions, which might be supported by the existing communication related distance functions of the NDF.
They can be used in combination with any algorithm that deals with the distance between points to bias the distance according to the communication relationship between the points. For example to find a minimum spanning tree use the communication relationship as the distance function. The tree then reflects the most desired connections between the nodes involved.

Summary

This section demonstrated that the two topology types have their advantages and disadvantages. Line topologies are only suitable for lines of devices and only if the communication relationships correspond to that decision. Tree topologies mostly provide better communication behaviour but not in all cases. Therefore the best choice will be a generic topology biased by cost and communication relationships.

4.2.3 The Influence of Partitioning

Because a description of a PROFINET application includes defined areas of a plant - grouping several nodes under different aspects - partitioning the plant into a meaningful decomposition is easy. For example partitioning along the boundaries of PLC areas would produce sub-problems with the nodes in one PLC area. According to the PROFINet Installation Guideline, the tool can then start connecting the devices in the PLC area. Afterwards, the tool can create interconnections between the PLC areas on a higher level in the network hierarchy by inserting switches. The inserted switches can be connected on the next higher level until one single switch links all established subnets. This procedure generates a hierarchical tree topology following the recommendation in the PROFINET Installation Guideline.

The procedure contains a problem concerning switch locations. Consider the worst case scenario depicted in Figure 4.9. Suppose the best solutions for two partitions A and B have been previously calculated as follows: for partition B insert a switch in the middle of the two devices to establish a connection \( e_1 \) and \( e_2 \) between them; for partition A leave the existing subnet (the single device) as it is. The next step in
the generation process is to establish a connection between these two partitions. This can be done simply by inserting $e_3$, which represents a local optimum for the problem structure at this level.

![Diagram](image)

**Figure 4.9: Interconnection problem in tree topologies**

Taking a broader view (see Figure 4.10), it is easy to see that moving the switch in the direction of the device in partition A would lead to a better solution (the global optimum) since it requires less cable length overall.

![Diagram](image)

**Figure 4.10: Ratio between local and global optima**

Let us analyse this phenomenon. First, we will calculate the difference between the solutions with local and global optimum in this worst case scenario. A relationship between the cable length saved and the distance $x$ to the former position of the switch in Figure 4.9 can be expressed by the following function $f(x)$:

$$f(x) = x - 2(c - r)$$

(4.1)

Using Pythagoras’ Theorem, we can substitute $c$ with $c = +\sqrt{x^2 + r^2}$ (a negative distance $c = -\sqrt{x^2 + r^2}$ is of no interest here) and get a function bound to the variables $x$ and $r$ only (see Equation 4.2).
\[ f(x) = x - 2\sqrt{x^2 + r^2} + 2r \quad (4.2) \]

To find the maximum cable length saved compared to the previous solution we need to equate the first derivative of \( f(x) \) (\( f'(x) \)) to zero and solve for \( x \), which leads to Equation 4.5 representing the best position for the switch.

\[ f'(x) = \sqrt{x^2 + r^2} - 2x \quad (4.3) \]
\[ 0 = \sqrt{x^2 + r^2} - 2x \quad (4.4) \]
\[ x = \frac{r}{\sqrt{3}} \quad (4.5) \]

Applying this to \( f(x) \) we get the overall ratio dependent on \( r \) (see Equation 4.6).

\[ f\left(\frac{r}{\sqrt{3}}\right) = (2 - \sqrt{3})r \approx 0.27r \quad (4.6) \]

This means that the ratio between the local and global optimum solution is approximately 13.5% of the distance between the two devices in partition B. This factor actually exerts very little influence on the final solution since

- In most cases the number of devices in a partition will be greater than two. The ratio between local and global optimum rapidly decreases with the number of devices in the partitions.

- Looking at those partitions with more than two devices, where one or more devices are located in the opposite direction of the partition to be connected as shown in the example in Figure 4.11, the solution is similar or nearly similar to the global optimum for the two partitions. This case occurs a lot more often than the worst case analysed above.
Figure 4.11: *Example for a case with more than two devices*

- The same effect can be observed in line topologies (see Figure 4.12).

Figure 4.12: *Switch positioning in line topologies*

- The rare occurrence of direct cabling also supports our argument. In most cases, cables are laid along runways (cable channels) which lead to a grid aligned cabling variation. In this case, the solution in Figure 4.9 is valid and optimal as any movement in the direction of partition A would lead to wasted cable length as depicted in Figure 4.13.

Figure 4.13: *Non-optimum solution using grid aligned cabling*

- Partitions with very small number of devices and large distances between the devices very seldom occur. Distances are more likely to be larger between devices on higher levels in the network hierarchy (consider backbones or other kinds of interconnecting network structures). But here there will be more nodes connected to the switches because there are more ports available in a switch at this level.
Thus variations in cable length based on the differences between global and local optima for tree topologies will be in an acceptable range. Generally the best strategy to get closer to the global optimum will be to avoid too much partitioning.

The generation of a meaningful line topology is more complicated. Partitioning the problem into sub-problems reduces the complexity of the problem but restricts the view of the algorithm so it cannot find a global optimum. For example, if the algorithm does not know anything about the position of the nodes in the neighbourhood of the sub-problem currently being processed, it cannot find the correct connection between the nodes in the sub-problem to build a suitable line topology.

This problem arises when two subnets are to be connected to each other: The number of ports of a switch is limited, therefore the algorithm must reserve a certain number of ports in the subnet for the linkage to the neighbouring subnets. Because it does not know the position of the nodes in the neighbouring subnets, it can not determine which switches to reserve ports in, etc.

For example, let us compare the interconnection variants between subnets A and B and subnets B and C in Figure 4.14. Any network component in this example has two ports. The solution for subnet A is optimal based on a view restricted by the area boundaries. But the solution for subnet C is optimal if a connection to a neighbouring subnet must be established since the interconnection to B is much shorter.

![Figure 4.14: Interconnection problem in line topologies](image)

This can be a real problem for line topologies spanning multiple partitions. But because of its segmented structure longer line topologies such as the one presented very seldom occur in Industrial Ethernet. A network planner will always try to avoid routing traffic through subnets, which would be the case in such longer line topologies.
Nevertheless, there are two ways to lessen this effect:

1. Try to improve the solution: Find pairs of switches that provide the best connection from the set of switches of all the partitions involved. In a second step reconfigure the sub-networks inside the partitions, for example, by inserting additional switches or reorganising the connections to provide open ports at the switches that will connect to another partition.

2. Increase the size of partitions: This technique reduces the frequency of the problem arising.

For our purposes the first technique is not an option as it requires modifying the partial solutions during the search process. This is forbidden according to the design rules in Section 3.1.5. Changing a partial solution during the search phase is also error-prone since partial solutions influence the properties of the eventual solution. It could be allowed if the algorithm can retain the relevant properties of the discarded partial solution or if all partial solutions involved could be revalidated to get a statement about the validity of the changes to be made. But this procedure requires changes in the search process while the latter is being used in the evaluation of the framework.

Realisation of the second technique is quite simple and will be considered in the design of the network generation algorithms. Partitions that are too small can be merged or the size of partitions proposed by the algorithm can be made bigger. But increasing the partition size increases the computing effort. The question is therefore what the optimal size of a partition might be. The answer depends on the maximum time a user is likely to wait on the results of the search process. Generally it will be impossible to estimate the computing time for the whole search process dynamically because the size of the search tree is unknown. Therefore the size of partitions should be a configurable parameter.

Partitioning can reduce problem complexity but especially in the case of longer line topologies spanning multiple subnets, partitioning can adversely affect the efforts to find a global solution. Changes in the search process are required if the optimal solution is necessary. Since such special topologies play a minor role in the domain of
Industrial Ethernet, the second technique proposed here will suffice for the evaluation of the Network Design Framework.

4.2.4 Network Generation Strategy

The overall strategy to generate the network will be based on using a set of tasks enclosed in separate modules so as to fit into the network design framework. The objective of this section is to identify the modules needed to realise this strategy.

The strategy starts with an initial partitioning step. This step creates an initial set of sub-problems to reduce the complexity of the original problem. As stated in Section 4.2.2 a partitioning according to given area boundaries (especially PLC areas) is a good start. If no areas are given, a partitioning according to communication relationships can be useful (see Section 4.2.3). To avoid the problems associated with partitioning and line topologies (see Section 4.2.3) the size of partitions should be configurable.

After the initial partitioning, the search process consisting of the production, validation, merging and further partitioning tasks follows.

A simple procedure can be used to support the generation of tree topologies: Further partition the sub-problem until the number of nodes fits the number of nodes of a switch. Insert the switch at its ideal position in the sub-problem and connect all nodes to it. The inserted switch will be moved to the parent problem. If the size of the parent problem matches the number of ports of a switch, the production rule can be applied again. Otherwise, the partition will be further partitioned. To provide a line interconnection of the partitions, the production modules should be able to leave a small number of ports in designated switches open. These switches can then be connected by a generic connection algorithm (for example Minimal Spanning Tree).

The production rules should also consider the selection of the distributor types. At the lowest level in the network only machine distributors will be used. Above this, field distributors should collect the connections to the machine distributors and at the top level, a building distributor can be inserted.

Interconnection between subnets (i.e. partial solutions) can be established in several
ways. The first method is to connect them with an additional level in the network, building a tree topology. The second method is to establish a line topology on top of the subnets and the third one is to use a hybrid method of both, for example line topology on the lower level and tree topology on higher levels in the network. Therefore interoperability between the algorithms for networks on lower level and those interconnecting networks must be preserved.

A powerful validation method that provides fast estimation of the three objectives: total cost, end-to-end latency and network load, is needed to control backtracking during the search phase. The generic partitioning method proposed for the initialisation phase can be used again for further partitioning during the search phase.

Because QoS capabilities and total solution cost are competing requirements the strategy must indicate a primary optimisation objective during the search phase. This direction can be changed for sub-problems if necessary. Because of economic concerns, the algorithm here is biased towards total cost optimisation. The search process will therefore start to optimise the solution based on the total cost. If the solution conflicts with the QoS requirements, the production modules will propose alternative solutions with better QoS properties.

According to the above discussion, the following modules are needed:

- **Partitioning Modules** for partitioning by given boundaries or by communication relationship and geometric distance.
- **Production Modules** for the insertion of switches in a set of nodes and for interconnection of nodes without insertion of switches.
- **Validation Modules** for the powerful intermediate and final validation of generated solutions.

These search modules are further analysed in the following sections.
4.2.5 Approaches for NDF Modules

4.2.5.1 Partitioning Modules

Recall that partitioning is needed for two purposes:

- To separate the devices by area membership during initial partitioning
- To decompose sets of nodes according to an abstract distance function (communication relationship or geometric distance).

Area-based Partitioning

Area-based partitioning provides a fast method to separate nodes using their area membership. In most cases PLC areas provide a good separation for groups with high communication relationship. For these cases a partitioning module that partitions according to one area membership (for example the PLC area) would be enough. The appropriate algorithm is simple model-based clustering (see Section 2.2.2). Devices not assigned to an area will be assigned to the nearest group by this partitioning module.

Distance-based Partitioning

Distance-based partitioning partitions a set of nodes according to communication relationship and geometric distance. Both the "communication related" and "geometric" distance function classes are nonlinear and so grid-based clustering (see Section 2.2.2) does not meet their needs. Density-based clustering methods are also not applicable because some nodes will be left unclustered, i.e., not assigned to a partition. Hierarchical methods can be used but they have a drawback. Hierarchical clustering decomposes the given problem into a dendrogram providing several partitioning granularities. If this dendrogram is stored for later use, it could save processing time, but if just one partition is needed, a lot of processing time is wasted. Partitioning-based clustering methods provide a more iterative procedure. After a few iterations a first partition is available that can be improved with further iterations of the algorithm. The drawback
of this method is that it is difficult to determine when enough iterations have been made and whether the partitioning is of acceptable quality. This leads to nonhomogeneous partitions with subsets that have different number of nodes. Therefore, in addition to the existing partitioning modules, another partitioning module based on a modified k-means algorithm which considers the given maximum of nodes in a cluster will also be implemented.

4.2.5.2 Production Modules

The production modules must support the following tasks:

- Insertion of switches in the sets of nodes (devices or switches).
- Establishment of a generic topology for a set of nodes.
- Interconnection of given subnets via the introduction of a higher network level.

Insertion of Switches

Insertion of switches should be applicable for sub-problems if the number of nodes equals the number of ports of a switch. To get a right-sized sub-problem, the switch insertion modules should propose appropriate partitions for themselves. If such a suitable sub-problem is given the module can propose its solution.

To conform to the main optimisation objective, the switch insertion modules sort the set of available switches by their cost per port. To come up with a complete set of switches for the search process, the module starts by proposing the first switch in the list with the lowest cost per port. If the solution produced is invalid, the module proposes the next switch in the list. This guarantees that all switches are considered and that a switch with more suitable properties for the given sub-problem is found when QoS requirements are violated.

Because the three categories of switches in PROFINET must be supported on different network levels, one production module will be provided for each category, inserting switches for that category only:
• The insertion module for machine distributors inserts a machine distributor if all nodes in the given sub-problem are devices. This ensures that MDs are inserted only at the lowest level in the network hierarchy.

• The insertion module for field distributors inserts field distributors only if the nodes in the given sub-problem are machine distributors.

• The insertion module for the building distributor inserts only when the root problem is reached by the search process and therefore the highest level in the network is reached.

The final switch insertion consists of three tasks:

• locating the best switch position by using an appropriate center function for the given cabling rule,

• connecting the switches to the nodes in the sub-problem and

• moving the switch into the parent problem to establish a connection to other subnets.

All these tasks are provided by the NDF.

**Generic Algorithm**

According to the results of the analysis in Sections 4.2.2 and 4.2.3 and the decisions summarised in 4.2.4 the generic network generation algorithm required demands the following features:

• Establishment of a minimum Steiner tree (see Section 2.2) with non-fixed steiner node locations.

• Insertion of steiner nodes.

• Consideration of an abstract non-linear distance function.

• The steiner nodes have maximum degree (number of open ports).
Currently no efficient algorithm exists which solves this problem with acceptable effort. The Steiner tree problem is known to be very complex and the additional requirements given above will increase the complexity so much that there will be no chance to find a solution by simply iterating through all possibilities. The development of algorithms for such complex problems is beyond the scope of this thesis. Instead we try to find a heuristic algorithm for this task.

