Ganzzahlige Programmierung und kernbasierte Lernverfahren für das Anschlusssicherungsproblem

Diplomarbeit

vorgelegt von

Christoph Jobmann

aus

Hamburg

angefertigt

im Institut für Numerische und Angewandte Mathematik

der Georg-August-Universität zu Göttingen

2008

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbsständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Göttingen, 23. Januar 2008

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Einleitung

Im täglichen Eisenbahnverkehr ist es bedauerlicherweise nicht immer möglich, Verspätungen zu vermeiden. Es besteht allerdings die Möglichkeit, daraus resultierende Folgeschäden in Form von verpassten Anschlüssen oder weiteren Verspätungen zu reduzieren. In der Praxis werden die notwendigen Entscheidungen üblicherweise basierend auf Erfahrungswerten und menschlicher Einschätzung getroffen.

Diese Arbeit beschreibt, auf welche Weise unter Zuhilfenahme mathematischer Modelle Computer verwendet werden können, um Empfehlungen für eine möglichst gute Entscheidung aussprechen zu können.

In Kapitel 1 wird das Anschlusssicherungsproblem vorgestellt, welches sich mit der Begrenzung der genannten Folgeschäden beschäftigt. Unter anderem wird ein aus der Modellierung resultierendes gemischt-ganzzahliges Programm eingeführt.

Verfahren zur Lösung des vorgestellten Programms werden in Kapitel 2 besprochen. Im Mittelpunkt steht dabei das Branch-and-Bound Verfahren zur Bestimmung der Optimallösung gemischt-ganzzahliger Programme. Dieses Lösungsverfahren hat allerdings den Nachteil, dass es im allgemeinen Fall eine sehr hohe Laufzeit aufweist. Da im Ernstfall schnell eine möglichst gute und nicht notwendig optimale Lösung benötigt wird, ist man auf Heuristiken angewiesen. Einige Heuristiken werden ebenfalls in Kapitel 2 vorgestellt. Diese Arbeit verfolgt unter anderem das Ziel, eine neuartige Heuristik zu entwickeln. Hierfür werden kernbasierte Lernverfahren eingesetzt, welche zunächst in Kapitel 3 kurz vorgestellt werden.

Die Anwendung dieser Heuristik auf das Anschlusssicherungsproblem wird in Kapitel 4 vorgestellt und untersucht. Anhand von Testdaten wird die Frage untersucht, ob Ansatz und Vorgehensweise sinnvoll sind oder nicht. Dies ist insbesondere aus dem Grund von Interesse, dass kaum Literatur zur Anwendung von kernbasierten Lernverfahren zur Analyse ganzzahliger Optimierungsprobleme vorliegt.

Für das Testen der Verfahren standen Fahrplandaten der Deutschen Bahn AG für einen Teil des Streckennetzes im Harzraum zur Verfügung. Die numerischen Ergebnisse wurden mit Hilfe eines C++ Programms unter Verwendung von MATLAB- und GLPK-Schnittstellen erzeugt. Einige grundlegende Konzepte des Programms und die verwendeten Schnittstellen werden in Kapitel 5 vorgestellt.

Numerische Ergebnisse für das Testgebiet sowie deren Visualisierung finden sich schließlich in Kapitel 6, welches zusätzlich diese Arbeit mit einer Gesamtbetrachtung abschließt.

Ergänzend findet man in Anhang A eine Auflistung der verwendeten Notation.

1 Anschlusssicherung

Das Anschlusssicherungsproblem wird in Abschnitt 1.1 vorgestellt. Dabei steht die Frage im Vordergrund, ob es unter Umständen sinnvoll ist, Anschlusszüge warten zu lassen, damit verspätete Reisende ihren Anschluss nicht verpassen und auf diese Weise ihr Ziel mit möglichst geringer Verspätung erreichen.

In Abschnitt 1.2 werden zwei Modelle vorgestellt, um die Fragestellung nach der besten Warten/Nichtwarten Entscheidung mit mathematischen Hilfsmitteln formalisieren zu können. Dies führt zu dem Begriff des *zulässigen modifizierten Fahrplans*, welcher eine Antwort auf vorhandene Quellverspätungen darstellen kann. Für beide Modelle werden Anforderungen in Form von linearen Ungleichungen vorgestellt, welche für solch einen Fahrplan erfüllt sein müssen.

Ausgehend von diesen linearen Ungleichungen als Nebenbedingungen ist es nur noch ein kleiner Schritt hin zu einem gemischt-ganzzahligen Programm, welches im Abschnitt 1.3 vorgestellt wird. Mit einer geeigneten Zielfunktion sollen hierdurch die Kundenverspätungen minimiert werden. Dieses Programm nimmt somit für das Anschlusssicherungsproblem eine zentrale Rolle ein.

Die Komplexität des Anschlusssicherungsproblems wird in Abschnitt 1.4 diskutiert. Dort wird darauf eingegangen, unter welchen Bedingungen das Problem schwer beziehungsweise leicht zu lösen ist.

Da im Allgemeinen nicht alle Züge von bestimmten Verspätungen betroffen sind, müssen auch nicht alle modellierten Fahrten in die Berechnungen eingehen, das Modell kann also reduziert werden. Abschnitt 1.5 befasst sich mit der Frage, wie jene Bereiche des betrachteten Modells aufgrund der aktuellen Verspätungs- und Entscheidungslage identifiziert werden können, welche tatsächlich für die weitere Berechnung von Bedeutung ist.

Grundlage und wesentliche Quelle für dieses Kapitel waren [26] und [25].

1.1 Das Anschlusssicherungsproblem

Verspätungen sind im Personentransport über die Schiene nicht immer vermeidbar: Oft reichen schon kleine Zwischenfälle wie zum Beispiel eine Signalstörung oder eine klemmende Fahrzeugtür, um eine signifikante Verspätung hervorzurufen. Dabei kann solch eine Verspätung unter Umständen durch schnelleres Fahren und verkürzte Haltezeiten in den jeweiligen Bahnhöfe reduziert, vielleicht sogar eliminiert werden. Trotzdem werden meistens zunächst einige Bahnhöfe verspätet angefahren. Dies ist ärgerlich für alle Fahrgäste, die auf die Nutzung des verspäteten Zuges angewiesen sind.

Besonders unangenehm ist es, wenn Fahrgäste in einem Bahnhof in einen weiteren Zug umsteigen wollen, dieser Anschluss aber aufgrund der Verspätung des ersten Zuges nicht ermöglicht werden kann, weil der Anschlusszug bei der Ankunft des verspäteten Zuges bereits abgefahren ist. In diesem Fall müssen diese Fahrgäste eine alternative Verbindung suchen oder auf den nächsten Anschlusszug warten. Bei umsichtiger Fahrplantaktung ist diese Wartezeit zum Glück üblicherweise begrenzt.

Eine Alternative kann sein, dass Anschlusszüge auf verspätete Züge warten. Dies hat den Nachteil, dass weitere Folgeverspätungen auftreten, welche wiederum andere Anschlusszüge beeinflussen und sich so die ursprüngliche Verspätung im gesamten Transportnetzwerk ausbreitet. In diesem Fall wären auch solche Fahrgäste von der Verspätung betroffen, die gar keinen Kontakt zu dem ursprünglich verspäteten Zug hatten.

Definition 1.1.1 *Für die Abfahrt oder Ankunft eines Fahrzeugs in einem Bahnhof trete durch einen bestimmten Vorfall zum ersten Mal eine Verspätung auf. Diese bezeichnet man als* **Quellverspätung** *oder auch* **Primärverspätung**.
Jede weitere Verspätung, welche nach Weiterfahrt oder Halten des Zuges beziehungsweise durch ermöglichte An-

schlüsse hervorgerufen wird, heißt **Folgeverspätung** *oder auch* **Sekundärverspätung**.

