Decentralized Throughput Optimization in Industrial Networks

Uwe Röttgermann

Technische Universität München

Institut für Informatik
der Technischen Universität München

Decentralized Throughput Optimization in Industrial Networks

Uwe Röttgermann

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaft (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:

Univ.-Prof. Dr. Christoph Zenger

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Dr. h.c. mult. Wilfried Brauer
2. Hon.-Prof Bernd Schürmann, Ph.D. (Univ. of Cape Town / Südafrika), Johann-Wolfgang-Goethe-Universität Frankfurt am Main

Die Dissertation wurde am 23. Dezember 2004 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 7. September 2005 angenommen.

Für meine Eltern.

Danksagung

An erster Stelle möchte ich meinen Doktorvater Herrn Univ.-Prof. Dr. Dr. h.c. mult. W. Brauer für seine Unterstützung und seine Geduld danken. Ohne seinen Zuspruch auch in schwierigen Phasen meiner Doktorarbeit wäre diese Arbeit wohl nie vollendet worden. Einen herrzlichen Dank auch der Siemens AG für ihre finanzielle Unterstützung und Herrn Prof. Schürmann in dessen Abteilung bei CT IC4 ich meine Arbeit schreiben durfte. In solch einem hervorragendem Team fällt es nicht schwer für schwierige Fragen den richtigen Ansprechpartner zu finden. Ganz besonders gilt mein Dank meinem Betreuer Herrn Dr. Sollacher. Viele Zusammenhänge wären mir ohne seine Hilfe nie klar geworden.
Meiner Freundin danke ich für ihre fortwährende Unterstützung, ihrem Einsatz und vor allem ihrer Geduld und dem Glauben an meinen Erfolg. Zu guter Letzt gilt mein größter Dank meinen Eltern für all ihre Unterstützung. In Gedanken bin ich bei meinem Vater, der viel zu früh von uns gegangen ist.

Kurzfassung

Die Optimierung des Durchsatzes ist eines der bedeutendsten Problemstellungen der industriellen Optimierung. Die kundenorientierte Produktion und die zunehmende Komplexität der Produktionsprozesse wurden in den letzten Jahren zu einer Hauptaufgabe der Optimierung. Als Konsequenz werden heute flexible, robuste und adaptive Ansätze benötigt, wie zum Beispiel die autonome und dezentrale Kontrolle von Teilprozessen.

Diese Arbeit stellt ein dezentralen Kontrollansatz vor, der mit Hilfe von intelligenten Agenten den Durchsatz in einem industriellen Prozess optimiert Als ein ersten Schritt der systematischen Untersuchung konzentriert sich die Arbeit auf single-commodity Netzwerke. Folgende Ergebnisse wurden erreicht:

i) Ein Konzept zur Modellierung realer single-commodity Produktionsnetzwerke: Diese Konzept erlaubt es alle möglichen single-commodity Produktionslinien mit Hilfe eines einfachen modularen Ansatzes zu modulieren. Industrielle Komponenten können von einem einzigen Basis-Modul abgeleitet werden

ii) Theoretische obere Grenze des Durchsatzes: Basierend auf dem realistischen Model eines industriellen Netzwerkes kann die theoretische mittlere obere Grenze des Netzwerkdurchsatzes berechnet werden. Es ist das Ziel dieser Durchsatzoptimierung den tatsächlichen Durchsatzes in Richtung der theoretischen oberen Grenze zu optimieren.

iii) Spezifikation eines Maximum-Durchsatz-Controllers: Zu jedem Modul gehört ein dezentraler agentenbasierter Controller. Dieser ist für die Verteilung der Warengüter zu seinen benachbarten Modulen verantwortlich. Es zeigt sich, dass der aktuelle mittlere Netzwerksuchsatz gegen die theoretische obere Grenze konvergiert, wenn alle lokalen Verteilungsregeln garantieren, dass so viele Waren verteilt wie möglich werden.

iv) Reduktion der Durchsatzoszillation: Aufgrund von Hysterese - Effekten der endlichen Bearbeitungszeit und der endlichen Puffergröße kommt es zu zeitlich variierenden Netzwerkdurchsätzen. Das vorgestellte Konzept kontrolliert lokal das Exportverhalten eines Agenten um die lokale Ausflussoszillation zu minimieren. Somit lässt sich auch die netzwerkweite Durchsatzoszillation vermindern.

v) Lokale Regeln optimal lernen: Regeln unterscheiden sich hauptsächlich in ihren konvergierenden Verhalten und ihrer Resourcen-Beanspruchung. In dynamischen

Umgebungen ist schnelle Adaption grundlegend wichtig. In Berücksichtigung dieser Vorraussetzung werden verschiedenen Optimierungsansätze miteinander verglichen: Learning to avoid blocking (LAB), Reinforcement Routing (RR), Ant-based Routing (AR) und ein Link State Routing (LSR) Ansatz. Der LSR Ansatz benutzt globales Wissen und konvergiert dementsprechend sehr schnell, belastet die Netzwerkressourcen sehr stark. RR und AR bedürfen weniger globale Netzwerkinformationen, Sie zeigen, dass sie schnell konvergieren und dabei sparsamer mit den Systemresourcen umgehen als der LSR Ansatz. Der LAB Ansatz benutzt hingegen nur lokale Informationen. Dementsprechend konvergiert er weniger schnell wie die anderen Ansätze, aber beansprucht auch sehr viel weniger Systemresourcen.

Abstract

Throughput optimization is an important problem of industrial optimization. Customer driven production and the increasing complexity of the production processes have become major challenges within recent years. As a consequence flexible, robust and adaptive approaches are required, e.g. autonomous and decentralized control of sub-processes.

This thesis presents a decentralized control approach based on intelligent agents optimizing the throughput in industrial production processes. As a first step of systematic investigation this thesis focuses on single commodity networks. The following results have been achieved:

i) A concept for modelling realistic single commodity production networks: This concept allows to model all possible kinds of single-commodity production lines using a simple modular approach. The industrial components can be derived from just one basic module.

ii) Theoretical upper bound of throughput: Based on this realistic model of an industrial network the average theoretical maximum bound for the network throughput is calculated. The goal of throughput optimization is to push the current network throughput towards its theoretical upper bound.

iii) Specification of maximum throughput controller: To each module a decentralized agent-based controller is associated. It is responsible for the distribution of commodities to its neighbour modules. It is shown that the current average network throughput converges to its theoretical upper bound if all local policies for commodity distribution guaranty that as much as possible units are exported.

iv) Reducing throughput oscillations: The outflow of components can vary in time because of hysteresis effects due to finite processing time and storage capacity. The concept introduced controls the export behaviour of the agents in order to minimize the outflow oscillations. This concept also reduces the complete network throughput oscillation.

v) Learning optimal local policies: Policies mainly differ in their convergence behaviour and resource requirements. In dynamic environments fast adaptation is mandatory. For this purpose several optimization approaches are compared: Learning to avoid blocking (LAB), Reinforcement Routing (RR), Ant-based Routing (AR) and a Link State Routing (LSR) approach. The LSR approach uses

global knowledge and therefore converges very fast, but also requires lots of system resources. RR and AR use less global network information; they show fast convergence while using less system resources than LSR. The LAB approach uses only local information; therefore, it converges not as fast as the other approaches, but requires much less system resources.

Contents

Chapter 1

Introduction

Over the past few years, analysis of successful manufacturing and retail companies shows that in today's hyper-competitive business environment, supply chain efficiency is a necessary condition for survival [SAP02]. In that respect, it is not astonishing that the optimisation of supply chains is itself a blended business and a huge number of companies like *SAP* or *i2* and *Oracle* offer *Supply Chain Management* (SCM) and *Enterprise Resource Planning* (ERP) software. Because of the increasing complexity of production and the necessity to adapt rapidly to marketplace changes, modern SCM and ERP software tools start to focus their attention on *adaptive supply chain networks* [SAP02]. Those tools are characterised by distributed control, parallel and dynamic information propagation and real-time analytics. In addition to the traditional supply-chain management solutions of planning, execution, coordination, and networking some new technologies like *agents* have become popular in the last years. Agents go along with the distributed control aspect of modern adaptive supply chain networks. Each local agent is responsible for a specified part of the complete process. The advantage is that each agent requires only a part of the complex global model which is in most cases simpler and easier to implement. One might visualise the agent approach as a huge number of intelligent entities. The entities work had in hand to fulfil a common goal, like i.e. ants organising their colony together.

Along with the advantages of this new agent technology, new problems occur. Local control implicates mostly local knowledge. If the action of the agent is based on local information, how should this interaction be designed to fulfil global objectives? A lot of different designs and optimization approaches are discussed in literature today [VL99, Kay01]. But there is no common design known, which is suitable for all purposes. In our view the purpose determines the agents design. This thesis introduces a network agent design and optimisation algorithms which deal with problems that occur when distributing commodities in an industrial environment. In the following we summarise the goals of the thesis.

1.1 Goals of this thesis

The goals of this thesis can be summarized into three main goals:

Decentralized design: As we already mentioned, the structure of modern facilities is very *complex*. Their production system consists of many subsystems each with non-trivial dynamical behaviour and highly dynamic interaction with other components. In order to scale down the complexity of the facility process we can split the control locally. In such a decentralised design local controllers communicate with others in their local neighbourhood on basis of the information received, and coordinate their actions to meet some global objectives. Then, they constitute a self-organising network. Another advantage of decentralised approaches is their *scalability*. Adding or removing components does not require new configuration of the network design. Using the same concept we can built out of the same set of simple modules different production scenarios, like i.e. warehouses, power plants, distribution centrers or an facility in automobile industry.

Finally, decentralised approaches show a large degree of *flexibility*. They adapt to changes or failures of components by self-organising. Therefore, the goal of this thesis is to introduce a multi-agent system which is able to control distribution processes in industrial environments.

Modular concept: The model of an industrial network has to be as exact as possible but also as simple as possible. Therefore, another goal of this thesis is to develop a suitable modular concept which allows realistic testing and realistic simulation of an industrial scenario. The concept should be built with modules which can be combined to construct all kinds of production systems.

Throughput Optimisation: Optimising distribution processes can have several different goals. Some optimisations try to optimize the allocation of buffers, others try to reduce the transportation times (respond times) of commodities and so on. In this thesis we put particular attention on the goal to optimize the commodity throughput. While doing so we focus on approaches with fast convergence toward its optimal throughput and fast adaptation to changing environments. Another focus is to minimize fluctuations in the resulting throughput.

1.2 Outline

This thesis consists of 6 chapters including the Introduction. The following gives a short outline of the contents of each chapter.

Chapter 2 is an overview about state of the art approaches to optimise and model industrial networks. This chapter presents the limitations of current approaches. Two parts in this chapter are presented in more detail. They are important for the other chapters. First, we present the Graph Theory in more detail. This traditional approach is used in further chapters to deduce some theoretical results and to specify the design

for the modular concept to model any kind of production process. Second, we discuss the multi-agent approach in more detail and list some of their popular routing approaches. We will see then why some approaches are more suitable for our purpose than others.

In chapter 3 we introduce a new architecture for the modelling of networks with industrial properties. This new architecture builds up out of an unique basic controller which can be combined to a complex network. This network shows equally behaviour like industrial networks of machines or shelves. The use of this architecture allows to simulate the commodity throughout of an network, e.g. a warehouse or an industrial plant.

Chapter 4 gives us the theory and the applications to maximise the commodity throughput from a specific source to specific sink. We introduce two controller concepts for optimisation. One controller stands out for its simple design and good results while the other guarantees maximum throughput. Different strategies (policies) for the decentralised controller are compared. Some of the policies are new, others are deduced by existing methods of chapter 2.

In chapter 5 we discuss the quality of the introduced optimisation approaches. For this we use a test network that is also introduced in chapter 5. Finally, chapter 6 summarize the results and gives a short outlook of further investigations.

Chapter 2

State of the art

The traditional approach to examine dynamic systems is to introduce a mathematical model which fits most of the interesting features. The problem is often the complex mathematical representation of the system. In such situations a possible solution is to simulate the system. Simulations experiment with an executable model of the system. A well-designed simulation has the advantage that the underlying equations are integrated numerically and the results are directly shown in the observables. In the last decade, simulations have become an essential part of scientific analysis as the computational power of computers has increased enormously.

The models of our interests consist of many parts which are heavily connected locally with each other. Many fields of inquiry, including applied mathematics, computer science, engineering, management, and operation research, are working on optimising such dynamic systems. The following section introduces some of the most important fields of optimisation. The following enumeration is neither complete nor a representative summary but focuses on important fields for this thesis. This discussion follows in part Parunak et al. [VSR98].

2.1 Approaches to investigate dynamic systems

2.1.1 Control theory

The *control theory* approach originated with Simon [Sim52] and is part of the work for which Simon was awarded with the 1978 Nobel Price in economics. Control theorists model systems with differential and difference equations and use mathematical tools, like Laplace and Z transforms, to study their dynamical behaviour. Modern control theory approaches use also Fuzzy Logic Controller and neural networks. One of the first who studied supply chains by simulations and control theory methods was Forrester [For68]. He and his students formulated supply chains by difference equations and then used their software tool *Dynamo* to sum them numerically. Their approach was so successful that it has led to a new branch of industry called *system dynamics*. Today, modern industrial simulation tools like *DYMOLA* or *MATLAB/Simulink* are used for modelling

and simulating dynamic systems. The control theory approach has the advantage that it is explicitly dynamic, even though the tools to study such equations make it most applicable to linear systems.

2.1.2 Operations research

Operations research is an approach which does not rely on linear assumptions and is able to handle linear as well as non-linear systems. In principle all mathematical tools to solve such abstract problems are used but the main focus is on optimization theory, game theory and statistical analysis. While not assuming linearity, this approach makes other strong assumptions about the underlying statistical distribution; and it also usually focuses on time averages and steady states rather than dynamical behaviour.

2.1.3 Graph theory

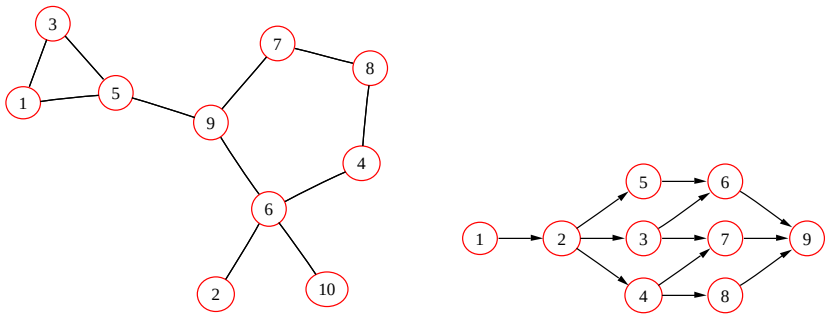
Graph Theory is a topological approach. Optimising flow in a network is a seminal application in *Graph Theory* [Die00, EWHPS96, AMO93, BG87] providing a rich number of tools to solve the maximum-flow problem. Graph Theory is very useful for the depiction of networks. Good overviews are found in standard literature [AMO93, BG87, Bol01, Die00, EWHPS96, FF74]. In a graph members are represented by nodes and the relationship between them is represented by edges connecting them. In this way, a graph G is a mathematical description of the relation of members. A mathematical definition is found in [Die00].

An example of a graph is shown in the left part of Fig. (2.1). Nodes are represented by circles and edges by lines connecting circles. This kind of graph is also called undirected, because all edges are equally matched in both directions.

A directed graph is shown on the right side of Fig. (2.1). Its edges are matched only in one direction. Arrows are representing the possible flow direction. A *path* through a graph G is an alternating sequence of distinct nodes and edges, beginning and ending with nodes [EWHPS96]. If in the graph there is no path of any length which has equal start and end nodes, then the graph is called *acyclic*. The directed graph shown in this figure is acyclic. The idea behind networks is that the edges of its graph carry some kind of flow - of water, electricity, data, commodities or similar [Die00]. These flows are usually restricted by capacities of the network edges. A definition of a network is found by Weinert [EWHPS96]:

Definition 1 (Network) A network $N = (V, E, c, s, t)$ is given by a directed graph $G(V, E)$ and two designated nodes, a *source* s and a *sink* t and a non-negative *capacity* function $c : E \to \mathbb{R}^+$.

The assumption of only one source and one sink is justified because for every set of sources and for every set of sinks we can define one *super-source* and one *super-sink* which connects to all sources and respectively to all sinks. An example of a small network is shown in Fig. (2.2). One node is specified as source and another one as sink. This network is connected and acyclic, nevertheless there are many different possible



(a) undirected (b) directed

Figure 2.1: Two different types of graphs: Undirected graph (a), directed graph (b)

paths from source to sink. A capacity is assigned to each edge which limits the possible throughput of the edge.
We define two kinds of sets now. A *Children Set* which contains all nodes that are connected by incoming edges of node n and a *Parent Set* which contains all nodes connected by outgoing edges of node n, see also Weinert [EWHPS96]. For a set $X \subseteq E$:

$$\begin{aligned} \text{Children Set:} \quad & \Gamma^+(X) = \{y \in V \setminus X | \exists\, x \in X : (x,y) \in E\} \\ \text{Parent Set:} \quad & \Gamma^-(X) = \{y \in V \setminus X | \exists\, x \in X : (y,x) \in E\} \end{aligned}$$

As an example see node 7 in Fig. 2.1(b). Its parents are nodes 3 and 4 and its child is node 9. A map $f : E \to \mathbb{R}^+$ assigning a real number to each edge is called *flow* of the network if

$$\forall n \in V \setminus \{s,t\} : \quad \sum_{u \in \Gamma^-(n)} f(u,n) = \sum_{w \in \Gamma^+(n)} f(n,w) \tag{2.1}$$

$$\forall e \in E : \quad 0 \leq f(e) \leq c(e) \tag{2.2}$$

The first equation is also known as Kirchoff's law. A flow f in a network $N(V,E,c,s,t)$ has the value $w(f)$:

$$w(f) = \sum_{u \in \Gamma^+(s)} f(s,u) - \sum_{w \in \Gamma^-(s)} f(u,s) \tag{2.3}$$

A flow f is called *maximal* if $w(f') \leq w(f)$ for all flows f' in network N [EWHPS96]. The *maximum flow problem* seeks a feasible solution that sends the maximum amount of flow from a specified source node s to another specified sink node t [AMO93]. The maximum flow is the maximum number of units the network is able to transport from source node s to sink node t. Graph Theory provides a rich number of tools to solve the maximum flow problem. Good overviews about algorithms for solving the maximum flow problem are found in Ahuja and Orlin [AMO93], Papadimitriou and Steiglitz [PS98] and

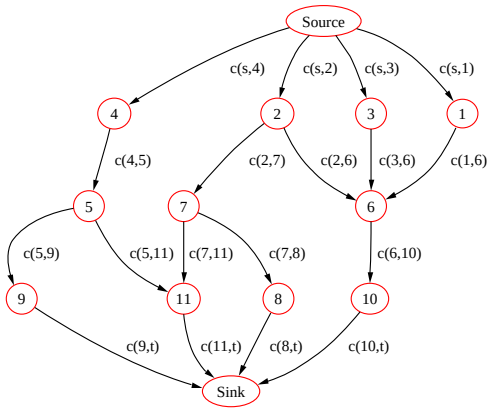


Figure 2.2: A small example (acyclic) network with 13 nodes. Two of them are designated as source s and sink t. Each edge has its own capacity.

in Goldberg's[1] notes about recent developments in maximum flow algorithms [Gol98]. The most famous tool is the algorithm of Ford and Fulkerson. The Ford-Fulkerson maximum flow labelling algorithm [FF56] was introduced in the mid-1950's, and became a seminal work. This algorithm is pseudo-polynomial. In the appendix A we introduce the maximum flow algorithm which is used in this thesis.

2.1.4 Distributed control

A new field which has become popular in recent years can be summarised by the term *distributed control*. It has been shown that centralised approaches have become impractical with the increasing complexity of real networks. The goal of distributed control is to automate processes locally. One of the biggest advantages is that the complexity of the process can be reduced locally because not all information is needed to solve sub-processes. Furthermore, distributed systems are more robust; local breakdowns of components do not in most cases affect the functionality of the complete system, the system reacts with more flexibility. An other advantage is scalability of distributed systems. The concept does not have to change because of the number of simulated entities; and adding or removing components does not require new configuration of the network design. Let us point out again that both concepts are theoretically identical but the agent formulation seems to easier because of the personalised view we are used to.

One concept for distributed control is a *multiagent system*. An agent is a local entity which acts on basis of local information. The agent is designed to fit the local property of the system. In sum, all agents behave like the dynamic system of interest, if the multiagent system is designed carefully. Theoretically, one can construct a set of Partial

[1]The fastest algorithm available at the moment seems to be the algorithm of Goldberg and Rao [GR97]. Its computational performance is of $O(min(n^{2/3}, m^{1/2})m) \cdot log(n^2/m)$ (m is the number of edges and n the number of nodes in the network)

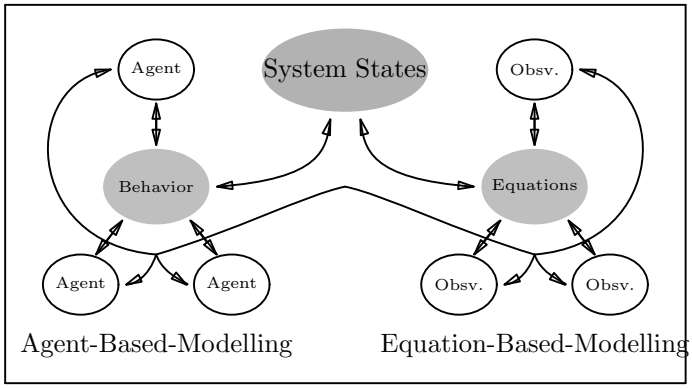


Figure 2.3: Multiagent System and Equation-Based-Modelling. Based on a discussion in Parunak et al.[VSR98]

Differential Equations (PDE) that completely mimics the behaviour of any multiagent system, see Baumgaertel et al. [BBV$^+$01]. Nevertheless, the use of agents has some advantages. It is often easier to set up rules and behaviours of a personalised entity than to develop system behaviour in mathematical expressions. Fig. (2.3) demonstrates the equality of both concepts. The left side shows a multiagent system where agents interact. The agents mimic the system properties. On the left side of Fig. (2.3) the corresponding non-agent system is shown. The observables are not represented by agents but directly related by mathematical expressions. These equations could either be algebraic, or ordinary differential equations over time or over time and space (partial differential equations).

Parunak et al. [VSR98] were among the first who used multiagent systems to analyse supply chains and logistic problems. Their DASHh (Dynamic Analysis of Supply Chains) is a multi-agent approach for simulation and understanding the behaviour of supply chains. For this Parunak et al. [VSR98] use supply chain examples, which are small enough to examine them still analytically but complicated enough to present non-linear behaviour. The agents implemented in DASCh are of three species: Company agents, PPIC agents and Shipping Agents. Company Agents represent the different firms that trade with each other. PPIC is the product planning and inventory controlling agent used by the Company agents. The agents model the delay and uncertainty dependent on both material and information between trading partners. The PPIC agents use simple forecasting approaches. An interesting result of their work is the inventory oscillation, which is also shown in my simulations. Their interest lies in understanding such non-linear behaviour.

In this work, a different multiagent approach is introduced to optimise the throughput of industrial networks with their specific characteristics. The next section presents a more detailed view about the multiagent optimization.

2.2 Multi-agent Approach

In our context of network optimisation we can roughly divide the distributed control approaches into two classes: *Market-based approaches* and *routing approaches*.

Market-based approaches

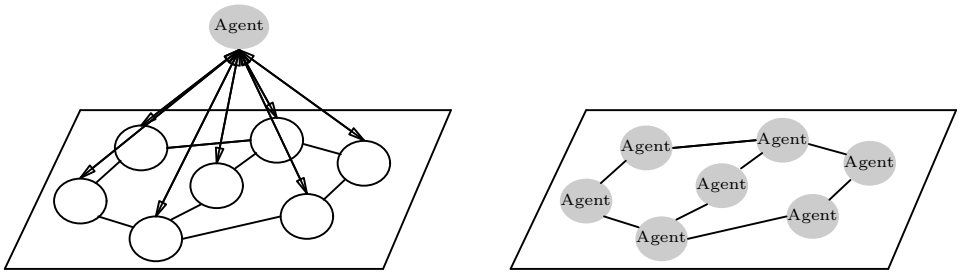
In market-based approaches agents negotiate the exchange of goods in a formalised procedure. The process is similar to real negotiations between producer and customers and includes voting, auctions, and general equilibrium market-based mechanisms. A good overview can be found at Sandholm [San99]. Market-based approaches have been successfully used in *distributed-resource-allocation* problems [CMM97, Cle96]. Their advantage is that the negotiations are based on real qualities like prices and demands. In that respect, those approaches are successful in imitating real human behaviour in negotiations. Thus, well-known economic techniques for market-controlling can be used. A drawback is that those negotiation processes become complicated if for example no common equilibrium (nash or pareto) for the negotiation members exists or system optimality is different from member optimality.