Human network planners cannot iterate through all the possibilities either. The network planner usually uses a straightforward method, e.g., start with small groups of nodes that have a dense communication relationship and can be connected to one single switch. Connect the nodes to the switch leaving one or two ports open to establish connections to neighbouring switches. Continue with the next group and so on. Then start connecting the switches to each other thereby establishing the network. If a conflict occurs, an attempt is made to resolve it using several alternatives depending on the type of conflict.

This approach is easily mapped to the following procedure:

1. Insert switches for matching numbers of nodes that are near each other according to an abstract distance function. This can be done by a switch insertion module such as those introduced in Section 4.2.5.2.

2. Establish a minimal spanning tree on the set of inserted switches and consider the number of free ports of a switch.

Because there is currently no known efficient algorithm to calculate a minimal spanning tree on a set of nodes with bound degree (in this example a maximum degree of two) we need an approximation. Kruskal’s algorithm [Kru56] is based on a very simple procedure. It first sorts all existing connections by distance. This sorted list is then iterated beginning with the shortest connection. If the connection does not establish a cycle in the network, it is added to the minimal spanning tree. At this decision point it is possible to additionally check if both nodes have enough free ports, to establish the given connection. Preconditions for this derived algorithm are:
• that the number of free ports in the given set of nodes is sufficient to establish a network including all nodes

• and that initially for every combination of nodes a connection is available in the sorted list.

This algorithm is not optimal but it provides an acceptable solution for small numbers of nodes with small distance values. A comparison to the optimum solutions of a brute force algorithm was made based on a simple implementation which showed an average deviation of 5% from minimum cabling costs. According to Siemens experts the cost to connect an automation device to the network is about 25 EUR, where 20 EUR is taken for the port at any switch and 5 EUR is calculated for a connecting cable of 10 meters. This means cabling costs are just under 20% of the total cost and there is only 1% deviation from the total cost when the algorithm generates a sub-optimal solution.

Interconnecting Partitions

Interconnecting partitions is very similar to establishing networks in subnets. Only a proxy coupled node of each subnet is moved to the parent problem and the production algorithm only needs to connect them in the same way as before.

A small number of ports of the proxy switch must therefore be left open, for example only one port for tree topologies and at least two ports to insert it in a line topology.

The production module must insert an additional switch to interconnect partitions with only one port left open. For a line topology, the generic production rule can be used. But any subnets generated, except the root problem, need to be interconnected on the higher level so an additional switch might be needed to act as proxy for the generated subnet in the parent problem.

Therefore an additional line producer is needed to insert a switch and connect the lines to this switch, which is moved to the parent problem afterwards.
4.2.5.3 Validation Modules

The validation algorithm only needs to validate the parameters of the objective function as the structural properties of the network are assumed to be met by the production rules. These properties include having a cycle-free network structure and having more ports than the number of physical links connected to the switch.

The validation algorithm calculates the parameters of the objective function and produces a verdict. Global parameters such as the global cost can be iteratively added with every new partial solution.

The parameters are calculated as follows:

- **Overall costs**: Sum of the costs for all physical links and switches.

- **End-to-end latency**: Sum of the latencies of the switches in the path connecting the two ends of a logical link.

- **Network load**: Calculate the network load of any physical link in the subnet by using the NDF algorithm to calculate the communication intensity.

4.2.5.4 Reporting and Export

A report is generally needed to get an extracted overview of the properties of the network or to get an export of data and import it to other software for further tasks such as analysis or writing a purchase order for the network components.

These tasks will be supported by suitable export modules:

- A simple report of a parts list of the required network components with calculated costs.

- A report providing an overview of the QoS properties end-to-end latency and network load.

- An export of network QoS properties for further analysis in a spreadsheet program.
The parts listing can be reported in a simple table, listing all types of components, the number of individual components and the cost per piece and in total.

A fast overview of the QoS properties will be provided using some kind of graphical representation. A frequency distribution looks adequate for this task because it shows the user in a single view the distribution of the end-to-end latency or network load. This is more expressive than simply showing statistics.

The export will be in a comma-separated value list as this format is supported by most spreadsheet programs.

4.3 Design

This section embodies the design of the modules to realise a PROFINET network planning tool on top of the Network Design Framework based on the results of the analysis in the previous section. It starts with the mapping tables to map a PROFINET system on the NDF data model. After that the different search modules that realise the production strategy are explained, namely the partitioning modules, the production modules and the validation module. In the last section the design of some modules for the generation of reports is presented.

4.3.1 Model Mapping Rules

This section defines how to map a PROFINET system description onto the NDF data models. The mapping rules are listed in separate tables and every table represents one part of the NDF data model.

Table 4.2 defines the mapping rules for the plant description. The left two columns reference the elements of the NDF model and their attributes. The right half lists the appropriate components in a PROFINET environment and the component properties to be set for the attributes of the proper model element.

Table 4.3 defines the mapping rules for the list of available material. Again, the left two columns reference the elements of the NDF model and their attributes and
<table>
<thead>
<tr>
<th>NDF</th>
<th>PROFINET</th>
</tr>
</thead>
<tbody>
<tr>
<td>element attributes</td>
<td>element attributes</td>
</tr>
<tr>
<td><strong>Node</strong></td>
<td><strong>IO Controller</strong></td>
</tr>
<tr>
<td>id type x,y,z</td>
<td>Generated unique id</td>
</tr>
<tr>
<td><strong>SRT Server Client data</strong></td>
<td>Logical communication link</td>
</tr>
<tr>
<td>id type x,y,z</td>
<td>Generated unique id</td>
</tr>
<tr>
<td><strong>Area</strong></td>
<td><strong>Logical areas</strong></td>
</tr>
<tr>
<td>id type x,y,z</td>
<td>Generated unique id</td>
</tr>
<tr>
<td><strong>Rack</strong></td>
<td><strong>Switch cabinet</strong></td>
</tr>
<tr>
<td>id capacity x,y,z</td>
<td>Generated unique id</td>
</tr>
</tbody>
</table>

**Table 4.2: Model mapping: Plant description**

<table>
<thead>
<tr>
<th>NDF</th>
<th>PROFINET</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SwitchType</strong></td>
<td><strong>Ethernet switch</strong></td>
</tr>
<tr>
<td>name processingTime numPorts</td>
<td>Product name</td>
</tr>
<tr>
<td>category rate cost environment</td>
<td>Average processing time [$\mu$s]</td>
</tr>
<tr>
<td><strong>CableType</strong></td>
<td><strong>Ethernet cable</strong></td>
</tr>
<tr>
<td>name rate cost environment</td>
<td>Product name</td>
</tr>
<tr>
<td>maxLength</td>
<td>Transfer rate [bps]</td>
</tr>
<tr>
<td>maxLoad</td>
<td>Cost of the switch [EUR] inside</td>
</tr>
<tr>
<td>maxTotalCost</td>
<td>Cost per meter [EUR/m] inside</td>
</tr>
<tr>
<td>maxEndToEnd</td>
<td>maximum cable length allowed [m].</td>
</tr>
</tbody>
</table>

**Table 4.3: Model mapping: Material List**

the right side lists the appropriate components in a PROFINET environment and the corresponding properties.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxEndToEnd</td>
<td>[$\mu$s]</td>
</tr>
<tr>
<td>maxLoad</td>
<td>[%]</td>
</tr>
<tr>
<td>maxTotalCost</td>
<td>[EUR]</td>
</tr>
</tbody>
</table>

**Table 4.4: Model mapping: Units for requirements**

Table 4.4 contains the units used for the requirements.

The output data needs no mapping on the PROFINET environment. The generated network simply contains the network components inserted from the material list, and
the units of network properties calculated depend on the units used in the input data and are the same as for the requirements in Table 4.4.

4.3.2 Partitioning Modules

Two partitioning modules will be developed (see Section 4.2.5.1 with partitioning based on given area boundaries or based on an abstract distance function such as the geographic distance or communication relationship of nodes.

Figure 4.15 depicts the UML class diagram for all area partitioning modules. The base class AreaPartitioningModule provides the common properties and methods of the area partitioning modules. It is derived from the NDF base class for partitioning modules called the PartitionModule (see 2.3). The partitionByArea() method partitions the given set of nodes by a given area type. Nodes that are not member of an area of the given type are assigned to the nearest partition afterwards. The involvedNodes method sums the nodes, which are part of an area of a defined type. This method is used to determine the rate value, which expresses the ability of the module to partition a given problem.

![UML class diagram: Area partitioning modules](image)

For each of the defined area types of PROFINET systems, one derived class provides
the specific area partitioning module:

- EmergencyAreaPartitioningModule
- OperationModeAreaPartitioningModule
- HMIPartitioningModule
- PLCAreaPartitioningModule

The specific area partitioning modules implement the abstract methods of the base class for partitioning modules, like `rate()` and `partition()` based on the methods of the base class `AreaPartitioningModule`.

A distance-based partitioning module called MinDistClusterPartitioner will also be realised with a modified k-means algorithm (see Figure 4.16). The module is derived from the NDF class `PartitioningModule` that is the base class of all partitioning modules. The distance in the algorithm will be calculated by the appropriate distance function for the defined cable rule, which considers the communication intensity between the nodes involved.

![UML class diagram: Distance-based partitioning module](image)

**Figure 4.16: UML class diagram: Distance-based partitioning module**

The algorithm starts with a partition of the given set of nodes in clusters with exactly n nodes. For each cluster a center is determined based on the given distance function. Afterwards these clusters are optimized by swapping nodes between neighbouring clusters. Neighbouring clusters are identified by the distance of their cluster
centers. The nodes to be swapped are identified based on an evaluation function, evaluating the new state of the clusters to be reached after swapping. The evaluation function calculates the overall distance of the members of the clusters to their center. A better partition is reached if the overall distance is lower.

4.3.3 Production Modules

The following production modules will be realised (see Section 4.2.5.2):

- Switch insertion modules for Machine Distributors, Field Distributors and Building Distributors.
- An interconnecting production module that produces a minimal spanning tree on top of a set of distributors.

Three types of switch inserting production modules are defined:

- MDInserter to insert machine distributors
- FDInserter to insert field distributors
- BDInserter to insert building distributors

All of these production modules are designed as depicted in Figure 4.17. They are derived from two classes: the NDF base class `ndf::ProductionModule` for all production modules and the base class called `InserterBase`, which provides generic function for the switch insertion, namely:

- `makeProposal()` creates a proposal for a given switch type and cabling rule.
- `produceSolution()` generates a solution using a given switch type and cabling rule.

`InserterBase` itself is derived from the two classes: `SwitchInserter` and `CableInstallerMngmt`. The class `SwitchInserter` provides methods to initialise (`initSwitchList()`) and manage (`getNextSwitchType()`) the subset of switches
Figure 4.17: **UML class diagram: Switch inserting production module** MDInserter

of a given distributor type and to insert a switch in a given set of nodes (insertCoupleNode()). The class CableInstallerMngmt provides a method to retrieve the right cable installation algorithm for a given globally defined cabling rule.

The interconnecting production module is called **ComposingProducer** because it composes a solution for a set of existing switches without inserting additional switches. As depicted in Figure 4.18 the ComposingProducer, derived from the NDF base class ndf::ProductionModule, uses an algorithm class called MSTCapacitatedKruscal that can calculate the minimal spanning tree (calculateMST()) of a given set of nodes. ComposingProducer also makes use of the CableInstallerMngmt class to get access to the appropriate NDF cable installer class for the globally defined cabling rule.

### 4.3.4 Validation Module

The validation module is realised in the class AnalyticValidation.

The validation module performs a validation based on simple but fast calculations. It uses the ndf::CommunicationIntensity algorithm class to calculate the net-
Figure 4.18: UML class diagram: Production module ComposingProducer

Figure 4.19: UML class diagram: Validation module AnalyticValidation
work load induced by the logical links. The end-to-end-latency is calculated by the
fixed average latency of the switches realising the physical connection between the
two ends of a logical link.

The class AnalyticValidation implements the two validation methods of the
base class ndf::ValidationModule called intermediateValidation() and finalValidation(). It is therefore able to do both intermediate validation of
the partial solutions and final validation of the whole generated network.

4.3.5 Export and Reporting

Three report modules will be realized (see Section 4.2.5.4): CSVExport, PartsListRe-
port and QoSReport.

Figure 4.20: UML class diagram: Export module CSVExport

The module CSVExport (see Figure 4.20) enumerates the entities in the solution
and exports the information as a line of comma separated values. The elements of the
collected information are listed in Table 4.5.

The line is appended to a file so data from multiple runs can be stored in one file. A
property in the tool configuration file defines the file name for the export. If the file
does not exist, the module automatically creates a table header with a description for
every value.

Figure 4.21 depicts the UML diagram of the class PartsListReport. This class
generates a HTML file with a parts list, organised in a HTML table.
The table contains

- for every material type in the material list the number of pieces (or meters in the case of cables) to realise the generated network plan
- the costs per position and
- the overall costs for all network components.

Finally, the class \texttt{QoSReport} (see Figure 4.22) realises the report module for the histograms of network QoS properties as described in Section 4.2.5.4.

The frequency distributions are calculated for end-to-end latency of logical links and network load of physical links. The generated report is a HTML page containing two diagrams with the calculated frequency distributions based on SVG graphics.
4.4 Implementation

This section contains some implementation-specific information related to the generation of proposals and the prevention of cycles in the search process. An overview of the programming effort is also given. Because all the implemented algorithms used were platform independent no additional libraries were needed.

Our strategy is to start with the cheapest solution possible. Hence the production module sorts the available switch types by their costs and starts by proposing a solution using the cheapest configuration of switches (assuming the number of available ports is sufficient). Each time a partial solution is discarded and the production module is notified, the module creates a new proposal using the next switch in the list. The module uses a ModuleData object that is attached to the description of the sub-problem (e.g., an instance of class Problem) to store the last index in the list of switch types. The evaluation of proposals is cost related and the best proposal gets the highest value. Therefore the actual value is calculated as $INT_{MAX} - \text{cost}$.