Definition 1.1.2 *Gegenstand des* **Anschlusssicherungsproblems** *ist die Frage, welche Kombination aus gehaltenen und verpassten Anschlüssen bei bekannten Quellverspätungen zu einer minimalen Benachteiligung der Kunden führt und auf welche Weise der vorgesehene Fahrplan angepasst werden muss.*

Wenn keine Quellverspätungen auftreten, ist das Problem besonders einfach[1], da in diesem Fall der vorgesehene Fahrplan ohne Abweichungen eingehalten werden kann. Das Verpassen eines Anschlusses muss noch nicht einmal angedacht werden.

Offen bleibt an dieser Stelle, nach welchen Gesichtspunkten zu optimieren ist, da sowohl gehaltene als auch verpasste Anschlüsse Vorteile und Nachteile mit sich bringen können. Halten aller Anschlüsse mag eine einfache, aber nicht notwendig gute Lösung darstellen. Gleiches gilt für das Verpassen aller Anschlüsse, die von einer Verspätung betroffen sind, das heißt bestmögliches Einhalten des Fahrplans.

Bemerkung 1.1.3 *Eine pünktliche Abfahrt des Anschlusszuges ist . . .*

- *gut für die Fahrgäste, welche ausschließlich den Anschlusszug nutzen, da für sie keine zusätzliche Verspätung entsteht,*
- *schlecht für jene Fahrgäste, die den verspäteten Zug nutzen und auf den Anschlusszug angewiesen sind, die so genannten* Umsteiger*, ihn aber verpassen.*

Wenn der Anschlusszug wartet, so dass der Anschluss gehalten wird, wobei allerdings eine Verspätung übertragen wird, ist . . .

- *gut für jene Fahrgäste, die den verspäteten Zug nutzen und in den Anschlusszug umsteigen,*
- *schlecht für all jene Fahrgäste, für die das Warten des Anschlusszuges eine zusätzliche Verspätung darstellt. Diese kann durch Ausbreitung von Folgeverspätungen auch an einem ganz anderen Bahnhof auftreten.*

Ein sinnvolles Ziel ist also die Bestimmung einer durchdachten Kombination aus Warten/Nichtwarten Entscheidungen. Hierzu kann das Problem als ein bikriterielles Optimierungsproblem betrachtet werden, bei dem eine Zielfunktion aus den Verspätungsminuten und eine aus den verpassten Anschlüssen resultiert. Abbildung 1.1 zeigt für ein examplarisches Verspätungsszenario die Menge der für das bikriterielle Problem zulässigen Lösungen, wobei jedem verpassten Anschluss das gleiche Gewicht zugeordnet wurde, ebenso jeder Verspätungsminute eines Zuges bei Ankunft und Abfahrt in einem Bahnhof. Jeder Datenpunkt repräsentiert eine Warten/Nichtwarten Entscheidung für das gesamte Netzwerk und die geringstmöglich resultierenden Folgeverspätungen für die selbe Menge von Quellverspätungen. Die durch ein $\diamond$ dargestellten Lösungen sind nichtdominiert, das heißt sie sind die besten Kandidaten für eine zu wählende Lösung, da alle weiteren Lösungen bei gleicher Verspätungsanzahl eine höhere Folgeverspätung beziehungsweise bei gleicher Folgeverspätung eine höhere Anzahl von verpassten Anschlüssen repräsentieren.

Da es jedoch sehr aufwendig ist, zunächst alle nichtdominierten Lösungen solch eines bikriteriellen Problems zu bestimmen und aus diesen eine besonders geeignete auszuwählen, ist es im Allgemeinen sinnvoller, sich auf eine Form von nichtdominierten Lösungen zu beschränken. Dies geschieht durch Untersuchung einer gewichteten Summe der einzelnen Zielfunktionswerte. Da Fahrgäste bei einem verpassten Anschluss auf den nächsten Zug im Fahrplantakt warten müssen, liegt die Wahl der Taktdauer als Gewicht für jeden verpassten Anschluss nahe, während die Verspätungsminuten einheitlich gewichtet werden. Dieser Ansatz führt zu dem Optimierungsproblem, welches ein zentraler Punkt dieses Kapitels und insbesondere von Abschnitt 1.3 ist. Eine ausführliche Behandlung des bikriteriellen Anschlusssicherungsproblems findet man in [27].

Zwar ist der Ansatz, die durchschnittlichen Kundenverspätungen zu minimieren, auch von sehr großem Interesse, da dies den Anfang des Weges zur besten Lösung aus Kundensicht darstellt. Allerdings werden hierfür Daten wie zum Beispiel Anzahlen der jeweiligen Umsteiger benötigt, die nicht immer gleich bleiben und zusätzlich umständlich zu erheben sind. Zusätzlich weist das resultierende Optimierungsproblem eine sehr viel komplexere Struktur auf, wie [26] zu entnehmen ist.

[1] Man könnte sogar sagen, es existiere gar kein Problem.

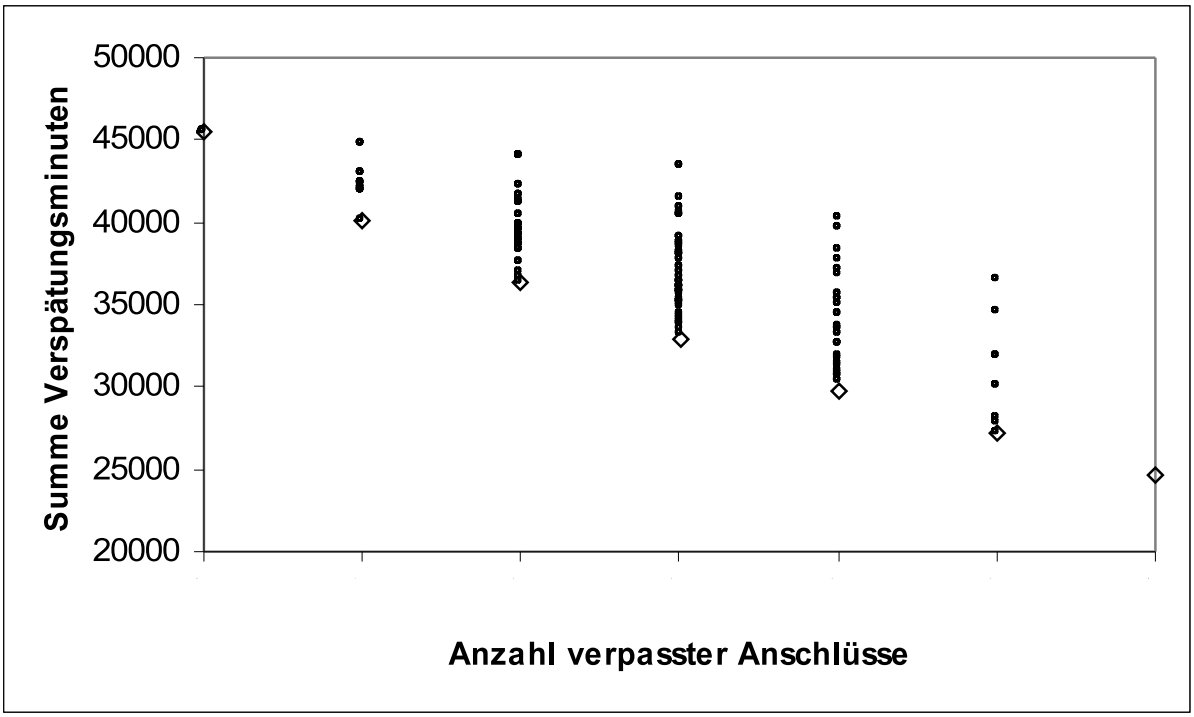


Abbildung 1.1: Lösungen für Anschlusssicherung als bikriterielles Problem

1.2 Modellierung des Anschlusssicherungsproblems

Dieser Abschnitt befasst sich mit der Betrachtung von Modellen zum Anschlusssicherungsproblem, welche jeweils zu einem gemischt-ganzzahligen Programm führen.