Routing approaches

A routing algorithm in industrial networks has to direct units from a source to a sink. The routing algorithm has to pay attention to different constraints which are imposed by the underlying network and its nodes' properties (chapter 3.1). A good overview about routing algorithms is found at Di Caro and Dorigo [CD97], who also classified routing algorithms. Following their discussion, routing algorithms can be classified into *centralised* or *distributed* and *static* or *adaptive*.

In a *centralised* routing approach a selected agent instructs all network agents to distribute their units. For the instruction the agent has to collect all information from all nodes. So, centralised algorithms are only useful if the delays necessary to gather information about the network state are negligibly low compared to the transportation time of units. In most industrial networks the delay of gathering compared to the unit distribution is low enough to be neglected. Examples are logistic systems like warehouses. The information exchange between forklifts and shelves could be implemented in such a way that the information is exchanged immediately but of course this is not possible for the exchange of goods.

In *distributed* or *decentralised* routing approaches each node has an agent which is responsible for the nodes' distribution of units. The agents gather information about their neighbourhood and about the network state. But information exchange occurs only among neighbours. Information is transported from agent to agent until it reaches its destination. Information distribution works equally as unit distribution. But it occurs much faster. Using the gathered knowledge the agents calculate a distribution policy only for their own nodes.



(a) centralised Approach (b) Decentralised Approach

Figure 2.4: Difference between central and decentralised routing. In a *centralised* approach (a) one selected agent has the task to determine the distribution policy for each network node. The selected agent has not to be member of the network. In a *decentralised* routing (b) agents are responsible for each node in the industrial network. They have to decide locally the distribution policy.

In *static* routing, the path of a unit is determined only on the basis of its source and its sink, without regard to the current network state. The path changes only in case of faulty edges or faulty nodes.

In *adaptive* algorithms, the path of a unit is adapted to time and spatially varying traffic conditions. As a drawback, they can cause oscillations in selected paths [BG87]. This fact can cause circular paths, as well as large fluctuations in measured performance [CD97].

Using this classification only four kinds of approaches are possible: *static-centralised*, *adaptive-centralised*, *static-distributed* and *adaptive-distributed*. In the last years, more and more attention has been drawn on decentralised approaches. The reason is that they are usually more robust than centralised approaches [BG87]. In decentralised routing, each node has an agent which decides a distribution policy based on the information exchange with its neighbours. A sketch of a decentralised network is shown on the right side of Fig. (2.4). If any agent fails, it has only influence on its local neighbourhood. All other agents are still able to distribute units based on the new network state. If the agent fails in a centralised approach, no new policy could be distributed on the network. This is shown on the left side of the figure. This agent serves to all nodes in the network. A failure of this central agent would cause a break down of the whole network. In this thesis I pay particular attention to decentralised routing approaches because of their robustness against failure of nodes and edges. Decentralised approaches can manage re-routing by themselves.

We have seen that the agent approaches could be classified into centralised, distributed, static and adaptive. Another classification is based on the degree of information which is available for agents. The agents use the information to get a perception of their environment. Usually, better data results in better solutions. On that respect, agents with more information about the system should perform better than agents with less knowledge. To simplify the graduation of the knowledge, two classifiers are introduced.

Overview about common routing algorithms:		
Knowledge:	Approaches:	Examples:
Global	Link State Routing	OSPF[Hen88], LVA[Beh97]
Local	Distance Vector Routing	RIP[Hen88], ARPANET
	Reinforcement-Learning	Bates[Bat95], Littman[BL94]
	Ant-Algorithms	AntNet[CD97]

Table 2.1: A small overview about common routing approaches. For each approach exist a large number of variations and extended applications.

Agent approaches can be classified with *global knowledge* and *local knowledge*.

In *global knowledge* (complete knowledge) approaches agents collect all information which is available globally. Information is mapped completely to get an internal representing map. As we will see, the complete internal knowledge about the system allows agents to optimize their distribution policy D. In small industrial systems *global knowledge* approaches maximise the network throughput very well. But information exchange needs system resources as well as storage capacities at the agent. For small and sparse industrial networks, these costs are still insignificant, but not in large and dense industrial complexes like distribution centrers. Spreading out the complete network state from node to node with reaching all agents takes a lot of resources. Additionally, the computational effort increases at least linearly at each node with shortest path algorithms and up to polynomial with maximum flow algorithms [Goo99, PS98].

In *local knowledge* approaches, agents have only a local impression of the complete network. Within their neighbourhood they share only knowledge among themselves and about incoming messages. It is not necessary to know each agent state. The goal of such approaches is to determine the minimum amount of exchanged knowledge which is necessary to fulfil the designed objectives of the agent system.

Most algorithms for the optimization of network flows use one of the four routing approaches which are listed in Tab. (2.1): *Link State Routing, Distance Vector Routing, Reinforcement Routing* and *Ant Algorithms*. Actually, most approaches are a combination of two or more of these general ones. They are used for a wide variety of networks like data networks or communication networks and of course the Internet. Depending on their purpose they differ in their implementation. A good overview about routing algorithms is also found at Heusse et. al. [HSGK98]:

Distance Vector Routing: The principles of distance vector routing (Bellman-Ford routing) are based on the principles of dynamic programming [Ber82]: an optimal path is made of sub-optimal paths. Each node i periodically updates its distance vector from the distance vector which is regularly sent by its neighbours as follows:

$$\mathcal{D}^i_{n,d} \leftarrow \min\{d_{i,n} + \mathcal{D}^n_{j,d} \,|\, \text{for all neighbours } n \text{ of } i\}\,, \mathcal{D}_{i,i} := 0 \qquad (2.4)$$

where $d_{i,j}$ is the assigned cost (i.e. distance) to the edge connecting node i with its neighbours n and $\mathcal{D}^i_{n,d}$ is the cost estimated by i for delivering a packet from

i to d passing neighbour n. It has been shown that this process converges in finite time to the shortest path with respect to the used metric if no edge cost changes after a given time [BG87]. These algorithms are not used anymore in modern Internet protocols, because the convergence is often too slow and the protocol is more adaptive to the appearance of new edges than to the failure of edges.[HSGK98, Tan96].

Link State Routing: Each node contains a dynamic map of the complete network. In that respect, it is a global knowledge approach. The dynamic map is used to estimate the optimal distances between nodes. Usually, graph theoretical algorithms like the Dijkstra algorithm[Dij59] are used. Each node periodically broadcasts its routing information to all other nodes using flooding mechanisms [BG87, Tan96]. All agents which are notified recompute their routing accordingly. Because protocols based on link state routing keep complete topology information at routers, they avoid long term looping problems of old distance vector protocols [Beh97]. The OSPF (Open Shortest Path First) protocol is a common TCP/IP routing protocol that provides robust and efficient routing [HSGK98] and is increasingly used in the Internet.

Reinforcement Routing: This is a version of Bellman-Ford routing; it performs the path relaxation steps online and asynchronously and measures path length by total delivery time and not by of the number of hops [BL94, LB93]. Once again, $\mathcal{D}^i_{n,d}$ is the cost to deliver a unit toward destination d passing its neighbour node n estimated by node i. Using reinforcement learning [SB98], the policy for unit distribution could be updated with local information only. Immediately after sending a unit to neighbour n, node i receives n's estimate of the cost associated with the remaining part of the trip, namely $\min_{j\in\mathcal{N}(n)}\{\mathcal{D}^n_{j,d}\}$ where $\mathcal{N}(n)$ is the neighbourhood of node n. The received estimate is used to update the node's own cost estimation $\mathcal{D}^i_{n,d}$:

$$\mathcal{D}^i_{n,d} \leftarrow (1-\eta)\cdot\mathcal{D}^i_{n,d} + \eta\cdot(d_{i,n} + \min_{j\in\mathcal{N}(n)}\{\mathcal{D}^n_{j,d}\}) \tag{2.5}$$

where $d_{i,n}$ is once again the cost for delivering a unit to neighbour node n and η is the so called *learning parameter* of the gradient descent. This routing approach was first introduced by Boyen and Littman [LB93] and is frequently used in different approaches [Bat95, CY96].

Ant Algorithm: Ant algorithms were first proposed by Dorigo[Dor92]. They are inspired by the observation of real ant colonies [DCG99]. Ants are insects which are optimizing their food path by using pheromones. They can find the shortest path between the food source and their nest. Artificial ants imitate such behaviour and are used to optimize paths in networks. A selected node emits artificial ants which move from one node to another until they reach their destination. The ants' decision which node they choose is based on two edge quantities: The *pheromone concentration* $\tau_{i,j}$ on an edge that connects node i and node j and the *heuristic*

value $\eta_{i,j}$ which could be i.e. $\eta_{i,j} = 1/d_{i,j}$ the reciprocal distance from node i to node j. Both values are weighted and averaged over all outgoing edges. The ant uses this distribution to decide its next node. If an ant reaches a destination a so called *backward ant* is initialized to return the exactly same path T. On its way back it deposits a quantity of pheromone $\Delta\tau_{i,j}$ on each edge that it has used:

$$\Delta\tau_{i,j} = \begin{cases} 1/L & : \text{ if } (i,j) \in T \\ 0 & : \text{ if } (i,j) \notin T \end{cases} \tag{2.6}$$

where T is the return-path done by the ant and L is the complete length of T. Additionally with pheromone evaporation the pheromone update on each edge by an ant is as follows:

$$\tau_{i,j} \leftarrow (1-\rho) \cdot \tau_{i,j} + \Delta\tau_{i,j} \tag{2.7}$$

where $\rho \in (0,1]$ is a *pheromone-trail-decay* coefficient. It has been shown that for a multi-ant system this approach successfully minimises the path length [Dor92]. The *AntNet* of Di Caro and Dorigo was shown to outperform OSPF and Bellman-Ford algorithms in their simulations [CD97].

Most of the common routing algorithms today are based on one or more of these approaches. They are used for routing protocols in the Intranet of companies or the world wide Internet architecture.

Chapter 3

Architecture for a decentralised control

Modern plants are complex systems. They are made of a multitude of different components which together fulfil a common task. Normally, the task is to manufacture products which are more or less complex conglomerates of different raw materials. To visualise the different steps which are needed to construct something, let us assume the product is a car. In this case, all workers and their machines work together to complete the new automobile. In a simplified view of the process, the production of cars starts with an incoming raw material like steel. The steel is used at each intermediate stop of the production network. The steel is formed in several punch presses, equipped then with different important parts like tyres or electronic and the engine block and is finally accomplished (lots of stops later) as a new automobile. As we see, the individual components of the network have to batch several different tasks: i.e. assembling tyres or lacquering coachworks. But all of them have in common that they work up with incoming materials and then pass the materials to following components in their network. In that respect, a few properties are needed by all components in the network. The properties are defined by the ability to import materials, to store them and to export them again. Of course this is a simplification and a generalisation of the components' functionalities but as we will see it is a sufficient specification for our purposes.

In a distributed approach, the single components are controlled by their own controller entity. We will discuss the design and configuration of the controller later in this thesis. At the moment we assume that the controller decides perfectly whether its component distributes commodities or not, and if so, the controller decides to which of the following components the commodities should be delivered. We will investigate controllers in more detail later in this chapter.

In the beginning, we focus our attention on *modelling* an industrial network. For this we introduce a generalised architecture for components in production environments. Our architecture should be precise enough to model realistic network processes but overdetailed. In network problems the most important question is how different components act together. From this follows that we model only properties which are important for

the interaction of components in the network: The ability to import, to store and to export commodities.

Here, we will present a modular concept for modelling industrial processes based on a simple basic module which allows to construct all necessary industrial components.

3.1 Modular Concept

A modular concept has to describe various different components in production processes like shelves, conveyor belts, and machines. It is only possible to describe each component with its own module if the number of different components in such a system is limited. Better concept are made of a limited number of basic modules. The combination of them allows to construct all kinds of components. In our approach the concept consists only of one single basic module.

This basic module and its combination describe only the components themselves but not their relationships. Such relationships can be described by using graph theoretical approaches like those already introduced in section 2.1.3.

In that respect our concept consists of two parts: A basic module which could be used to construct all kinds of industrial components and the graph theoretical description of the relationship of the components and their basic modules.

3.1.1 Graph Theory

Local controllers distribute the incoming material to following components. The path from component to component through the production system depends on the primary objective of the industrial system and could either be more or less fixed and set in advance or be very flexible and individual each time. In a logistic system for example, the freight, which could be anything from letters to automobiles, could take several different paths from the client to its destination. The relationship between the components and all possible paths through the industrial systems is subsumed under the generic term *topology*. Such topologies are best represented by networks like those in section 2.1.3. Such a network $G(V, E)$ is shown in Fig. (3.1).

The *nodes* $n \in N$ represent the components of the industrial complex and their *edges* $e \in E$ represent the possible ways for the transport of commodities. Transportation is usually done by transportation systems like conveyor belts or fork lifts. Of course it is impossible to transport an infinite number of units along an edge in a finite time interval. With other words, the flow along that edge $e \in E$ has an upper bound, called *edge-capacity* $c(e)$. In a warehouse for example the transportation of goods is done by fork-lifts. They are able to carry only a specific number of goods at once. An edge defines the start and destination point for a fork-lift in a warehouse. If a fork-lift finds the fastest possible path (not necessarily the shortest) toward its destination we have an upper bound of transportable goods via this edge, which is the capacity of that edge in our warehouse.

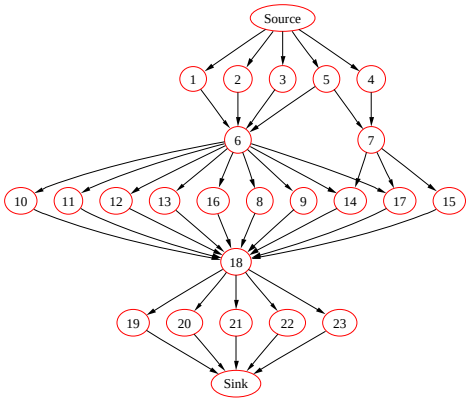


Figure 3.1: A sketch of an industrial complex. The directed graph illustrates the relationships between each component and illustrates all possible paths for the commodities through the system.

3.1.2 Basic Module

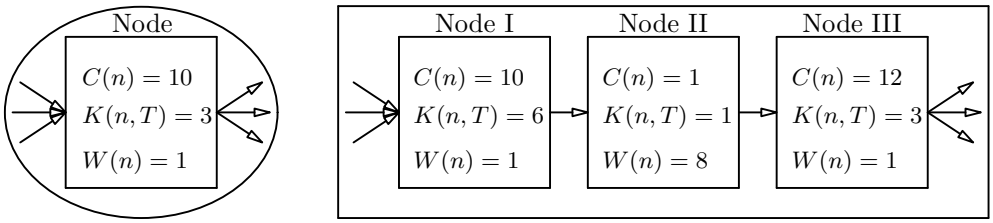
Depending on their task, components differ in their properties. Shelves, for example are able to store lots of units simultaneously. On the other side some machines only work on one unit simultaneously. Another property is the time components spend on units. They spend various different times, from a short pass-through to a long term storage. Therefore, we need a generalised concept to describe all the possible properties of components. Such an approach has to be applicable to all kinds of industrial systems, like warehouses, factories, distribution- and logistic systems. The basic module we present here only consists of three quantities:

1. *Node Capacity* C is the maximum number of units a component could work on simultaneously. A shelf for example is able to store lots of units at once. In this case, the node capacity is the maximum storage amount of the shelf.

2. *Residence Time* W is the minimum time a unit stays at the component. In the example of a producing machine the residence time is just the time needed for processing this unit. The total time a unit stays inside a component is the sum of the residence time and the delay until exportation.

3. *Inventory Level* K is the current amount of units inside the component. In contrast to both other quantities the inventory level is in all scenarios a variable in time. The others change only in the case of damage or uncertainty.

This basic module is a construct for all components, like a brick in a wall. From this follows that a component is the conglomeration of basic modules (at least of one basic module). Graph theory is the cement that joins the basic nodes together. In that respect, basic modules are also called *basic nodes*. In this thesis, a component represents an industrial entity (like a machine or a shelf). It can consist of lost of basic modules.



(a) Component - Shelf (b) Component - Machine

Figure 3.2: Two kind of components. The component shelf (a) consists only of one basic module. In contrast to the shelf component a machine (b) consists of three basic modules which are linked by edges (with their own capacity!)

For illustration purpose we demonstrate the construction of a shelf and a machine:

The component *shelf* in Fig. 3.2(a) consists only of one basic node n. The task of a shelf is to store units temporarily. In this example the component shelf could store at most $C(n) = 10$ units. Loading and unloading takes time. This shelf for example needs at least one time step $W(n) = 1$ for loading and unloading. At time T the current amount $K(n, T)$ of stored units inside the shelf was 3. In contrast to residence time and node capacity the current amount is a variable quantity which could change between $0 \leq K(n, T) \leq C(n)$.

The component *machine* in Fig. 3.2(b) is a more complicated one. It consists of three basic nodes which are identical with real subcomponents of an industrial machine. The basic nodes are connected by edges which could also have their own capacities for transportation. The capacities of the inner edges are set to infinity. The first node is the entrance of the machine. Its task is to store all incoming units until they could be produced further. In this way, an entrance is a shelf. Similarly, a shelf stores all outgoing units until they are transported further. Meanwhile, the core of the machine produces units. In this example, the machine is able to process one unit at a time, which takes 8 time steps. Here, the core is the bottleneck of the machine. The machine runs only when the entrance shelf can deliver a unit to the core and the exit shelf can still store another unit. In that respect, shelves are buffers.

Buffers are of incredible importance of industrial network investigations. Machines unfortunately tend to break down at unpredictable times. Such a break down not only influences the state of the machine itself but also influences the state of connected neighbours. The child machine has to stop as well when running out of goods as it cannot import units from the node which has stopped. Its children machines are confronted with the same problem then. In the end the whole production facility stops. To restart an already stopped facility is mostly a long-drawn-out process which takes lots of hours and is quite expensive. A way to avoid or to buffer that is to install storage places which have enough goods in stock to deliver units while the machine is being repaired. A perfect buffer is one which breaks the correlation between the interruptions of machines. In real facilities, this is also a question of costs. Stocking places are expensive as they

need space and service. The goal of optimisation is then to minimise buffer sizes while preserving uncorrelation between the interruptions of machines.

3.1.3 Extended Network

The previous discussion has shown that different industrial components like shelves and machines can be constructed out of a basic node with only three quantities (residence time, node capacity and the inventory amount). The graph theory provides the theoretical description of relationships between such components. Both, the graph theory and the concept of a basic module allows us to describe industrial networks. For this case a new definition of network is needed to implement all found properties:

Definition 2 (Extended Network) A network $N(V, E, c, C, W, s, t)$ is given by a directed graph $G(V, E)$ and two designated nodes, a *source* s and a *sink* t and a non-negative *edge-capacity* function $c : E \to \mathbb{R}^+$. Additionally a non-negative *node-capacity* function $C : V \to \mathbb{R}^+$ and a non-negative *residence-time* function $W : V \to \mathbb{R}^+$ are defined.

On the contrary to Definition 1 (pp. 6), the extended network has three additional quantities: The edge-capacities which limit the transportable number of units via an edge, and two node-specific quantities. One quantity describes the nodes' ability to work with several units simultaneously and the other defines the time the node spends at least at each unit. Another quantity of interest is the current amount of units inside a node. The *inventory-level* is a non-negative function $K : V \times \mathbb{R}^+ \to \mathbb{R}^+$ of a vertex and of time.

Most of all existing systems are dynamic. This means that their state changes over time. The load of a buffer changes over time as the inflow and outflow rates change. We discrete our system into small equal distant time steps Δt. If we choose Δt small enough we can assume that our flow rates are constant between time T and the following iteration step $T + \Delta t$. At time T we can calculate system levels like buffer sizes $K(T)$ of our nodes. And with our knowledge of former flow rates and levels we are able to decide our next actions according to a given policy. To choose Δt infinitesimal small would lead back to a continuous description. Anyway we prefer difference equations instead of differential equations, as this is the way we build up our computer model. All following equations are easily transformed into differential equations.

In section 3.1.1 we already defined a *Children Set* $\Gamma^-(X)$ which contains all nodes connected by incoming edges of node n and a *Parent Set* $\Gamma^+(X)$ which contains all nodes connected by outgoing edges of node n.

Now, for a set $X \subseteq E$: A map $f : E \to \mathbb{R}^+$ is called *flow* of the network if

$$\forall e \in E : \quad 0 \leq f(e) \leq c(e) \tag{3.1}$$

$$\forall n \in V \setminus \{s, t\} : \quad 0 \leq K(n) \leq C(n) \tag{3.2}$$

$$\forall n \in V \setminus \{s, t\} : \quad \lim_{N_T \to \infty} \frac{1}{N_T} \sum_{T=0}^{N_T} I(n, T) = \lim_{N_T \to \infty} \frac{1}{N_T} \sum_{T=0}^{N_T} O(n, T) \tag{3.3}$$

This is different from traditional graph theory (section 2.1.3). The flow f is only defined if all three conditions are valid. The first condition is the classic assumption that a flow via an edge can not exceed the capacity of the edge. The second condition is a property of the nodes in industrial networks expressing upper bounds of the inventory level. The third condition says that the inflow $I(n)$ has to equal the outflow $O(n)$ of a node n in average. The average condition is important for understanding that the nodes' inflow and the nodes' outflow could temporarily differ. As long as the inventory level $K(n)$ of node n has not reached its upper bound $C(n)$, the inflow of units could exceed the amount of out-flowing units. Or, as long as the inventory level $K(n)$ is larger than zero, the outflow could exceed the inflow.

The inventory level $K(n,T)$ of node n at time $T \geq 0$ changes according to the net effect of inflows and outflows at node n:

$$K(n, T + \Delta t) = K(n,T) + \Delta t \cdot [I(n,T) - O(n,T)] \tag{3.4}$$

This continuity equation Eq. (3.4) was first introduced by Forrester [For68][1] and is a common way to model inventory levels in Inventory Theory [JB01].

The inflow rate $I(n,T)$ of node n at time T is given by the sum of all incoming flow rates $f(y)$ from all parent nodes. These flow rates are restricted by the transportation capacities of their edges. Therefore, the inflow rate $I(n,T)$ has an upper bound which is the sum of all edge-capacities of the nodes' incoming edges. The same holds for the outflow rate $O(n,T)$ at time T, which is also bounded by the sum of all edge-capacities of their outgoing edges:

$$I(n,T) = \sum_{y \in \Gamma^-(n)} \min\{f((y,n),T), c((y,n))\} \tag{3.5}$$

$$O(n,T) = \sum_{y \in \Gamma^+(n)} \min\{f((n,y),T), c((n,y))\} \tag{3.6}$$

3.2 Controller Architecture

In section 3.1 we have developed a concept to describe the topology and the functionality of production processes with all its important properties in a computer model. The concept bases on a basic node which is sufficient to construct modularly all kinds of real industrial components like shelves and machines. The task of such a component is to distribute units to its neighbours. If there is more than one neighbour which units can be delivered, component has to be instructed to which of them it has to send units. For this, an agent-based approach is chosen. The agent is responsible for the unit inflow and for the unit outflow of its component. All its decisions are based on the internal

[1] Forrester has called the continuity equation (3.4) *level-equation*, because if the simulations stop every inflow or outflow immediately, the inventory level $K(n)$ is still measurable. Values of equations which are not measurable during simulation pause are called *rate* equations.

properties of its component and its own distribution policy. In the next chapter we will explore agent policies which optimise the complete commodity (unit) throughput of the network. In this chapter, the agent control which is independent from the overall goal of optimisation is introduced itself.