Because production modules are allowed to merge a sub-problem with its parent problem the process can run in an endless cycle of partitions and merges while apparently doing progress. To prevent such situations the parent problem is marked as changed if a sub-problem is transformed in a partial solution. If a production module would like to propose a merge with the parent, it first checks whether the current sub-problem is marked as changed, e.g., if any sub-problem of this problem has been changed and the current solution is thereby already in another state. This prevents pro-
<table>
<thead>
<tr>
<th>File</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>AreaPartitioningModule.cpp</td>
<td>295</td>
</tr>
<tr>
<td>AreaPartitioningModule.h</td>
<td>68</td>
</tr>
<tr>
<td>MinDistCluster.cpp</td>
<td>89</td>
</tr>
<tr>
<td>MinDistCluster.h</td>
<td>45</td>
</tr>
<tr>
<td>MinDistPartitioningModule.cpp</td>
<td>147</td>
</tr>
<tr>
<td>MinDistPartitioningModule.h</td>
<td>69</td>
</tr>
<tr>
<td>EmergencyAreaPartitioningModule.cpp</td>
<td>116</td>
</tr>
<tr>
<td>EmergencyAreaPartitioningModule.h</td>
<td>29</td>
</tr>
<tr>
<td>HMI AreaPartitioningModule.cpp</td>
<td>115</td>
</tr>
<tr>
<td>HMI AreaPartitioningModule.h</td>
<td>28</td>
</tr>
<tr>
<td>InitialPLCAreaKMeansClusterPartitioningModule.cpp</td>
<td>65</td>
</tr>
<tr>
<td>InitialPLCAreaKMeansClusterPartitioningModule.h</td>
<td>16</td>
</tr>
<tr>
<td>InitialPLCAreaPartitioningModule.cpp</td>
<td>82</td>
</tr>
<tr>
<td>InitialPLCAreaPartitioningModule.h</td>
<td>29</td>
</tr>
<tr>
<td>InitialPriorityPartitioningModule.cpp</td>
<td>137</td>
</tr>
<tr>
<td>InitialPriorityPartitioningModule.h</td>
<td>41</td>
</tr>
<tr>
<td>OperationModeAreaPartitioningModule.cpp</td>
<td>117</td>
</tr>
<tr>
<td>OperationModeAreaPartitioningModule.h</td>
<td>28</td>
</tr>
</tbody>
</table>

| Total number of LoC | 1516 |

Table 4.6: Lines of code for partitioning modules

...duction modules from proposing merges when the current problem has just been cut from the parent.

The programming effort in lines of code for all implemented sources of the extensions are listed in tables 4.6, 4.7, 4.8 and 4.9). The total number of lines of code is 4678.
<table>
<thead>
<tr>
<th>File</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CableInstallerMngmt.cpp</td>
<td>24</td>
</tr>
<tr>
<td>CableInstallerMngmt.h</td>
<td>17</td>
</tr>
<tr>
<td>ComposingProducer.cpp</td>
<td>250</td>
</tr>
<tr>
<td>ComposingProducer.h</td>
<td>67</td>
</tr>
<tr>
<td>FDInsert.cpp</td>
<td>70</td>
</tr>
<tr>
<td>FDInsert.h</td>
<td>32</td>
</tr>
<tr>
<td>FDLineProducer.cpp</td>
<td>66</td>
</tr>
<tr>
<td>FDLineProducer.h</td>
<td>30</td>
</tr>
<tr>
<td>InserterBase.cpp</td>
<td>190</td>
</tr>
<tr>
<td>InserterBase.h</td>
<td>42</td>
</tr>
<tr>
<td>LineProducerBase.cpp</td>
<td>292</td>
</tr>
<tr>
<td>LineProducerBase.h</td>
<td>88</td>
</tr>
<tr>
<td>LineProducer.cpp</td>
<td>57</td>
</tr>
<tr>
<td>LineProducer.h</td>
<td>46</td>
</tr>
<tr>
<td>MDInsert.cpp</td>
<td>67</td>
</tr>
<tr>
<td>MDInsert.h</td>
<td>31</td>
</tr>
<tr>
<td>MDLineProducer.cpp</td>
<td>64</td>
</tr>
<tr>
<td>MDLineProducer.h</td>
<td>31</td>
</tr>
<tr>
<td>SwitchInserter.cpp</td>
<td>147</td>
</tr>
<tr>
<td>SwitchInserter.h</td>
<td>45</td>
</tr>
<tr>
<td>Total number of LoC</td>
<td>1656</td>
</tr>
</tbody>
</table>

Table 4.7: Lines of code for production modules

<table>
<thead>
<tr>
<th>File</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnalyticValidation.cpp</td>
<td>460</td>
</tr>
<tr>
<td>AnalyticValidation.h</td>
<td>67</td>
</tr>
<tr>
<td>MappingTable.cpp</td>
<td>55</td>
</tr>
<tr>
<td>MappingTable.h</td>
<td>41</td>
</tr>
<tr>
<td>Total number of LoC</td>
<td>623</td>
</tr>
</tbody>
</table>

Table 4.8: Lines of code for partitioning modules

<table>
<thead>
<tr>
<th>File</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSVExportModule.cpp</td>
<td>198</td>
</tr>
<tr>
<td>CSVExportModule.h</td>
<td>97</td>
</tr>
<tr>
<td>PartsListReport.cpp</td>
<td>158</td>
</tr>
<tr>
<td>PartsListReport.h</td>
<td>38</td>
</tr>
<tr>
<td>QoSReport.cpp</td>
<td>321</td>
</tr>
<tr>
<td>QoSReport.h</td>
<td>71</td>
</tr>
<tr>
<td>Total number of LoC</td>
<td>883</td>
</tr>
</tbody>
</table>

Table 4.9: Lines of code for export modules
Chapter 5

Evaluation

This chapter is about the evaluation of the framework and the results achieved by the task of this thesis, i.e., parallelisation of the search process and application of the network planning tool for PROFINET network planning. The evaluation targets five evaluation objectives:

- The quality of generated solutions
- The performance of the search process
- The speedup reached by parallelisation
- The usability for end users
- The adequacy of the framework design for the network planning task.

The chapter starts with a section on the development of the required tools and the evaluation procedures for the different evaluation objectives. The subsequent sections summarise the results of the evaluation.

5.1 Evaluation Process and Supporting Tools

Performance measurements are needed for performance evaluation and speedup analysis. To get an overview of the performance characteristics of the search process and its
relationship to the way the search process performs its task, several input parameters should vary over the test sequences, for example, the structure of the problem instance (number and position of nodes and the number of logical links between the nodes), the kind of tool configuration (e.g. number and collection of search modules) and so on. This procedure demands a test problem generator to generate random problem instances in a meaningful manner. It should be possible to generate realistic plant descriptions with nodes distributed over the plant which are logically connected to each other given some defined logical areas.

A suitable environment is needed to automate the execution of the planning tool with the different configurations and to collect the results. One only needs to enter a given directory and run the planning tool with the given set of input files in that directory. This ensures that outputs are always collected with the right input files in the same directory.

The environment described can be used to evaluate the performance, speedup and the quality of the solution. The speedup evaluation will be achieved by comparing the performance of single task runs and multitask runs. Therefore the number of parallel tasks is another important parameter to be varied. Ideally, the quality of the solution should be compared to an optimum solution but finding the optimum solution is not feasible for bigger problems due to the problem complexity. Therefore it is important to be able to approximate the properties of the optimum solution.

### 5.1.1 Problem Generator

The problem generator is needed to generate meaningful random plant descriptions as input to the network generation tool. The generated plant should be constructed as realistically as possible.

An automation plant consists of a set of nodes distributed over the whole plant and structured by more or less separate areas. The areas are dependent on the machinery structure, in most cases square- or ellipsoid-shaped and of different sizes. The devices in the shapes are not always evenly distributed. Especially in a generic model of line
topologies, nodes are more likely to be located near the line (see example in Figure 5.1). The generator therefore creates node areas of different sized squares and ellipsoids supporting different probability functions to control the distribution of the nodes inside a given shape.

The geometric boundaries and the overall number of nodes in the plant is configurable and controls the size of the problem instance for the evaluation runs. The number of nodes in an area or alternatively the number of areas is another parameter that is configurable. Other parameters control the amount of internal communication between nodes within areas and external communication between nodes in different areas separately in terms of logical links and their network load parameters.

5.1.2 Automated Execution

The execution of a set of prepared problem instances is needed for certain tasks of the evaluation. Prepared problem instances consist of the plant description generated by the problem generator and additional configuration input to the network generation tool: the material list, the requirements and the tool configuration.

Automated execution provides post processing in order to analyse the newly produced results for a problem instance. Such post processing can be supported by a script that will be triggered by the tool after each run of the network planning tool.

The execution environment should provide a log of the automated execution to keep
track of the progress.

5.1.3 Optimum Solution Generation

Because an efficient algorithm for the generation of the optimum solution is unknown, a brute force algorithm is developed here to perform this task. The task is, in detail, to find an optimal set of switches to be inserted in a set of nodes and a network connecting all switches and nodes to each other fulfilling the given requirements regarding cost, end-to-end latency and network load, generally optimising towards minimal costs. The algorithm will enumerate all possible network structures, calculate the properties and select the optimum solution by comparing the property values.

An optimal set of switches of the same type can be derived from the rule of equivalent port usage for line and tree topologies, depicted in Figure 4.1 in Section 4.2.2.1. The number of ports needed to establish a line topology equals the number of ports of any tree topology for the same set of nodes and switches. The number of ports needed for a line topology equals the number of nodes to be connected to the line plus the number of physical links to establish the line between the switches (refer to Figure 5.2 for an example). To establish the line of switches, excluding the two switches at the two ends of the line, every switch uses two ports to connect to a neighbouring switch, so only \( p - 2 \) ports of the switches are left to connect the terminal nodes to the switches. Therefore the formula for the number of switches needed for a given set of terminal

![Figure 5.2: Example of a line topology](image)
nodes is:

\[ f(p, n) = \begin{cases} 
  0 & \text{if } n \in \{1, 2\} \\
  \frac{n-2}{p-2} & \text{if } n \geq p \geq 3
\end{cases} \]  

(5.1)

where \( p \) is the number of ports available at a switch and \( n \) is the number of nodes to be connected by the line topology. The two ports, which must be left open at the two ends of the line are considered in the term \( n - 2 \), which reduces the total number of nodes to be connected by the line topology for the given problem by two.

Given an optimum set of switches the task of enumerating all possible networks can be achieved by the following procedure:

1. Enumerate all networks for the switches only.

2. For any established network of switches, enumerate all possible assignments of nodes to switches. Consider the number of ports left open.

3. For all established networks with switches and nodes connected, determine the optimum position for all switches.

A suitably efficient implementation of this brute force procedure was developed for the thesis (see Appendix B for more details).

5.1.4 Estimating the Properties of an Optimum Solution

One way to estimate the properties of an optimum solution is to formulate approximate equations based on the solutions for smaller problem sizes to express relationships between required output and given input parameters such as number of nodes, number of logical links and so on. But this method will be very imprecise when a large number of independent variables is involved.

To obtain better estimates for bigger problem sizes, a Monte Carlo Simulation was employed. All possible solutions for small problem sizes are enumerated and their properties are analysed and stored in a histogram. From the histogram, a normal distribution will be approximated and the position of the optimum solution in relation to
statistics such as the average and the standard deviation will be determined. According to the principles of the Monte Carlo Simulation, the distribution of the properties for larger sized problems can be estimated using a smaller number of samples. Based on this distribution and the average and the standard deviation for this set of samples, an estimation for the properties of the optimum solution should be possible.

Applying this approach to the network planning problem failed in the given time frame. Unfortunately the shape of the distribution of the properties varies very much depending on the size and the structure of the problem and there is no obvious relationship between the input parameters and the properties. Therefore another approach was used to evaluate the quality of the solution generated for problems with bigger number of nodes. This is based on generating lower bounds estimates on the properties of the generated solution.

A NDF extension was developed for this purpose. It calculates lower bounds for the number of switches to be inserted, the overall length of cable to be used and the overall costs as follows:

- Cabling: The least possible length of cable is estimated by a minimal spanning tree for the devices in the problem description.

- Switches: the smallest number of switches is estimated by a heuristic using the least number of machine distributors on field level and the least number of field distributors to connect them to each other. This estimate considers the planning of tree and line network topologies.

- Costs: The cost estimate considers tree and line topologies. The costs depend on the number of switches to be inserted (see previous bullet) and the costs of the cables needed to establish a network.

- Load: The lowest possible network load is calculated at the device with the highest network load at its logical links.
5.2 Speedup from Parallelisation

This section discusses the measurements carried out for the evaluation of the proposed parallel network generation algorithm. The algorithm was evaluated by determining the speedup compared to the single threaded version of the network planning tool.

Measurements were made for the following two cases:

- **Forward only:** This is the case where the solution is found on the first path through the search tree - the case that was intended to be optimized by the chosen parallelisation approach.

- **Backtracking:** To evaluate the assumptions made in the analysis of the parallelisation approach, the speedup in the backtracking case was also measured.

5.2.1 Measurement Environment

All measurements were made on a personal computer with the following properties:

<table>
<thead>
<tr>
<th>Processor</th>
<th>Intel Core 2 Quad (Q8200) with 2.33GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front side bus</td>
<td>1333MHz</td>
</tr>
<tr>
<td>Main memory</td>
<td>4GB</td>
</tr>
<tr>
<td>Operating System</td>
<td>Linux (Ubuntu), Kernel 2.6.27-9-generic (optimized for i686)</td>
</tr>
</tbody>
</table>

Table 5.1: Hardware

The measurements were made using the following set of modules for the generation of automation networks:

- **Partitioning Modules**
  - InitialPLCAreaPartitioningModule for initial partitioning according to the given PLC area boundaries
  - KMeansPartitioningModule for generic partitioning

- **Production Modules**
  - MDInserter to insert machine distributors in sets of automation devices
- MDLineProducer to establish connected lines of up to four machine distributors
- FDIInserter to insert field distributors connecting lines of machine distributors
- FDLineProducer to establish lines of field distributors
- A ComposingProducer to establish minimal spanning trees for sets of nodes with free ports

- Validation Modules
  - The AnalyticValidation validation module for interim and final validation

For the proposed algorithm, the speedup is influenced mainly by two factors: the number of tasks available for processing at a certain time in the schedule and the load generated by the processing of a task. The number of tasks depends on the number of sub-problems generated by partitioning, and the processing load depends on the size of the sub-problem. Therefore a set of different problem instances was created for the measurements varying the size of the problem (comp. Table 5.2) and the number of devices in a PLC area (comp. Table 5.3).