In Abschnitt 1.2.1 wird zunächst ein Modell vorgestellt, das sich an jenem Netzwerk orientiert, welches sich aus Bahnhöfen und Verbindungen zwischen diesen zusammensetzt. Derartige Netzwerke sind als Netzpläne verbreitet, eine Nutzung dieses Modells erfolgt daher besonders intuitiv.

Als nächster Schritt wird dieses Transportnetzwerk in Abschnitt 1.2.2 zu einem Ereignis-Aktivitäts-Netzwerk verfeinert, welches die einzelnen Zugverbindungen inklusive Umsteigemöglichkeiten in den Vordergrund stellt. Es ist weniger intuitiv, dafür aber eleganter als das transportnetzorientierte Modell. Außerdem eignet es sich besonders gut für rechnergestützte Lösungsansätze. Zusätzlich ist anzumerken, dass dieses Modell auch für allgemeinere Netzwerke verwendbar ist, sofern sie kreisfrei sind.

1.2.1 Transportnetzwerk

Die Orientierung an einem Netzwerk aus Bahnhofsverbindungen ist naheliegend, da dieses als Netzplan üblicherweise bereits vorhanden ist. Außerdem ist die Struktur weniger kompliziert als bei anderen Modellansätzen, so dass ein vertrauter Umgang auch ohne umfangreiche Einarbeitung möglich ist: Durch eine an die vorliegenden geographischen Verhältnisse angelehnte Darstellung können Bahnhöfe von Interesse schnell identifiziert werden. Problematisch ist allerdings, dass die zeitliche Struktur eines Fahrplans im Rahmen solch eines Bahnhofsverbindungsnetzwerks schwierig zu veranschaulichen ist.

Definition 1.2.1 (nach [26]) *Sei V eine Menge von Bahnhöfen und $E \subseteq V \times V$ eine Menge von ungerichteten Bahnhofsdirektverbindungen. Ein* **Transportnetzwerk** *ist ein ungerichteter Graph $TN = (V, E)$, das heißt ein*

Netzwerk mit Bahnhöfen als Knoten und Direktverbindungen als Kanten.

Notation 1.2.2 (nach [26]) *Sei für solch ein $TN = (V, E)$ eine Menge F von Fahrzeugen gegeben. Bezeichne für ein beliebiges Fahrzeug $g \in F$*

- *$V^g \subseteq V$ die Menge der Bahnhöfe, an denen Fahrzeug g hält und*
- *$E^g \subseteq V^g \times V^g$ die Menge der Direktverbindungen, welche das Fahrzeug g nutzt.*

Hat man in einem Bahnhof $v \in V$ die Möglichkeit, von Fahrzeug $g \in F$ in Fahrzeug $h \in F$ umzusteigen, so wird dieser Anschluss durch (g, h, v) beschrieben. Die Gesamtheit aller solcher Anschlüsse lässt sich als eine Menge $\mathcal{U} \subseteq F \times F \times V$ beschreiben.

Dabei kann angenommen werden, dass jedes Fahrzeug $g \in F$ in jedem Bahnhof höchstens einmal hält. Dafür kann unter Umständen eine zeitabhängige Darstellung der Bahnhöfe notwendig werden.

Definition 1.2.3 (nach [26]) *Ein* **Fahrplan $\mathbf{\Pi}$** *ist gegeben durch Zahlen $\Pi_{\mathrm{arr}_g^v}$, $\Pi_{\mathrm{dep}_g^v} \in \mathbb{N}_0$ für alle $g \in F$, $v \in V^g$, wobei gilt:*

- *$\Pi_{\mathrm{arr}_g^v}$ ist die geplante Ankunftszeit des Fahrzeugs g an Bahnhof v,*
- *$\Pi_{\mathrm{dep}_g^v}$ ist die geplante Abfahrtszeit des Fahrzeugs g von Bahnhof v.*

Ein Fahrplan $\mathbf{\Pi}$ heißt **zulässig**, *wenn die folgenden Bedingungen (1.1), (1.3) und (1.5) erfüllt sind.*

Dabei können die Zeitangaben des Fahrplans einheitlich wahlweise in Sekunden oder Minuten gemacht werden. Bei der Implementierung zum Anschlusssicherungsproblem wurde eine sekundengenaue Darstellung gewählt.

Wartebedingungen: Betrachte Fahrzeug $g \in F$ in Bahnhof $v \in V^g$.
Sei $L_g^v \in \mathbb{N}_0$ die vorgegebene Mindestwartezeit des Fahrzeugs g. Diese kann beispielsweise den Fahrgästen ermöglichen, das Fahrzeug zu betreten oder zu verlassen, oder sie stellt sicher, dass der Fahrer eine Pause machen kann. Die geplante Wartezeit des Fahrzeugs g in Bahnhof v ist $\Pi_{\mathrm{dep}_g^v} - \Pi_{\mathrm{arr}_g^v}$. An einen zulässigen Fahrplan ist daher die folgende Bedingung zu stellen:

$$\Pi_{\mathrm{dep}_g^v} - \Pi_{\mathrm{arr}_g^v} \geq L_g^v. \tag{1.1}$$

Durch Einplanung einer großzügigen Wartezeit kann sich eine Pufferzeit s_g^v ergeben:

$$s_g^v = \Pi_{\mathrm{dep}_g^v} - \Pi_{\mathrm{arr}_g^v} - L_g^v. \tag{1.2}$$

Fahrtbedingungen: Betrachte die Fahrt eines Fahrzeugs $g \in F$ von Bahnhof $u \in V^g$ nach $v \in V^g$.
Sei $L_g^{uv} \in \mathbb{N}_0$ die Zeit, die Fahrzeug g bei maximaler Fahrtgeschwindigkeit für die Fahrt von u nach v benötigt, die sogenannte Mindestfahrtzeit. Die geplante Fahrtzeit ist $\Pi_{\mathrm{arr}_g^v} - \Pi_{\mathrm{dep}_g^u}$. An einen zulässigen Fahrplan ist daher die folgende Bedingung zu stellen:

$$\Pi_{\mathrm{arr}_g^v} - \Pi_{\mathrm{dep}_g^u} \geq L_g^{uv}. \tag{1.3}$$

Durch Einplanung einer großzügigen Fahrtzeit kann sich eine Pufferzeit s_g^{uv} ergeben:

$$s_g^{uv} = \Pi_{\mathrm{arr}_g^v} - \Pi_{\mathrm{dep}_g^u} - L_g^{uv}. \tag{1.4}$$

Umsteigebedingungen: Betrachte schließlich einen Umsteigevorgang von Fahrzeug $g \in F$ nach Fahrzeug $h \in F$ im Bahnhof $v \in V^g \cap V^h$, das heißt einen Anschluss $(g, h, v) \in \mathcal{U}$.
Sei $L_{gh}^v \in \mathbb{N}_0$ die Mindestumsteigezeit, das heißt die Zeit, die für das Verlassen des Fahrzeugs g, das Aufsuchen des richtigen Bahnsteigs und das Betreten von Fahrzeug h als mindestens notwendig erachtet wird. Die geplante Umsteigezeit ist $\Pi_{\mathrm{dep}_h^v} - \Pi_{\mathrm{arr}_g^v}$. An einen zulässigen Fahrplan ist daher die Bedingung zu stellen:

$$\Pi_{\mathrm{dep}_h^v} - \Pi_{\mathrm{arr}_g^v} \geq L_{gh}^v. \tag{1.5}$$

Durch Einplanung einer großzügigen Umsteigezeit kann sich eine Pufferzeit s_{gh}^v ergeben:

$$s_{gh}^v = \Pi_{\mathrm{dep}_h^v} - \Pi_{\mathrm{arr}_g^v} - L_{gh}^v. \tag{1.6}$$

Aus der Ganzzahligkeit von Angaben zu Fahrplanzeiten, Mindestwartezeit, -fahrtzeit und -umsteigezeit ergibt sich, dass auch die eingeführten Pufferzeiten ganzzahlig sind.