The decentralised agent-based controller has three main abilities. The first ability is local communication with its surroundings. The agent collects information about its neighbours and the environment's topology (section 3.2.2). The second ability is reasoning about the collected information. Using the information, the agent learns to optimise its policy for commodity distribution (section 3.2.3). And the third ability is the ability to distribute reliable commodities among neighbours (section 3.2.4). Three abilities together describe the functionality which is needed for a decentralised controller of production processes. This agent model is based on an abstract model introduced in the next section.

3.2.1 Agent Model

In the early 70's, Aoki [Aok71] already used the term *agent* as a synonym for a linear controller in a decentralised control system. In modern AI research, agents are described as autonomous intelligent entities. Even though the term *agent* is not clearly defined, a possible definition can be found in literature [JSW98]: An *agent* is a computer system situated in some environment and is capable of flexible autonomous action in order to meet its design objectives. Following the definition of the authors there are three key-properties for agents: *situatedness, autonomy* and *flexibility*. Situatedness means that the agent receives sensory input from its environment and that it can perform actions which change the environment in some way. Autonomy means that the agent is able to fulfil its objectives.

Many process control systems are already situated in autonomous computer systems which monitor a real world environment and perform actions to modify it when conditions change. According to Jennings [JSW98], situated and autonomous computer systems are considered agents if they are capable of flexible actions. Agents seem to be flexible if they are responsive, pro-active and social, which all enables them to interact with other agents. Responsive means that agents perceive their environment and respond to changes in a time.

Pro-active is not just a response to the environment, but a goal-directed behaviour: agents not just react, they take the initiative, when appropriate. Flexibility means that agents adapt their policy, using for example learning methods.

There are various designs for agents. Good overviews about agent architectures are found in Wooldridge [Woo99] and Jennings [JSW98]. The architectures of agents differ depending on the aspect from which a problem is investigated. An example of an agent architecture for logistic processes is found in [Rev01] or [VSR98]. For our purpose, the main aspects are what information is required an how it is used. Agents are controllers which use input information to choose suitable actions. Internal states could be taken into consideration which would result in adaptive and learned acting. This aspect does not demand a complex and extensive agent architecture. A small abstract design is

sufficient. Wooldridge [Woo99] introduced an abstract model of an agent architecture which is sufficient for our purpose. This abstract architecture is shown in Fig. (3.3). It consists of only three functions, a *see* function, a *next* function and an *action* function. Following the notation of Wooldridge we can easily formalise these functions:

1. Agents are surrounded by their environment which could consist of other agents or objects that influence the agents. In our industrial context, the agent is surrounded by other controllers that offer and demand goods. This environment can be characterised as a set $S = \{s_1, s_2, \ldots\}$ of environment states. Agents perceive the environment with their senses. The function *see* captures the agent's ability to observe its environment. The output of the function *see* is a percept of the given input. It could happen that two different environment states s_1 and s_2 are mapped to the same percept. Also noise could distort the impression of the environment. The agent has only an imperfect knowledge about its surrounding. *See* is a function which tries to map the environment to a non-empty set P of percepts:

   $$see : S \rightarrow P$$

   Such a map can be represented by simple filters, clustering networks [Koh89] or even pattern recognition neural networks [DR02, Haf97].

2. The perceived environment influences the state of the agent. A state is an internal value which reflects the experience and the beliefs of the agent as well as internal dynamic constraints. During its lifetime, an agent learns about its action and the response of its environment. A function *next* maps an internal state J and a percept P to their internal state J:

   $$next : J \times P \rightarrow J$$

   Examples are Kalman filters, recurrent neural networks or reinforcement learning [WS99, MMT00]. They map states and percepts on to updated states.

3. Based on this updated internal state J, an agent decides its action. Again, its action influences its environment again and the environment influences the agent. The cycle continues. The action-selection function is defined as

   $$action : J \rightarrow A$$

   where A is a set of actions. For example in Q-Learning [SB98], a reinforcement learning approach, the action is chosen out of an action-state table.

In that respect, an agent consists of three functions namely a see-function, a next-function and an action-function. The see function collects information, the next function uses this information and the action function acts according to the used information. In our respect, agents are just information processors but still autonomous, flexible and situated in their environment. The environment and the agent's objectives constraint

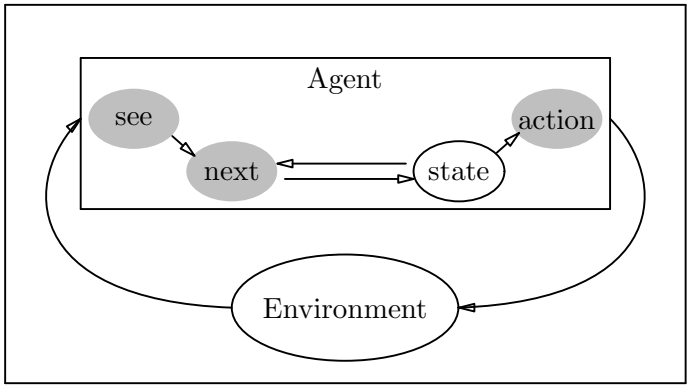


Figure 3.3: Abstract architecture for intelligent agents (based on figure from Wooldridge in [Woo99]).

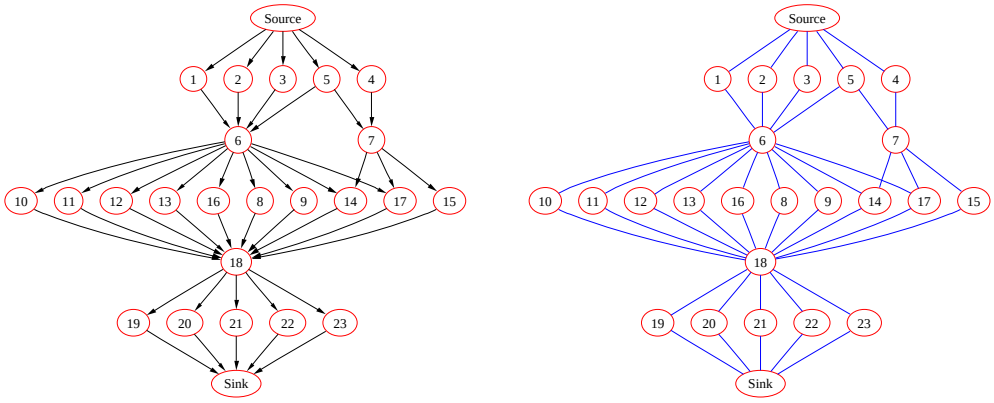
the agent's behaviour. An agent which is responsible for an industrial component in a production process has to be aware of the component's properties.

The task of an agent is to distribute a number of units to its neighbours. The agent's action is constrained by the topology of its environment which determines the possible material flow and the agent itself is constrained by the internal properties of its component. In the following we will determine the three functions *see, next* and *action* for the purpose of industrial production processes. Together they define the design of the decentralised controller in our production process environment.

3.2.2 Local Communication

The *see* function of an agent captures the communication with its neighbours and its local environment. The material flow is defined by the production process topology. The network defines the possible paths from one node to another. Commodities can only be exchanged between neighbours that are directly connected. Additionally, commodities can only be delivered to children nodes. As agents are responsible for nodes, they have to communicate with the nodes' neighbours, respectively their agents, to arrange a delivery. An agent has to negotiate its commodity inflow and its commodity outflow.

Local communication is direct communication between neighbours only. As a result of that we distinguish between two kinds of networks, see also Fig. (3.4): A material flow network and a communication network. Both are similar but differ in an important detail. In contrast to the material flow network, the communication network is undirected. Communication occurs in both directions: from parents to children and from children to parents. For some optimisation approaches it is necessary to obtain information from agents other than their neighbours. For this, a message can be sent if the paths through the network are known. The message will be sent by hopping from neighbour to neighbour until it reaches the addressee. Such a behaviour is used by ant-like approaches.



(a) Material Flow Network (b) Communication Network

Figure 3.4: The topology of an industrial network. On the left side the material flow network (a) is shown which allows commodities only been send in direction of edges. On the right side the communication network (b) is shown. Communication occurs undirected between neighbours.

An information exchange can not be done once but it repeated regularly. It could happen that meanwhile neighbours change or that the transportation capacities of their connections become weaker or stronger. The agent has to collect all information which is necessary to send a unit successfully. The amount of this information depends on the optimisation approach. Some agents need only a very limited knowledge about the neighbours' states or network properties; others use this knowledge heavily. For example Link State approaches (section 4.4.3) exchange the knowledge about all nodes and network states. This information is flooded throughout the network.

In our model are only *two* kinds of message types that are exchanged between neighbours: *Assertions* and *Offers*. An agent can send an assertion to one of its neighbours requesting information about the current state of this neighbour and perhaps about some additional quantities defined by the optimisation approaches. The neighbour has to send this information immediately. The other kind of message is an offer. An agent offers a commodity to one of its children. The child agent accepts or rejects this unit because of internal node properties.

The agent has to *screen* its surrounding topology regularly. This information contains the number of neighbours, the identities of neighbours and the current capacity of the edges.

Assertion: Agent n demands knowledge about state and additional information from agent $y \in \Gamma^-(n)$. The agent y sends this information to n.

Offer: Agent n offers $y \in \Gamma^-(n)$ a unit u. Agent y has to reject if $C(y) = K(y,T)$, which means that there is no place for an additional unit. If $C(y) > K(y,T)$ then the agent y accepts the unit.

Screen: Agent n screens the local topology for its transportation capacities. The topol-

ogy returns the upper bounds for units which can be sent to each neighbour of agent n.

The information which is stored in a percept P is gained from all kinds of messages. This percept is the agent's view of its local surrounding.

3.2.3 Policy Learning

Assume an agent has some units that are ready for delivering. Which of the following nodes should get how many units? If there is only one child node, it is easy for the agent to make a decision. But the question is, which of the neighbours should get the unit if there is more than one neighbour. The answer depends on the agent's objective. If the objective is to minimise the inventory levels of the components, then the agent should send its units to nodes with low inventory levels. But a different objective could result in the selection of different nodes. Additionally, in dynamic environments the best suitable following node changes over time. The agent needs an adaptive distribution policy in order to reach its objective.

A set of internal states J represents the beliefs, desires and experience of an agent. They are needed to distribute successfully the units among children nodes. The internal state is updated by using the captured percepts P from the environment and the internal states J: $P \times J \rightarrow J$. Percepts are the internal representation of the local communication. The information is exchanged between the agents and between agents and the network topology. The agent n uses its internal states J to choose a child for offering a commodity. Different optimisation algorithms for choosing a child are possible. Best suited algorithms are those which are able to adapt to changing conditions. For this reason, learning algorithms are promising approaches. A good overview about learning in multiagent systems is found in [WS99]. An online-optimisation algorithm maps the internal states $J(n)$ of agent n and its *action-value* function $D(n)$ on an updated *action-value* function $D(n)$:

$$\text{learning:} \quad J(n) \times D(n) \rightarrow D(n) \tag{3.7}$$

The action-value function $D(n)$ of an agent n is a weight vector with $D_j(n) \geq 0$ and $j \in \Gamma^-(n)$. The name *action-value* indicates the similarity with the well known action-value function of reinforcement learning (RL) [SB98]. $D(n)$ is similar to the RL Q-values that are assigned to states-action pairs. In this context, the action-value $D(n)$ is a table storing weights for each of the agent's neighbours. A *policy* is derived from D to select a child. A commodity is more likely to be delivered to a child with hight weight than to a child with a low weight. The probability to get a unit from agent n is equal for all neighbours of n if their weights $D_j(n)$ are equal. To avoid that no unit can be delivered because of the agent's policy $D(n)$, we demand that at least one weight $D_j(n)$ of all children $j \in \Gamma^-(n)$ has to be larger than zero:

$$\text{For all agents } n: \quad \sum_{j \in \Gamma^-(n)} D_j(n) > 0 \tag{3.8}$$

The derived policy is the intention of the agent. An unit can only be sent if the receiver accepts it. An agent may reject a unit if its storage capacity is exhausted or if a unit's capacity of transportation to its neighbour is exhausted. In chapter 4 we will deduce bounds that can be used here as a decision support. With the help of those bounds, the agent can predict the ability of its children nodes to accept units in certain ranges. In chapter 4 we will use these theoretical bounds together with online-optimising approaches to maximise the throughput of industrial networks.

3.2.4 Commodity Distribution

The task of the *action* function of an agent n is to distribute the commodity units to its children nodes. The functionality of commodity distribution does not know by itself to which of the following nodes it shall deliver. This knowledge is derived from the agent's action-value D which weights all children nodes. The agent offers each unit which is to be exported to one of its neighbours based on the policy derived from D. The offer can be accepted or rejected by the selected neighbour. If the offer is accepted and the topology still allows sending units, the agent transfers the unit to its neighbour. A generic commodity distribution function of agent n is shown by the following pseudo code:

```
1   function commodityDistribution(n, D(n), U(n), T)
2   foreach y ∈ Γ⁻(n) do f((n,y),T) = 0 end
3   foreach unit u of U(n) do
4       select children j ∈ Γ⁻(n) using policy derived from D(n).
5       offer j unit u and get answer B1 = C(j) − K(j,T).
6       screen rest-capacity B2 = c((n,j)) − f((n,j),T) of edge (n,j).
7       if (B1 > 0) & (B2 > 0)
8           send unit u to neighbour j.
9           f((n,j),T) = f((n,j),T) + 1/Δt.
10      end
11  end
```

Inputs for this commodity distribution function are the agent's policy $D(n)$ and the set of units $U(n)$ that are to be distributed during the time step $T \cdot \Delta t$. For each unit $u \in U(n)$, the agent n selects one of its children agents. Then n offers the units and screens to its surrounded topology. The answers are given by $\mathcal{B}_1$ and $\mathcal{B}_2$ where $\mathcal{B}_1$ informs n about the amount of units that can be accepted by the selected neighbour j and $\mathcal{B}_2$ informs n about the available transportation capacity towards j which is defined by the surrounding topology. If the offer is accepted ($\mathcal{B}_1$ and $\mathcal{B}_2$ are greater zero) the agent sends the unit to its selected neighbour. This procedure is looped over each of its exportable units of $U(n)$.

Until now, we have not specified the method for deriving the policy from D. The commodity distribution also depends on the controller objective and therefore we put off the specification until chapter 4 where the controller objective is introduced. In that

chapter, we will also introduce online-optimiser for updating D, which has been omitted, too.

Local Communication, Policy Learning and Commodity Distribution are the three basic modules for decentralised agent-based controllers. In chapter 4, after the introduction of the controller's objective, we will specify its architecture. Its general layout consists of the functions which have been presented in this chapter. The agent-based controller learns from its local environment (policy learning) by collecting information (local communication) and distributing units (commodity distribution). The result of learning is an adapted policy derived from D. This policy tells all agents how to distribute commodities among neighbours and to fulfil the overall objective of the network.

Chapter 4

Maximum Throughput Policies

In chapter 3 we learnt to construct complicated production processes with a simple basic module and we learnt to control them with a decentralised approach. But the controller architecture has not been clearly defined yet because its design depends also on the primary objective of the controller. The objective of the controllers in this thesis is to increase the throughput of the complete production process and, if possible, to maximise it. Responsible for the distribution of commodities is the agent's policy which is derived from D, see chapter 3. D has to be adapted locally for each controller with online-optimisation approaches. For the adaptation of D, the agents have only access to local information that they get from their neighbours, from their local environment (chapter 3.2) and from some known bounds that help the agent in distributing commodities. Those bounds are introduced in the following section 4.1.1. If the total throughput of the industrial network reaches its theoretical maximum, we call the set of all local polices a *maximum throughput policy* (MTP). And we will see that different local policies reach the same theoretical bound. In that respect, a MTP is one out of the set of all MTPs that are possible for a certain network.

4.1 Theoretical Results

This section focuses on the maximum throughput objective. How many commodities can an agent deliver towards its children? The answer depends on the properties of its children and the properties of the childrens children and so on. In the first part of this section we deduce such bounds for a single agent and for agents in a connected network. In the second part of this section we use the results to calculate the average maximum throughput of the complete production network.

4.1.1 Bounds in maximum local throughput

The inventory level $K(n,T)$ of an agent's component n at time T has already been introduced in section 3.1.3:

$$K(n,T+\Delta t) = K(n,T) + \Delta t \cdot [I(n,T) - O(n,T)]$$

where $I(n,T)$ and $O(n,T)$ are the current inflow and outflow rates. We already know that the inflow and the outflow are restricted by the amount of units which can be imported and exported via all edges that are connected with the component (see also Eq. (3.5) and Eq. (3.6)). But the upper bounds of a component's inflow or outflow are not only restricted by the capacity of its edges. There is also a linear relation between flow, residence time and inventory level. This relation is called *Little's Law* in Inventory Theory [JB01]:

$$InventoryLevel = (FlowRate) \cdot (ResidenceTime)$$

The flow rate increases if the inventory level increases and decreases with the increase of residence time. This relation is only valid if the inventory capacity is unlimited. The maximum inventory level and the residence time of a unit at any node are already given by the system properties. We have to find an upper bound for the inflow rate and the outflow rate based on the buffer size and the residence time of a node.

Let us first have a look at the inflow rate. The inflow rate for a node n at time T is given by the number of units that are to import during the next time interval Δt. The maximal inflow rate is then given by the number of free unused slots in the inventory buffer at time T and the number of units that are to export during the next time interval Δt, if we are able to import and export at the same time:

$$I(n,T) \leq [C(n) - K(n,T)]/\Delta t + O(n,T) \tag{4.1}$$

A unit can not be delivered immediately but after the residence time W. Machines for example only export units that are processed. $K_{t \geq W}(n,T)$ is the number of units which have already stayed as long or longer than the residence time W at the buffer of node n at time T:

$$O(n,T) \leq K_{t \geq W}(n,T)/\Delta t \tag{4.2}$$

These upper bounds still fit Eq. (3.2) as we can easily proof if we insert the upper bounds in Eq. (3.4):

$$K(n,T+\Delta t) \leq K(n,T) + \Delta t \cdot ([C(n) - K(n,T)]/\Delta t + O(n,T) - O(n,T)) = C(n)$$

Let $N(V,E,c,C,W,s,t)$ be a network as defined in definition 2 with a node set V and an edge set E. The *maximum flow problem* in an industrial network is defined similarly as in traditional graph theory (section 3.1.1):

Definition 3 (Maximum flow problem) The *maximum flow problem* in an industrial network is to send the maximum flow from the source node s to a sink node t, while preserving the flow bound constraints Eq. (3.1), Eq. (3.2) and Eq. (3.3) on all edges of E and nodes of $V \backslash \{s,t\}$.

The maximum flow is the maximal number of units the network is able to transport from source node s to sink node t. To reach such a theoretical value in real systems all sources have to produce as many units as their children nodes are able to import and

all nodes have to import and export as many units as possible. Then, a system is called to be under *maximum-load* or *heavy-load*.

We already know the bounds of possible inflow and outflow for a single node in our system. These bounds are restricted by the inventory level $K(n)$ and the residence time $W(n)$ of a unit at a node n. Additionally, the inflow and outflow are restricted by the edge-capacities of the node. In Eq. (4.1) we have seen that the inflow of a node at time T in our system is equal or lower than the sum of the unused free storage places in node buffer plus the amount of units we export during the next time interval. A single node can only export units that have stayed their minimum residence time $W(n)$ inside the buffer. Thus, the upper bound for the number of exportable units is given by Eq. (4.2). If our node always exports as much as possible then Eq. (4.2) can be simplified by the amount of units our node has imported $W(n)$ time steps before. A maximised flow through a node n is then given by its maximal inflow $I^*(n,T)$ and its maximal outflow $O^*(n,T)$. If the edge-capacities of edges linking our node are large enough, we get:

$$O^*(n,T) \stackrel{!}{=} I(n,T-W(n)) \tag{4.3}$$

$$I^*(n,T) \stackrel{!}{=} [C(n)-K(n,T)]/\Delta t + O^*(n,T) \tag{4.4}$$

An industrial node, which imports and exports as much as possible, has an oscillating outflow. The period of the oscillation is given by Eq. (4.3) and is the residence time $W(n)$ of a node. Two values are interesting: The first is the maximal outflow $O^*_{\max}(n)$ of a node n in time T, which is bounded by $\langle O^*(n)\rangle \leq O^*_{\max}(n) \leq C(n)$. Here, $\langle O^*(n)\rangle$ is the average outflow and the second value of interest. $\langle O^*(n)\rangle$ equals the number of units that are expected to be exported from node n. And this is easily deduced by Little's Law:

$$\langle O^*(n)\rangle = \lim_{N\to\infty} \frac{1}{N}\sum_{t=0}^{N} O^*(n,t) = \frac{C(n)}{W(n)} \tag{4.5}$$

The example of the node in Fig.(4.1) shows the storage amount and the outflow of a node with working-time $W = 5$, a maximal buffer-size $C(n) = 20$ units and an initial amount of $C_0 = 12$ units, that has just been imported. The node is exporting and importing as much as possible. The storage room is completely filled and the outflow is oscillating with a period of $W = 5$ time-steps. In this example, the maximum outflow of our node is about 12 units, which is reached only every 5th time step. The rest of the time we get less units.

An agent, which is responsible for a node, tries to import and export as much as possible. Doing so, the agent follows the rules given by Eq. (4.4) and Eq. (4.3). If its outflow and inflow are not restricted by other nodes in a network, the agent is gets the average maximal throughput which is given with Eq. (4.5). In a connected network, the agents' action-range is not only restricted by the topology and the agents' properties but also by the throughput capacities of the connected neighbours. The nodes' outflow is limited by the amount of units its children nodes accept. In this way, our node may

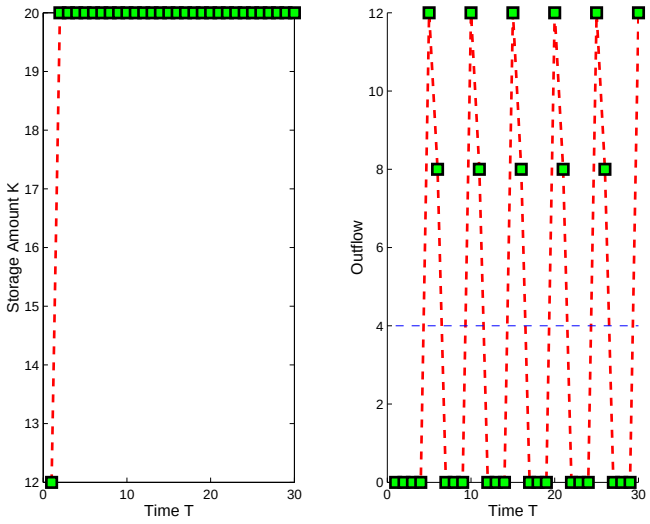


Figure 4.1: A node which imports as much as possible and also exports as much as possible has an oscillating outflow and a maximally used buffer. Here, our node has a buffer size C of maximal 20 units, an initial amount of 12 units and a residence time W of 5 time steps. The outflow oscillates with a period of $W = 5$. The average outflow (blue line) matches perfectly the estimate $C/W = 4$ units per time step.

not be able to export as much as it would like. In a connected network Eq. (4.2) has to be modified:

$$O(n,T) \leq \min\{K_{t\geq W}(n,T)/\Delta t, \sum_{y\in\Gamma^+} I((n,y),T)\} \tag{4.6}$$

Here, $I((n,y),T)$ is the inflow that node $y \in \Gamma^+(n)$ accepts from node n. The maximal possible outflow at time T is limited to the lower value of the number of the node's exportable units and the sum of units which might be accepted by its children nodes.