<table>
<thead>
<tr>
<th>Configuration identifier</th>
<th>Size [amount of nodes]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>500</td>
</tr>
<tr>
<td>2</td>
<td>1000</td>
</tr>
<tr>
<td>3</td>
<td>2000</td>
</tr>
</tbody>
</table>

Table 5.2: Problem sizes

<table>
<thead>
<tr>
<th>Devices per area</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>50</td>
</tr>
</tbody>
</table>

Table 5.3: Devices per area

All combinations of problem sizes and number of devices per area were tested using the following process controller configurations:
All unnecessary processes and daemons on the system were shut down to prevent unintentional influences by the system.

### 5.2.2 Speedup for the Forward-Only Case

All measurement results were calculated over 50 runs to increase the precision. The speedup was calculated by the formula given in Section 2.4.3 with the duration of the search process of the single threaded process used as bias. The standard deviation for each value is depicted with error bars in the line graphs and was calculated by the formula given in Equation 5.3 ($E(p)$). Both the deviation of the search time of the single threaded process controller ($e(0)$) and that of the multithreaded process controller ($e(p)$ with $p$ indicating the number of worker threads) in its particular configuration (e.g. with one, two, three and four worker threads) were taken into account. The formula considers the fact, that the calculated speedup is actually a sum of the real speedup $S(p)$ and an deviation interpreted as error $E(p)$ derived from the deviations of the calculated averages for the single threaded process $T(0)$ and the multithreaded process $T(p)$ during speedup calculation (cp. Equation 5.2).

$$S(p) \pm 1/2E(p) = \frac{T(1) \pm 1/2e(1)}{T(p) \pm 1/2e(p)}$$

$$\pm E(p) = 2\left(\frac{T(1) \pm 1/2e(1)}{T(p) \pm 1/2e(p)} - S(p)\right)$$

### Table 5.4: Devices per area

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>The single threaded process controller as reference</td>
</tr>
<tr>
<td>M1</td>
<td>The multithreaded process controller with one thread</td>
</tr>
<tr>
<td>M2</td>
<td>The multithreaded process controller with two threads</td>
</tr>
<tr>
<td>M3</td>
<td>The multithreaded process controller with three threads</td>
</tr>
<tr>
<td>M4</td>
<td>The multithreaded process controller with four threads</td>
</tr>
</tbody>
</table>
The measurement results for the different configurations are given in the next three line graphs (see Figures 5.3, 5.4 and 5.5). The single threaded process (S) is always indicated by the red line with speedup 1. The multithreaded versions with one to four processors are indicated with the identifier given in Table 5.4. The total number of nodes is indicated by the horizontal axis and the speedup is given by the vertical axis. The multithreaded controller with one worker thread (M1) always marks the lower boundary for the speedup.

![Graph showing speedup with 10 devices per area](image)

Figure 5.3: *Speedup with 10 devices per area*

In the first configuration set with 10 devices per area (see Figure 5.3), the speedup for multithreaded process is below 1, which means that the processing time is higher than without parallelisation.

In the second configuration set with 20 devices per area (see Figure 5.4), the speedup is also low, but in most cases is greater than or equal to the single threaded process. It also shows that the speedup seems to be a bit higher if the size of the problem is bigger.

With 50 devices per area (see Figure 5.5) the speedup is greater than 1 in all cases when the multithreaded process controller has more than one worker thread in the
Figure 5.4: *Speedup with 20 devices per area*

Figure 5.5: *Speedup with 50 devices per area*
thread pool. The highest speedup is reached with the largest problem size and the largest number of devices per area. This usually increases the number of available tasks and the processing load per tasks. The best speedup is around 1.7, which is still poor.

The reason for the poor speedup rate is the bigger control overhead compared to the parallelisable processing effort for transformation tasks in the search process. For each task there is a non-parallelisable effort for scheduling and backtracking control purposes. The validation, also performed by the control thread, runs in parallel to the search tasks. If the control overhead is bigger than the processing effort for a search task, the time savings compared to M1 are low.

The current production modules are still very simple and therefore lightweight. More complex production modules, which accept larger amounts of nodes in a sub-problem and therefore produce better overall solutions, are possible. With these production modules, the processing effort per task will rise while the overall control overhead goes down. Therefore it was of interest to measure the speedup for such heavy-weighted production modules simulated in the analysis by synthetically generated load using a function that wasts a defined amount of instructions without operational effect. To determine the minimum possible overhead, the processing effort for a production module was continually increased with every test run until the difference of the search time with the current load to the previous search time was around zero, i.e.:

\[ 0 \approx (t_{S(i)} - t_{M1(i)}) - (t_{S(i+1)} - t_{M1(i+1)}) \]

where \( t_{S(i)} \) indicates the search time for the single threaded process in iteration \( i \) and \( t_{M1(i)} \) the search time for the multithreaded process in the same iteration \( i \).

Table 5.5 lists the measurement and calculation results to determine the ideal load to get minimal overhead. The columns contain the following values (left to right order):

- **load**: This is the processing effort for a run without artificially increasing the load (e.g. \( load = T_m(n + 1)/T_m(1) | n = row \)). In the first row, the original pro-
cessing effort is given, therefore the relational load is 1.0. The artificial load has been increased up to 402 times of the original load (see last row, first column).

- $T_m$: the average processing time for a production step in the single threaded version of the network planning tool

- $T_S$: the overall processing duration for the single threaded version of the network planning tool

- $T_{M1}$: the overall processing duration for the run using the multithreaded version of the planning tool with only one worker thread

- $O_A$: the absolute overhead calculated by the formula $O_A = T_S - T_M$.

- $O_R$: the relative overhead to the whole processing time as calculated by $O_R = \frac{O_A}{T_S}$.

Table 5.5: Control overhead in the worst case

<table>
<thead>
<tr>
<th>load</th>
<th>$T_m$ [s]</th>
<th>$T_S$ [s]</th>
<th>$T_{M1}$ [s]</th>
<th>$O_A$ [s]</th>
<th>$O_R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>0.00000784127</td>
<td>0.008521</td>
<td>0.012486</td>
<td>0.003965</td>
<td>0.465321</td>
</tr>
<tr>
<td>10.228751210</td>
<td>0.0000802064</td>
<td>0.017907</td>
<td>0.021911</td>
<td>0.004004</td>
<td>0.223600</td>
</tr>
<tr>
<td>46.831954518</td>
<td>0.000367222</td>
<td>0.054837</td>
<td>0.059305</td>
<td>0.004468</td>
<td>0.081478</td>
</tr>
<tr>
<td>55.783438142</td>
<td>0.000437413</td>
<td>0.063737</td>
<td>0.068669</td>
<td>0.004932</td>
<td>0.077380</td>
</tr>
<tr>
<td>61.550616163</td>
<td>0.000482635</td>
<td>0.072856</td>
<td>0.078064</td>
<td>0.005208</td>
<td>0.071483</td>
</tr>
<tr>
<td>73.049646295</td>
<td>0.000572802</td>
<td>0.080352</td>
<td>0.086961</td>
<td>0.006609</td>
<td>0.082251</td>
</tr>
<tr>
<td>80.878480144</td>
<td>0.00063419</td>
<td>0.089558</td>
<td>0.096294</td>
<td>0.006736</td>
<td>0.075214</td>
</tr>
<tr>
<td>81.787389033</td>
<td>0.000641317</td>
<td>0.097495</td>
<td>0.104903</td>
<td>0.007408</td>
<td>0.075983</td>
</tr>
<tr>
<td>89.158899005</td>
<td>0.000699119</td>
<td>0.105785</td>
<td>0.113771</td>
<td>0.007986</td>
<td>0.075493</td>
</tr>
<tr>
<td>97.784415025</td>
<td>0.000766754</td>
<td>0.114768</td>
<td>0.123182</td>
<td>0.008414</td>
<td>0.073313</td>
</tr>
<tr>
<td>113.391708231</td>
<td>0.000889135</td>
<td>0.121117</td>
<td>0.131186</td>
<td>0.010069</td>
<td>0.083134</td>
</tr>
<tr>
<td>114.686268933</td>
<td>0.000899286</td>
<td>0.130402</td>
<td>0.139608</td>
<td>0.009206</td>
<td>0.070597</td>
</tr>
<tr>
<td>119.313708111</td>
<td>0.000935571</td>
<td>0.137683</td>
<td>0.148390</td>
<td>0.010707</td>
<td>0.077766</td>
</tr>
<tr>
<td>157.897891541</td>
<td>0.00123812</td>
<td>0.177231</td>
<td>0.187960</td>
<td>0.010729</td>
<td>0.060537</td>
</tr>
<tr>
<td>242.761695491</td>
<td>0.00190356</td>
<td>0.258360</td>
<td>0.269420</td>
<td>0.011060</td>
<td>0.042808</td>
</tr>
<tr>
<td>321.053859898</td>
<td>0.00251747</td>
<td>0.340131</td>
<td>0.350611</td>
<td>0.010480</td>
<td>0.030812</td>
</tr>
<tr>
<td>402.408028291</td>
<td>0.00315539</td>
<td>0.419488</td>
<td>0.431319</td>
<td>0.011831</td>
<td>0.028203</td>
</tr>
</tbody>
</table>

The estimates were made using a problem with 100 nodes and 10 nodes per area. All absolute times are averages over 100 runs. As the table shows, at about 113 times the original overhead, the absolute overall overhead $O_A$ is fixed at around 0.01 seconds.
which means that the ideal load is reached at this point. Because a factor of 113 seems a bit high the result was analysed further.

The production modules in this scenario were very simple: connect a given set of nodes to a switch and position it at the right location, roughly summarized. Production modules that consider larger sets of nodes (bigger parts of the problem) and therefore produce better overall solutions will need more complex network generation algorithms. Assuming a runtime complexity of \( O(n^2) \) with \( n \) as the number of nodes, which is quite good for a network generation algorithm, the ideal load for a production module no longer looks unrealistic. If this algorithm needs four nodes nearly the same time, as the current simple production module consumes, a 113 times increased load would correspond to around \( \sqrt{113} = 10.63 \) times of the number of nodes (e.g. around 42 nodes). The formula for this estimate is given in Equation 5.7 with \( T(n) \) as time function for \( n \) nodes with a constant time factor \( t \).

\[
T(n) = n^2 \cdot t \text{ for } 1t = \frac{1}{4^2} \tag{5.5}
\]

\[
\frac{T(n)}{T(x \cdot n)} = \frac{(x \cdot n)^2 \cdot t}{n^2 \cdot t} = 113 \tag{5.6}
\]

\[
x = \sqrt{113} = 10.63 \tag{5.7}
\]

With a minimal overhead, the speedup was significantly higher than expected and up to 3.3 in the best case, which is near the ideal speedup of four (for four processors).
Figure 5.6: Speedup with 10 devices per area and minimum control overhead

Figure 5.7: Speedup with 20 devices per area and minimum control overhead
5.2.3 Speedup in the Backtracking Case

Measurement of the speedup during backtracking is a bit complicated. Basically two distinct cases will be analysed: (1) the worst case where the full search tree is enumerated and no solution is found and (2) the case where there is a minor amount of backtracking, which will occur more often.

Full enumeration of the search tree means enumerating all solutions for a problem provided by the given set of modules. This is in most cases impossible, because the intermediate validation interrupts the solution due to conflicts with the requirements before the last switch has been connected. Therefore a pseudo validation module has been introduced which produces the same processing load but accepts all partial solutions except the last one, which would solve the root problem in the problem tree.

The goal of these measurements is to get the average speedup for the parallel search process with backtracking. The analysis showed that the speedup in the backtracking case varies depending on the level where the backtracking step occurs. The full enumeration of the search tree runs through all levels of backtracking of the tree and the
speedup for the full enumeration therefore provides an average value for a backtracking step occurring anywhere in the search tree.

Because the full enumeration is a synthetic scenario some measurements with minor backtracking were made for comparison. The number of production and backtracking steps were counted in both cases to get an indication of the relation between the speedup for full enumeration and that for more realistic backtracking cases.

Full enumeration of huge problems needs too much time. Therefore a small problem with 18 nodes in 2 areas was used to demonstrate the behaviour. The network planning tool was configured to use 5 different production modules. Therefore in every node in the search tree (each existing sub-problem) a minimum of 5 child nodes exists. A possible complexity estimation for the resulting search tree considered the number of transformations processed during the full enumeration.

<table>
<thead>
<tr>
<th>config</th>
<th>forward</th>
<th>backward</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6899</td>
<td>6899</td>
<td>0.834646</td>
<td>0.768621</td>
<td>0.767224</td>
<td>0.764470</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>0.934672</td>
<td>0.278785</td>
<td>0.306174</td>
<td>0.271280</td>
</tr>
<tr>
<td>2</td>
<td>419</td>
<td>419</td>
<td>0.819508</td>
<td>0.599327</td>
<td>0.583814</td>
<td>0.563229</td>
</tr>
<tr>
<td>3</td>
<td>1021</td>
<td>1021</td>
<td>0.793275</td>
<td>0.669235</td>
<td>0.664048</td>
<td>0.644958</td>
</tr>
<tr>
<td>4</td>
<td>494</td>
<td>494</td>
<td>0.807281</td>
<td>0.757536</td>
<td>0.850298</td>
<td>0.775307</td>
</tr>
<tr>
<td>5</td>
<td>86</td>
<td>66</td>
<td>0.899061</td>
<td>0.884642</td>
<td>0.937366</td>
<td>0.911406</td>
</tr>
<tr>
<td>6</td>
<td>22</td>
<td>2</td>
<td>0.953027</td>
<td>1.200207</td>
<td>1.399035</td>
<td>1.419852</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>0</td>
<td>0.954140</td>
<td>1.191468</td>
<td>1.435283</td>
<td>1.461052</td>
</tr>
</tbody>
</table>

Table 5.6: Speedup in case of backtracking

Table 5.6 lists the measurement results. The columns in left to right order, contain the following values:

- **config**: the configuration number used for the measurement run. The first configuration (0) used the pseudo validation module which results in a full enumeration. All remaining runs used the normal validation module with different maximum end-to-end latency requirements. These are:

- **forward**: the number of transformation steps (partition, production, merge)

- **backward**: the number of backtracking steps occurred during search process

- **M1**: the speedup using the multithreaded controller with one worker thread
• **M2**: the speedup using the multithreaded controller with two worker threads

• **M3**: the speedup using the multithreaded controller with three worker threads

• **M4**: the speedup using the multithreaded controller with four worker threads

Using 5 production modules and a static number of nodes in the search tree, the number of transformation operations of a depth first search performing a full enumeration equals $O(s^5)$ with $s$ as the number of existing nodes (e.g. sub-problems or levels) in the tree. According to the given number of transformation steps of 6899, the problem instance used in these measurement runs establishes a search tree with an average depth of around 5.5 during full enumeration if all 5 production modules provide a transformation proposal for each sub-problem. But the last row shows an actual number (20) of minimal transformation steps for a valid solution without backtracking. This is greater than 5.5 as estimated, which shows that the actual number of proposals is usually lower than 5.