Bisher wurden noch keine Verspätungen berücksichtigt. Bezeichne nun für ein Fahrzeug $g \in F$ und einen Bahnhof $v \in V^g$ eine Quellverspätung mit $d_g^v \in \mathbb{N}_0$. Die Menge aller verspäteten Ankünfte[2] wird bezeichnet mit

$$\mathcal{E}_{\mathrm{del}} = \{(g, v) : d_g^v > 0\}.$$

Dabei gelte die Konvention $d_g^v = 0$ für $(g, v) \notin \mathcal{E}_{\mathrm{del}}$.

Angenommen, ein Fahrzeug $g \in F$ erreicht Bahnhof $v \in V^g$ mit Quellverspätung $d_g^v > 0$. Dann würde eine einfache Erhöhung des Wertes ${\Pi_{\mathrm{arr}}}_g^v$ um d_g^v im Allgemeinen zu einem nicht mehr zulässigen Fahrplan führen. Man ist daher neben einer *Warten/Nichtwarten Entscheidung* für alle Anschlüsse an einem *modifizierten Fahrplan* $\mathbf{x}$, bestehend aus Angaben ${x_{\mathrm{arr}}}_g^v$, ${x_{\mathrm{dep}}}_g^v$ für alle $g \in F$, $v \in V^g$, interessiert, welcher bezüglich der betrachteten Zielfunktion die besten Eigenschaften aufweist. Dabei ist anzumerken, dass ein modifizierter Fahrplan dann zulässig ist, wenn Wartebedingungen analog zu (1.1) und Fahrtbedingungen analog zu (1.3) erfüllt sind, das heißt die Verspätung eines Fahrzeugs wird auf den jeweils nächsten Bahnhof korrekt übertragen. Da an dieser Stelle nicht explizit gefordert wird, dass Anschlusszüge auf möglicherweise verspätete Züge warten, müssen Umsteigebedingungen analog zu (1.5) in einem zulässigen modifizierten Fahrplan nicht notwendig erfüllt sein.

Definition 1.2.4 (nach [26]) *Bei vorgegebenem Fahrplan* $\mathbf{\Pi}$ *und einer Menge von Quellverspätungen* $d_v^g \geq 0$ *mit* $v \in V$, $g \in F$ *heißt ein* **modifizierter Fahrplan** $\mathbf{x}$ **zulässig**, *wenn die folgenden Bedingungen gelten:*

$$\begin{aligned}
{x_{\mathrm{arr}}}_g^v &\geq {\Pi_{\mathrm{arr}}}_g^v + d_g^v \quad \forall g \in F,\ v \in V^g, && (1.7)\\
{x_{\mathrm{dep}}}_g^v &\geq {\Pi_{\mathrm{dep}}}_g^v \quad \forall g \in F,\ v \in V^g, && (1.8)\\
{x_{\mathrm{dep}}}_g^v - {x_{\mathrm{arr}}}_g^v &\geq L_g^v \quad \forall g \in F,\ v \in V^g, && (1.9)\\
{x_{\mathrm{arr}}}_g^v - {x_{\mathrm{dep}}}_g^u &\geq L_g^{uv} \quad \forall g \in F,\ (u, v) \in E^g. && (1.10)
\end{aligned}$$

Durch die Bedingungen (1.7) und (1.8) wird sichergestellt, dass der modifizierte Fahrplan auf dem ursprünglichen aufbaut und dass Quellverspätungen berücksichtigt werden. Bedingung (1.9) sorgt dafür, dass ein Fahrzeug, das einen Bahnhof verspätet erreicht, diesen auch mit dieser Verspätung verlässt. Bedingung (1.10) schließlich gewährleistet, dass ein Fahrzeug, das einen Bahnhof mit einer Verspätung verlässt, diese auch bei der Ankunft im nächsten Bahnhof hat. Die transferierte Verspätung kann sich dabei maximal um die jeweilige Pufferzeit reduzieren. (1.9) und (1.10) ergeben sich aus (1.1) beziehungsweise (1.3), angewendet auf $\mathbf{x}$ statt auf $\mathbf{\Pi}$.

Bemerkung 1.2.5 *Nach Definition 1.2.4 breiten sich Folgeverspätungen eines Zuges ausschließlich von einem Bahnhof zum jeweils nächsten folgenden sowie innerhalb eines Bahnhofs auf die Anschlusszüge aus. Dies entspricht einer Ausbreitung entlang verschiedener Pfade im Transportnetzwerk, die sich zu Bäumen zusammenfassen lassen.*

Wie bereits erwähnt wurde, ist das Aufrechterhalten von Anschlüssen nicht notwendig, um einen zulässigen modifizierten Fahrplan zu erhalten. Nun wird betrachtet, wie man Warten/Nichtwarten Entscheidungen in die Anforderungen an einen zulässigen modifizierten Fahrplan aufnehmen kann.

Definition 1.2.6 (nach [26]) *Ein Anschluss* $(g, h, v) \in \mathcal{U}$ *in einem Bahnhof* $v \in V$ *gilt als* **gehalten**, *wenn Fahrzeug* $h \in F$ *auf Fahrzeug* $g \in F$ *wartet, das heißt wenn*

$${x_{\mathrm{dep}}}_h^v - {x_{\mathrm{arr}}}_g^v \geq L_{gh}^v.$$

Ansonsten gilt der Anschluss als **verpasst**.

Um die zulässigen Lösungen des Anschlusssicherungsproblems beschreiben zu können, muss nun noch ein Zusammenhang zwischen Warten/Nichtwarten Entscheidungen und dem modifizierten Fahrplan hergestellt werden. Hierzu werden für $g, h \in F$, $v \in V$ weitere Variablen ${y_{\mathrm{dep}}}_g^v$, ${y_{\mathrm{arr}}}_g^v$ und z_{ghv}, in Vektorschreibweise $\mathbf{y}$ und $\mathbf{z}$, definiert:

$$\begin{aligned}
{y_{\mathrm{dep}}}_g^v &= \text{Verspätung, mit der Fahrzeug } g \text{ Bahnhof } v \text{ verlässt,}\\
{y_{\mathrm{arr}}}_g^v &= \text{Verspätung, mit der Fahrzeug } g \text{ Bahnhof } v \text{ erreicht,}\\
z_{ghv} &= \begin{cases} 0 & \text{wenn Anschluss } (g, h, v) \in \mathcal{U} \text{ gehalten wird,}\\ 1 & \text{wenn Anschluss } (g, h, v) \in \mathcal{U} \text{ verpasst wird.} \end{cases}
\end{aligned}$$

[2] Es ist zwar grundsätzlich möglich, auch für Abfahrten Quellverspätungen zuzulassen, dies würde aber eine umständlichere Notation erfordern, worauf an dieser Stelle verzichtet wird. Eine quellverspätete Abfahrt lässt sich schließlich ohne wesentliche Abstriche auch als verspätete Ankunft am Ende der ausgehenden Fahrkante darstellen.