The following table summarises the possible ranges for the inflow and outflow decisions of an agent that is responsible for a node in a network:

Import/Export range for a network node	
Inflow:	$0 \leq I(n) \leq \min$ { Eq. (4.1), Eq. (3.5) }
Outflow:	$0 \leq O(n) \leq \min$ { Eq. (4.6), Eq. (3.6) }

4.1.2 Maximum global throughput

The assumption of nodes working under heavy-load allows us to calculate the maximum flow from source to sink in the network. With the use of the equations previously deduced and the assumption of heavy-load working nodes we can determine the inflow of all nodes n at each time T:

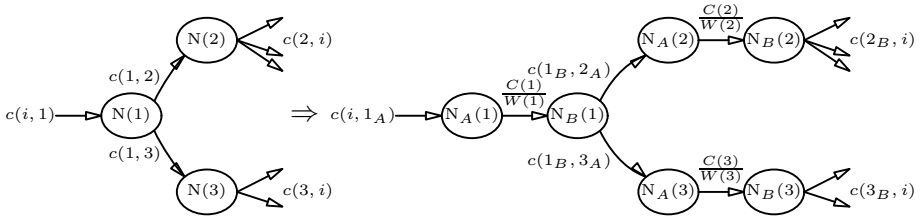


Figure 4.2: For the calculation of an average maximum flow in an industrial network the graph has to be extended. Every node is split into two new nodes. For example node $N(1)$ is split up into a node $N_A(1)$ which inherits all incoming links and a node $N_B(1)$ which inherits all outgoing links. Both are connected by a direct link from $N_A(1)$ to $N_B(1)$ and get an edge-capacity based on Eq. (4.5) of $\frac{C(N(1))}{W(N(1))}$. The average maximum flow through the network can now be calculated by standard graph tools like the Ford-Fulkerson Algorithm.

$$I(n,T) = [C(n) - K(n,T)]/\Delta t + O(n,T) \tag{4.7}$$

$$O(n,T) = \min\{\frac{K_{t\geq W}(n,T)}{\Delta t}, \sum_{j\in\Gamma^-} \min\{I(j,T), c(n,j)\}\} \tag{4.8}$$

$$K(n,T) = K(n,T-1) + \Delta t \cdot [I(n,T) - O(n,T)] \tag{4.9}$$

Already for small networks this is a difficult task as can be seen easily in the complexity of the equations used. It is impossible to find an analytic solution for larger, more complicated networks. Another way to solve this problem is to use tools delivered by Graph Theory introduced in Section 3.1.1.

First, we need a suitable map of our network model in addition to the traditional network that has been derived from Definition 1. The traditional network was presented as a set of nodes and a set of edges linking the nodes. Edges have finite capacities which limit the total maximal throughput from a designated source to a designated sink. Inflow equals outflow and because of this the average maximal throughput of a node is given by Eq. (4.5), which says that the average maximum outflow of our node is given by the relation of the node's capacity and the residence-time it has to spend for incoming units. The capacity of the nodes' throughput could be represented by splitting up each node N into two new nodes, see Fig. (4.2). The nodes arising from node N are N_A and N_B. Node N_A has all incoming links of node N and node N_B has all outgoing links. The node capacity is represented by a link between N_A and N_B assigned with the real number $C(N)/W(N)$ of Eq. (4.5). This is done for all nodes in our network. Extending our graph doubles the cardinality of the nodes set but it does *not* change the topology as this is given by the incoming and outgoing links. But in this new representation the nodes contain no limiting capacities. Applied on this new network, classic Graph Theory methods return the maximum flow from a given source to a given sink. This is the average maximum outflow which is expected for our network model.

An example solution is shown in Fig. (4.3). Already small networks produce highly

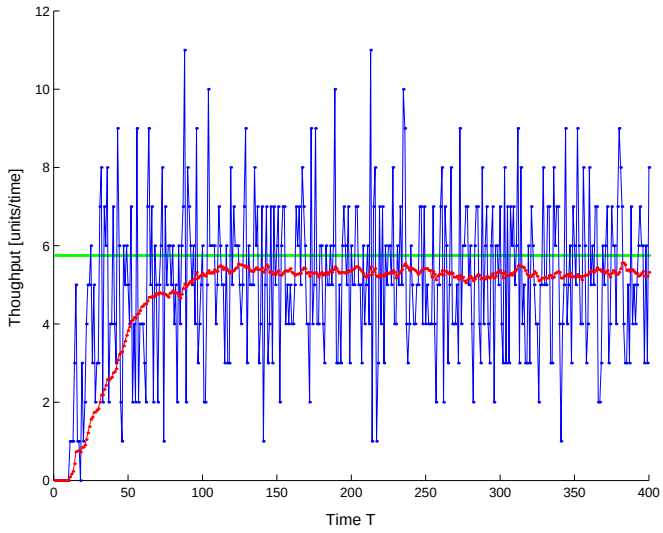


Figure 4.3: A small industrial network with 21 nodes can already produce a fluctuating throughput. In this example, the agent that is responsible for a node, exports its already produced units to its children equally. The agent imports and exports as much as possible. The horizontal line represents the theoretical average throughput and the fluctuating line presents the current throughput. The thick line represents the average of the current throughput and demonstrates that without an optimised strategy the theoretical possible average outflow is not reached.

fluctuating outflow. In this figure the throughput of the small test network of chapter 5.1 is shown. Each node has a limited storage capacity and a finite residence time to hold units. The finite buffers and residence times lead to a highly non-linear relationship between the members of our network. The nodes in this example distribute units among their children equally while preserving the bounds which have been deduced in the previous section. A second straight line shows the calculated average maximal throughput of the system after splitting up the network nodes and using traditional maximum flow algorithms, or in this case, the algorithm which has been introduced in section 3.1.1. On average, our system can not get more throughput as this bound but it is possible to have larger peaks instead. At a single moment, it is possible that all parent-nodes of our sink are exporting simultaneously more than half of their storage-capacity. At this moment, we get a larger sink-inflow as our expected average maximum flow. But then, the nodes have a lower minimum outflow depending on their residence time. This leads to an average equal or smaller than the average expected. In this example, the agents weight equally with their children nodes. No child node is favoured and because of this the agent tries to send units to its neighbours equally. In this case this policy does not lead to an optimised throughput. In the next section we present decentralised control concepts which lead to an average throughput that converges to the theoretical maximum throughput.

4.2 Reducing Outflow Oscillation

A node which imports and exports like produces an oscillating outflow as shown in Fig. (4.1). Such an oscillatory behaviour is also seen in supply chains investigated by Parunak et. al. [VSR99]. Their multi-agent approach is called DASHh (Dynamic Analysis of Supply Chains) [VSR98] and is used for simulation and for understanding supply chain's behaviour. The child nodes have to be prepared for very high inflow during the oscillation peak and for low inflow between the peaks. A more smooth flow would be desirable. A simple policy for reducing the amplitude of an oscillation is to limit inflow or outflow to the calculated average throughput $\langle O^*(n) \rangle$ of a node n but this is only correct if Eq. (4.5) results in an integer. If the expected average throughput is not an integer, we can still minimise the oscillation with a probabilistic approach. In this case, our average throughput reaches the theoretical value asymptotically, which is trivial to show. Here we adapt the inflow but it would also work for the outflow:

$$I^*(n,T) \begin{cases} = \min\{\text{Eq. (4.4)}, \frac{C(n)}{W(n)}\} & : \frac{C(n)}{W(n)} \in \mathbb{N} \\ \approx \min\{\text{Eq. (4.4)}, p \cdot \text{ceil}(\frac{C(n)}{W(n)}) + (1-p) \cdot \text{floor}(\frac{C(n)}{W(n)})\} & : \frac{C(n)}{W(n)} \notin \mathbb{N} \end{cases} \tag{4.10}$$

The second part of Eq. (4.10) is the probabilistic control mechanism to reduce oscillation with the probability function p which is 1 with the probability $\frac{C(n)}{W(n)} - \text{floor}(\frac{C(n)}{W(n)})$ else 0. The second part does not avoid the oscillation but it minimises the amplitude.

An example of such a non-oscillation control mechanism is shown in Fig. (4.4). It is the same example as in Fig. (4.1) and it shows that such a control mechanism minimises the outflow to the theoretical solution. This was only possible though, because the ratio of storage capacity and the residence time were integers. The mechanism needs some time-steps to get control over the outflow but then it exports exactly the average that is theoretical possible. maximal outflow.

This approach does not prevent oscillations in networks. It is able to reduce the amplitudes of the oscillations caused by the nodes themselves. But it does not reduce oscillations caused by the topology of the network. It reduces efficiently the throughput oscillation in industrial networks. Fig. (4.5) demonstrates this with an example industrial network with 48 nodes. Agents are responsible for the distribution of units to nodes' children. The agent model and the agents' (optimal) local policy is discussed in the next chapter. Here, only the dynamics is of interest: On both parts of the figure, the total network throughput, measured in units per time, is plotted. The curves show the averages over 50 runs. On the upper part, the agents follow their local policy without a local mechanism to prevent node outflow oscillations. The error-bars show that the throughput is fluctuating around the theoretical average throughput bound (see next section). On the lower part of the figure, the same network - but using the local policy of Eq. (4.10)- produces a more smooth curve of the network throughput. This time, the error-bars are smaller. The oscillation is efficiently reduced by the local control mechanism.

The control mechanism to reduce fluctuations works in this version independently from the control mechanisms for the distribution of units that are introduced in the next

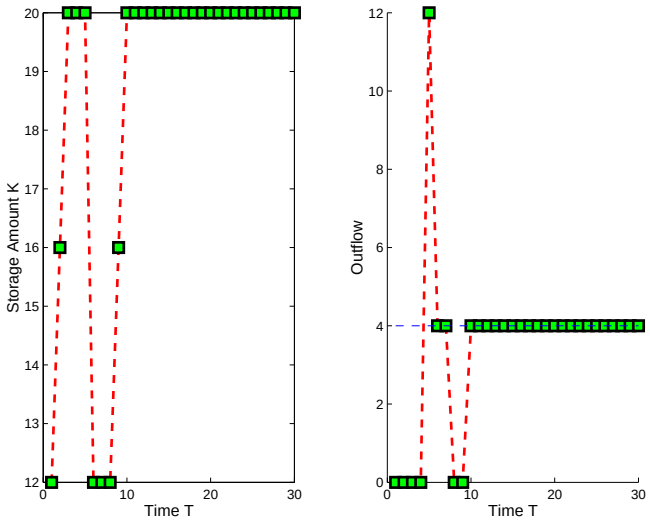


Figure 4.4: This demonstrates the same example as in Fig. (4.1). This time a non-oscillation control mechanism as introduced in Eq. (4.10) is used to minimise the oscillation amplitude. Here, we can avoid oscillation completely as $\frac{C(n)}{W(n)} = 4 \in \mathbb{N}$.

sections.

4.3 Controller Types

In chapter 3 we introduced the model of a decentralised controller but we have not specified its design yet. Each component or node in the production network has its own agent. The controller agents in our model are *semi-active*: They can control the outflow of units but not the inflow of units. If a unit is offered and the current storage level K allows import, then the agent can not deny this unit. On the other hand, the agent controls its unit outflow almost completely. Following the deduced bounds of section 4.1.1 the possible outflow $O(n,T)$ of node n at time T is:

$$0 \leq O(n,T) \leq \min\{K_{t \geq W(n)}(n,T), \sum_{y \in \Gamma^-(n)} \min\{I((n,y),T), c((n,y))\}\} \tag{4.11}$$

where $I((n,y),T) = [C(y) - K(y,T)]/\Delta t + O(y,T)$ with $y \in \Gamma^-(n)$. $O(n,T)$ is the range in which the agent can export units. Eq. (4.11) also tells how many units each neighbour accepts (but only if its is known that its neighbours' neighbours also import or not).

During the time interval Δt, all agents are simultaneously trying to export units considering the bound in Eq. (4.11). The exportation of units from one node to another is a parallel process. Because information exchange occurs much faster in production processes than at commodity exchange, the transportation of units can be negotiated during one time interval Δt. We assume that the message exchanges are synchronised

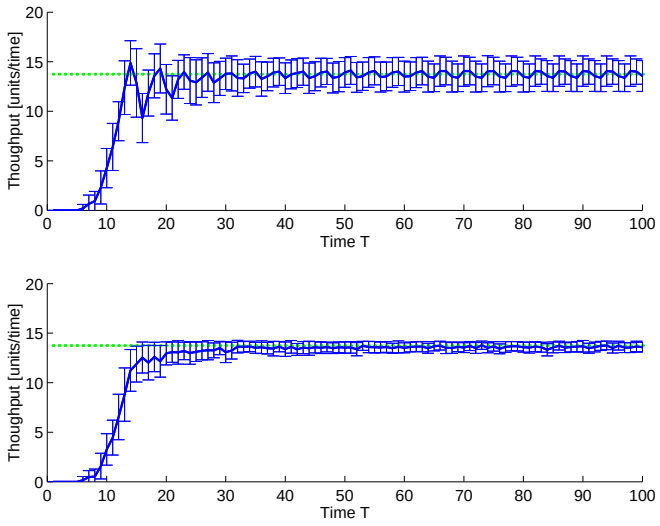


Figure 4.5: Demonstration of the local policy to reduce network oscillation. Shown is the total network throughput [units/time] as an average over 50 runs of an industrial random network with 48 nodes. The upper part is the result of the simulations without a local agent-policy for reducing oscillation and the lower part with such an agent-policy. The result shows that the approach efficiently reduces the oscillations, but is does not prevent them.

and that the requests are answered following the order of their arrival. This is a first-in first-out (FIFO) policy.

At the beginning of the current time step $T \cdot \Delta t$, an agent n can export up to $K_{t \geq W(n)}(n, T)$ units. This is the number of units which have stayed at least the residence time $W(n)$ at node n. Although we investigate networks with an unique type of commodity we can distinguish units by their age. Instead of selecting a unit randomly out of the set of all exportable units, the agent tries to offer the oldest units. Like the requests in the message exchange, the commodities are handled by a FIFO policy, too.

In the following, we present two kinds of controllers. Both controllers are based on the design which is sketched in chapter 3. The first controller is small and efficient in almost every network but it can not guarantee that the throughput is optimised in all production networks. The second controller can guarantee this but it is looses its simple design.

4.3.1 ST Controller

A policy for distributing units can be derived from D in several different ways. One way would be to select always the neighbour with the largest weight in the agent's action-value table D. But it has been shown in many fields that such an approach is less successful than an approach where the neighbour is chosen out of a probabilistic distributions [SB98]. A probabilistic approach transforms the weights of D into probabilities.

```
1   function STController(n, D(n), U(n), J(n), T)
2   foreach y ∈ Γ⁻(n) do f((n,y),T) = 0 end
3   assert information P(n) from all children y ∈ Γ⁻(n)
4   foreach unit u of U(n) do
5       choose children j ∈ Γ⁻(n) by policy derived from D(n) with Eq. (4.12).
6       offer j unit u and get answer B1 = C(j) − K(j,T).
7       check rest-capacity B2 = c((n,j)) − f((n,j),T) of edge (n,j).
8       if (B1 > 0) & (B2 > 0)
9           send unit u to neighbour j.
10          f((n,j),T) = f((n,j),T) + 1/Δt.
11      end
12  learn new policy D(n) from the set {B1, B2, J(n), P(n)}
13  end
```

Figure 4.6: A decentralised controller which is designed to distribute commodities among neighbours.

Then, the selection of a receiver is a random process. Such an approach is called an *exploration strategy*[1]. The idea is that an agent only learns about other actions and the change of its environment if the agent also chooses another action sometimes. A probabilistic distribution ensures that not always the neighbour is chosen which is estimated to be best. Better solutions could be found by choosing different neighbours because agents' estimations are often wrong in dynamic environments. The probability $p_u(n,j)$ of sending a unit u out of a set of distributable units U to a neighbour $j \in \Gamma^-(n)$ might be given by:

$$\forall j \in \Gamma^-(n): \quad p_u(n,j) = \frac{(D_j(n))^\beta}{\sum_{j\in\Gamma^-(n)}(D_j(n))^\beta} \tag{4.12}$$

where β is a non-linearity parameter which stresses the actual policy and $D_j(n)$ is the entry of children $j \in \Gamma^-(n)$ of n in the action-value table $D(n)$. The policy for choosing a neighbour for the unit u is derived by drawing a neighbour out of the distribution of Eq. (4.12). A controller should try to find a client for each of its units. Each time-step the agent starts with the oldest unit and offers it to a child node. The child nodes are drawn like described above. In this design, each of the units that have been declared for exportation is offered only once to a neighbour. So the agent finishes its tasks when it has offered all units exactly once. Thus, this controller design is responsible for the average selection of neighbours. And a learning approach triggers the average selection of clients.

The design of the controller is shown in Fig. (4.6): First, the controller asserts an update of general information from all its children. Then, the agent tries to export all units U which have been chosen for exportation. The amount of units of U is given by

[1]Those exploration strategies are very common in reinforcement learning approaches. In Q-Learning for example, the most used exploration strategy is a Boltzmann distribution triggering the Q-estimates by a temperature. For more see R. Sutton and A. Barto [SB98].

$0 \leq |U(n)| \leq K_{t \geq W(n)}(n,T)$. For each unit $u \in U$ a child node is chosen by using a policy derived from $D(n)$. Let us assume that neighbour $j \in \Gamma^{-}(n)$ is chosen. The agent n offers neighbour j the unit u. As an answer, the agent gets a value $\mathcal{B}_1$ which tells how many units neighbour j still accepts. Another property which has to be checked is the current transportation capacity $\mathcal{B}_2$ via the connecting edge from n to j. If both values $\mathcal{B}_1$ and $\mathcal{B}_2$ are larger than zero, the agent sends the unit u to its neighbour n. The current storage amount K is updated automatically for both agents n and j by using the known continuity equation (Eq. 3.4):

$$\forall y \in \{n, j\}: \quad K(y,T) \leftarrow K(y,T) + \Delta t \cdot F(y) \tag{4.13}$$

where $F(y) = 1$ if $[f((y,j),T) - f((n,y),T)] > 0$ and $F(y) = -1$ if $[f((y,j),T) - f((n,y),T)] < 0$ and otherwise $F(y) = 0$. Notice that the flow $f((n,j),T)$ is updated in line 10 by adding $1/\Delta t$. The flow is a rate measured in units per time-step. So we have to add a rate, too. The loop continues until each unit has been offered.

4.3.2 MTP Controller

As already mentioned, the communication in production processes occurs much faster than the exchange of commodities and because of this it is possible to send much more messages than commodities during the time interval Δt. As communication is much faster than commodity exchange, communication could be used to negotiate the exchange of commodities among neighbours. The decentralised controller of the previous section already uses this property of communication. It sends a request to a neighbour node to ask for acceptance of a commodity. The neighbour node answers to the request based on its current internal inventory level K. It rejects the request if its current inventory level K equals its complete capacity C. At this moment no additional unit can be imported but a few moments later (still during Δt) it could be possible again when commodities might have been exported to a neighbour node. But the present controller of section 4.3.1 will not retry its offer. Only one request for each unit is sent during Δt. The new controller that we present in this chapter will retry its requests as long as possible so that its neighbour nodes could accept additional units.

Following Eq. (4.1), we already know how many units a neighbour node j of n can import during the Tth time interval of the simulation:

$$I(j,T) \cdot \Delta t = [C(j) - K(j,T)] + O(j,T) \cdot \Delta t \tag{4.14}$$

During the time interval $T \cdot \Delta t$, the agent j can import as many units as free storage places are available and as many units as are exported during the time interval Δt. The problem is to know how many units the neighbour will export during the time interval. This is a difficult task as Eq. (4.11) shows, even if we assume that all agents respect the boundary condition of *maximum-load* that forces them to export as many units as possible. The exportation of units always depends on the agents' neighbours, their possible importation depends on their neighbours and so on. Without knowing how many units the neighbour node can export, the controller has to retry its offers

endlessly[2]. The controller needs a termination criterion which allows him the decision to stop requesting neighbours. Such a termination criterion could be given if all neighbours inform the agent when they are *blocked*, which means that no additional unit can be imported in this time interval. Then, the agent does not send requests to its neighbours any longer. As we will see, the blocking information is spread out in the network like in a dynamic programming approach. An agent itself recognises that it is blocked, when all possible neighbours are blocked or no transfer to them is possible and when the internal buffer is filled:

Definition 4 An agent n is called *blocked* if its import $I(n,T)$ becomes zero:

$$I(n,T) = [C(n) - K(n,T)]/\Delta t + O(n,T) \; = \; 0$$

To be zero, two conditions have to be valid: First, the buffer has to be filled completely with units. For this, $K(n,T) = C(n)$ and additional units can only be accepted if other units are exported. The outflow $O(n,T)$ is the second condition and has to be zero. Therefore, we need to know the precondition for which no outflow of a node is possible. Of course no unit will be exported if there is no unit to export: $U(n) = \emptyset$ where $U(n)$ is the set of units which is chosen for export. The export of units can still be impossible even if $U(n) \neq \emptyset$. It becomes impossible to export units to a neighbour j if either the transportation capacity $c((n,j))$ is exhausted or if j is blocked. We notice here that it is not sufficient to screen only the buffer contents of the neighbour because its import depends also on its export. Let us summarise the conditions for the termination of an agent's offering:

$$\text{Termination condition:} \qquad T_e = \begin{cases} 0 & : \; \sum_{j \in \Gamma^-(n)} [c((n,j)) \cdot \overline{b_n}(j)] = 0 \\ 0 & : \; U(n) = \emptyset \\ 1 & : \; \text{else} \end{cases} \tag{4.15}$$

where b_n is a vector which contains the *blocked* status of all children nodes of n. The entry $b_n(j) = 1$ if neighbour j of n is blocked else zero and $\overline{b_n}(j) = 0$ is the opposite. An agent stops its offering if either no unit is left for exportation or no unit can be exported anymore or if both is the case. In Eq. (3.8) we demand that at least one child of a node n has to get a weight entry in the action-value table $D(n)$ that is larger than zero. Thus, we can exclude the case that no units can be delivered because of the derived policy from $D(n)$. To guarantee termination we have to tighten this precondition by forcing all children nodes of n to have a weight entry in $D(n)$ larger than zero:

$$\forall n \in V \forall j \in \Gamma^-(n) : \; D_j(n) > 0 \tag{4.16}$$

The MTP controller is shown in Fig. (4.7). The only difference from the controller previously introduced in section 4.3.1 is a single line. Instead of one offer per unit, the agent continues offering until no unit is left for exportation or until transportation to child nodes becomes impossible. We claimed that this controller algorithm terminates

[2]This implies the question about synchronisation and the length of a time interval Δt. In our parallel process we assume that a complete iteration step Δt has to spend at least as long as the slowest agent needs for its predefined tasks.

```
1   function MTPController(n, D(n), U(n), J(n), T)
2   foreach y ∈ Γ⁻(n) do f((n,y)) = 0 end
3   assert information P(n) from all children y ∈ Γ⁻(n).
4   while T_e > 0 do
5       choose children j ∈ Γ⁻(n) by policy derived from D(n) with Eq. (4.12).
6       offer j unit u and get answer B_1 = C(j) − K(j,T).
7       check rest-capacity B_2 = c((n,j)) − f((n,j)) of edge (n,j).
8       if (B_1 > 0) & (B_2 > 0)
9           send unit u to neighbour j.
10          f((n,j)) = f((n,j)) + 1/Δt.
11      end
12  learn new policy D(n) from the set {B_1, B_2, J(n), P(n)}
13  end
```

Figure 4.7: Pseudo-code for the MTP controller. In comparision to the controller design of Fig. (4.6) only line 4 has been changed. Now, the offering runs until no unit can be exported anymore.

its offering of units. We are going to concrete this with the following theorem:

Theorem 1 All MTP controllers terminate after a finite number of loops.

Proof (of Theorem 1) We are only interested in controllers that have units to export. Otherwise the second part of the termination condition becomes active: $U(n) = \emptyset$ and the controllers stop offering immediately. It is also easy to see that a controller j terminates if all its children nodes have terminated: On one hand j, terminates if it has allocated all of its export-able units. On the other hand, if the controller j does not run out of units, it will exhaust the edge capacity $c((j,k))$ of each child $k \in \Gamma^-(j)$ or fill up the inventory level of k until k is blocked. In both cases j terminates because no additional commodity unit is deliverable anymore.

Because of the acyclic topology of the network (see Definition 2) there is at least one controller which has only the sink as its child. This controller terminates because it can allocate all of its units at sink - or as the case may be - because its outgoing edge capacity is exhausted. In general, all controllers either exhaust the edge-capacity of an edge that is connected with the sink or they run out of units and terminate.