As shown in the table, the number of transformation steps (forward) varies very much from the lowest allowed maximum latency (config 1) to the highest possible maximum latency (config 7). The number of steps for a full enumeration (6899) is much higher than the maximum number of steps in the realistic scenarios (e.g. with restricted or unrestricted maximum latency and normal validation), namely configuration 3 with 1021 steps.

When the number of forward steps equals the number of backward steps, no solution for the given problem was found due to the usage of the pseudo validation for full enumeration (see row 0) or the given maximum latency in the requirements.
The results have been presented as a chart in Figure 5.9 where the number of the configuration is given on the x-axis and the speedup is given on the y-axis. Again, the speedup for single threaded and the different multithreaded runs are shown in independent line graphs in the figure.

![Chart with speedup in case of backtracking](image)

**Figure 5.9: Chart with speedup in case of backtracking**

As expected in the analysis, the speedup in the backtracking case is very low. During full enumeration, which gives an average speedup for backtracking in any node in the search tree, it is below 1.0. Only in those cases where a solution was found (configuration 5-7) is the speedup nearly or above 1.0, and this is because of the speed advantage from using the first path constructed (forward-only case).

The earlier backtracking occurs the lower is the total number of backtracking steps to be performed in the search tree, but the higher is the number of partial solutions produced unnecessarily in advance because of the downstream validation. Therefore with multiple worker threads the speedup with configurations 1 to 3 is much lower than the speedup with one worker thread only. The worst case is shown in configuration 1 where the speedup is around 0.27 with multiple worker threads, which means that the search process took about 3.7 times longer.
The control overhead grows with the number of backtracking steps because of the additional effort for re-scheduling and validation order organisation purposes. This is shown by the speedup of the measurements for the runs using the multithreaded controller with only one worker thread, which is lower when the number of backtracking steps is higher.

5.3 Quality of Solutions

The quality of the generated solutions is evaluated by comparison to the optimal solution for smaller problem sizes and to the estimated values of the properties.

5.3.1 Comparison to Optimum Solution

Optimum solutions are generated with the brute force algorithm explained in Section 5.1.3. The brute force algorithm is very complex and very time-consuming when the number of nodes increases. For eight nodes it took less than a second, for ten nodes it took about five seconds and for twelve nodes it took more than five minutes. The computation for 14 nodes did not finish within twelve hours. Therefore the maximum number of nodes to be processed by the brute force algorithm is set to twelve.

The aim of this sub-section, besides the evaluation of solutions for smaller problems, is to evaluate the estimated values (see Section 5.1.4) too. Therefore in all comparisons the estimated values for the properties were also considered.

Table 5.7 gives the minimum, maximum and average (min, max, avg) for the properties of the objective function (cost, latency and load) for problems with eight, ten and twelve nodes (see horizontal sections in the table). For each problem size the statistical values for 100 generated problems are determined by the following methods:

- optimum: Determination by the brute force algorithm.
- line.e: Estimation for a line topology.
- tree.e: Estimation for a tree topology.
- line.g: Generation by the planning tool with a line topology configuration.
- tree.g: Generation by the planning tool with a tree topology configuration.

<table>
<thead>
<tr>
<th>config</th>
<th>costs</th>
<th>latency</th>
<th>load</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>avg</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>optimum</td>
<td>253.6</td>
<td>349.4</td>
<td>285.9</td>
</tr>
<tr>
<td>line.e</td>
<td>330.1</td>
<td>388.8</td>
<td>350.1</td>
</tr>
<tr>
<td>tree.e</td>
<td>400.1</td>
<td>458.8</td>
<td>420.1</td>
</tr>
<tr>
<td>line.g</td>
<td>412.6</td>
<td>695.0</td>
<td>516.5</td>
</tr>
<tr>
<td>tree.g</td>
<td>335.1</td>
<td>530.2</td>
<td>383.4</td>
</tr>
<tr>
<td>optimum</td>
<td>333.0</td>
<td>425.2</td>
<td>373.8</td>
</tr>
<tr>
<td>line.e</td>
<td>409.4</td>
<td>473.7</td>
<td>435.0</td>
</tr>
<tr>
<td>tree.e</td>
<td>399.4</td>
<td>463.7</td>
<td>425.0</td>
</tr>
<tr>
<td>line.g</td>
<td>493.8</td>
<td>851.6</td>
<td>649.6</td>
</tr>
<tr>
<td>tree.g</td>
<td>416.5</td>
<td>622.1</td>
<td>485.1</td>
</tr>
<tr>
<td>optimum</td>
<td>413.3</td>
<td>513.9</td>
<td>456.3</td>
</tr>
<tr>
<td>line.e</td>
<td>489.5</td>
<td>568.8</td>
<td>516.2</td>
</tr>
<tr>
<td>tree.e</td>
<td>479.5</td>
<td>558.8</td>
<td>506.2</td>
</tr>
<tr>
<td>line.g</td>
<td>583.7</td>
<td>957.3</td>
<td>763.5</td>
</tr>
<tr>
<td>tree.g</td>
<td>503.4</td>
<td>790.0</td>
<td>594.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>config</th>
<th>costs</th>
<th>latency</th>
<th>load</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>optimum</td>
<td>253.6</td>
<td>349.4</td>
<td>285.9</td>
</tr>
<tr>
<td>line.e</td>
<td>330.1</td>
<td>388.8</td>
<td>350.1</td>
</tr>
<tr>
<td>tree.e</td>
<td>400.1</td>
<td>458.8</td>
<td>420.1</td>
</tr>
<tr>
<td>line.g</td>
<td>412.6</td>
<td>695.0</td>
<td>516.5</td>
</tr>
<tr>
<td>tree.g</td>
<td>335.1</td>
<td>530.2</td>
<td>383.4</td>
</tr>
<tr>
<td>optimum</td>
<td>333.0</td>
<td>425.2</td>
<td>373.8</td>
</tr>
<tr>
<td>line.e</td>
<td>409.4</td>
<td>473.7</td>
<td>435.0</td>
</tr>
<tr>
<td>tree.e</td>
<td>399.4</td>
<td>463.7</td>
<td>425.0</td>
</tr>
<tr>
<td>line.g</td>
<td>493.8</td>
<td>851.6</td>
<td>649.6</td>
</tr>
<tr>
<td>tree.g</td>
<td>416.5</td>
<td>622.1</td>
<td>485.1</td>
</tr>
<tr>
<td>optimum</td>
<td>413.3</td>
<td>513.9</td>
<td>456.3</td>
</tr>
<tr>
<td>line.e</td>
<td>489.5</td>
<td>568.8</td>
<td>516.2</td>
</tr>
<tr>
<td>tree.e</td>
<td>479.5</td>
<td>558.8</td>
<td>506.2</td>
</tr>
<tr>
<td>line.g</td>
<td>583.7</td>
<td>957.3</td>
<td>763.5</td>
</tr>
<tr>
<td>tree.g</td>
<td>503.4</td>
<td>790.0</td>
<td>594.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>config</th>
<th>costs</th>
<th>latency</th>
<th>load</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>optimum</td>
<td>253.6</td>
<td>349.4</td>
<td>285.9</td>
</tr>
<tr>
<td>line.e</td>
<td>330.1</td>
<td>388.8</td>
<td>350.1</td>
</tr>
<tr>
<td>tree.e</td>
<td>400.1</td>
<td>458.8</td>
<td>420.1</td>
</tr>
<tr>
<td>line.g</td>
<td>412.6</td>
<td>695.0</td>
<td>516.5</td>
</tr>
<tr>
<td>tree.g</td>
<td>335.1</td>
<td>530.2</td>
<td>383.4</td>
</tr>
<tr>
<td>optimum</td>
<td>333.0</td>
<td>425.2</td>
<td>373.8</td>
</tr>
<tr>
<td>line.e</td>
<td>409.4</td>
<td>473.7</td>
<td>435.0</td>
</tr>
<tr>
<td>tree.e</td>
<td>399.4</td>
<td>463.7</td>
<td>425.0</td>
</tr>
<tr>
<td>line.g</td>
<td>493.8</td>
<td>851.6</td>
<td>649.6</td>
</tr>
<tr>
<td>tree.g</td>
<td>416.5</td>
<td>622.1</td>
<td>485.1</td>
</tr>
</tbody>
</table>

Table 5.7: Comparison with optimum and estimated values

The first three sections were obtained from a configuration with a partitioning module using a K-Means algorithm. This partitioning module generates a given number of partitions with minimum distance between the nodes within a partition but the number
of nodes in each partition may differ. According to the generation procedure, this leads to many cases where the number of nodes in a sub-problem is lower than the number of ports of the switch inserted in the sub-problem. Therefore solutions of configurations with this partitioning module use more switches than the optimum solution or even more than estimated.

Another partitioning module based on Ward’s algorithm was developed and used for the three remaining sections titled with the postfix “- MinDist”. This partitioning module optimizes according to the given distance function and evenly distribute the nodes across the sub-problems. Configurations with this partitioning module generate the better solutions in comparison to the optimum solution.

The network load determined from the generated solutions and from the estimation algorithm are the same for each set of problems of the same size. The reason is that all logical links in a PLC area end in the PLC, controlling that PLC area. By separating the communication of PLC areas through the initial partitioning according to area boundaries, only a little traffic is merged outside the PLC areas, and most of the traffic occurs at the physical link between the PLC and the connecting switch. The same happens when using the brute force algorithm because of the lower distance between nodes with logical linkage (the devices and their controller). The heaviest traffic to be expected is therefore the sum of the traffic induced by the set of logical links connected to a PLC which is the same as what the estimation algorithm calculates.

Table 5.8 lists the relationships between the generated solutions for line topologies (gl) and tree topologies (gt) and the optimum values (opt) and estimated values. Because there is no estimation for the latency, the table contains no relations between estimated and measured latency.

The table validates the conclusions from Table 5.7. The quality of generated solutions using the K-Means partitioning module is worse than the solutions with the MinDist partitioning module in costs and latency. The network load is always the same and therefore the ratio is always 1.0. But the ratios show another positive development very clearly. The average ratio between generated and optimum solution is lower if the number of nodes is higher. This relationship will be further analysed with bigger
Table 5.8: Relation between optimum, estimated and generated values

Table 5.8: Relation between optimum, estimated and generated values

<table>
<thead>
<tr>
<th>config</th>
<th>costs</th>
<th>latency</th>
<th>load</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>avg</td>
</tr>
<tr>
<td>8 nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gl/opt</td>
<td>1.43</td>
<td>2.31</td>
<td>1.80</td>
</tr>
<tr>
<td>gt/opt</td>
<td>1.10</td>
<td>1.70</td>
<td>1.34</td>
</tr>
<tr>
<td>gl/est</td>
<td>1.23</td>
<td>1.89</td>
<td>1.47</td>
</tr>
<tr>
<td>gt/est</td>
<td>0.84</td>
<td>1.22</td>
<td>0.91</td>
</tr>
<tr>
<td>el/opt</td>
<td>1.10</td>
<td>1.30</td>
<td>1.23</td>
</tr>
<tr>
<td>10 nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gl/opt</td>
<td>1.41</td>
<td>2.18</td>
<td>1.73</td>
</tr>
<tr>
<td>gt/opt</td>
<td>1.08</td>
<td>1.65</td>
<td>1.30</td>
</tr>
<tr>
<td>gl/est</td>
<td>1.19</td>
<td>1.91</td>
<td>1.49</td>
</tr>
<tr>
<td>gt/est</td>
<td>1.02</td>
<td>1.43</td>
<td>1.14</td>
</tr>
<tr>
<td>el/opt</td>
<td>1.07</td>
<td>1.23</td>
<td>1.17</td>
</tr>
<tr>
<td>12 nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gl/opt</td>
<td>1.33</td>
<td>2.07</td>
<td>1.67</td>
</tr>
<tr>
<td>gt/opt</td>
<td>1.07</td>
<td>1.64</td>
<td>1.30</td>
</tr>
<tr>
<td>gl/est</td>
<td>1.16</td>
<td>1.79</td>
<td>1.48</td>
</tr>
<tr>
<td>gt/est</td>
<td>1.02</td>
<td>1.48</td>
<td>1.17</td>
</tr>
<tr>
<td>el/opt</td>
<td>1.06</td>
<td>1.18</td>
<td>1.13</td>
</tr>
<tr>
<td>8 nodes - MinDist</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gl/opt</td>
<td>1.26</td>
<td>2.14</td>
<td>1.54</td>
</tr>
<tr>
<td>gt/opt</td>
<td>1.21</td>
<td>1.74</td>
<td>1.51</td>
</tr>
<tr>
<td>gl/est</td>
<td>1.02</td>
<td>1.71</td>
<td>1.26</td>
</tr>
<tr>
<td>gt/est</td>
<td>0.85</td>
<td>1.23</td>
<td>1.02</td>
</tr>
<tr>
<td>el/opt</td>
<td>1.10</td>
<td>1.30</td>
<td>1.23</td>
</tr>
<tr>
<td>10 nodes - MinDist</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gl/opt</td>
<td>1.21</td>
<td>1.92</td>
<td>1.49</td>
</tr>
<tr>
<td>gt/opt</td>
<td>1.18</td>
<td>1.51</td>
<td>1.39</td>
</tr>
<tr>
<td>gl/est</td>
<td>1.01</td>
<td>1.58</td>
<td>1.28</td>
</tr>
<tr>
<td>gt/est</td>
<td>1.04</td>
<td>1.39</td>
<td>1.22</td>
</tr>
<tr>
<td>el/opt</td>
<td>1.07</td>
<td>1.23</td>
<td>1.17</td>
</tr>
<tr>
<td>12 nodes - MinDist</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gl/opt</td>
<td>1.17</td>
<td>1.70</td>
<td>1.39</td>
</tr>
<tr>
<td>gt/opt</td>
<td>1.15</td>
<td>1.47</td>
<td>1.24</td>
</tr>
<tr>
<td>gl/est</td>
<td>1.01</td>
<td>1.50</td>
<td>1.23</td>
</tr>
<tr>
<td>gt/est</td>
<td>1.01</td>
<td>1.33</td>
<td>1.11</td>
</tr>
<tr>
<td>el/opt</td>
<td>1.06</td>
<td>1.18</td>
<td>1.13</td>
</tr>
</tbody>
</table>

Figure 5.10 and 5.11 contain histograms for the cost ratio between generated and optimum solution for 8 and 12 nodes. In both cases the cost ratio for around 35% of the generated solutions is greater than 1.3. and between 65% and 70% are at least around 1.6 (8 nodes) and 1.5 (12 nodes) of the optimum cost. The number of better solutions is higher for bigger problems as already seen in the table above.
Figure 5.10: Cumulative histogram on cost ratio (8 nodes)

Figure 5.11: Cumulative histogram on cost ratio (12 nodes)
To analyse the reason for the diversity of the solutions Figure 5.12 depicts the relation of optimum cost to the cost of generated solutions as points. The X axis shows the optimum cost for a problem and the y axis shows the cost of the generated solution. Each point represents the cost for the solution of a problem in the optimum case and of the generated solution. The figures show significant clusters of points, each in a diagonal oval shape. This shows that changes in line lengths result in proportionally low costs increases but there are higher cost differences due to the difference in number of switches (because a switch produces significantly more costs).