Bei den Verspätungen ist es hier unerheblich, ob es sich um Quell- oder Folgeverspätungen handelt.

In vielen Fällen ist es praktischer, direkt mit den Verspätungsvariablen anstatt den modifizierten Fahrplanangaben zu arbeiten. Für $g \in F$, $v \in V^g$ bestehen dann offensichtlich die Beziehungen

$$\begin{aligned} x_{\mathrm{dep}_g^v} &= \Pi_{\mathrm{dep}_g^v} + y_{\mathrm{dep}_g^v}, \\ x_{\mathrm{arr}_g^v} &= \Pi_{\mathrm{arr}_g^v} + y_{\mathrm{arr}_g^v}. \end{aligned}$$

Mit Hilfe der neuen Verspätungsvariablen y lässt sich Definition 1.2.6 so umformulieren, dass ein Anschluss $(g, h, v) \in \mathcal{U}$ als **gehalten** gilt, wenn

$$y_{\mathrm{arr}_g^v} - y_{\mathrm{dep}_h^v} \leq s_{gh}^v. \tag{1.11}$$

Satz 1.2.7 (nach [26]) *Sei $M \in \mathbb{N}_0$ ein hinreichend großer Parameter. Dann ist $(\mathbf{y}, \mathbf{z})$ genau dann zulässig, wenn die folgenden Bedingungen erfüllt sind:*

$$\begin{aligned} y_{\mathrm{arr}_g^v} &\geq d_g^v \quad \forall (g,v) \in \mathcal{E}_{del}, && (1.12)\\ y_{\mathrm{arr}_g^v} - y_{\mathrm{dep}_g^v} &\leq s_g^v \quad \forall g \in F,\ v \in V^g, && (1.13)\\ y_{\mathrm{dep}_g^u} - y_{\mathrm{arr}_g^v} &\leq s_g^{uv} \quad \forall v,u \in V^g,\ g \in F : (u,v) \in E^g, && (1.14)\\ -Mz_{ghv} + y_{\mathrm{arr}_g^v} - y_{\mathrm{dep}_h^v} &\leq s_{gh}^v \quad \forall (g,h,v) \in \mathcal{U}, && (1.15)\\ y_{\mathrm{dep}_g^v}, y_{\mathrm{arr}_g^v} &\in \mathbb{N}_0 \quad \forall g \in F,\ v \in V^g, \\ z_{ghv} &\in \{0,1\} \quad \forall (g,h,v) \in \mathcal{U}. \end{aligned}$$

Durch die Bedingungen (1.12) wird sichergestellt, dass alle Quellverspätungen berücksichtigt werden. An dieser Stelle ist noch einmal anzumerken, dass das Anschlusssicherungsproblem trivial ist, wenn keine Quellverspätungen $d_g^v > 0$ vorliegen: In diesem Fall sind die oben genannten Bedingungen erfüllt, wenn keine zusätzlichen Verspätungen und keine verpassten Anschlüsse auftreten, das heißt $y_{\mathrm{arr}_g^v} = y_{\mathrm{dep}_g^v} = 0\ \forall g \in F, v \in V^g$ und $z_{ghv} = 0\ \forall (g,h,v) \in \mathcal{U}$. Zusammen mit $y_{\mathrm{arr}}, y_{\mathrm{dep}} \geq 0$ ist (1.12) äquivalent zu (1.7) und (1.8).

Bedingungen (1.13) und (1.14) stellen eine Umformulierung der Bedingungen (1.9) und (1.10) unter Verwendung der Verspätungsvariablen und Pufferzeiten dar. Bedingung (1.15) schließlich stellt sicher, dass bei jedem gehaltenen Anschluss Verspätungen korrekt transferiert werden und sich bestenfalls um die vorgesehene Pufferzeit verringern können.

Der Parameter M muss hinreichend groß gewählt werden, so dass jede resultierende Folgeverspätung kompensiert werden kann. Dabei ist allerdings zu beachten, dass ein zu großer Wert unter Umständen dazu beitragen kann, dass einzelne Lösungsverfahren weniger effizient sind. Wie jedoch später (siehe Bemerkung 1.2.16) noch hervorgehoben werden wird, können Verspätungen sich ohne Einfluss anderer Verspätungen nicht weiter verschlechtern. Daher ist die Wahl $M = \max\limits_{g \in F, v \in V} d_g^v$ ausreichend.

1.2.2 Ereignis-Aktivitäts-Netzwerk

In diesem Abschnitt wird ein Modell vorgestellt, welches ein Ereignis-Aktivitäts-Netzwerk verwendet. Dies ist eine Technik, die üblicherweise im Bereich der Projektplanung Anwendung findet. Auf geeignete Weise werden *Ereignisse* analog zu den Knoten eines Graphs definiert, welche durch Kanten in Form von geeigneten *Aktivitäten* verbunden sind. Durch dieses Modell ist es möglich, die den Fahrplan betreffenden Größen alternativ zu strukturieren und weiter gehende Analysen auf diese Weise zu erleichtern.

Grundlage des Modells sind je eine Menge von Bahnhöfen V und Fahrzeugen F, welche die Bahnhöfe nach einem mit Ankunfts- und Abfahrtszeiten vorgegebenen Fahrplan $\mathbf{\Pi}$ anfahren, sowie Angaben zu Minimaldauer in Form von L_g^v, L_g^{uv} und L_{gh}^v für Fahrzeuge $g, h \in F$ und Bahnhöfe $u, v \in V$.

Notation 1.2.8 (nach [26]) *Jede* **Ankunft** *eines Fahrzeugs $g \in F$ in einem Bahnhof $v \in V^g$ wird als ein Ereignis aufgefasst. Die Gesamtheit aller* **Ankunftsereignisse** *wird zusammengefasst in der Menge*

$$\mathcal{E}_{\mathrm{arr}} = \{(g, v, \mathrm{arr}) : \textit{Fahrzeug } g \in F \textit{ erreicht Bahnhof } v \in V^g\}.$$

Zusätzlich wird jede **Abfahrt** *eines Fahrzeugs $g \in F$ von einem Bahnhof $v \in V^g$ als ein Ereignis aufgefasst. Die Gesamtheit aller* **Abfahrtsereignisse** *wird zusammengefasst in der Menge*

$$\mathcal{E}_{\mathrm{dep}} = \{(g, v, \mathrm{dep}) : \textit{Fahrzeug } g \in F \textit{ verlässt Bahnhof } v \in V^g\}.$$

Die Gesamtheit aller Ereignisse wird mit

$$\mathcal{E} = \mathcal{E}_{\mathrm{arr}} \cup \mathcal{E}_{\mathrm{dep}}$$

bezeichnet. Diese Ereignisse können als Knoten eines Graphen angesehen werden. Für jedes Ereignis $e \in \mathcal{E}$ ist die geplante Zeit des Eintretens gegeben durch

$$\Pi_e = \begin{cases} \Pi_{\mathrm{arr}}{}_g^v & \textit{falls } e = (g, v, \mathrm{arr}) \in \mathcal{E}_{\mathrm{arr}}, \\ \Pi_{\mathrm{dep}}{}_g^v & \textit{falls } e = (g, v, \mathrm{dep}) \in \mathcal{E}_{\mathrm{dep}}. \end{cases}$$

Bezeichne außerdem für jedes Ereignis $e \in \mathcal{E}$ die Variable d_e seine Quellverspätung, das heißt $d_e = d_g^v$ falls $e = (g, v, \mathrm{arr}) \in \mathcal{E}_{\mathrm{arr}}$ und 0 sonst.