We have shown until now that at least two controllers terminate, namely the controller which is connected only with the sink and the sink itself (no commodity export). Because of the acyclic structure of the network, there is at least one parent node whose children set is completely covered by the *blocked* nodes. This parent node terminates as well. This procedure iterates until the whole network is covered. The network has only a finite number of nodes and each controller has only a finite number of units to deliver (except source). And because of this the information is spread out through the network in a finite number of hops. All controllers terminate after a finite number of loops. □

We call such a controller a *maximum throughput policy* (MTP) controller because it guarantees a maximum average network throughput independent for the policy derived from D at each controller. We are going to proof this now:

Theorem 2 If all controllers are MTP controllers, all policies result in an average maximum network throughput.

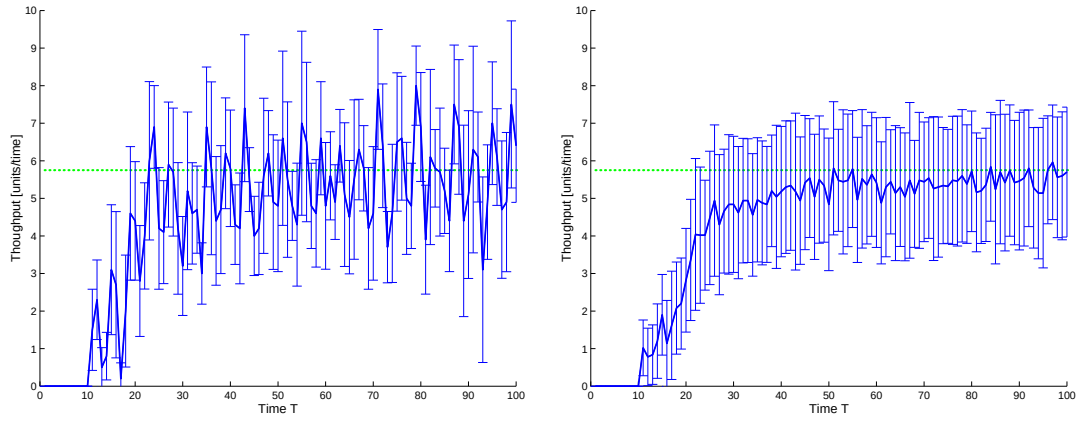
Proof (of Theorem 2) For this proof we look at the set P of all *paths* from source to sink which differ at least in one node. With the help of P we can specify the complete flow F_{total} from source to sink:

$$F_{\text{total}} = \sum_{p \in P} f_p, \qquad \sum_{p \in P \wedge e \in P} f_p \leq c(e), \qquad \sum_{p \in P \wedge n \in P} f_p \leq C(n)$$

where f_P is the flow via a path p of the set P. F_{total} is the sum of all flows f_P via all paths of P from source to sink. F_{total} holds the boundary conditions that the total flow via an edge e can not exceed the edge capacity $c(e)$ and the total flow via a node n can not exceed the node capacity $C(n)$. In section 4.1.2 we saw that nodes can be expressed as edges with special edge-capacities. Therefore, in the following we only deal with edge capacities. With the help of the Ford-Fulkerson theorem [FF56] (see Appendix A), we see that each flow f_P has an upper bound that is equal to the minimum edge-capacity of all edges on the path P. In other words, the Ford-Fulkerson theorem says that the maximum throughput from source to sink equals the minimum sum of the capacities of those edges which have to be removed to disconnect source and sink. The theorem is in this form only valid if all edge-capacities are expressible as integers.

A MTP controller derives its policy from the action-value table D. A precondition demands that all weights for the controller's children are larger than zero. This implies that if the controller has an average number of commodities to send then the average flow to any child node is also larger than zero. Let us choose the weights of D arbitrarily and let us count the frequency of the use of edges. This frequency in time corresponds to the average flow along the edges.

The goal now is to increase the source outflow incrementally until no further increase is possible. Then, a so called *minimum-cut*, is found namely the edges which are exhausted and, if removed, which disconnect source and sink. In the beginning, no capacity of any path has been exhausted by the flow f_P. Then, we increase the flow incrementally on all paths until a controller reports that it can not increase the flow f_P for a given path. The controller has now detected a bottleneck and gets rejections for this offer. The MTP controller continues to offer until another child is drawn randomly based on the action-value table D. This child accepts the unit or it terminates because of the termination conditions. If it terminates it becomes another bottleneck and the parent controller will also detect this. In other words, the MTP controller changes its distribution policy and increases other flows f_P instead. This is done by all controllers until no further policies are found which can handle an increasing average source outflow. Now, we have used on every path at least one edge to its capacity. If we remove



(a) without local oscillation reducing (b) with local oscillation reducing

Figure 4.8: *MTP: Static - Equal Distribution.* On the left side is shown the oscillating behaviour of the throughput of a system which consists of MTP controllers. The right side shows the system with MTP controllers but this time the local oscillation reduction method of section 4.2 is implemented.

all those edges, we disconnect source and sink for other flows f_P and because of this we have maximum average throughput. □

Oscillations

One drawback of the MTP controller is that the throughput usually oscillates. The oscillation is much more distinctive than the throughput oscillation of the Single Trial (ST) controller of section 4.3.1. The reason is the property of the MTP controller to offer units as long as no termination condition is valid. Thus, the controller exports as much as possible each time step. In the beginning, the source's children are without any commodities so they accept the maximum amount of them. In the following time step, part of their buffers are filled with commodities that have to stay at least the nodes' residence times. Thus, the source commodity outflow is possibly reduced. When the first units that have stayed longer than the residence time, are exported, then at least the same amount of units can be imported again and so on. This results in an oscillation.

The oscillation is shown in Fig. 4.8(a). The blue line is the average throughput in a static scenario of the test network of chapter 5.1 (100 runs). The oscillation behaviour is undesirable in most real cases. Thus, we use the local outflow oscillation reduction method of section 4.2. The result is shown in Fig. 4.8(b). This time, the current throughput shows that the oscillation is reduced. But as we have seen in section 4.2, a network with MTP controllers that use the oscillation reduction method can not reach its theoretical average throughput. It will result in a slightly lower throughput. But in most cases the difference is negligible. We use this kind of local oscillation reduction in combination with every MTP controller.

4.4 Policy Optimisation

In the previous section we introduced two different controller types. The MTP controller, for example, ensures a maximum throughput for any kind of distribution policy. Although every policy results in a maximised throughput, it is still important to optimise the distribution policy. If the policy is chosen badly, the throughput converges very slowly to its theoretical bound. Policies that converge in a short time are needed especially in dynamic environments. These best policies are not known in advance but have to be learned by the controllers themselves. The goal of the next section is to present several different *policy optimisation* algorithms.

4.4.1 Equal Distribution (ED)

Both types of controllers of the previous section distribute their exportable units based on a probability distribution. The policy to select a child is derived from the action-value table D with the help of Eq. (4.12). Children with larger weights in D are more likely chosen than others. The agent controls the average flow via its outgoing edges with the adaptation of those weights. In the simplest approach, the weights stay all equal in size. This means that the agent does not give priority in shipment to any children and that their selection is uniformly distributed. We use this approach to benchmark online optimisation approaches of the following subsections.

4.4.2 Learning to Avoid Blocking (LAB)

Assume that controller n has u units which it can export. But which of its neighbours get units and how many? Responsible for the agent's decision is its policy which is derived from its action-value table D. Depending on the entries of D, the agent selects one of its neighbours and offer it a unit. This neighbour can reject the offer or accept it. Obviously, the agent has to find neighbours which accept units. Neighbours which reject units *block* the traffic. Therefore, the goal of a *learning* approach has to be *to avoid blocking* neighbours.

The following approach is based on the learning approach from P. Bak and D. Chialvo which is called *Learning from Mistakes* [CB97]. The idea of their approach is that an agent learns by punishment. Each time the agent chooses an action that turns out to be wrong, the agent is punished. The agent remembers this in following situations in which the punished action could be chosen again. On the other hand, an action which turns out to be successful is specially marked. This mark tells the agent that this action has once turned out to be successful. If this specially marked action is chosen again and it turns out that this has not been an successful choice, then the action is punished less then without being marked. The agent remembers actions which turned out to be successful. After some time the agent forgets which nodes have been marked. The game starts again.

With the help of Eq. (4.12) the agent n selects one of its neighbours randomly. The policy $D(n)$ weights the neighbours. Neighbours that are known to accept more units

should be weighted better or rather with a larger weight than neighbours that accept less units. Let us assume that all neighbours of n are initially weighted equally. None of its neighbours is preferred. If a unit can not be delivered to a selected neighbour because it is exceeding the capacity of the connecting edge or because the neighbour rejects the unit, the weight for selecting this neighbour has to be reduced. The result is that this neighbour will be chosen with lower probability at the next loop. Agent n adapts its policy $D(n)$ each time an offer is rejected by a neighbour node j:

$$D(n,j) \leftarrow D(n,j) \cdot (1 - \frac{1}{1+k}) \tag{4.17}$$

where $D(n,j)$ is the weight in policy $D(n)$ of agent n for its neighbour j and u is the number of units that have been marked as export-able at the beginning of the time interval Δt. The parameter k is a kind of learning parameter. To avoid too small weighted entries in D we standardise the policy D by its largest entry each loop:

$$\forall j \in \Gamma^-(n) : D(n,j) \leftarrow D(n,j)/\max\{D(n)\}$$

In the beginning, the agent has no information about which of its neighbours would accept most commodities. In the absence of such knowledge, the agent weights all neighbours equally.

4.4.3 Adaptive Link State Routing (ALSR)

In a link state approach the complete knowledge about the network state is distributed to each agent in the network. The distribution is calculated based on an internal map which represents the complete network. Changes in topology or network states have to be flooded throughout the network. An agent who receives such a notification, which is called a *link state advertisement* (LSA), updates its internal map and broadcasts this information to its neighbours. The distribution policy D is in most link state routing approaches calculated by a shortest path algorithm. A famous example of such an algorithm is the OSPF protocol introduced by Moy [Moy98]. It is the current routing algorithm in Internet.

Because protocols based on link state routing keep complete topology information at routers, they avoid long term looping problems of old distance vector protocols [Beh97]. However, the requirement that the complete topology is broadcasted to every agent does not scale well [ERH92]. There are two main scaling problems: flooding requires excessive communication resources, and computing the routes that use complete topology databases requires excessive processing resources [Beh97]. A lot of enhancements have been implemented in current routing protocols to overcome the problem of scaling. First of all they are multi-casted which means that not every node calculates the routing policy for itself. Neighbour nodes are put together in blocks and routing is calculated for the whole block. This saves processing resources. Other approaches like *Link Vector Algorithms* (LVA) [Beh97] try to overcome the problem of scaling by coding the messages in a special way while communicating.

Using shortest paths algorithms is only one possibility for an agent to identify the following node for its units. Another possibility would be the use of maximum flow algorithms. In both cases, the agent uses the information to update its action-value table D. The agent-based controller chooses a child node based on its policy derived from D with the help of Eq. (4.12). Traditional maximum flow algorithms like the algorithm in the appendix A are not able to handle production networks with their particular properties. Properties of industrial components like residence times or node capacities are not taken into account. Fortunately, we have already developed in section 4.1.2 a suitable transformation of a production network into a traditional network representation. For this, each component is divided into two traditional nodes and a connecting edge with the capacity of the maximum theoretical throughput of that node. This average maximum node throughput has been found already in Eq. (4.5) and is the ratio of node capacity and residence time. When a maximum flow algorithm is applied to the new network representation, the average maximum throughput between two selected nodes is returned. Algorithms like the one in Fig. (A.2) can be modified to return also the flow on each edge. We interpret these edge flows of the maximum flow algorithms as the frequency in which the edge is used. And this can be taken as an advice for the controllers' commodity distribution. The ratio of the flow along the outgoing edges of a node represents the weights of the action-value table D. Consequently, each agent calculates its updated policy D as follows:

1. Update network states and transform production network $N(V, E, c, C, W, s, t)$ into a traditional network $N'(V, E, c, s, t)$ using technique of section 4.1.2.

2. Calculate maximum edge flows $\mathcal{F}$ of network $N'(V, E, c, s, t)$ using a maximum flow algorithm. Network flow from network source to network sink is calculated.

3. Use outgoing edge flows $\mathcal{F}_i^{\text{out}}$ of own node i as weights in D to derive distribution policy.

4.4.4 Reinforcement Routing (RR)

The Routing Information Protocol (RIP) [Hen88] is an example of a Distance Vector Routing (DVR) algorithm [AMM01, Bel58, BG87, CRKGLA89]. The principles of that approach are based on the principles of dynamic programming [Ber82]: A problem of size k could be solved by first solving the sub-problem of size 0, then of size 1 and so on. This way, the problem of size k can be solved gradually. In our purpose, dynamic programming says that an optimal path is made of sub-optimal paths. In a DVR algorithm the nodes exchange their routing vectors that represent shortest path distances. In the case of our industrial network with a common destination, a routing vector contains the node's last estimate of the shortest distance to the network sink. Each agent periodically updates its distance vector from the distance vectors regularly sent by its neighbours. Following the notation of Heusse et al [HSGK98], the distance $\mathcal{T}_{n,d}^i$ is the distance estimation of agent i to the destination d by sending units via neighbour n. Each agent i periodically

updates its distance vector as follows:

$$\mathcal{T}^i_{n,d} \leftarrow d_{i,n} + \min_{j \in N(n)}\{\mathcal{T}^i_{j,d}\}, \quad \text{with } \mathcal{T}^i_{n,n} = d_{i,n} \tag{4.18}$$

where $d_{i,n}$ is the known distance between i and its neighbour n. It has been shown that this process converges in finite time to the shortest paths with respect to the used metric if no edge cost changes after a given time [BG87]. But DVR algorithms are not used anymore in modern Internet protocols like OPSF. The reason is that the convergence is often too slow and a DVR protocol is more adaptive to adding new edges than to failure of existing edges [Tan96, HSGK98]. Today, they are mainly used for intra-domain routing, e.g. in the Routing Information Protocol (RIP) supplied with the BSD version of UNIX [DCG99].

Littman and Boyan [LB93] propose an online and asynchronous version of this distance vector routing based on reinforcement learning [SB98]. Once again, $\mathcal{T}^i_{n,d}$ is the cost estimated by agent i for delivering a unit to its destination d via its neighbour n. The agent chooses the neighbour $n \in \Gamma^-(i)$ which has the smallest cost estimation toward destination d, where $\Gamma^-(i)$ is the set of children of i. The idea is that children which are closer to destination d have also a better estimation of their cost for delivering a unit to d. Thus, it allows the agent to update its own estimation based on the cost estimation of the neighbouring node which has received the unit. The agent does not have to wait until the unit has reached its final destination d. The update is as follows [HSGK98]:

$$\mathcal{T}^i_{n,d} \leftarrow (1-\eta) \cdot \mathcal{T}^i_{n,d} + \eta \cdot (d_{i,n} + \min_{j \in \Gamma^-(n)}\{\mathcal{T}^n_{j,d}\}) \tag{4.19}$$

where η is the so called *learning rate* of reinforcement learning approaches. Boyer and Littman introduced *Q-Routing* [BL94] which has shown better results as simple shortest paths algorithms in communication networks. But again, this approach has shown that its convergence is still slow and not applicable in the context of fast changing environments. To speed up the convergence, the agents need more knowledge for their commodity distribution. One possible way is the *multiple round trip routing* approach of Heusse et al. [HSGK98]. They use *forward* and *backpropagating mobile* agents to transfer additional information. The forward agents in their approach share the same queues as the units and use the same routing policies. On their way to their destination they keep track of the costs between hops. The backpropagating agent retraces the units' way back to the source and updates routing policies at each agent. Instead of forward propagating mobile agents we can use the commodities if they are marked properly. Such a proper stamp has to include three kinds of information: The identification number of the intermediate node, a time stamp of the unit's arrival, and a time stamp of the unit's departure. If a unit arrives at its destination, which is the sink t in our model, it launches a backpropagating message retracing the way back to inform all nodes on the path and updates is distance estimations like follows:

$$\mathcal{T}^i_{n,t} \leftarrow (1-\eta) \cdot \mathcal{T}^i_{n,t} + \eta \cdot d_{n,t} \tag{4.20}$$

where $\mathcal{T}^i_{n,t}$ again is the distance estimation of agent i to send a unit to its sink t via the way of neighbour n and η is the learning rate again. The reward $d_{n,t}$ is the average

delay from n to t of all units which arrived at the same time step at t. This approach is successfully tested on several different network types. And this kind of update rule is similar to the ant-based approach we will present in the next subsection.

For our purpose we use both reinforcement approaches; multiple round trip routing and Q-Routing. The commodities are marked properly like described above and are used as forward propagating agents. On their path they keep track again of the travel times between two nodes. A travel time between two nodes i and j is the number of time steps a unit u has to stay at node i and the time u needs to be transferred to node j. In this approach, the agent has to learn to optimise two estimates. The first is $\langle \mathcal{T}^n_{n,t} \rangle$ the average distance estimation from n to t via all neighbours. All back propagating agents update the nodes' agent's estimate like follows:

$$\langle \mathcal{T}^n_{n,t} \rangle \leftarrow (1 - \eta_1) \cdot \langle \mathcal{T}^n_{n,t} \rangle + \eta_1 \cdot \langle d_{n,t} \rangle \tag{4.21}$$

where $\langle d_{n,t} \rangle$ is the average travel time from n to t and η_1 the learning rate. The second estimate is the distance of its neighbours to sink. Their distances can be learned like in Eq. (4.19):

$$\forall j \in \Gamma^-(n) \; : \; T^n_{j,t} \leftarrow (1 - \eta_2) \cdot T^n_{j,t} + \eta_2 \cdot (W(n) + \langle T^j_{j,t} \rangle) \tag{4.22}$$

where $T^n_{j,t}$ again is the estimated travel time for a unit from node n to t via the neighbour j and η_2 is again the learning rate. This update estimate the travel time for a unit from the agent n to its neighbour j and then the travel time to sink. The travel time for a unit is given by the time $W(n)$ that a unit has to stay at least at n and the estimated time from its neighbour to sink. This approach has two update rules which are updated in different time scales. The first rule is updated irregularly by back-propagating mobile agents and the second regularly each time step.

To derive its decision, the agent uses its action-value table D. We use the weights in $\mathcal{T}^n_{j,t}$ for the children $j \in \Gamma^-(n)$ as the entries for the children in D: $D(n, j) = 1/\mathcal{T}^n_{j,t}$.

4.4.5 Ant Routing (AR)

Ant algorithms are successfully applied for finding shortest paths in combinatorial optimisations and communication network problems [DCG99]. In 1991 Dorigo introduced the first *Ant Colony Optimisation* (ACO) heuristic called *Ant System* (AS) [Dor92]. It is based on the observed phenomenon that real ants are able to optimise the paths from their nest to its food location by very simple rules. Artificial ants imitate the behaviour of real ants to perform similar routing tools. The Ant System has been applied to a lot of different approaches. A good overview is found in Di Caro and Dorigo [DCG99]. Di Caro and Dorigo also introduce an ACO called *AntNet* [CD97] to optimise networks. They have shown that their approach outperforms OSPF and Bellman-Ford network algorithms in communication networks. Heusse et al. [HSGK98] use similar forward and backpropagating ants but they use a different procedure for updating the routing table. Here, a forward ant is not necessary because the system is running under heavy load and the units themselves can be used as forward ants if they are marked properly. Like in

the Reinforcement Routing approach a proper stamp has to include three kinds of information: The identification number of the intermediate node, a time stamp of the unit's arrival, and a time stamp of the unit's departure. The source emits as many commodity units as possible, which are directed by the agent-based controllers of section 4.3 towards their destination. At each intermediate agent the units are forwarded by a distribution policy. If a unit reaches its destination, a so called *backward ant* is initialised to return to source exactly on the same path. On its way back it deposits a quantity of pheromone $\Delta\tau$ on each edge that it has used. In this way, the edge pheromone concentration on an edge on the path of the returning ant is updated as follows:

$$\tau_{i,j} \leftarrow (1-\rho)\cdot\tau_{i,j} + \Delta\tau_{i,j} \tag{4.23}$$

where $\tau_{i,j}$ is pheromone concentration on the edge connecting node i and node j. The coefficient $\rho \in (0,1]$ is a pheromone trail decay coefficient which ensures that less used edges become less attractive in time. An agent n updates its action-value table $D(n)$ based on this pheromone concentration $\tau_{n,j}$ on the edges to its neighbours j and additionally based on a second criteria which is called the *heuristic value* $\eta_{n,j}$:

$$D(n,j) \leftarrow \frac{[\tau_{n,j}]^{\alpha}\cdot[\eta_{n,j}]^{\beta}}{\sum_{y\in\Gamma^{-}(n)}[\tau_{n,y}]^{\alpha}\cdot[\eta_{n,y}]^{\beta}} \tag{4.24}$$

where $D(n,j)$ is the weight for n's children j, and α and β are two parameters that control the relative weight of pheromone concentration and heuristic value.

This update procedure for the agent's policy contains two parameters which have to be adapt for the purpose of the network optimisation. The first is the quantity of pheromone $\Delta\tau$ that the returning ant deposits on its way back to source. We choose it to be proportional to the unit's transportation time L:

$$\Delta\tau_{n,j} = \begin{cases} 1/L^{k} & : \text{ if } (n,j)\in T \\ 0 & : \text{ if } (n,j)\notin T \end{cases} \tag{4.25}$$

where $\Delta\tau_{i,n}$ is the pheromone concentration which is added to the edge connecting node i and neighbour node n. The pheromone concentration is only added if the edge is on the path T of the back propagating ant and L^{k} is the travel-time of the unit k which has launched the backward ant.

The heuristic value $\eta_{i,n}$ represents a second type of information obtained by the environment. The first information has been the pheromone concentration which is directly correlated to the usage of a specific path. The heuristic parameter can be used to obtain additional information like the condition or state of a node or edge. In our case, we try to avoid that some paths are overloaded. For this, the second parameter $\eta_{i,n}$ of the edge connecting node i and its neighbour n is chosen as follows:

$$\eta_{n,j} = \min\{c(n,j), \frac{C(j)}{W(j)}\} \tag{4.26}$$

where $c(n,j)$ is the capacity of the edge connecting node n and its neighbour node j.

Chapter 5

Analysis

In the previous chapter we introduced several different throughput online optimisation approaches. In this section we present the results of the approaches that were applied to a test network. This demonstration network will be introduced in the section 1 of this chapter.

We investigate all approaches in two scenarios. The first scenario is in a static environment and it is of interest to determine the *quality* of the results of each of the approaches. In this context, quality is understood as the distance from the converged average throughput to its theoretical bound. Some of the approaches take very long to converge. Therefore, the quality (as a converged value) is an insufficient measure for our purpose. Because of this, we determine the quality in a specified time interval. We start with an unlearned system at time zero and move to a learned system at a time at which we guess most approaches have converged sufficiently already.

Another measure of interest is the communication effort. As already discussed in the previous chapters, the communication effort is the amount of messages that are needed to negotiate the exchange of commodities. Combined with the quality, the communication effort is a measure for the efficiency of an approach. Less communication is an indication for better resource allocation. This becomes an important aspect in large and complex networks.

In a second scenario, we are interested in the behaviour of our approaches in a dynamic environment. If the topology of the networks changes suddenly, the distributing agents have to find new customers for their commodities. In a dynamic environment the key feature is adaptedness. Only approaches which adapt fast to changing conditions are successful.

5.1 Demonstration Network

We have introduced several different optimisation approaches in the previous chapter. We applied them on lots of random generated networks but only a few of them have typical behaviour for industrial networks. To demonstrate the advantages and disadvantages of each of the approaches, we introduce a typical industrial topology network that

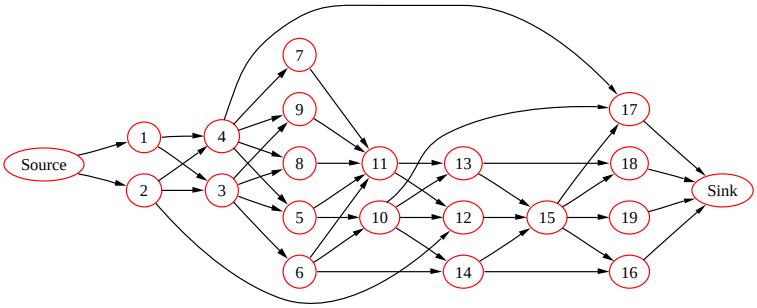


Figure 5.1: The topology of the test network for benchmarking the algorithms.

shows the common behaviour of supplying networks. Here, the demonstration network has the typical shape of a production process. In definition 2 (pp. 19) we demanded that production processes are acyclic. There is no reason for a unit to visit a node twice. The whole production process is mostly straight forward.