Cumulative Histogram

Figure 5.12: *Optimum to generated costs (12 nodes)*

A rule of thumb for the maximum latency estimation is to allow at most ten hops (switches) between two communicating devices. For the measurements the latency for each switch type were configured to be 100 $\mu$s to allow an easy determination of the number of hops in the path between the ends of a logical link.

Using the MinDist partitioning module the ratio between latency of optimum solutions and tool generated solutions ranges between 0.6 and 1.5 in the best case (MinDist with 12 nodes) and the average is at 0.88. The reason is that by default the tool produces more latency optimized solutions because of the consideration of the PLC areas and therefore the cost is higher.

The histogram (see Figure 5.13) shows that in most cases the latency is equal to the latency of the optimum solution.
Another important aspect to be analysed is the quality of the estimates. The ratio between optimum and estimated latency was always 1.0 as mentioned above. The latency estimate is therefore very good for this type of problems. The average ratio of the costs is between 1.23 and 1.17 (see Table 5.8) and therefore the estimate is around 20% above the optimum value. Additionally, the ratio decreases with an increased number of nodes, which means that it should be more exact for bigger problems. This is analysed in the next section.

5.3.2 Evaluation of Bigger Problem Sizes

The purpose of this sequence of analyses is to determine the quality of tool generated solutions for bigger problem sizes. Because the brute force algorithm is unable to determine the optimum values for such problem sizes, we used the estimated values for properties (see Section 5.1.4). As the KMeans clustering module produced poor solutions, only the MinDist partitioning module was used in all the test runs.

The benchmark for the end-to-end latency was 1000 $\mu$s due to the rule of thumb using at least 10 hops between two jointly communicating devices.

Table 5.9 gives an overview of the quality values for problems with 50 to 400 nodes. Again the cost, the latency and the network load is given in the table for each of the
following tool configuration:

- **line.e** and **tree.e**: The first two rows in each section of the table list the estimated values for tree and line topologies.

- **line.g** and **tree.g**: The next two rows list the properties of the tool generated solutions using tree and line topologies configurations without boundaries in the requirements file (i.e. without backtracking).

Table 5.9: Absolute and estimated values (50 - 400 nodes)

<table>
<thead>
<tr>
<th>config</th>
<th>costs [EUR]</th>
<th>latency [μs]</th>
<th>load [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>avg</td>
</tr>
<tr>
<td>line.e</td>
<td>2053</td>
<td>2138</td>
<td>2091</td>
</tr>
<tr>
<td>tree.e</td>
<td>1863</td>
<td>1948</td>
<td>1901</td>
</tr>
<tr>
<td>line.g</td>
<td>2264</td>
<td>2783</td>
<td>2504</td>
</tr>
<tr>
<td>tree.g</td>
<td>2153</td>
<td>2342</td>
<td>2108</td>
</tr>
<tr>
<td>line.gr</td>
<td>2264</td>
<td>2783</td>
<td>2515</td>
</tr>
<tr>
<td>tree.gr</td>
<td>2153</td>
<td>2342</td>
<td>2108</td>
</tr>
</tbody>
</table>

100 nodes

| line.e | 4093 | 4191 | 4149 | - | - | - | 0.437 | 1.210 | 0.738 |
| tree.e | 3633 | 3731 | 3689 | - | - | - | 0.437 | 1.210 | 0.738 |
| line.g | 4592 | 5502 | 4944 | 700 | 1700 | 1119 | 0.437 | 1.210 | 0.739 |
| tree.g | 4124 | 4578 | 4249 | 500 | 1100 | 763 | 0.437 | 1.210 | 0.739 |
| line.gr | 4592 | 5469 | 4996 | 700 | 1600 | 974 | 0.437 | 1.210 | 0.739 |
| tree.gr | 3924 | 4628 | 4254 | 500 | 1000 | 803 | 0.437 | 1.210 | 0.739 |

150 nodes

| line.e | 6141 | 6256 | 6195 | - | - | - | 0.470 | 1.176 | 0.824 |
| tree.e | 5341 | 5456 | 5395 | - | - | - | 0.470 | 1.176 | 0.824 |
| line.g | 6965 | 7859 | 7364 | 900 | 2000 | 1279 | 0.470 | 1.176 | 0.837 |
| tree.g | 5998 | 6763 | 6356 | 600 | 1200 | 865 | 0.470 | 1.176 | 0.825 |
| line.gr | 7072 | 8170 | 7456 | 900 | 1600 | 1086 | 0.470 | 1.176 | 0.833 |
| tree.gr | 5998 | 6763 | 6365 | 600 | 1000 | 849 | 0.470 | 1.176 | 0.825 |

200 nodes

| line.e | 8179 | 8310 | 8236 | - | - | - | 0.605 | 1.344 | 0.893 |
| tree.e | 7189 | 7320 | 7246 | - | - | - | 0.605 | 1.344 | 0.893 |
| line.g | 9178 | 10510 | 9762 | 900 | 2500 | 1406 | 0.605 | 1.344 | 0.929 |
| tree.g | 7954 | 8972 | 8412 | 700 | 1200 | 950 | 0.605 | 1.344 | 0.897 |
| line.gr | 9178 | 10781 | 9904 | 900 | 1900 | 1199 | 0.605 | 1.512 | 0.933 |
| tree.gr | 7954 | 8972 | 8438 | 700 | 1000 | 906 | 0.605 | 1.344 | 0.897 |

400 nodes

| line.e | 16314 | 16437 | 16388 | - | - | - | 0.806 | 1.512 | 1.074 |
| tree.e | 14404 | 14527 | 14478 | - | - | - | 0.806 | 1.512 | 1.074 |
| line.g | 18858 | 20394 | 19519 | 1400 | 2500 | 1804 | 1.277 | 3.158 | 2.125 |
| tree.g | 16062 | 17301 | 16632 | 900 | 1400 | 1126 | 0.941 | 2.923 | 1.962 |
| line.gr | 18858 | 20837 | 19675 | 1200 | 2300 | 1593 | 1.176 | 3.058 | 2.048 |
| tree.gr | 16214 | 17568 | 16812 | 800 | 1000 | 979 | 0.941 | 2.923 | 1.948 |
- **line.gr** and **tree.gr**: The last two rows list the properties of the generated solutions with the best latency near or beyond 1000 µs in a given maximum processing time of 10 seconds (i.e. with backtracking).

As the table shows that the costs are closer to the estimated values and they grow even closer with a greater number of nodes. The maximum network latency calculated for generated solutions of problems with 50 nodes exceeded the allowed maximum of 1000 µ using the tool configuration, which generates line topology. But when a maximum latency was given in the requirements file, a valid solution could be found for all problems with 50 nodes. The tree configuration always generated valid solutions for such small problem sizes. The latency of solutions with line configuration gets worse with larger numbers of nodes. At 150 nodes the tool, configured for line generation, was no longer able to find a valid solution for all problems in the given 10 seconds. As shown in Figure 5.14 over 57% were beyond 1000 µ and many were close to the given benchmark. At 400 nodes solutions with the lowest latency already passed the mark even with restricted maximum latency but the tool still generated valid solutions with the tree configuration in all cases.

![Histogram](image)

**Figure 5.14: Latency in solutions for 150 nodes (line.gr)**

Now, the network load is as low as expected and in some cases not equal to the estimated network load as observed for the very small problem sizes in the previous section.
In Table 5.10 the property values of the generated solutions for problem sizes between 50 and 400 nodes have been put in relation to the estimated values for cost and network load and the required maximum value for the latency of 1000 $\mu$s. All solutions generated with line topologies configuration have the prefix `gl` and those generated with the tree configuration have the prefix `gt`. Those with postfix `r` are configured with a restriction on the maximum network latency. It is obvious that the relationship between the estimated and actual costs of the generated solution gets a stable average for larger numbers of nodes which ranges from 1.15 to 1.20 depending on the configuration. The average of the latency and network load is constantly growing with the number of nodes. Only the tree configuration with restriction in the maximum network load keeps its good quality.

The table (see 5.11) shows the absolute values for problem sizes of 1000 and 5000.
<table>
<thead>
<tr>
<th>config</th>
<th>costs [EUR]</th>
<th>latency [$\mu$s]</th>
<th>load [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>avg</td>
</tr>
<tr>
<td>1000 nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>line.e</td>
<td>81253</td>
<td>81461</td>
<td>81353</td>
</tr>
<tr>
<td>tree.e</td>
<td>71413</td>
<td>71621</td>
<td>71513</td>
</tr>
<tr>
<td>line.g</td>
<td>95033</td>
<td>98160</td>
<td>96701</td>
</tr>
<tr>
<td>tree.g</td>
<td>80403</td>
<td>82933</td>
<td>81889</td>
</tr>
<tr>
<td>line.gr</td>
<td>95208</td>
<td>98329</td>
<td>96893</td>
</tr>
<tr>
<td>tree.gr</td>
<td>80707</td>
<td>83051</td>
<td>81995</td>
</tr>
<tr>
<td>5000 nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>line.e</td>
<td>202750</td>
<td>203002</td>
<td>202878</td>
</tr>
<tr>
<td>tree.e</td>
<td>177810</td>
<td>178062</td>
<td>177938</td>
</tr>
<tr>
<td>line.g</td>
<td>238698</td>
<td>244335</td>
<td>241559</td>
</tr>
<tr>
<td>tree.g</td>
<td>201058</td>
<td>206307</td>
<td>203730</td>
</tr>
<tr>
<td>line.gr</td>
<td>238840</td>
<td>244645</td>
<td>241741</td>
</tr>
<tr>
<td>tree.gr</td>
<td>201216</td>
<td>206307</td>
<td>203765</td>
</tr>
</tbody>
</table>

Table 5.11: Absolute and estimated values (1000 and 5000 nodes)

With 5000 nodes the network load very much exceeds the given maximum capacity of 100 Mbit. This has two reasons:

1. There are too many logical links crossing the plant. Therefore a huge number of logical links merge at the top most physical links and leads to high network load on these segments. In some of these cases it is impossible to establish a valid network under such restrictive constraints.

2. The hierarchical network structure used by the planning tool to interconnect PLC areas encourages cases where especially those crossing logical links are merged on the top most physical links.

This is not a serious problem because in such big networks a backbone with enough capacity is usually planned. This is a rule that could also be considered in the network planning tool in the future.

Another problem is that the latency, even with the tree configuration, is rising above the acceptable amount of 1000 $\mu$s due to the same reason: the load is rising too much.

Table 5.12 lists the ratio of estimated, required and actual values for generated solutions of problems with 1000 and 5000 nodes. As assumed before, based on median
Table 5.12: Relation between estimated and actual values (1000 and 5000 nodes)

Problem sizes, the ratio between the estimated optimum cost and the actual cost is fixed between 1.15 and 1.20 for larger number of nodes.

Table 5.13: Absolute and estimated values without inter area communication

To show the better network load of solutions where the load problem was solved with a backbone or simply does not exist, the bigger problems have been reconfigured without inter PLC area communication. The results are given in Table 5.13. The
The overall network load is much lower and even for 5000 nodes still under 2.7%.

The latency is again in acceptable ranges using the tree configuration on 1000 nodes but not with 5000 nodes. This has been determined as the upper bound for the number of nodes in such kind of network planning problems to be solved by the tool with the given production and partitioning modules in the given configuration and maximum processing time.

The analysis showed that the network planning tool with the extensions developed in this thesis is able to generate valid solutions. The probability to find a solution in a time frame of 10 seconds is high for problems below 400 nodes. Between 400 and 1000 nodes the network planner must do minor manual interventions search for the right configuration. Between 1000 and 5000 nodes, increasing manual interventions such as the insertion of backbones are necessary. The upper bound to find valid solutions without manual interventions is set at problem sizes of 5000 nodes.

### 5.4 Evaluation of the Extensions

This section contains all performance measurements to evaluate the extensions to the framework developed in this thesis (see Sections 4.3.2, 4.3.3 and 4.3.4).

Table 5.14 provides an overview of the total processing time for different problem sizes with and without restrictions in the requirements. Because a total iteration of the search tree is already too long to be measured for small problems, the processing time for the runs with restricted maximum load was set to a maximum duration of 10 seconds, which is a realistic response time acceptable to a user. Generally the frequency of backtracking conditions during the search process heavily depends on the structure of the network and is therefore impossible to be estimated, which results in very diverse processing times. From this point of view the maximum boundary of 10 seconds is at least a way to get information about the performance in a realistic use case.

The measurements here were made with the line configuration only, because the re-
suits were very similar to those from the runs with the tree configuration.