Notation 1.2.9 (nach [26]) *Die* **Fahrt** *eines Fahrzeugs $g \in F$ von Bahnhof $u \in V^g$ nach $v \in V^g$ ohne Zwischenhalt wird als eine Aktivität a aufgefasst. Sie beginnt mit dem Abfahrtsereignis $e_1 = (g, u, \mathrm{dep}) \in \mathcal{E}_{\mathrm{dep}}$ und endet mit dem Ankunftsereignis $e_2 = (g, v, \mathrm{arr}) \in \mathcal{E}_{\mathrm{arr}}$. Die geplante Dauer der Fahrt ist gegeben durch $\Pi_{e_2} - \Pi_{e_1}$, die Mindestdauer durch $L_a = L_g^{uv}$. Die Gesamtheit aller* **Fahrtaktivitäten** *wird zusammengefasst in der Menge*

$$\begin{aligned} \mathcal{A}_{\mathrm{drive}} \quad = \quad & \{((g, u, \mathrm{dep}), (g, v, \mathrm{arr})) \in \mathcal{E}_{\mathrm{dep}} \times \mathcal{E}_{\mathrm{arr}} : \textit{Fahrzeug } g \in F \textit{ fährt direkt} \\ & \textit{von Bahnhof } u \in V^g \textit{ nach } v \in V^g\}. \end{aligned}$$

Auch das **Warten** *eines Fahrzeugs $g \in F$ im Bahnhof $v \in V^g$ wird als eine Aktivität a aufgefasst. Sie beginnt mit dem Ankunftsereignis $e_1 = (g, v, \mathrm{arr}) \in \mathcal{E}_{\mathrm{arr}}$ und endet mit dem Abfahrtsereignis $e_2 = (g, v, \mathrm{dep}) \in \mathcal{E}_{\mathrm{dep}}$. Die geplante Wartedauer ist gegeben durch $\Pi_{e_2} - \Pi_{e_1}$, die Mindestdauer durch $L_a = L_g^v$. Die Gesamtheit aller* **Warteaktivitäten** *wird zusammengefasst in der Menge*

$$\mathcal{A}_{\mathrm{wait}} = \{((g, v, \mathrm{arr}), (g, v, \mathrm{dep})) \in \mathcal{E}_{\mathrm{arr}} \times \mathcal{E}_{\mathrm{dep}}\}.$$

Schließlich wird auch jede Möglichkeit zum **Umsteigen** *von einem Fahrzeug $g \in F$ in Fahrzeug $h \in F$ im Bahnhof $v \in V^g \cap V^h$ als eine Aktivität a aufgefasst. Diese beginnt mit dem Ankunftsereignis $e_1 = (g, v, \mathrm{arr}) \in \mathcal{E}_{\mathrm{arr}}$ und endet mit dem Abfahrtsereignis $e_2 = (h, v, \mathrm{dep}) \in \mathcal{E}_{\mathrm{dep}}$. Die geplante Zeitspanne zum Ermöglichen eines Umsteigevorgangs ist gegeben durch $\Pi_{e_2} - \Pi_{e_1}$, die Mindestdauer durch $L_a = L_{gh}^v$. Die Gesamtheit aller* **Umsteigeaktivitäten** *wird zusammengefasst in der Menge*

$$\begin{aligned} \mathcal{A}_{\mathrm{change}} \quad = \quad & \{((g, v, \mathrm{arr}), (h, v, \mathrm{dep})) \in \mathcal{E}_{\mathrm{arr}} \times \mathcal{E}_{\mathrm{dep}} : \textit{es ist möglich, im Bahnhof } v \in V^g \cap V^h \\ & \textit{von Fahrzeug } g \in F \textit{ nach } h \in F \textit{ umzusteigen}\}. \end{aligned}$$

Die Gesamtheit aller Aktivitäten wird mit

$$\mathcal{A} = \mathcal{A}_{\mathrm{drive}} \cup \mathcal{A}_{\mathrm{wait}} \cup \mathcal{A}_{\mathrm{change}}$$

bezeichnet. Die Pufferzeit s_a für eine Aktivität $a = (e_1, e_2) \in \mathcal{A}$ ist gegeben durch

$$s_a = \Pi_{e_2} - \Pi_{e_1} - L_a.$$

Da jede Aktivität mit je einem Ereignis beginnt und endet, können die Aktivitäten mit den gerichteten Kanten eines Graphen identifiziert werden.

$\mathcal{A}_{\mathrm{change}}$ lässt sich auch darstellen durch

$$\mathcal{A}_{\mathrm{change}} = \{((g, v, \mathrm{arr}), (h, v, \mathrm{dep})) : (g, h, v) \in \mathcal{U}\},$$

also kann $\mathcal{U}$ mit $\mathcal{A}_{\mathrm{change}}$ identifiziert werden.

Definition 1.2.10 (nach [26]) *Sei eine Menge F von Fahrzeugen, eine Menge V von Bahnhöfen sowie ein zulässiger Fahrplan $\mathbf{\Pi}$ mit Ankunfts- und Abfahrtszeiten der Fahrzeuge in den Bahnhöfen gegeben. Ein* **Ereignis-Aktivitäts-Netzwerk** *$\mathcal{N} = (\mathcal{E}, \mathcal{A})$ ist ein gerichteter Graph mit Knotenmenge $\mathcal{E}$ und Kantenmenge $\mathcal{A}$.*

Bemerkung 1.2.11 (nach [26]) *Mit Hilfe eines Ereignis-Aktivitäts-Netzwerks $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ und Mindestdauern L_a, $a \in \mathcal{A}$, lässt sich Definition 1.2.3 umformulieren. Ein Fahrplan $\mathbf{\Pi} \in \mathbb{N}_0^{|\mathcal{E}|}$ heißt* **zulässig**, *wenn für alle $a = (e_1, e_2) \in \mathcal{A}$ gilt:*

$$\Pi_{e_2} - \Pi_{e_1} \geq L_a.$$

Dieses Ereignis-Aktivitäts-Netzwerk ist sehr viel komplexer als das in Abschnitt 1.2.1 vorgestellte Transportnetzwerk, da nun jedem Bahnhof eine Vielzahl von Ereignissen zugeordnet sind, die durch Warte- und Umsteigeaktivitäten untereinander verbunden sind, sowie Fahrtaktivitäten, welche die Ereignisse in verschiedenen Bahnhöfen miteinander verknüpfen. Da es sich bei dem Ereignis-Aktivitäts-Netzwerk um die zeitexpandierte Variante des Transportnetzwerkes mit gleichem Fahrplan handelt, können darin keine gerichteten Kreise entstehen. Dies ist für weitergehende Überlegungen von Bedeutung.