An important property of those shapes is that they are built of connected *layers*. A *layer* contains a number of parallel working components, for example machines. Nodes with a layer are not connected among themselves (acyclic) but they are connected with other layers or components. In most real production processes the majority of connections are from one layer to its successive layer. Only a limited number of so called *shortcut links* exists, which connect layer-nodes with layers more far away.

The test network in Fig. (5.1) represents such an industrial network. Each node in the network could be an industrial component like a shelf, a conveyor belt or a machine. But in the test example the nodes are single basic-components as introduced in section 3.1. The three quantities of the basic nodes are the node-capacity C, the current inventory-level $K(T)$ and the residence-time W of a unit. The edge-capacities c are the upper bounds for the unit transportation. The quantities are random numbers (positive but not necessarily integers). The node capacity C is chosen randomly within the range:

$$0 < C(n) \leq W(n) \cdot \min\{ \sum_{y \in \Gamma^+(n)} c((y,n)), \sum_{z \in \Gamma^-(n)} c((n,z))\} \tag{5.1}$$

This upper bound is easy to deduce from the Little's Law. Thus, capacities larger than the upper bound of Eq. (5.1) are not reasonable. Only the node-capacities for the source and the sink are infinitely large. At the beginning of each test, all inventory levels of all nodes are empty expect the inventory level of the source node which is completely filled.

5.1.1 Simulation Scenarios

The behaviour of the optimisation algorithms has to be tested in static and in dynamic environments. For this we define two test scenarios:

Static Scenario: In this scenario no node or edge fails. The convergence of the network throughput and its maximum is tested with each algorithm.

Dynamic Scenario: In the dynamic scenario the network topology changes every 50th time step. Sometimes a node fails and other times a node appears. The Fig. (5.1) shows the test network. The influence of a topology-change event depends on the selected node. Thus, we have to make sure that all algorithms are tested with the same sequence of node changes. For this, we choose nodes which have large influence at the network flow. The following sequence holds the nodes which fails or (re-)appears every 50th time step. The first node change starts an time step zero: $[4\downarrow, 4\uparrow, 15\downarrow, 6\downarrow, 15\uparrow, 6\uparrow, 7\uparrow]$. Where the symbol $\downarrow$ stands for a node's failing and the symbol $\uparrow$ stands for the (re-)activation of the node with that number. Of particular interest in this scenario is the time the current throughput takes to converge to the new theoretical network throughput.

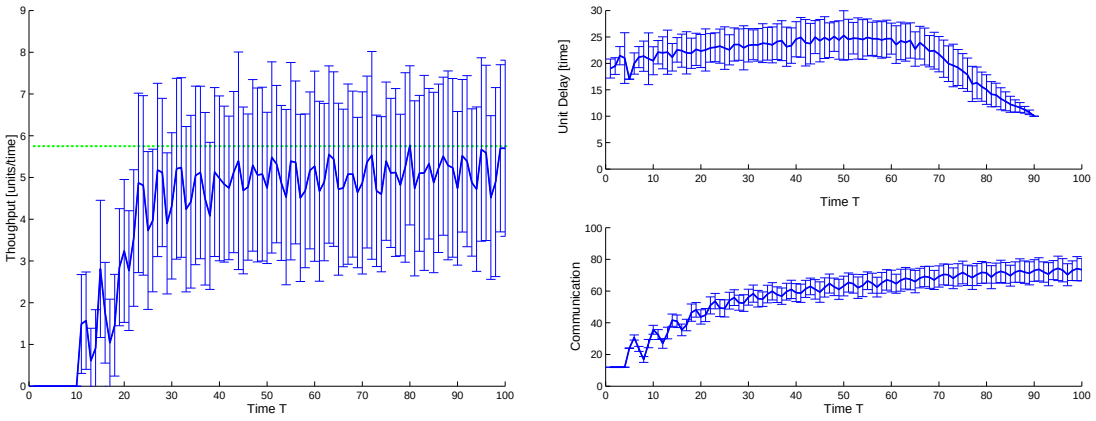
5.1.2 Quantities

We investigate the differences of optimisation approaches by comparing some of their quantities. Obviously the main focus for our purpose is the network throughput. This quantity is a direct measure for the quality of our approaches. But other quantities are also important if we assess the utility and the reliability of approaches in real environments. Some of those important quantities are the transportation delay of commodities on their path through the network or the number of messages which have to be exchanged to fulfil the designed objectives. In the following we discuss the observed quantities.

Throughput

The *throughput* is measured as the number of commodities which arrive at the sink during a time step. Because of hysteresis effects (see section 4.1) the throughput usually fluctuates. Therefore, two throughput behaviours are of interest: First, the global maximum throughput and second, the average network throughput. The design of a production line has to cope with peak throughput requests. The costs for the preparation of peak throughput depends on the size of the global peak. In that respect one goal of network throughput optimisation is to reduce the maximum peak throughput without reducing the average network throughput. The average network throughput specifies how many commodities can be drawn out of the network. The goal is to maximise the average network throughput towards its theoretical upper bound.

An example for the throughput analysis is shown in Fig. 5.2(a) for the static scenario. The thick (blue) line is the sample of 100 runs of the simulation and shows the average network throughput. The error-bars are the standard deviation of all 100 samples. The horizontal line shows the theoretical maximum bound of the throughput. In general, we can classify solutions by their adaptation speed and their quality. An optimisation approach results in a good quality solution if the difference of the network throughput and its theoretical bound becomes small.



(a) Throughput (b) Transportation Delay and Communication Effort

Figure 5.2: (**Static Scenario**): On the left part of the figure, the throughput is shown. It is the number of incoming units at the sink at each time step T. The horizontal line is the theoretical maximum bound. On the right side two graphs are shown: In the upper part the transportation times (unit delay) for units through the complete network. The other (blue) line shows the average transportation time of units initialised on the journey at that time step. In the lower part of the figure the communication effort is shown in the amount of messages which are exchanged during a time step. For more information see text.

The adaptation speed is a kind of measure (we introduced) that counts the time steps until the throughput has practically finished converging. As we see, this is a subjective measure. For example, in Fig. 5.2(a) the throughput stays after 40 time steps almost at the same throughput level of about 5 units per time step. Also, another quantity is of interest. As already discussed, the fluctuations are unwelcome. Solutions which have a small standard deviation present more continually throughput. Therefore, another goal of optimisation is to reduce the fluctuations shown here by the standard deviation.

Communication Effort

Messages have to be exchanged to negotiate the delivery of commodities. The amount of all messages in the network during a time interval is called the *communication effort*. The concept of chapter 3.1 assumes that the transfer of commodities via edges is instantaneous. The edges only limit the amount of commodity units that can be transferred during a time interval. If we would like to implement transportation times explicitly, we have to construct intermediate components like conveyor belts or forklifts with a finite residence time W. If the transportation of units does not take time, then the most important time taking tasks of an agent is reasoning about its delivery decision (learning approach) and its message transfer. A message in our context is the tuple of request and answer. In chapter 4 we introduced two kinds of controllers which negotiate their commodity distribution differently. Both controllers have to send at least as many

messages as units they intend to export. This corresponds to the minimum number of messages which have to be exchanged if the receivers accept all offers. The first kind of controller (STP Controller) sends only this amount of offers even if some receivers do not accept commodities. The second kind of controller (MTP Controller) continues offering until all units are allocated or the agent gets to the conclusion that no further commodity can be accepted by its neighbours. Therefore, if an agent of the second type does a good job and optimises well its distribution policy then it will reduce its amount of messages needed to export its units.

Independent of the controller type (STP or MTP) there are two different communication efforts. The first is the communication effort described above. Agents negotiate with neighbours for exchanging goods. Some approaches are able to communicate with agents other than their neighbours. They collect more information, like traffic, on the further path to sink. This communication differs from the first as it affects the communication traffic more globally. Due to this, we distinguish between local and global communication.

As an example, the communication efforts for a static scenario are shown in Fig. 5.2(b). In the lower part of the figure the communication effort is shown. The number of messages, which are needed to export the units of all agents in the network is plotted during each time interval Δt. In the beginning, the network is not filled with units. Therefore, there is only a limited number of messages needed to pass them. With increasing time, the nodes become more and more filled with units and their controllers have more units to export. This results in an increasing communication effort with time.

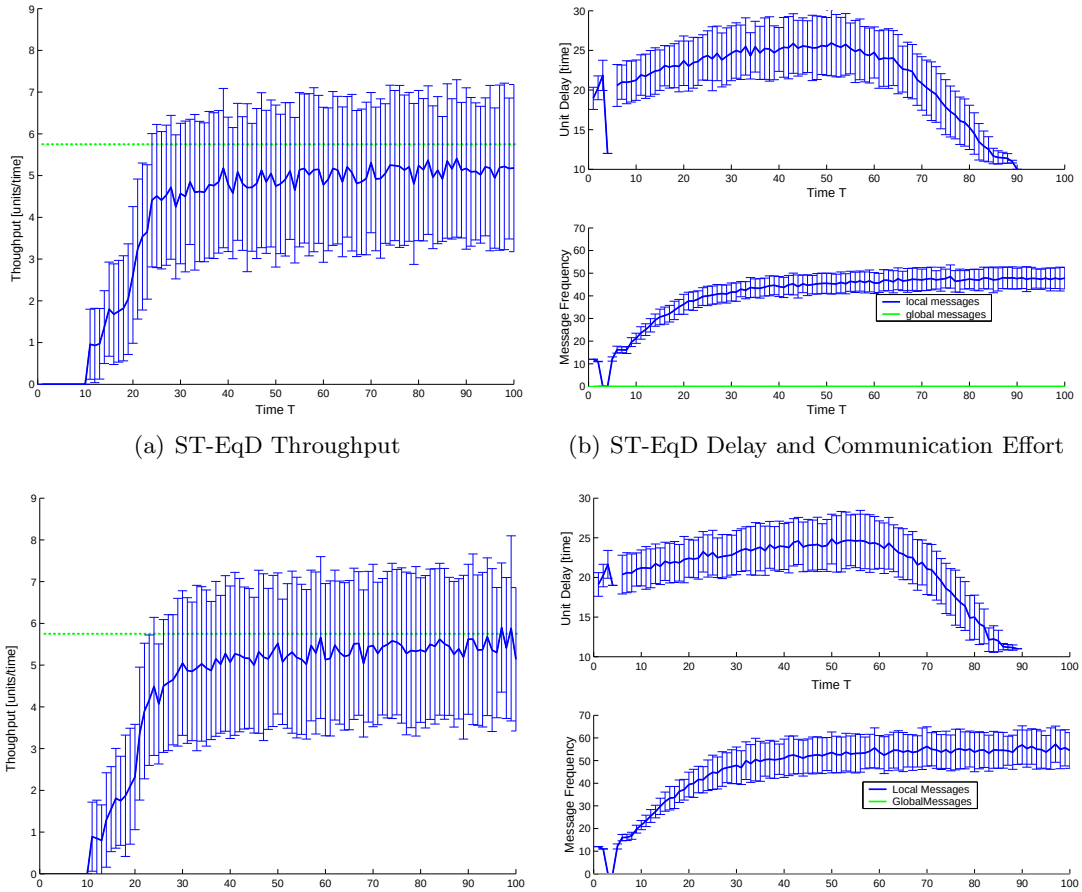
Transportation Delay

Another important quantity in the investigation of optimisation approaches for networks is the *transportation time* (unit delay) for a unit beginning its journey at the source and ending it at the sink. Although the minimization of the transportation time is not a primary goal in this thesis, this quantity is useful for selecting different approaches. In the upper part of Fig. 5.2(b), the transportation times for static and dynamic scenarios are plotted. The thick (blue) curve shows the average transportation times for units initialised at each time step. The decreasing tails of both curves are because of statistical reasons. The statistic counts only units which were able to reach the sink within the simulation. In the tail, only the fastest units are recorded. It should also be remarked that at nodes with a large inventory level the unit has to wait longer for export than in less filled nodes. The reason is that first those units are exported which arrived earlier (FIFO - First In First Out).

5.2 Static Scenario

5.2.1 Equal Distribution

The easiest approach for the commodity distribution at each local controller is to weight all child nodes equally. Hence, for an agent n, the probability to choose one child node



(a) ST-EqD Throughput

(b) ST-EqD Delay and Communication Effort

(c) MTP-EqD Throughput

(d) MTP-EqD Delay and Communication Effort

Figure 5.3: *Static Scenario - Equal Distribution Policy*

out of its children is always given by the constant probability $1/|\Gamma^-(n)|$. This approach neither takes care of actual states of nodes' children nor of available transportation capacities. Therefore, it distributes the commodities equally among the network. In Fig. 5.3 we compare all results for both controller types. Here, the agents distribute their commodities with the *EqD* approach. As we expected while using the MTP controller, the network throughput converge to the networks theoretical upper bound, see Fig. 5.3(c). Whereas the ST controller converges towards a lower level than the theoretical upper bound, see Fig. 5.3(a). The reason for this seems to be that a ST controller easily delivers to blocked children with the same probability as to children with large local throughput. And because the controller has only one try to offer a unit successfully,the controller can not avoid bottlenecks. The result is that the theoretical maximum average throughput is not reached.

The transportation times of the commodities have the same order of magnitude for both controller types, see upper parts of Fig. 5.3(b) and of Fig. 5.3(d). A closer look shows that the commodity delay (transportation time) is a little bit smaller with the MTP controller than with the ST controller. In contrast to the MTP controller commodity, a commodity at the ST controller is offered just once and, if denied, it has to stay another time step. And obviously, the commodity of the MTP controller has additional chances to be transferred during that time step.

In principle, we expect a larger local communication effort for the MTP controller than for the ST controller. The MTP controller sends messages to its child nodes until all commodities are allocated or no customer is available anymore. The ST controller on the contrary just once sends an offer for each of its commodities. In the lower parts of Fig. 5.3(b) and of Fig. 5.3(d), the number of messages which are needed to negotiate completely the commodities transfer is plotted. The *EqD* optimisation does not need any global information and therefore no global messages either. The comprehension of the local message transfers shows that indeed the needed message frequency is slightly than larger for the MTP controller than the ST controller.

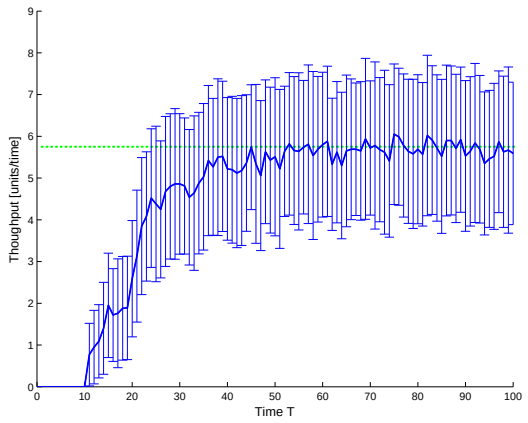
5.2.2 LAB

The LAB approach is a *Learning from Mistakes* [CB97] approach. In that respect, the agent is punished if it selects a neighbour which denies the commodity offer. Therefore, the agent learns to avoid blocking neighbours.

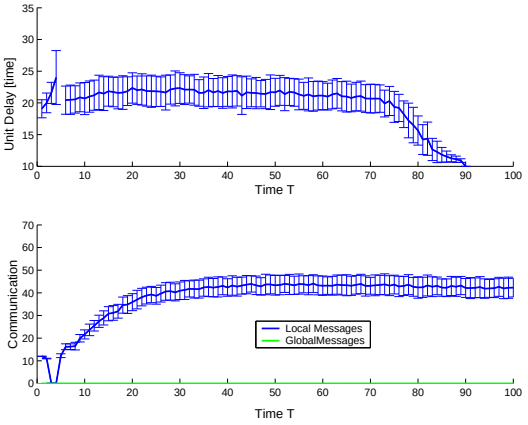
For the ST controller the current network throughput is shown in Fig. 5.4(a). The throughput of the MTP controller is shown in Fig. 5.4(c). As expected, this time, the actual throughput is better than the throughput of the *EqD* policy. The LAB approach avoids successful bottlenecks. We compare now the ST controller and the MTP controller: At first glance, there is no significant difference in throughput optimisation between them. For the test network, the throughput quality of MTP controllers is only up to 2% better than the controller type of section 4.3.1. But only the MTP controller guarantees the convergence towards the theoretical upper bound. In network topologies different than the test network (see for example the appendix) the ST controller converges to a lower level than the theoretical upper bound.

The LAB approach avoids bottlenecks. Blocked nodes are the bottleneck of network. They hold in a lot of cases commodities longer than other nodes. In that respect, it is not surprising that the transportation times of both controllers using LAB are smaller than the transportation times using the *EqD* approach. The times are shown in the upper parts of Fig. 5.4(b) and of Fig. 5.4(d). The comprehension of the transportation time with both controllers shows again no significant difference but a slightly slower delay when using the MTP controller.

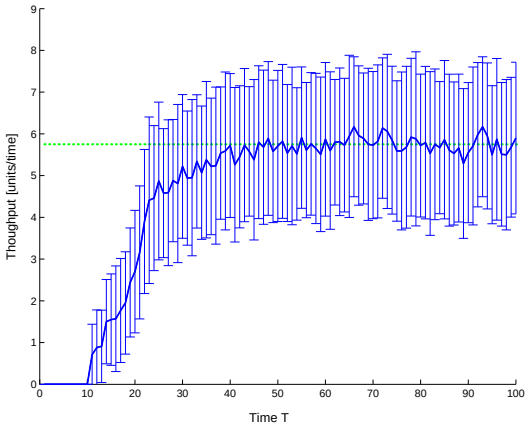
The LAB is a completely local approach. Thus, there is no global information exchange and the LAB controller sends only messages to its child nodes. In the lower parts of Fig. 5.4(b) and of Fig. 5.4(d), the communication efforts for both controller types are shown. Again, like in the previous discussed *EqD* approach, the message frequency (local) at the MTP controller seems to be slightly larger than at the ST controller but a



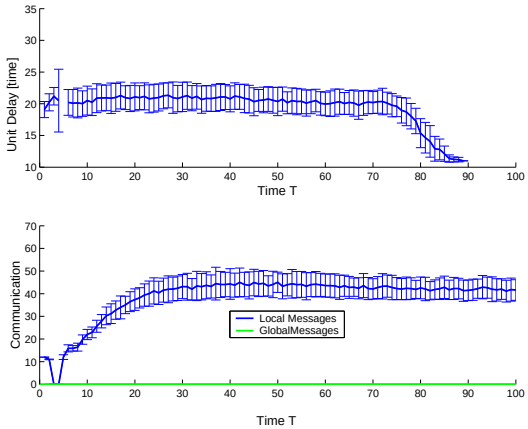
(a) ST-LAB Throughput



(b) ST-LAB Delay and Communication Effort



(c) MTP-LAB Throughput



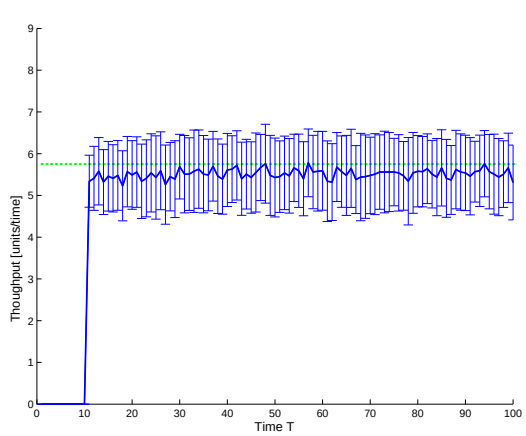
(d) MTP-LAB Delay and Communication Effort

Figure 5.4: *Static Scenario - Learning to Avoid Blocking*

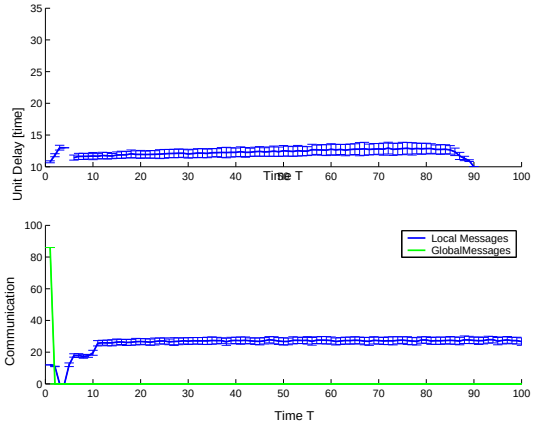
the differences are too small to distinguish them seriously.

5.2.3 Adaptive Link State Routing

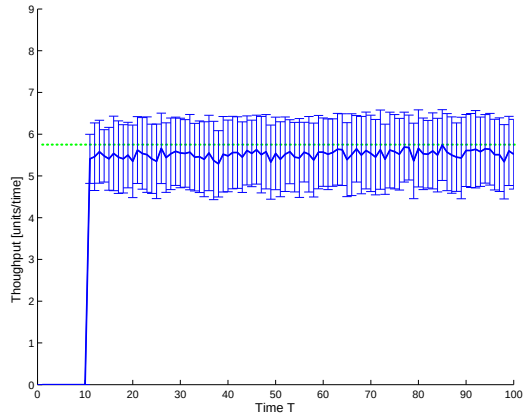
Link State approaches use global topology information to calculate new routing information. Thus, the agents have complete knowledge about all other agents' states and the network topology. The agent is informed each time the network changes by a link state advertisement, which is flooded throughout the network. With this new information the agent calculates its new routing policy.



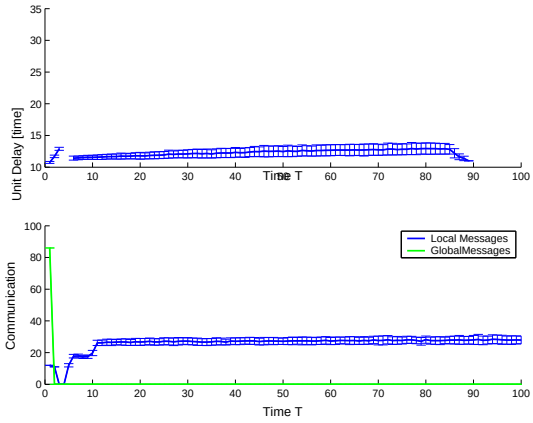
(a) ST-LSR Throughput



(b) ST-LSR Delay and Communication Effort



(c) MTP-LSR Throughput



(d) MTP-LSR Delay and Communication Effort

Figure 5.5: *Static Scenario - Adaptive Link State Routing*

In the figures Fig. 5.5(a) and Fig. 5.5(c) the actual throughputs of the Link State Routing approach for both controller types are shown. Obviously, the throughput converges for both controller types very fast. This is not astonishing if we remember that the LSR approach uses global topology information. The agents know very early their

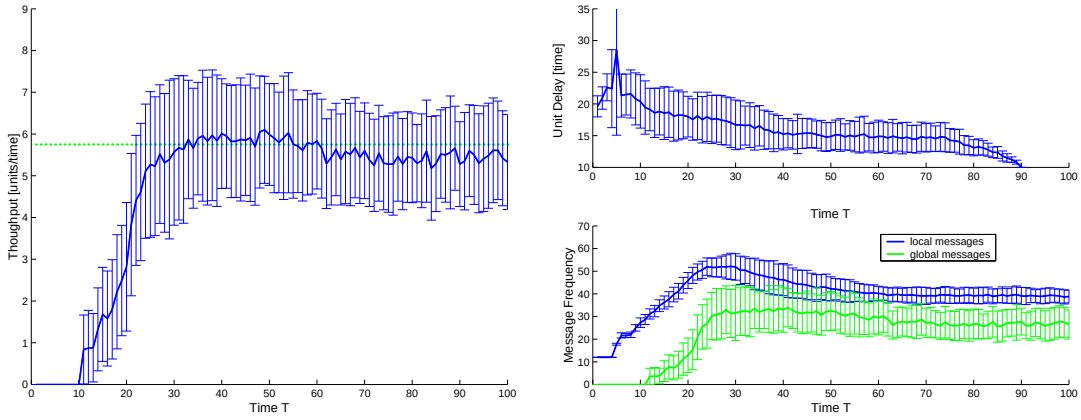
best distribution policy. But the throughput converges towards a slightly lower level than the theoretical upper bound. The reason is not the link state approach itself but the way of our implementation. Each agent has an internal map of the complete network topology. With the help of traditional maximum flow algorithms, the agent calculates a set of edge flows which are maximising the network throughput, see Section 4.1.2. The agents use this calculated edge-flows as entries in their internal action-state table D. The agents derive then, on the base of D their distribution policy. In that respect, the calculated edge-flows correspond with the average amount of commodity units which are sent via the edges towards its children. The calculated edge flows are zero if the graph theoretical tool has not used these edges to optimise the network throughput. Thus, the agents interpret this as an advice not to send commodities to children with a zero calculated edge-flow. But the preconditions for the MTP controller do not allow to avoid any children at all. The probability to choose a child node has to be larger than zero. Trivial, but unsuitable, is to add an infinitesimally small value ϵ to all weights which are equal to zero. The disadvantage of this solution is that the probability to choose this neighbour is very low. This would result in a huge communication effort because the MTP controller would send messages to already blocked neighbours and get rejections until the child is chosen with such a low probability. A better solution is to ignore those child nodes which have weight entries in D equal to zero. The drawback is that not as many units as possible are exported. The resulting actual average throughput is smaller than the theoretical bound. Nevertheless, we use this solution and accept reduced throughput.