<table>
<thead>
<tr>
<th>nodes</th>
<th>min</th>
<th>max</th>
<th>avg</th>
<th>stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>without restriction</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>0.0094</td>
<td>0.0182</td>
<td>0.0113</td>
<td>0.0017</td>
</tr>
<tr>
<td>100</td>
<td>0.0188</td>
<td>0.0353</td>
<td>0.0236</td>
<td>0.0040</td>
</tr>
<tr>
<td>150</td>
<td>0.0276</td>
<td>0.0774</td>
<td>0.0386</td>
<td>0.0076</td>
</tr>
<tr>
<td>200</td>
<td>0.0381</td>
<td>0.0910</td>
<td>0.0543</td>
<td>0.0101</td>
</tr>
<tr>
<td>400</td>
<td>0.0928</td>
<td>0.1810</td>
<td>0.1306</td>
<td>0.0188</td>
</tr>
<tr>
<td>1000</td>
<td>1.1889</td>
<td>1.9689</td>
<td>1.4341</td>
<td>0.1424</td>
</tr>
<tr>
<td>5000</td>
<td>7.3358</td>
<td>10.1957</td>
<td>8.7627</td>
<td>0.6412</td>
</tr>
<tr>
<td>with restriction</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>0.0094</td>
<td>1.1630</td>
<td>0.0230</td>
<td>0.1152</td>
</tr>
<tr>
<td>100</td>
<td>0.0188</td>
<td>2.0392</td>
<td>0.1250</td>
<td>0.3318</td>
</tr>
<tr>
<td>150</td>
<td>0.0288</td>
<td>5.5542</td>
<td>0.4685</td>
<td>1.0115</td>
</tr>
<tr>
<td>200</td>
<td>0.0401</td>
<td>4.8155</td>
<td>0.5584</td>
<td>1.0170</td>
</tr>
<tr>
<td>400</td>
<td>0.0988</td>
<td>5.0099</td>
<td>0.6474</td>
<td>0.9841</td>
</tr>
<tr>
<td>1000</td>
<td>1.2181</td>
<td>5.2010</td>
<td>2.0664</td>
<td>0.9944</td>
</tr>
<tr>
<td>5000</td>
<td>7.5543</td>
<td>12.7292</td>
<td>9.7308</td>
<td>1.2322</td>
</tr>
</tbody>
</table>

Table 5.14: Processing time for 50 to 5000 nodes

The standard deviation for the tests with no restrictions is low. Small deviations are always possible due to the random distribution of logical links across the plant. The average value ranges from 0.01s for 50 nodes without restriction to around 9s for 5000 nodes. In Figure 5.15 the processing time is depicted as a graph which shows a higher gradient above 500 nodes but could still be perceived as linear.

Figure 5.15: Processing time for 50 to 5000 nodes without restrictions

The standard deviation for the tests with restrictions in network load and processing
time is significantly higher due to the varying amount of backtracking steps, as men­tioned above. Another graph, in Figure 5.16, has been prepared for comparison. Here
the behaviour is different below 500 nodes but the overall graph could be approximated
with a line, as in the case with no restrictions.

![Graph showing processing time for 50 to 5000 nodes with restrictions](image)

Figure 5.16: Processing time for 50 to 5000 nodes with restrictions

Table 5.15 shows the amount of processing time consumed by each module during
the search process for a problem with 5000 nodes 46408 logical links and 632 areas.

The columns of the table contain the following values:

- module: Name of the module
- avg: Average processing time for a transformation operation of the module.
- stddev: Standard deviation of the processing time for a transformation operation.
- total: Total processing time consumed by the module.
- quota: Percentage of the whole run time of the tool.

It shows that the performance of the modules is good also in a scenario with a huge
problem size. Obviously the MinDist partitioning module consumes the most process­
ing time due to the large amount of processing time per transformation step of around
8ms and its frequent usage. The area partitioning module also has a larger amount of
processing time consumption at 11 ms. All production modules and also the validation module took less than 1 ms because their algorithms are of low complexity. Another reason is that the production modules process only problems of minor sizes while the partitioning module usually have to process larger numbers of nodes. The same holds for the area partitioning module which processes the largest number of nodes in its role as initial partitioning module. But it is called only once and therefore has minor influence on the total processing time. The validation module has an acceptable performance but it is used for every partial solution produced and adds 11.8% to the total processing time.

5.5 Usability for End Users

The network planning tool is of course a prototype and not yet a professional product developed for end users. The purpose of this section is therefore a preliminary evaluation towards an end user friendly planning tool based on current information.

To evaluate the usability the analysis starts with a description of the current procedure to generate a network for a given description of a plant.

To start, the user needs a description of the plant with the devices, their communication behaviour and grouping in areas. Manual writing of such a description is too time consuming, therefore the import of such information from existing CAD plans and from development files of the automation plant will be the right choice. Actually the current development in the automation domain follows the idea of Product Lifecycle Management systems (PLM) for automation systems (cp. [RA06], [K06] or [LFM06])
which supports the tasks and management of data associated with a manufactured automation plant in its whole life cycle. The integration of a network planning tool into such a PLM is therefore just a consequence.

The next step is to write a list of available materials including at least the switch types and cables. Such a configuration file can be saved from a previous session or a workmate.

The user then defines the objective function for the properties of the network. The end to end latency can be estimated by the cycle time of the automation device and the protocol behavior. A rule of thumb is to have a maximum of 10 switches in a logical link. Another estimate is possible given the information of the material list. The network load can be chosen without consideration of the automation devices. The current search procedure of the planning tool starts with the solutions with lowest costs, therefore it makes no sense to assign a boundary for the costs.

Next, the user chooses a tool configuration to produce acceptable solutions. He will make a copy of this configuration because during the planning procedure he will change many parameters to get the best solution.

To get a starting point for the properties of the network and to make sure at least one possible network exists for the given problem (there are some cases where no solution is possible) the user does one run without any restrictions in the requirements file. The user then proceeds to reduce the restrictions in the requirements file step by step to reflect the properties that are not as required, and repeats the network generating process.

If the planning tool does not generate a valid solution in acceptable time the user may change the tool configuration and try other modules, module configurations or global configurations such as the weighting of communication related logical distance versus geometrical distance of devices which is used in many algorithms in the tool.

This procedure is complicated for the user and still complicated to be automated because the decisions for the changes in the configuration rely heavily on the analysis of the previously generated solution and the knowledge of the modules. Only the
step-wise restriction of the requirements (e.g., end-to-end latency) can be automatically performed to ensure the generation of a solution, which is at least near the given boundary in case no matching solution was found.

The analysis of the quality of the generated solutions in the previous section, which was entirely performed with the latter method with automatic restriction of the requirements, showed that for smaller problem sizes no changes in the configuration are needed to find a valid solution. This might not be the best solution possible for the planning tool but it provides an acceptable approximation for a non-professional user during the development of the automation plant and assures him that a valid network for his plant is still possible. The more professional network planner then has the possibility to improve the network by changing the configuration or by inserting networks for subareas or backbones on his own.

5.6 Adequacy of the NDF for Network Planning Algorithms

The network design framework as a basis for the network planning tool for PROFINET is evaluated by the work of this thesis, and this section is a summary of the lessons learnt during the development of the modules to realise the chosen generation procedure.

The overall strategy of the search process of the NDF is to partition the whole problem into smaller sub-problems to reduce run-time complexity. As shown in the analysis in Section 4.1 of the application and proven by the quality evaluation in Section 5.3 the partitioning has the drawback that the solution does not reach the optimum because the production modules lose the big picture. But it is still in an acceptable range to the optimum solution and a human network planner will not do any better. A planner looks for clusters of nodes with strong communication relationship and starts by connecting them before interconnecting on higher levels. The PROFInet Installation Guideline suggests a similar procedure using the PLC area boundaries as grouping indicator.
Therefore the strategy is proven to be the right way with all the advantages derived from it, namely the clear separation between the process controller and the planning modules such as partitioning modules, production modules and validation modules.

The separation of the process controller allowed an easy integration of the multi-threaded process controller. Only a few changes to the NDF were needed to make some of the data structures thread-safe (see Section 3.3).

The best-first search algorithm currently used in the NDF has some drawbacks in huge search trees as in all depth-first search algorithms. It is usually correct, as shown in the evaluation of the quality, where each solution was found in less than 10 seconds, but it requires sensible handling to keep the number of sub-problems low. The developer of production modules should always try to use as much heuristics as possible to reduce the number of possible solutions for a given problem. An improvement of the search algorithm by application of the Branch and Bound method would certainly be another answer to this side effect.

The development and integration of the production modules and partitioning modules during this work was of manageable complexity thanks to their separation.

The proposal created by the production modules to propose a transformation provides an evaluation of the proposal in costs only, but the requirements for the network consider costs, end-to-end latency and network load. The strategy used in the network planning tool for PROFINET was to start with the network with lowest costs and optimize by backtracking towards network quality of service, but another strategy could start with higher network quality of service. In this case an evaluation based on costs would be inadequate. Therefore, the proposal should be more flexible.

Another problem is the management of different hierarchical levels in the network, for example where to place the different distributor types (MD, FD and BD), but this is a common problem in network planning.

More low-level advantages are:

- the possibility to attach some context information to sub-problems for later use,
- the rich set of algorithms,
- the object pool mechanism, which reduces the number of memory allocations and
- the consequent usage of the factory pattern and smart pointers.

One big advantage is the graphical user interface which is able to visualise every transformation step in the network and thereby greatly helps when debugging the implemented modules.
Chapter 6

Conclusion

The thesis is based on a framework for network design algorithms, called Network Design Framework (NDF), developed in a collaboration between the Laboratory of Distributed Systems at Wiesbaden University of Applied Sciences, Germany (where the author was one of the two main researchers on the project) and Siemens AG.

One purpose of the thesis is to develop a network planning tool for an Industrial Ethernet environment and thereby evaluate the applicability of a method to reduce network design complexity by using heuristics and partitioning as supported by the NDF. Another goal is to develop a parallelisation strategy for the NDF to ensure a good response time for big problem sizes as well.

To develop a suitable parallel algorithm for the search process of the NDF two approaches were considered: One constructs multiple paths of the search tree in parallel from Kumar et al. and the other constructs one path through the search tree at a time in parallel. As the parallelisation of the construction of a single path through the search tree seemed more promising for a best-first search when the probability to find the solution on the first path is high, the second approach was selected. The assumption that most solutions are found on the first path or with less backtracking steps was supported by the results of the evaluation process. But more intelligence and heuristics in the production modules are necessary to give good estimations of their solutions, thereby preventing backtracking as much as possible.

As a consequence of the parallelisation of the search algorithm a design rule for the
development of the production, partitioning and validation modules was defined, which states that all data associated with the state of the search process must be managed by the process controller to ensure that a module always has exclusive access to the data needed to perform its specific task.

To realise the parallel search algorithm a multi-threaded process controller was developed for the NDF as an alternative to the existing single threaded process controller. This consists of four different subsystems:

- A messaging subsystem that provides asynchronous messaging
- A task execution subsystem that manages the execution of tasks by threads of a thread pool
- A task scheduling subsystem that manages a schedule of tasks and controls the execution of the tasks
- A backtracking control subsystem that is responsible for validation and determining the backtracking order.

Integration of the new process controller in the NDF was straightforward and only marginal concurrency related changes were needed in the NDF.

An analysis of various Industrial Ethernet protocols showed up PROFINET as an appropriate representative for the application. Consequently PROFINET was analysed in more detail with a view to formulating useful heuristics and a network generation strategy.

The generation strategy is composed of the following building blocks:

- An initial partitioning step according to given boundaries of logically separated control areas (so called PLC areas).
- Further partitioning by a generic partitioning algorithm.
- Insertion of machine distributors near the automation devices.
- Insertion of field distributors to connect machine distributors.
• Insertion of field distributors to connect field distributors.

• Establishing a minimal spanning tree on given subsets of any kind of distributor.

A set of modules were developed to generate tree- or line-oriented topologies with proper configuration. The data model was enhanced to provide protocol specific information to generate powerful estimates of the network load. Furthermore a corresponding validation module was developed to support fast validations of the partial solutions generated. Additionally several export modules were defined to test the capabilities of the framework in this regard.

To evaluate the parallel algorithm and the network planning tool a number of auxiliary tools were created:

• A problem generator that generates network planning problems in the form of realistic automation plant structures

• An automated execution environment for testing

• A brute force algorithm to determine the properties of optimal solutions which is realized as another process controller module to the NDF for comparison.

• A method to estimate the properties of networks which is also based on the NDF.

The evaluation of the parallel algorithm confirmed that the chosen approach for the acceleration of the search process is generally right for the case with low appearance of backtracking steps and in the ideal case a speedup of 3.3 is reachable. This speedup will be reached if the performance consumption of the modules is high. With the current modules and their low performance needs, the tool only managed to attain low speedup values. The parallelisation of the backtracking steps contributes little, which is why the overall speedup for huge problem sizes with a few backtracking steps is greater than 1.0 in practice.

To achieve more optimal solutions and at the same time the way reduce the probability of backtracking steps it would be quite reasonable to develop production modules that process a larger number of nodes in one transformation operation. Assuming
such production modules have a comparatively low complexity of $O(n^2)$, our evaluation shows that if a production module processes only 10.6 times more nodes in one transformation operation, it will be possible to reach the potential maximum speedup. Therefore the approach developed for the parallel algorithm must be considered as generally right.

When a generation strategy that deals with heavy backtracking is required, the approach of Kumar et al. is recommended as an alternative. But due to the tendency to large state spaces the approach of Kumar et al. is not appropriate for conventional end user hardware with their maximum of 4 processing units available today, which is another reason for the selection of the approach used in this thesis. Furthermore, Kumar et al. will produce more non parallelisable overhead, which reduces the reachable speedup using production modules of low runtime complexity thereby making the approach questionable.

The quality of the solutions achieved by the network planning tool was evaluated based on generated problem descriptions. For a small number of nodes a comparison to the optimal properties is possible using the brute force algorithm. However, applying the brute force algorithm is very complex, making the determination of optimal properties already impossible for 14 nodes. Instead, the module that estimates lower bounds for the optimal properties was used, which gave good performance when compared to the results of the brute force algorithm for small problem sizes. This demonstrates that an increase in the number of nodes only results in a small deviation from the optimal cost value.

The quality of the solutions is predominantly good. For 8-12 nodes the deviation ranges from 1.8 to 1.24 times the optimal value, depending on the configuration of the network planning tool. This is comparatively high but for larger number of nodes, e.g., 50, the ratio settled between 1.1 and 1.2 of the estimated value. The deviation in costs is mainly caused by deviations in the number of switches as pointed out in the evaluation. A partitioning module that maintains a fairly stable bound on the maximum number of nodes per sub-problem proved to be an effective instrument to prevent unused switch ports in the network. The end-to-end latency and the network load rose
rapidly with a greater number of nodes ($\geq 1000$). This is mainly caused by logical
links connecting PLC areas. With more nodes there is higher probability that longer
chains of logical links are formed, which could not be mapped to a physical network
structure by the network planning tool even if a solution actually exists. Due to the
character of automation plants, heavier communication load with tight quality of ser­
vice requirements tend to appear inside PLC areas more often than between such areas
and therefore this problem is of less importance in this domain. Furthermore the heav­
ier network load is rather a logical consequence of more communication crossing the
whole network and can usually be solved by the integration of a backbone with higher
throughput.