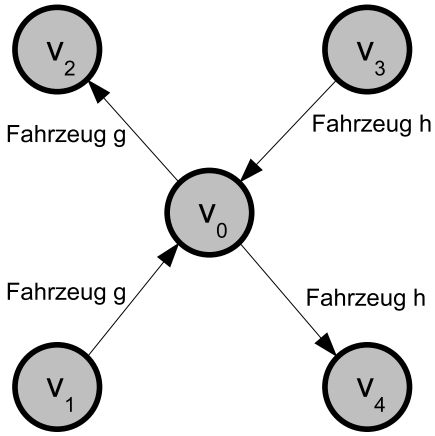


Abbildung 1.2: Zwei Züge in einem Bahnhof – Modellierung als Transportnetzwerk (nach [26])

Beispiel 1.1 (nach [26]) *Betrachte einen Bahnhof, in dem von zwei verschiedenen Bahnhöfen aus je ein Zug eintrifft. Fahrgäste können zwischen den jeweiligen Zügen umsteigen. Dann fahren die Züge weiter, jeder zu einem anderen Bahnhof. Die Modellierung dieser Situation nach den verschiedenen Ansätzen wird in den Abbildungen 1.2 und 1.3 verdeutlicht.*

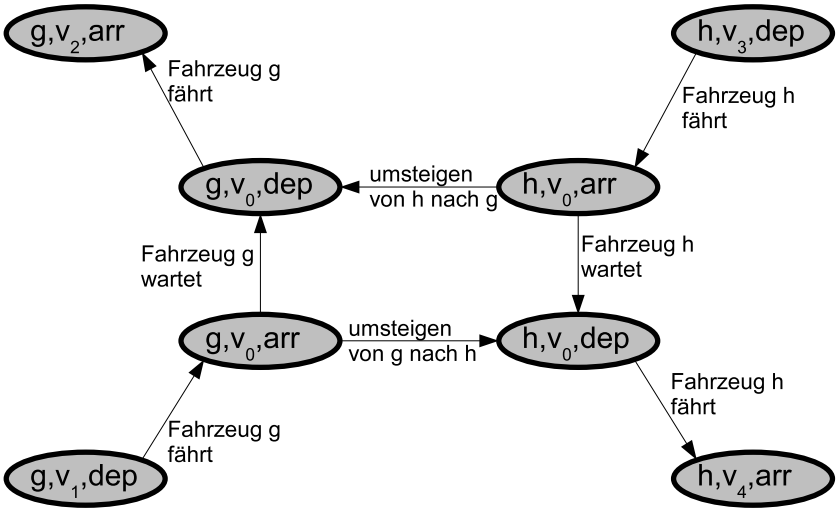


Abbildung 1.3: Zwei Züge in einem Bahnhof – Modellierung als Ereignis-Aktivitäts-Netzwerk (nach [26])

Das erste Modell enthält lediglich den Bahnhof, in dem die Züge aufeinander treffen, sowie den jeweils vorhergehenden und nachfolgenden Bahnhof mit Direktfahrkanten, also fünf Knoten und vier Kanten. Aus dem Netzwerk ist nicht unmittelbar erkennbar, dass die Züge im Bahnhof warten oder dass ein Umsteigen möglich sein soll. Auch Zeitangaben sind nicht Teil des Netzwerks. Dafür ist es allerdings sehr übersichtlich.

Das zweite Modell enthält hingegen für jeden Zug eine Abfahrt von einem Startbahnhof, eine Ankunft und Abfahrt im Umsteigebahnhof, und eine Ankunft im Endbahnhof. Zu jedem Zug gehören zwei Fahrtaktivitäten und eine Warteaktivität, außerdem liegen zwei Umsteigeaktivitäten vor. Es gibt also insgesamt acht Ereignisse und acht Aktivitäten. Es sind sehr viel mehr Details im Netzwerk erkennbar. Bei umfangreicheren Modellen verliert man aus diesem Grund bedauerlicherweise leicht den Überblick.

Definition 1.2.12 (nach [26]) *Sei $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ ein Ereignis-Aktivitäts-Netzwerk, also insbesondere ein azyklischer Graph. Dann lässt sich sowohl für die Ereignisse als auch für die Aktivitäten eine strikt partielle Ordnung $\prec$ und eine partielle Ordnung $\preceq$ definieren:*

1. *Seien $e_1, e_2 \in \mathcal{E}$ beliebig.*
   - *$e_1 \prec e_2$ wenn es in $\mathcal{N}$ einen Pfad gibt, der Ereignis e_1 vor e_2 enthält.*
   - *$e_1 \preceq e_2$ wenn $e_1 \prec e_2$ oder $e_1 = e_2$.*
2. *Seien $a_1, a_2 \in \mathcal{A}$ beliebig.*
   - *$a_1 \prec a_2$ falls in $\mathcal{N}$ einen Pfad gibt, der die Aktivität a_1 vor a_2 enthält.*
   - *$a_1 \preceq a_2$ falls $a_1 \prec a_2$ oder $a_1 = a_2$.*
3. *Sei $\mathcal{E}^0 \subseteq \mathcal{E}$. Dann ist $e_1 \in \mathcal{E}^0$ ein* **minimales Element** *von $\mathcal{E}^0$, wenn es kein $e_2 \in \mathcal{E}^0$ gibt mit $e_2 \prec e_1$.*
4. *Sei $\mathcal{A}^0 \subseteq \mathcal{A}$. Dann ist $a_1 \in \mathcal{A}$ ein* **minimales Element** *von $\mathcal{A}^0$, wenn es kein $a_2 \in \mathcal{A}^0$ gibt mit $a_1 \prec a_2$.*

Bemerkung 1.2.13 (nach [26]) *Für jede nichtleere Menge $\mathcal{E}^0 \subseteq \mathcal{E}$ (oder $\mathcal{A}^0 \subseteq \mathcal{A}$) existiert ein minimales Element, da das Netzwerk azyklisch ist. Dieses minimale Element ist nicht notwendig eindeutig. Allerdings ist die Relation $\prec$ genau dann eine totale Ordnung auf der Menge $\mathcal{E}^0$ beziehungsweise $\mathcal{A}^0$, wenn diese Menge in einem Pfad enthalten ist.*

Es ist möglich, einen Zusammenhang zwischen der partiellen Ordnungsrelation $\prec$ und dem Fahrplan $\mathbf{\Pi}$ herzustellen. Insbesondere kann eine beliebige Menge $\mathcal{E}^0 \subseteq \mathcal{E}$ unter Berücksichtigung von $\prec$ angeordnet werden, sofern jede Mindestdauer L_a echt größer als Null ist. Die Grundlage hierfür stellt das folgende Lemma dar.

Lemma 1.2.14 (nach [26], Beweis in dieser Arbeit hinzugefügt) *Sei $\mathbf{\Pi}$ ein zulässiger Fahrplan für $\mathcal{N}$.*

1. *Für $e_1, e_2 \in \mathcal{E}$ gilt stets, dass aus $e_1 \prec e_2$ die Eigenschaft $\Pi_{e_1} \leq \Pi_{e_2}$ folgt.*
2. *Sei $\mathcal{E}^0 \subseteq \mathcal{E}$, $e_1 \in \mathcal{E}^0$ beliebig. Wenn $\Pi_{e_1} = \min\limits_{e_2 \in \mathcal{E}^0} \Pi_{e_2}$ und $L_a > 0$ für alle $a = (e_2, e_1) \in \mathcal{A}$ mit $e_2 \in \mathcal{E}^0$, dann ist e_1 ein minimales Element von $\mathcal{E}^0$ in Bezug auf $\prec$.*

Beweis: Sei $\mathbf{\Pi}$ ein zulässiger Fahrplan für ein Ereignis-Aktivitäts-Netzwerk $\mathcal{N} = (\mathcal{E}, \mathcal{A})$.