The transportation times of the ST or the MTP controller are shown in the upper parts of Fig. 5.5(b) and of Fig. 5.5(d). The transportation times for commodities are small. With both controller types the transportation times are much smaller than in the previous approaches. The agent knows an optimal distribution solution from the very beginning of the simulation. The consequence is that the transportation times for the commodities are small.

The lower parts of Fig. 5.5(b) and of Fig. 5.5(d) show that the local communication efforts (dark curves) are small. This shows that the controllers only need a minimum number of *message interchanges* to negotiate the commodity exchange. The amount of messages needed to negotiate the commodity exchange depends on the efficiency of the negotiation and on the number of controllers and units participating the negotiation. LSR uses only a small number of controllers and distributes its units efficiently. However, in the beginning of the simulation, the controllers exchange network topology information throughout the network. This results in a large global message exchange which is shown by the grey curves. The small peak is the amount of messages needed to flood the link state advertisement throughout the network. In a static environment like this scenario, a link state advertisement has to spread only once throughout the network. But already for this short advertisement the communication effort increases strongly. The application of such techniques as multi-casting [Beh97] would help to reduce the scaling problem. But this kind of approach is not used here because it is not known how agents arrange suitably in a self-organised hierarchy. This lets us suggest that for highly dynamic scenarios such an approach becomes nearly unusable because of the needed communication effort.

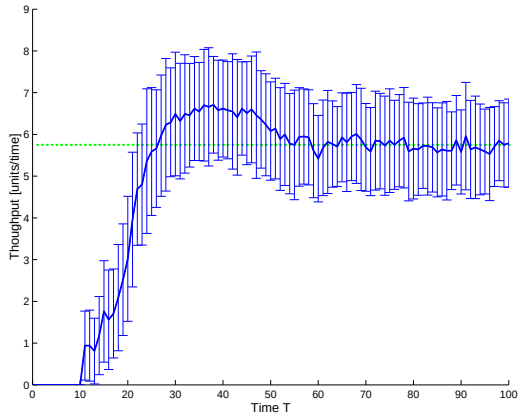
5.2.4 Reinforcement Routing

In the Reinforcement Routing (RR) approach the agent learns not only through the distance estimations of its neighbours but also through information about the actual times for the delivering of the commodities to their final destination. The RR approach uses less global information than the LSR but more than the LAB or EqD approach. For this, we expect better throughput results than with LAB or EqD but worse throughput results when using the LSR approach.

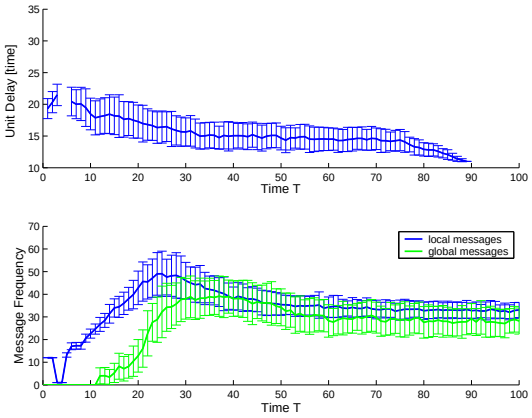


(a) ST-RR Throughput

(b) ST-RR Delay and Communication Effort



(c) MTP-RR Throughput



(d) MTP-RR Delay and Communication Effort

Figure 5.6: *Static Scenario - Reinforcement Routing*

In figures Fig. 5.6(a) and Fig. 5.6(c) the current throughputs for the ST and the MTP controllers are shown. Both controller types converge fast while the ST controller throughput converges towards a lower level than its theoretical upper bound. One reason might be that the RR approach does not take care of the fast changing inventory levels of the controllers' children. The controller's distribution policy is based on the transportation times of commodities to their sink. These transportation times correspond

with the inventory levels only indirectly and with a delay. The ST-RR controller has only one try to offer each of the controller's commodities. If the inventory levels of the controller's children are not taken into account, especially the complete filled levels, then offers are rejected. Thus, less commodity units are delivered and the current throughput converges towards a lower level than theoretically possible. Obviously, this problem can not occur using the MTP controller, because we have already proofed that each MTP Controller optimises the network throughput, see section 4.3.2. An interesting behaviour of the RR controller is also shown in Fig. 5.6(a) and in Fig. 5.6(c). The current throughput increases fast in the beginning and for a short time becomes larger than the theoretical upper bound. This behaviour can explained be by the RR controller design and the network topology: In the beginning, the inventory levels of all nodes are empty and the transportation times of the commodities depends only on the residence times of the components on their path towards sink. The controllers learn to choose neighbours with small residence times. This works well until the first inventory levels are filled up. Due to the network topology, those nodes stay filled up for definite time-steps. During these time-steps, the filled-up nodes can not import additional units. Thus, the parent nodes of those filled up nodes can not export as much as before and the consequence is that the complete current network throughput decrease until its average is lowerthan the theoretical upper bound again. The comprehension of both controller types shows that such an overlap is larger when using the MTP controller than the ST controller. If some following nodes are blocked, because filled up, the MTP controller chooses another node, if possible.

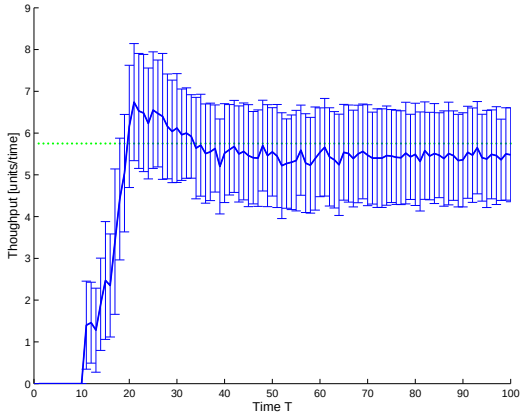
The previously discussed approaches are characterised by their use of local information only. This includes the Link State approach which has only a single global communication burst in the beginning of the simulation (static environment). The reinforcement approach (and the ant approach which we will discuss in the following subsection) uses global information, too. These messages are back-propagated from the sink implying transfer times from one node to another on their path to sink. Global communication stresses network communication resources more than local communication. Due to the difference of influence we plot them separately (see 5.6(b) and 5.6(d)).

Let us first have a closer look at the local communication. After the start-up period, the effort for the communication is at its maximum ($T \approx 20$). The controllers have not learned yet to distribute their commodities efficiently. Because of this, they need to distribute a lot of messages among their neighbours. Henceforth, the controllers learn to optimise their distribution and therefore the local communication effort converges to a lower level.

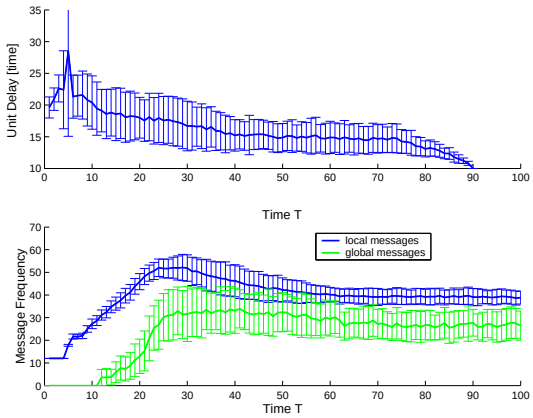
Controllers getglobal information only if previously sent commodities have already reached their destination (in our case always the sink). A response is sent back to the sender of the commodities. In the beginning, only a few commodities reach the sink and we count only a few number of global messages. Later, the global information converges to the maximum upper bound which corresponds to the network throughput. Now, the global communication is of significance for the total network communication load.

5.2.5 Ant Routing

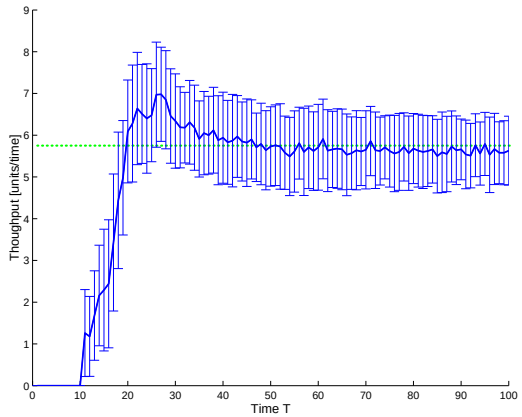
The ant routing approach measures the traffic density and the capacity of paths in the network. With that knowledge the agents identify bottlenecks towards their receiver. This knowledge does not only come from local information but also of information from nodes which are close to the network sink. The agent learns to weight its outgoing paths depending on the capacity of nodes which are closer to network sink.



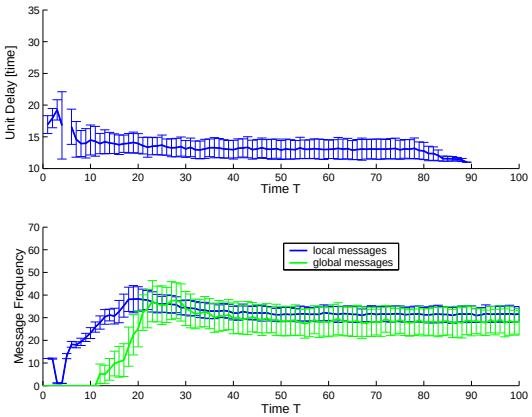
(a) ST-AR Throughput



(b) ST-AR Delay and Communication Effort



(c) MTP-AR Throughput



(d) MTP-AR Delay and Communication Effort

Figure 5.7: *Static Scenario - Ant Routing*

The reinforcement algorithm and the ant algorithm show not only similarities in architecture but also in behaviour. The throughput of the STP and the MTP controller are shown in Fig: (5.7(a)) and Fig. (5.7(c)). Both controller types converge fast while again the ST controller throughput converges towards a somewhat lower level than its theoretical upper bound. The reason is again the delay of information and the distribution failures of units. For short time again the average throughput becomes larger than the theoretical (average) upper bound . Until free slots are available in

adjacent nodes the controllers can exchange goods. If all slots are filled the controllers are forced to wait until new slots are opened again. This results in a decreasing average throughput again.

The communication efforts are shown in Fig. 5.7(b) and Fig. 5.7(d). Again, we distinguish between local and global communication. The local communication effort is smaller with the use of the MTP controller. Obviously, it is of advantage to distribute commodities carefully among neighbours. The global communication effort is now equally large with the local communication effort. Both are not comparable in their influence of the network though. Of course this assumption is only valid if communication is distributed like commodities in the network topology. A different communication structure like a communication bus in a ring structure would behave differently.

The reinforcement approach and the ant approach are successful in reducing the transportation times of commodities from source to sink. They explicitly optimise these delays.

5.3 Dynamic Scenario

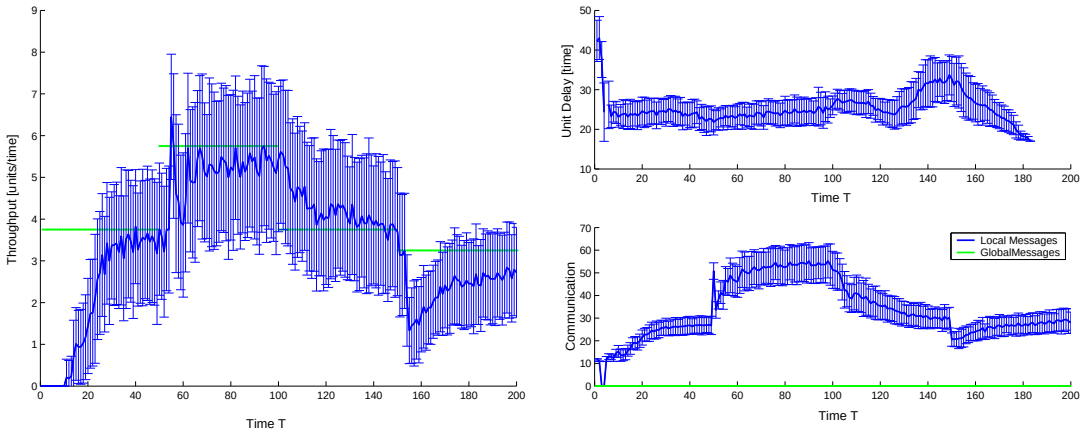
In modern facilities, components fail and transportation capacities break down unexpectedly. In that respect, the topology of a production network changes dynamically. The goal of this section is a comparison of the optimisation approaches in a dynamic scenario. They are tested with same network and same network dynamic which were introduced in chapter 4. Hence, the comparison is narrow but homogeneous.

In section 5.1, we defined an order of node failures. Such a sequence of fails is an example of an arbitrary dynamic system. This can be used as a sample scenario to test different approaches. For each approach, we analyse not only the throughput again but also the communication effort and the average transportation time of commodities in the network.

In the previous section we have seen the similar behaviour of STP and MTP controller. In this section we focus our interests on the MTP controller because this controller type guarantees optimal results.

5.3.1 Equal Distribution

In Fig. (5.8) we examine the results of the equal distribution policy in a dynamic scenario. On the right side, the actual throughput is shown again. As expected the equal distribution policy adapts the distribution in a way that the actual throughput converges to its theoretical bound. But in the case of increasing throughput the time between two node changes is too short to reach a good throughput quality possibilities. In the case of a reduced theoretical average throughput, the actual throughput decrease slowly towards its theoretical bound. The reason is that the buffers have been filled before the node failed. The stored commodities can exported if they have stayed at least the residence-time. Even if a node fails the controllers have inventory to export for a short time. This is a typical behaviour of a buffer. On the left side of the figure, the transportation



(a) MTP-EqD Throughput

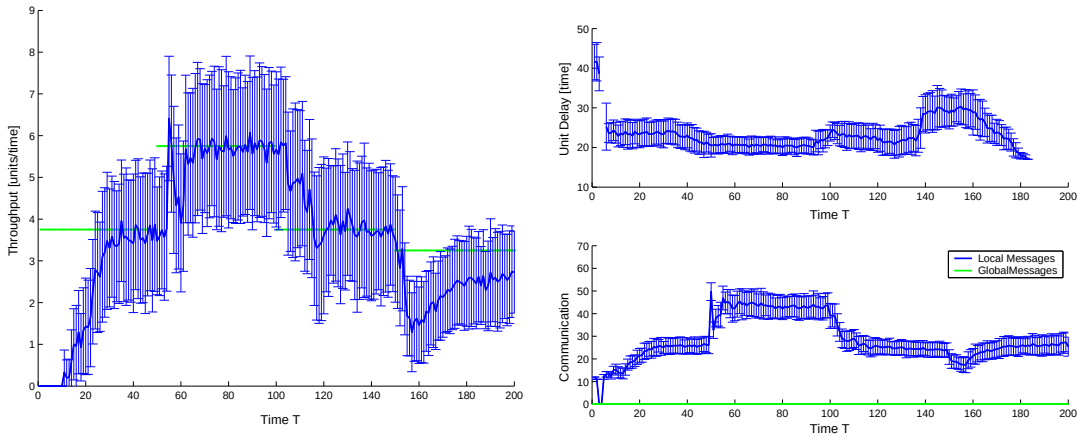
(b) MTP-EqD Delay and Communication Effort

Figure 5.8: *Dynamic Scenario - Equal Distribution Policy*

times for commodities and the communication effort of this approach are plotted. As we expect if more nodes are active, more communication is needed to distribute them. The failure of a node results in decreasing communication effort. Like the throughput, the communication effort shows again buffer effects. The communication is not reduced immediately in the case of a node fail but converges slowly to its lower bound. The reason is the same as in the case of throughput: The nodes buffer commodities in a way that even at failures enough units are available to proceed some time. As long as the buffers are filled, the negotiations proceed unchanged. Another interesting point is that the transportation time is nearly unimpressed of node changes during the first 150 time steps. The reactivation or fail of a node only influences the transportation time if the alternative nodes differ significantly in their residence times. Such a significant change is only shown in the last node change. Thus, the transportation time suddenly increases.

5.3.2 LAB

Fig. (5.9) shows actual average throughput as well as transportation times and communication effort in the case of the LAB approach. At a fist glance, there are not many differences between the dynamic behaviour of the equal distribution approach of Fig. (5.8) and this LAB approach. Especially the transportation times and communication efforts behave similar. But a more closer comprehension shows that the LAB approach needs less communication effort. By avoiding children which are usually blocked, the LAB approach needs less messages to negotiate its commodity distribution. Also the transportation times are shorter than the transportation times at the equal distribution policy. The reason is that the average inventory levels of all nodes are less filled than with the equal distribution policy. LAB chooses children with a large average local throughput. Those children have small residence times in a lot of cases. And because the controller



(a) MTP-LAB Throughput (b) MTP-LAB Delay and Communication Effort

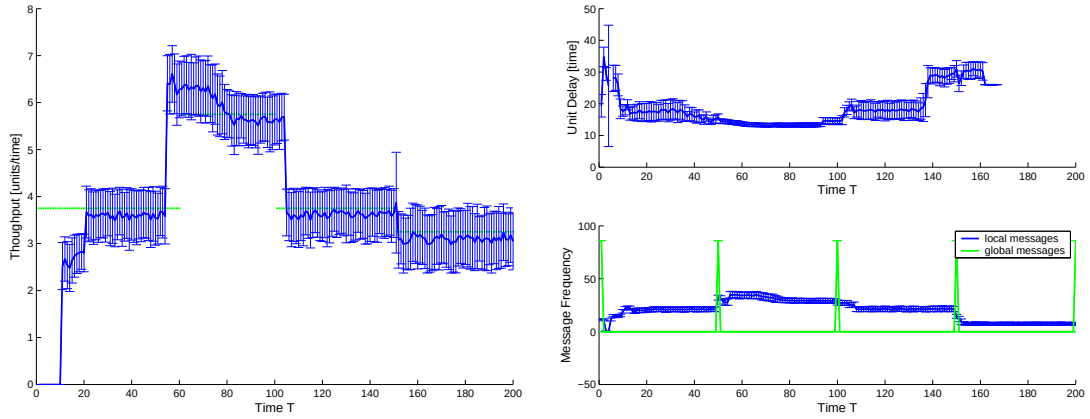
Figure 5.9: *Dynamic Scenario - Learning to Avoid Blocking*

chooses its commodities by age (FIFO), small inventory levels are an advantage for the transportation time of commodities. On the left side, the actual average throughput is plotted. The comprehension with the throughput of the equal distribution policy shows that the LAB approach adapts, as expected, faster than the equal distribution policy. In this example, the time between two node changes is long enough to converge with good quality to its theoretical bound.

5.3.3 Adaptive Link State Routing

In Fig.(5.10) the results of the examination for the link state approach are presented. On the left side we see again the actual average throughput and on the right side the transportation times for the commodities and the communication effort. As expected, the throughput when using the Link State approach converges very fast to its theoretical bound. The result is not surprising as the Link State approach uses the complete global topology and traffic knowledge to calculate the best commodity distribution. But even for those small test networks the computational effort drives the users computer equipment to long calculation periods. Even if we use for each agent its own processor (and memory), this approach will fail if the network size increases dramatically. Remember that each agent uses the complete network topology to find best distribution paths. For this the LSR approach behaves like a centralised approach. But LSR approach shows robustness due to network fails.

Another drawback is the broadcast mechanism. This is shown in the communication effort at the left side of Fig. (5.10). At each node change, the communication increases strongly because of the link state advertisements which are spread throughout the network. In more dynamic environments this approach would become impractical.



(a) MTP-LSR Throughput (b) MTP-LSR Delay and Communication Effort

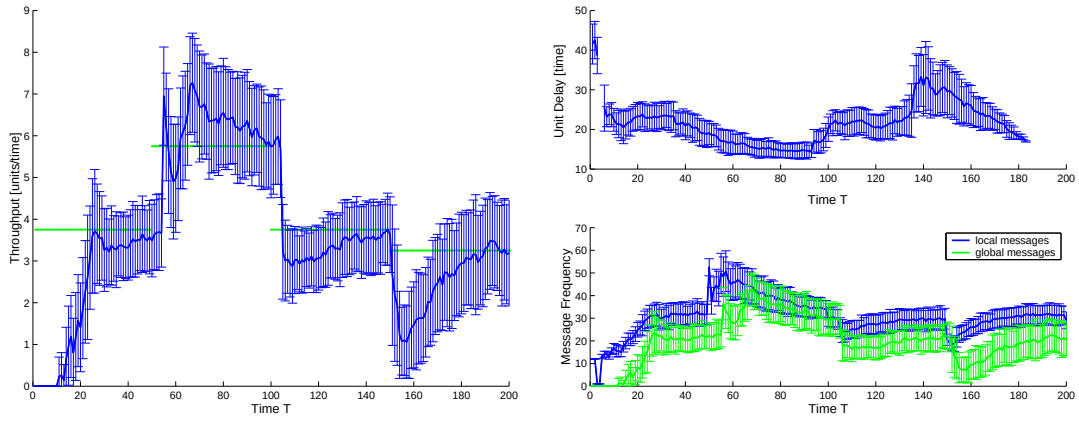
Figure 5.10: *Dynamic Scenario - Adaptive Link State Routing*

5.3.4 Reinforcement Routing

In Fig.(5.11), the results for the reinforcement approach are presented. The throughput is shown on the left side of the figure. For each network change, the approach reaches the theoretical bound within the 50 time steps. The influence of the topology change increases from the EqD approach to the LAB and the reinforcement approach. Each network failure (or recover) changes dramatically fast the current throughput, too. The reason for this is that learned systems are more specialised. What does this mean? Let us have a close look at a node which does not learn but distributes all its commodities equally among neighbours. This node does not recognise the failure of any neighbour and it still (tries to) deliver that neighbour. Here, the agent can not prefer faster paths along other neighbours. In a learned system, goods are distributed to those neighbours only which promise best transfer times. In the case of a node failure, the intelligent agent chooses its customer among those who were not failed. This increases the delivering time and the throughput.

5.3.5 Ant Routing

In Fig.(5.12) the results of the examination from the ant routing approach are presented. As expected, the results are quite similar to the results from the reinforcement approach. The most interesting aspect is that the ant approach shows again more effect on network changes than the reinforcement approach. Also, the ant approach shows faster adaptedness to the network state. This is shown for the commodity throughput in an faster converging towards the theoretical bound.