With backtracking the planning tool was able to solve the problem automatically up
to 400 nodes. Assuming a solution exists for the problem with logical links between
PLC areas as described above, it was possible even with 5000 nodes to solve the gener­
ated planning problems with low number of backtracking steps in the given maximum
processing time of 10 seconds as expected. Performance tests showed that a maxi­
mum processing time of 10 seconds and an average of around 9 seconds was normal.
Processing time for more than 1000 nodes was more than 2 seconds with a major por­
tion taken up by input/output operations: reading the problem instance and writing the
solution to the file system.

Usability for end users was also evaluated. This is in relation to dealing with the long
processing times of the best-first search when problems that are impossible or have
hardly reachable solutions were presented. Here the user approximates a reachable
and good solution by repeated runs with stepwise adjustment of the requirements until
either a desired solution is found or the tool configuration must be changed to achieve
further improvement. In the worst case, parts of the network such as backbones must
be inserted manually, as mentioned above. Because the latter case sometimes requires
more professional knowledge, even though many parameters are handled by the net­
work planning tool, it remains of limited use for the casual end user. But the response
time and the probability of solutions for smaller problems as well as the graphical user
interface are assessed as positive features from the users’ point of view.
The GUI proved to be beneficial for the development of modules too. The development of the network planning tool allowed an evaluation of the capabilities of NDF, which was considered adequate except for some minor outstanding improvements. The strategy of problem partitioning and solving sub-problems also proved to be applicable to network planning problems of the Industrial Ethernet domain.

Work on the following would constitute promising directions for future research, providing enhancements to the NDF and the network planning tool:

- **Production modules that can handle larger number of nodes:** In particular a full usage of the available ports of switches is important to reduce overall costs. The number of unused ports as well as the quality of the overall solution depends on the degree of partitioning. Therefore production modules that support the processing of larger sub-problems are of evident advantage.

- **Algorithm to find near optimal solutions for 20 or more nodes:** Working on the brute force algorithm threw up an algorithm to iterate over all possible networks for a set of nodes without identity such as the switches needed to establish a network between a set of devices. The number of networks for such exchangeable nodes is low compared to the number of possible assignments of devices to a given set of switches. When combined with a modified cluster algorithm to find a suitable assignment of devices to switches it should be possible to find a solution near the optimum solution within a few milliseconds.

- **Parallel algorithm that supports backtracking:** It will be interesting to combine the approach of Kumar et al. and the approach taken in this thesis to parallelise the backtracking case.

- **Support of a Branch and Bound search algorithm:** An efficient alternative to a best-first search is a Branch and Bound algorithm if it is applicable to the given problem. Therefore a process controller that supports branch and bound could be a good enhancement to the NDF.

- **Improvement of the proposal evaluation:** Improving the proposals can support
other network generation strategies favouring more quality of service oriented solutions rather than costs.

- **Enhanced automated execution environment:** A realisation of the method presented to find approximate solutions for given requirements would be a valuable enhancement in terms of usability.

- **Support for manual integration of network parts:** This would be an improvement for the case where the network planning tool is not able to find a solution without the assistance of the user.

The network planning tool developed in this thesis provides a useful tool for the professional network planner of soft real-time applications on PROFINET. This work demonstrated the general applicability of the method and the framework in the domain of network planning problems. If the proposed improvements are implemented, which amongst other things will increase the efficiency of the parallel search algorithm, its adoption to the end user domain will be possible. Moreover by integrating the network planning tool into a Product Lifecycle Management System, which is an imperative consequence of current developments in the automation sector, the network planning tool will significantly increase productivity in the planning process of automation plants in the future.
Bibliography


[RA06] Hamidullah Riaz Ahmad, Fan Yuqing. Managing product data and design flow process. In *Proceedings of the Sixth International Conference on*


Appendix A

Publications

One paper [Mac08] has been published as full paper on a conference for process automation and automation systems in Emden (Germany) called Automatisierungstage.
Appendix B

Algorithms to Realise Brute Force

Network Generation

The algorithms in this appendix are developed to realise the brute force procedure designed in Section 5.1.3, namely:

1. Enumerate all networks for a set of switches
2. enumerate all possible assignments of nodes to switches for an established network of switches. Consider the number of ports left open.
3. determine the optimum position for all switches in a network with switches installed and nodes connected.

The algorithms are explained in the same order in the following sections.

B.1 Networks for a Fixed Set of Nodes

The first task in the proposed procedure is to enumerate all possible networks for a fixed set of switches.

The generated networks must satisfy the following requirements:

- Valid port assignment: The number of ports of a node must be greater than or equal to the degree of the node.
- **Cycle free**: The network must have no cycles.

- **Connected**: All network elements must be connected to the network. No part is allowed to be isolated.

A valid network with \( n \) nodes must have \( n - 1 \) edges. Given a set of all possible edges \( E \) between the nodes \( n \in N \) the set of combinations of \( n - 1 \) edges \( e \in E \) represents the set of all networks including invalid networks. But a large number of networks have the same structure with only the nodes exchanged among each other (for example mirrored network structures).

Another approach is to vary the degree \( d \) of a node (number of connected edges) and enumerate all network structures (sets of edges) satisfying the defined degrees for the nodes (see example in Figure B.1).

![Networks with nodes of equal degrees](image)

**Figure B.1:** Networks with nodes of equal degrees

Consider the example in Figure B.1: the set of all possible connections is defined as \( e_{ij} \) with \( i, j \in N \{1..n\} \) and \( j > i \).

Using the first approach it is easy to enumerate the list of all possible sets of \( n - 1 \) edges (in this case 6) as shown in Table B.1.

Every row in the table represents a possible network structure. The columns show the edges \( e_{ij} \) (indicated by a 1) used by the structure. Network structures that do not meet some requirement are ignored.

The idea is to interpret each row as a number consisting of the digits 1 and 0 with the least significant digit rightmost in the sequence. This number is then increased at each step while satisfying the constraint that the sum of the digits (i.e., the number of edges) is \( n - 1 \).

Suppose \( m = n - 1 \) edges are required, the algorithm (see Listing B.1) works as follows:
Table B.1: List of possible edge combinations for six nodes

1. Initialisation Step: Start with the smallest possible number, i.e., set the rightmost m digits to 1 and set the rest of the digits to 0

2. Incremental Step: Starting from the rightmost digit, find the first non-zero digit dd with 0 to its immediate left. Set dd to 0 and set the digit to its left to 1. The digits to the right of dd are then reassigned as follows: Starting from the rightmost digit, set as many digits to 1 as necessary to make up a row total of m; set the rest to 0.

The algorithm terminates when the largest possible number is reached (see the last row in Table B.1).

The transformation into a network structure works as follows: Initialise a data model with the set of n nodes. Add the edges to the nodes according to the indices. The generated network must have enough ports, be cycle-free and fully connected. The first requirement is implicitly met by the algorithm and the second by the way the network is constructed: Every subnet is represented by a tree of nodes connected to
it. Two nodes are only connected if they belong to two different subnets, ensuring that there are no cycles. In this case, the two subnets are merged and the result is represented by a tree constructed by attaching one tree to the root of the other tree.

Listing B.1: Enumeration of edge combinations

```c
1 declare e[0..max];
2 void init(m){
3     for(i=m; i<max; i++){}
4         crosstotal=crosstotal-e[i];
5         e[i] = 0;
6     }
7     for(i=max; crosstotal=max.edges;--i){
8         e[i] = 1;
9         crosstotal++;
10     }
11 }
12 int increment(){
13     for(i=max; e[i]==0;--i);
14     if(i!=max.edges){
15         e[i]=e[i]-1;
16         for(i=i-1; e[i]!=0;--i);
17         e[i]=e[i]+1;
18         init(i);
19         return PROCEED;
20     } else{
21         return END;
22     }
23 }
24 void enumerate(){
25     crosstotal=0;
26     init(0);
27     while (PROCEED==increment()){
28         // new combination of edges
29     }
30 }
```

The third requirement follows since a cycle-free network of $n$ nodes and $n - 1$ edges is necessarily fully connected.

Now, an algorithm is needed to enumerate all possible combinations of degrees for the network. The sum of the degrees of the nodes always equals twice the number of
edges in a network. Here it is equal to \((n - 1) \times 2\). We use an algorithm similar to B.1, but in this case the row total is \((n - 1) \times 2\) and the minimum value for a digit is 1 since every node is connected.

Listing B.2: Enumeration of degree combinations

```c
max_degrees=(n-1)*2;
declare d[0..n-1];

void reinit(i){
crosstotal=max_degrees;
v=d[i];
for(i=i+1; i<n-1; i++){
crosstotal=crosstotal-d[i];
d[i]=v;
crosstotal=crosstotal+v;
}
d[i]=d[i]+max_degrees-crosstotal;
}

int inc(){
for(i=n-2; i>=0; --i){
if(d[i]<d[i+1]){
d[i]=d[i]-1;
}
}
for(i=i-1; i>=0; --i){
if(d[i]<d[i+1]){
d[i]=d[i]+1;
}
}
return i;
}

void enum(){
for(i=0; i<n-1; i++){
d[i]=1;
}
d[i]=max_degrees-n+1;
while(true){
i=inc();
if(i<0) break;
else reinit(i);
}
```

B-5
The initializing and incremental rules are as follows:

- Initialize all digits to 1. The rightmost digit is increased to make the row total equal to \((n - 1) \times 2\).
- The rightmost digit dd bigger than its left neighbour is decreased by one; the neighbour is increased by one.
- The digits to the right of dd are re-initialized: all digits are set to the value of dd except the rightmost digit which is assigned a value that makes the row total \((n - 1) \times 2\).

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table B.2: List of degree combinations for networks with six nodes

The algorithm terminates when the reinitializing step can not be successfully executed. The list of degree combinations for the given example is shown in Table B.2.

**B.2 Assignment to Groups of Different Size**

To enumerate all the possible assignments of the device nodes to a network of switches simply enumerate all combinations of combinations of devices.

For example, there are 10 ways \((1/2 \times \binom{6}{3})\) to connect exactly three out of six nodes to each of two switches (see Table B.3).

Each row shows a possible distribution of the nodes \(n \in \{1..6\}\) onto two groups. The first column is the index of the iteration.
Table B.3: Combinations for two groups with three out of six nodes

To enumerate all possible combination of nodes in a sequence of \(i\) groups, enumerate all combinations in the remaining \(i-1\) groups for any combination in the group \(i\) in a list of groups.

For groups of different sizes, the algorithm must be slightly modified: Groups of the same size are bundled in a set and handled by the algorithm as before, i.e., independent of the other sets of groups. This ensures enumeration of the combinations of all sets of groups of the same size. To get all the required combinations, all the different combinations of the different sets are combined with each other. Thus any combination of the first set is combined with every combination of the second and the resulting combinations of these two sets are combined with the next set of groups and so on.

\section*{B.3 Optimum Switch Positioning}

The connections between all network nodes are already established when the positioning of switches takes place. This fact downsizes the task onto a geometric problem depending on the cabling rule that defines the distance between two nodes.

The optimum position of a switch is simply the position where the total distance between the switch and all its connected devices is minimal. This position can be found by the center function provided by the NDF which matches the given cabling rule.

When more than one switch is used to establish the network, switches are connected
to each other, and the positions of the nodes (consisting of devices and switches) connected to a switch are not fixed. One way to solve this problem is to regard the network as a hierarchy. Every switch in this hierarchy has a set of children and a parent. At the lowest level, the set of children are the devices.

Given this hierarchy, an approximation can work like this: Start with an initialisation phase. Beginning at the lowest level walking upward towards the root, position each switch at the center of their children. At this stage the switches are located near their optimum position. To find the optimum position, a min-max algorithm can be applied. Again the algorithm walks through the hierarchy and positions the switches. But this time, the set of all nodes including the parent node is used for the calculation of the center position. If all switches are positioned again, the difference in the overall cable length to the former solution is calculated. If the difference to the former intermediate solution is bigger than a predefined precision of the overall cable length the last procedure is repeated to get the next (further optimized) solution.
Appendix C

Examples of Data Flow and Functional Programming

C.1 Data Flow Example

Listing C.1 shows a simple example of source code in the imperative style.

Listing C.1: Imperative programming language example

```plaintext
1 S = X + Y;
2 S = 3 * S;
3 S = S/3 + F(S);
```

Rewriting the code according to the rules of the Data Flow paradigm (assignment only once) requires the introduction of new variables (see C.2).

Listing C.2: Single assignment convention example

```plaintext
1 S := X + Y;
2 S1 := 3 * S;
3 S2 := S/3 + F(S);
```
This form of programming leads to special problems: For example, a very often used construct for iterations in imperative languages such as the `for` statement is not applicable in this way.

The set of statements in Listing C.3 can be reordered in a data flow graph shown in Figure C.1 by the application of the transformation rules.

Listing C.3: Decomposition Example

```plaintext
1 P := X + Y
2 Q := P / Y
3 R := X * P
4 S := R - Q
5 T := R * P
6 RESULT := S / T
```

Figure C.1: Decomposition according to Data Flow rules

The graph shows which instructions can be executed in parallel (instructions on the same level in the tree) and which of them must be synchronised until the appropriate prerequisites are calculated.
C.2 Functional Programming Example

The example in Listing C.4 shows a simple sequence of functions which can be de­
composed in a wood of trees of operations to be processed (see Figure C.2).

Listing C.4: Example for a sequence of function call statements

```
1  A( B() , C( D( E() , 4 ))) ;
2  F( C() , D( E() , 4 )) ;
```

![Diagram](image)

Figure C.2: Decomposition of a functional program

Intermediate results of Z1 and Z2 and Y1 and Y2 are the same because no side effects
are allowed. Therefore it is also allowed to reuse the intermediate result. This leads to
a tree without redundant operations (C.3).

![Diagram](image)

Figure C.3: Elimination of redundancy in a functional program
The process can now begin to “reduce” the trees by parallel execution of the functions from top to bottom.