(a) MTP-RR Throughput

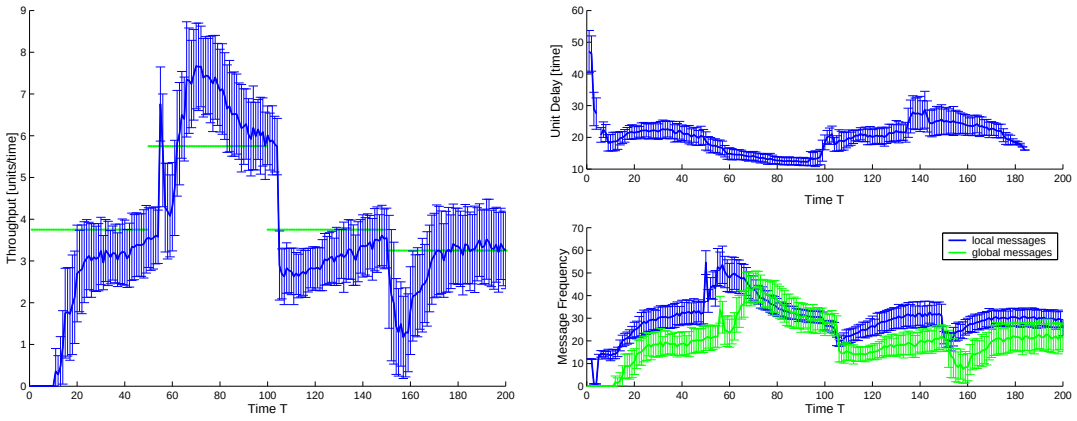
(b) MTP-RR Delay and Communication Effort

Figure 5.11: *Dynamic Scenario - Reinforcement Routing*

5.4 Conclusion

In this section we summarise the results of the previous examination. We used two kinds of controllers for our examination. The Single-Try-Policy (STP) controller and the Maximum-Throughput-Policy (MTP) controller. We have seen that the second controller, the MTP controller, guarantees an average maximum network throughput. In that sense, all approaches which used the MTP controller proofed convergence to their theoretical bound. The STP controller of section 4.3.1 can not guarantee maximum throughput for all networks and all kinds of approaches. But in a lot of cases (depending on size and complexity of the network), especially in acyclic directed networks and using suitable optimisation approaches, the STP controller results in good solutions. Due to this and depending on the problem it could be favourable to implement the STP controller. The most advantage of a STP controller is the simplicity of its implementation. The STP is much easier to implement than the MTP controller. But if maximum throughput is obligatory, the choice has to be the MTP controller.

We presented also five different optimisation approaches. In Fig. (5.13) we compared the approaches due to their learning speed in a static scenario. And in Fig. (5.14) we compare the approaches in the dynamic scenario. Not surprisingly, the Link State Routing proofs to be the fastest approach. But for this approach each agent holds a complete map of the network to calculate the optimal routing. In large and complicated networks, Link State Routing requires controllers with large computational power. Especially in dynamic environments, a huge number of messages has to be exchanged to update all controllers' knowledges. The problem of link state advertisements is a well known problem in recent literature. It has to be good considered to implement Link State approaches. In short: Quality vs. communication and computational effort. But we have seen that other approaches results also in fast adaptation without the drawback



(a) MTP-AR Throughput

(b) MTP-AR Delay and Communication Effort

Figure 5.12: *Dynamic Scenario - Ant Routing*

of network advertisement flooding or huge computational power at each controller. The ant-based approach and the reinforcement approach show fastest learning and adaptedness in dynamic environments. As both algorithms are similar (i.e. back-propagating agents, delay time measure) they show similar behaviour. They are faster than the LAB approach. But this is reasonable if we remember that only the LAB approach is strictly limited to local knowledge. Back-propagating mobile agents collect additional network state information which is used to distinguish neighbours. Such knowledge is not available for the LAB approach. A real fair comprehension would include same knowledge ranges for all approaches. All approaches have proven to optimise the network throughput. Depending on the purpose and the external constraints we can use one of the presented methods.

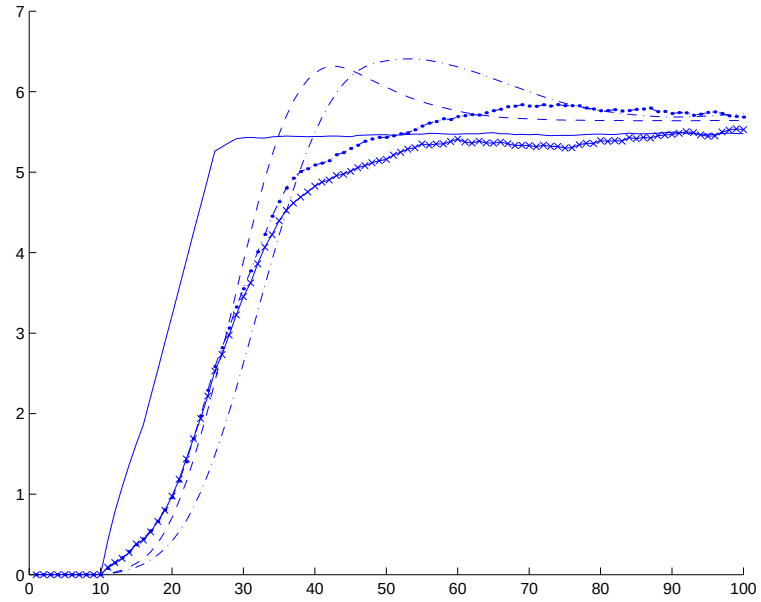


Figure 5.13: *Static Scenario-Throughput Comprehension* From left to right the different optimisation approaches are plotted: Link State Routing, Ant-based Routing, Reinforcement Routing, LAB and the non-learning approach of equal commodity distribution.

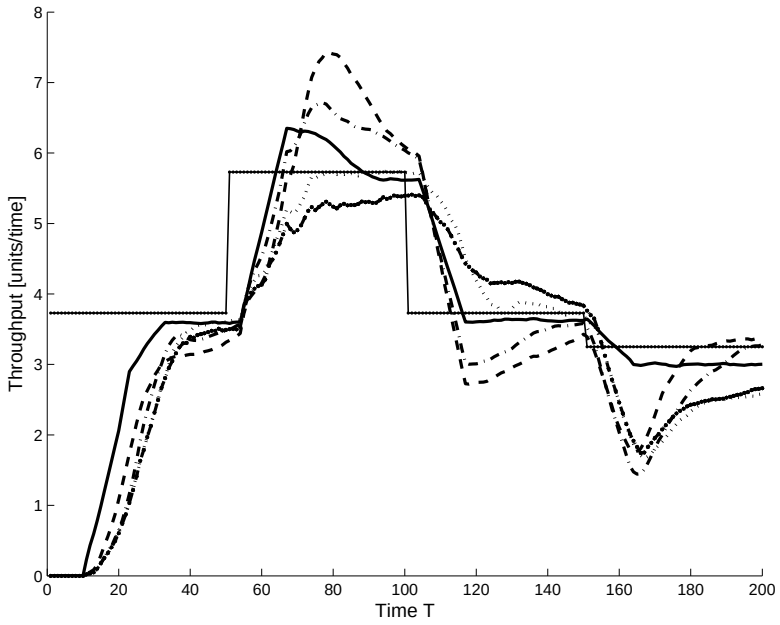


Figure 5.14: *Dynamic Scenario-Throughput Comprehension* From left to right the different optimisation approaches are plotted: Link State Routing, Ant-based Routing, Reinforcement Routing, LAB and the non-learning approach of equal commodity distribution.

Chapter 6

Summary and Outlook

The goal of this thesis was the decentralised throughput optimisation of production processes. This overall goal can be divided into three main goals:

1. The first goal was to indentify production process properties and to find a suitable representation of those properties as a modular concept for modelling purpose.
2. The second main goal was the description of a suitable decentralised controller architecture which fits the optimization problem.
3. And the third goal was the optimisation of the network throughput using the decentralised controller architecture combined with a realistic model of production processes that has been derived from the modular concept.

The first goal of this thesis was solved in chapter 3 by identifying suitable quantities which enable to construct, out of of a basic module, complex industrial network models. Industrial networks contain hundreds of different components like shelves or machines. Each of them has different tasks and properties. For large networks, it becomes very quickly impractical to design a representative module for each of those components. A clean design should be able to build complex components out of just a handful simple components which can be combined together. A better concept consists out of just a few basic modules. The problem is to identify properties in the industrial processes which are typical for all kinds of components and can still be combined at will. The concept which was introduced in this thesis consists out of just one basic module which contains only three properties. Each component in the production area has an upper bound for material which can be worked on simultaneously. Some components like shelves can work on several commodities simultaneously. They store them. This quantity we call *node-capacity*. Another property is the time a component spends at least on a commodity until it can be exported again. An example are machines which use incoming material. This quantity we call *residence time*. And the third quantity is the current number of commodities the component is working on simultaneously: the *inventory level*. All three quantities together are the properties of the basic module which can be combined using edges and their capacities introduced in traditional graph theory. The basic module

allows to construct modularly all kinds of component types which are typical for real production processes like shelves, machines or robots.

The second goal of this thesis was the design of a decentralised controller. An abstract design for such a controller in a production environment is presented in the second part of chapter 3. The specific design of a controller depends on its purpose. In that respect, the concrete controller design depends on the third goal of this thesis namely the throughput optimization. In this thesis, we presented two kinds of controller types. The first controller shows a clean design which allows simple implementation. It was shown that this controller optimizes network throughputs by using suitable optimisation approaches. A drawback of this controller is that it can not guarantee maximum throughput for any kind of distribution policy. Thus, there are networks which can not be optimised by using this controller type. Therefore, we developed an advanced controller type which guarantees maximum throughput independently of the distribution policies [1]: The MTP controller.

Although the MTP controller guarantees maximum throughput for all policies, the use of optimisation algorithms is not unimportant. Aspects like adaptation are still important properties. To measure the quality of different optimisation algorithms we introduced an approach to calculate the average theoretical throughput of the production process. For this, we mapped our production network to a traditional graph theory network. The maximum throughput problem can be solved then by traditional graph theory tools like the Ford-Fulkerson algorithm - like in this thesis - by an adaptation of the Karp and Edmonds algorithm presented in the appendix A. In this thesis, we presented several different optimisation algorithms: three learning algorithms and two non-learning optimisation algorithms. Some of them are new approaches and others are adaptations of recent works. We have shown that all of them are suitable to achieve the goal of throughput optimisation. An additional problem which occurs in such production systems is the oscillation in the current network throughput. In this thesis, a local mechanism was introduced which reduces efficiently the oscillation of the current network throughput.

All introduced concepts - the modular concept of section 3.1, the abstract controller architecture of section 3.2, the specific controller types of section 4.3 and the throughput optimisation algorithm of section 4.4 - represent a successful decentralised throughput optimisation of production networks. But there are also aspects which have not been investigated in this thesis. First of all, the approaches in this thesis are limited to single-commodity networks. It would be interesting to extend the processes of this thesis to multi-commodity networks.

[1] We have to remark that there are strategies which are no policies in our multi-agent understanding.

Appendix A

Maximum Flow Algorithm

In Fig. (A.2) we present a variant of the maximum flow algorithm of Edmonds and Karp [EK72] and Ford and Fulkerson [FF56, FF74]. The idea is as follows: Between a source s and a sink t exist different paths. If two paths have no node in common except their end node, we call them *disjoint* or *node-independent*. A network is *connected* if there is a path between every pair of nodes. A connected directed (acyclic) *multigraph* is shown on the left side of Fig. (A.1), that is a graph which has more than one edge between nodes.

The *connectivity* $\kappa'(G)$ is the smallest number of edges that when removed from a multigraph G disconnects source and sink. This is shown on the right side of the figure. Removing the grey coloured edges disconnects the multigraph. With the help of the Menger theorems [Men27] for node-independent paths, Douglas R. White and Frank Harary have rewritten the famous Ford-Fulkerson Theorem ([FF56]) [WH01]:

> *The maximum node-flow between any pair of nodes in a multigraph G equals the minimum number of edges in node-independent paths whose removal disconnects G.*

Using this, our multigraph of Fig. (A.1) has a maximum throughput of $f_{\text{thr}} = \kappa'(G) = 4$ if each edge has a single capacity of one. In the case of unit capacities the task of a maximum flow algorithm is to find the connectivity $\kappa'(G)$ of a network G. Each network G with rational capacities $c \in \mathbb{Q}$ could be described obviously as a unit multigraph

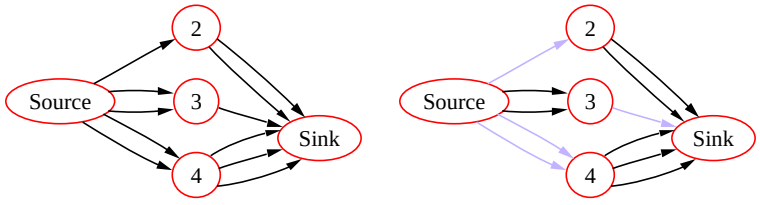


Figure A.1: Demonstration of the Ford-Fulkerson Theorem [FF56] rewritten by White and Harary [WH01]. The grey edges on the right side are the minimum cut of the multigraph.

```
1   function f = maxflow(𝒜, s, t)
2   f = 0;
3   con = 1;
4   while con,
5       A = 𝒜; A(A ≥ 1) = 1;
6       p = shortestpath(A, s, t);
7       if isempty(p),
8           con = 0;
9       else
10          for k=1:length(p)-1
11              c(k) = 𝒜(p(k),p(k+1));
12          end
13          R = min(c);
14          for k = 1:length(p)-1
15              𝒜(p(k),p(k+1)) = 𝒜(p(k),p(k+1)) - R;
16              𝒜(p(k+1),p(k)) = 𝒜(p(k+1),p(k)) + R;
17          end
18           f = f + R;
19      end
20  end
```

Figure A.2: Max-flow algorithm in language Matlab. This algorithm calculates the minimal set of edges whose removal disconnects the source s and sink t and that is equal to the maximum flow (Ford-Fulkerson Theorem).

network M. In this way a capacity $c = 3.1$ between two nodes in a non-multigraph could be represented as 31 edges connecting both nodes in a multigraph.

The idea of the algorithm is to find a node-independent path p from source to sink in the network G and to determine the connectivity $\kappa'(G_p)$ of this path. The connectivity is given by the minimal capacity of all capacities along p. As an example see the path from source to sink over node 4 in Fig. (A.1). The weakest connection is from source to node 4, as there are only two possible edges. In the corresponding multigraph M of G all connection sets (the set of edges connecting same nodes) are reduced by $\kappa'(G_p)$ edges. This disconnects completely the nodes of the weakest connection. A remapping also disconnects the path in G. This is done as long as no connecting path between source and sink in G is found anymore. The sum of all connectivities equals the maximum flow from source to sink in G. That corresponds to the Ford-Fulkerson Theorem.

Instead of running through all possible paths from source s to sink t we could better look for the shortest ones. The advantage is that it is more likely to find a small connectivity on longer paths than on short paths. To understand this let us take a very long and circuitous path. On this, we have lots of pairs of nodes and therefor a higher probability to find a very small set of edges connecting a pair. But this is a disadvantage because, in average, we now reduce less edges from the multigraph and we have to sum

connectivities in more loops as if we would choose always short or shortest paths from s to t. Those shortest path algorithms are also provided by graph theory. The most important shortest path algorithms are found in Johnson [Joh77] and Ahuja and Orlin [AMO93].

Fig. (A.2) shows the maximum flow algorithm in matlab language. The graph G is given by the *adjacency matrix* $A = A(G)$ which is a $n \times n$-dimensional matrix with elements $a_{ij} = 1$ if $[i, j] \in E$ and $a_{ij} = 0$ if $[i, j] \notin E$. The multigraph M is represented by the *weighted adjacency matrix* $\mathcal{A} = \mathcal{A}(M)$ which is given by the adjacency matrix but with elements $\mathcal{A}_{ij} = c_{ij}$ if $[i, j] \in E$. With the capacity c_{ij} of the edge between node i and the node j. In each loop, line 6 returns the current shortest path p in A. The connectivity of this path in $\mathcal{A}$ equals the number of currently used edges in the multigraph $\mathcal{A}$. If we remove these edges from $\mathcal{A}$ than we also remove at least one edge on the path in A. This is a kind of edge marking. Marked edges are not taken into account for shortest path search anymore. Until no more paths in A are found, the connectivities of all found paths of $\mathcal{A}$ are summed and updated like described above. The final result equals the maximum throughout of the graph.

This presented maximum flow algorithm is used for examining the theoretical maximum throughput of networks in industrial areas. It can be used also as decision support for a decentralised control mechanism which has to maximise total throughput.

Bibliography

[AMM01] Kaizar A. Armin, John T. Mayes, and Armin R. Mikler. *Agent-Based Distance Vector Routing*, volume 2164. Lectures Notes in Computer Science, 2001.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, Englewood Cliffs, NJ [u.a.], 1993.

[Aok71] M. Aoki. Some control problems associated with decentralized dynamic systems. *IEEE Trans. Aut. Control*, 16:515–516, 1971.

[Bat95] John W. Bates. Packet routing and reinforcement learning: Estimating shortest paths in dynamic graphs. Unpublished manuscript, 1995.

[BBV+01] H. Baumgaertel, S. Brueckner, H. Van Dyke Parunak, R. Vanderbok, and J. Wilke. Agent models of supply networks dynamics. In Terry Harrison, editor, *The Practice of Supply Chain Management*. Kluver, 2001.

[Beh97] J. Behrens. *Distributed Routing for Very Large Networks based on Link Vectors*. PhD thesis, University of California, Santa Cruz, may 1997.

[Bel58] Richard E. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[Ber82] D. P. Bertsekas. Distributed dynamic programming. *IEEE Trans. Automat. Control*, 27:610–616, 1982.

[BG87] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, 1987.

[BL94] Justin A. Boyan and Michael L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 671–678. Morgan Kaufmann Publishers, Inc., 1994.

[Bol01] Béla Bollobás. *Random graphs*. Cambridge University Press, Cambridge [u.a.], 2001.

[CB97] Dante R. Chialvo and Per Bak. Learning from mistakes. *Neurosience*, pages 1137–1148, 1997.

[CD97] G. Di Caro and M. Dorigo. AntNet:A mobile agents approach to adaptive routing. Technical report, IRIDIA, Université Libre de Bruxelles, 1997.

[Cle96] S. Clearwater, editor. *Market based control: a paradigm for distributed resource allocation*. World Scientific, 1996.

[CMM97] Anthony Chavez, Alexandros Moukas, and Pattie Maes. Challenger: A multi-agent system for distributed resource allocation. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 323–331, New York, 5–8, 1997. ACM Press.

[CRKGLA89] Chunhsiang Cheng, Ralph Riley, Srikanta P. R. Kumar, and J. J. Garcia-Luna-Aceves. A loop-free extended bellmann-ford routing protocol without bouncing effect. In *Proc. ACM SIGCOMM '89*, pages 224–236, Austin, TX, sep 1989.

[CY96] Samuel P. M. Choi and Dit-Yan Yeung. Predictive Q-routing: A memory-based reinforcement learning approach to adaptive traffic control. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 945–951. The MIT Press, 1996.

[DCG99] Marco Dorigo, Gianni Di Caro, and Luca M. Gambarella. Ant algorithms for discrete optimization. *Artificial Life, MIT Press*, 5(2), 1999.

[Die00] Reinhard Diestel. *Graph theory*. Number 173 in 2. Springer, New York, NY [u.a.], 2000.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[Din70] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Mathematics Doklady*, 11:1277–1280, 1970.

[Dor92] Marco Dorigo. *Optimzation, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Dipartimento di Elettronica e Informazione, 1992.

[DR02] G. Deco and E.T. Rolls. *Computational Neuroscience of Vision*. Oxford University Press, 2002.

[EK72] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *JACM*, 19(2):248–264, 1972.

[ERH92] Deborah Estrin, Yakov Rekhter, and Steven Hotz. Scalable inter-domain routing architecture. In *SIGCOMM*, pages 40–52, 1992.

[EWHPS96] T. Emden-Weinert, S. Hougardy, B. Kreuter H. J. Prömel, and A. Steger. Einführung in Graphen und Algorithmen. *Online Skript, Humbold Universität, Berlin*, 1996.

[FF56] L. R. Ford Jr. and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[FF74] Lester R. Ford and Delbert R. Fulkerson. Flows in networks. In *A Rand Corporation research study*, pages XII, 194. Princeton Univ. Press, Princeton, NJ, 1974.

[For68] J. W. Forrester. *Industrial Dynamics*. MIT Press, Cambridge, MA, 1968.

[Gol98] Andrew V. Goldberg. Recent developments in maximum flow algorithms. Technical report, NEC Research Institute, Inc., 1998.

[Goo99] Gerhard Goos. *Vorlesung über Informatik: Bd.2, Objektorientiertes Programmieren und Algorithmen*. Springer-Verlag, Berlin, Heidelberg, New York, 2 edition, 1999.

[GR97] A.V. Goldberg and S. Rao. Beyond the flow decomposition barrier. In *Proc. 38th IEEE Annual Symposium on Foundation of Computer Science*, pages 2–11, 1997.

[Haf97] M. Haft. *Selbstorganisiernede neurale Netze auf Grundlage von Informationsmaßen*. PhD thesis, Fakultät der Physik, Technische Universität München, München, 1997.

[Hen88] C. Hendrick. Routing information protocol; RFC1058. *Internet Request for Comments*, 1(1058), jun 1988.

[HSGK98] M. Heusse, Dominique Snyers, Sylvain Guérin, and Pascale Kuntz. Adaptive agent-driven routing and load balancing in communication networks. Rr-98001-iasc, ENST de Bretagne, BP 832, Brest Cedex, France, 1998.

[JB01] Paul A. Jensen and Jonathan F. Bard. Inventory theory. *Operations Research Models and Methods*, 2001.

[Joh77] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the Association for Computer Machinery*, 24(1):1–13, January 1977.

[JSW98] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[Kay01] Russel Kay. Supply chain management. www.computerworld.com, 2001.

[Koh89] T. Kohonen. *Self-Organization and Associative Memory*. Springer, Heidelberg, third edition, 1989.

[LB93] Michael Littman and Justin Boyan. A distributed reinforcement learning scheme for network routing. Technical Report CS-93-165, Carnegie Mellon University, School of Computer Science, 1993.

[Men27] Karl Menger. Zur Allgemeinen Kurventheorie. *Fundamenta Mathematicae*, 10:96–115, 1927.

[MMT00] Peter Marbach, Oliver Mihatsch, and John N. Tsitsiklis. Call Admission Control and Routing in Integrated Services Networks: Using Neuro-Dynamic Programming. In *IEEE Journal on Selected Areas in Communications*, volume 18, pages 197–208, 2000.

[Moy98] J. Moy. RFC 2328: OSPF version 2, 1998.

[PS98] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Englewood Cliffs, New Jersey, U.S.A., unabridged edition, may 1998.

[Rev01] Wendelin Reverey. Entwicklung einer Simulationsumgebung zur Optimierung von Logistikprozessen. Master's thesis, Universität Hannover and Technische Universität München and Siemens AG CT IC 4, sep 2001.

[San99] Thomas W. Sandholm. Distributed rational decision making. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, chapter 5, pages 201–258. The MIT Press, Cambridge, MA, USA, 1999.

[SAP02] SAP White Paper. Adaptive supply chain networks. 2002.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[Sim52] H. A. Simon. On the application of servomechanism theory in the study of production control. *ECONOMETRICA, Journal of the Econometric Society*, 20(2), 1952.

[Tan96] Andrew S. Tannenbaum. *Computer Networks*. Prentice-Hall, Inc, Upper Saddle River, NJ, 1996.

[VL99] Holger Voos and L. Litz. An new approach to optimal control using market-based algorithms. In *Proccedings of the European Control Conference ECC'99*, Karlsruhe, 1999.

[VSR98] H. Van Dyke Parunak, R. Savit, and R. L. Riolo. Agent-based modeling vs. equation-based modeling: A case study and users' guide. In Jaime S. Sichman, Rosaria Conte, and Nigel Gilbert, editors, *Proceedings of Multi-agent systems and Agent-based Simulation (MABS-98)*, volume 1534 of *LNAI*, pages 10–25, Berlin, 1998. Springer.

[VSR99] H. Van Dyke Parunak, Robert Savit, and Rick L. Riolo. DASCh: Dynamic analysis of supply chains, final report. Technical report, CEC,ERIM, 1999.

[WH01] Douglas R. White and Frank Harary. The cohesiveness of blocks in social networks: Node connectivity and conditional density. *Submitted to Sociological Methodology 2001*, 2001.

[WN01] Douglas R. White and M. E. J. Newman. Fast approximation algorithms for finding node-independent paths in networks. *SFI Working Papers*, 2001.

[Woo99] Michael Wooldridge. Intelligent agents. In Gerhard Weiss, editor, *Multi-agent Systems: A Modern Approach to Distributed Artificial Intelligence*, chapter 1, pages 27–78. The MIT Press, Cambridge, MA, USA, 1999.

[WS99] Gerhard Weiss and Sandip Sen. Learning in multiagent systems. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, chapter 6, pages 259–298. The MIT Press, Cambridge, MA, USA, 1999.