1. Seien $e_1, e_2 \in \mathcal{E}$ beliebig mit der Eigenschaft $e_1 \prec e_2$. Im Netzwerk existiert also nach Definition 1.2.12 ein Pfad $p = e_1 \dots e_2$. Seien dabei $e_1', e_2' \in \mathcal{E}$ beliebige aufeinander folgende Ereignisse auf diesem Pfad, das heißt es gibt eine Aktivität $a \in \mathcal{A}$ mit $a = (e_1', e_2')$. Da $\mathbf{\Pi}$ als zulässig vorausgesetzt wird, gilt $\Pi_{e_1'} + L_a \leq \Pi_{e_2'}$ und wegen $L_a \in \mathbb{N}$ erst recht $\Pi_{e_1'} \leq \Pi_{e_2'}$.
   Induktiv ergibt sich nach Betrachtung aller Aktivitäten entlang des Pfades p, dass tatsächlich $\Pi_{e_1} \leq \Pi_{e_2}$ gilt.
2. Es gelte die oben genannte Voraussetzung. Angenommen, e_1 ist kein minimales Element von $\mathcal{E}^0$.
   Nach Bemerkung 1.2.13 enthält $\mathcal{E}^0$ stets mindestens ein minimales Element $e_{\min}$ und insbesondere eines mit $e_{\min} \prec e_1$. Unterscheide nun folgende Fälle:
   a) $\Pi_{e_{\min}} < \Pi_{e_1}$: Dies widerspricht der Voraussetzung $\Pi_{e_1} = \min\limits_{e_2 \in \mathcal{E}^0} \Pi_{e_2} \leq \Pi_{e_{\min}}$
   b) $\Pi_{e_{\min}} = \Pi_{e_1}$: Wegen $e_{\min} \prec e_1$ gibt es in $\mathcal{N}$ einen Pfad $p = e_{\min} \dots e_1$. Sei $e_2 \in \mathcal{E}$ der unmittelbare Vorgänger von e_1 auf p, das heißt $p = e_{\min} \dots e_2 e_1$ und es existiert eine Aktivität $a = (e_2, e_1) \in \mathcal{A}$. Es gilt $e_{\min} \preceq e_2 \preceq e_1$, nach 1. somit $\Pi_{e_{\min}} \leq \Pi_{e_2} \leq \Pi_{e_1}$ und in diesem Fall sogar $\Pi_{e_{\min}} = \Pi_{e_2} = \Pi_{e_1}$. Da $\mathbf{\Pi}$ zulässig ist, muss also $L_a = 0$ gelten. Dies steht im Widerspruch

zur zweiten Hälfte der Voraussetzung.

c) $\Pi_{e_{\min}} > \Pi_{e_1}$: nach 1. nicht möglich, da $e_{\min} \prec e_1$.

Unter den genannten Voraussetzungen ist e_1 also ein minimales Element von $\mathcal{E}^0$.

□

Es ist anzumerken, dass die Umkehrung der Aussagen von Lemma 1.2.14 im Allgemeinen nicht gilt. Wenn nun allerdings die Ereignisse $\mathcal{E}^0 = \{e_1, e_2, \ldots, e_{|\mathcal{E}^0|}\} \subseteq \mathcal{E}$ nach Planzeit sortiert sind, das heißt

$$\Pi_{e_1} \leq \Pi_{e_2} \leq \ldots \leq \Pi_{e_{|\mathcal{E}^0|}},$$

und es ist zusätzlich die Voraussetzung $L_a > 0$ für alle $a = (e_i, e_j) \in \mathcal{A}$ mit $e_i, e_j \in \mathcal{E}^0$ erfüllt, dann gilt $e_k \not\prec e_{k-1}$ $\forall k = 2, \ldots, \left|\mathcal{E}^0\right|$. Eine Menge von Aktivitäten kann auf ähnliche Weise angeordnet werden.

Wie bereits bei dem auf dem Transportnetzwerk basierenden Modell werden Quellverspätungen als bekannt vorausgesetzt, das heißt es gibt eine Menge $\mathcal{E}_{\text{del}} \subseteq \mathcal{E}$ mit $d_e > 0$ für alle $e \in \mathcal{E}_{\text{del}}$. Ansonsten gelte $d_e = 0$ für $e \in \mathcal{E} \setminus \mathcal{E}_{\text{del}}$. Die Verspätung des Ereignisses $e \in \mathcal{E}_{\text{del}}$ hat nun zur Folge, dass der tatsächliche Eintrittszeitpunkt nicht mehr mit dem geplanten Zeitpunkt Π_e zusammenfällt. Eine Erhöhung von Π_e um d_e kann dann zu einem unzulässigen Fahrplan führen. Die folgende Definition ist eine Umgestaltung von Definition 1.2.4, basierend auf der Notation zum Ereignis-Aktivitäts-Netzwerk. Sie ist insbesondere äquivalent zu Definition 1.2.4.

Definition 1.2.15 (nach [26]) *Ein* **modifizierter Fahrplan x** *mit Angaben $x_e \in \mathbb{N}_0$ für alle $e \in \mathcal{E}$ heißt* **zulässig**, *wenn die folgenden Bedingungen gelten:*

$$x_e \geq \Pi_e + d_e \quad \forall e \in \mathcal{E}, \tag{1.16}$$

$$x_{e_2} - x_{e_1} \geq L_a \quad \forall a = (e_1, e_2) \in \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{drive}}. \tag{1.17}$$

Bedingung (1.16) stellt sicher, dass kein Ereignis früher eintreten darf als geplant, außerdem werden hierdurch die Quellverspätungen für alle $e \in \mathcal{E}_{\text{del}}$ berücksichtigt. Mit Hilfe der Bedingungen (1.17) wird dafür gesorgt, dass die Verspätungen korrekt von einem Ereignis auf das nächste entlang der Fahrt- und Warteaktivitäten übertragen wird.

Bemerkung 1.2.16 *Auch an dieser Stelle ist analog zu Bemerkung 1.2.5 anzumerken, dass Folgeverspätungen sich ausschließlich baumförmig entlang von einzelnen Pfaden im Netzwerk ausbreiten. Entlang eines Pfades kann sich die Verspätung eines Ereignisses verglichen mit der Verspätung des Vorgängerereignisses nur dann vergrößern, wenn es zusätzlich Teil eines anderen Pfades ist, über den eine größere Folgeverspätung übertragen wird, oder wenn das Ereignis selbst quellverspätet ist. Im Allgemeinen nimmt allerdings die Verspätung entlang eines Pfades ab oder bleibt gleich. Insbesondere in Kombination mit der in Definition 1.2.12 vorgestellten Ordnungsrelation wird dies in Abschnitt 1.5 noch von Bedeutung sein.*

Auch die Formulierung der Bedingungen in Satz 1.2.7 wird an die neue Notation angepasst. Hierzu werden zunächst folgende Entscheidungsvariablen eingeführt:

$$y_e = \text{Verspätung des Ereignisses } e \in \mathcal{E}$$

$$z_a = \begin{cases} 0 & \text{wenn Umsteigeaktivität } a \in \mathcal{A}_{\text{change}} \text{ gehalten,} \\ 1 & \text{wenn Umsteigeaktivität } a \in \mathcal{A}_{\text{change}} \text{ verpasst.} \end{cases}$$

Satz 1.2.17 (nach [26]) *Sei $M \in \mathbb{N}_0$ ein hinreichend großer Parameter. Dann ist $(\mathbf{y}, \mathbf{z})$ mit $\mathbf{y} \in \mathbb{N}_0^{|\mathcal{E}|}$ und $\mathbf{z} \in \{0,1\}^{|\mathcal{A}_{\text{change}}|}$ eine zulässige Lösung für das Anschlusssicherungsproblem, wenn die folgenden Bedingungen erfüllt sind:*

$$y_e \geq d_e \quad \forall e \in \mathcal{E}_{\text{del}}, \tag{1.18}$$

$$y_{e_1} - y_{e_2} \leq s_a \quad \forall a = (e_1, e_2) \in \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{drive}}, \tag{1.19}$$

$$-M z_a + y_{e_1} - y_{e_2} \leq s_a \quad \forall a = (e_1, e_2) \in \mathcal{A}_{\text{change}}. \tag{1.20}$$

Notation 1.2.18 (nach [26]) *Die Menge der zulässigen Lösungen des Anschlusssicherungsproblems wird bezeichnet durch*

$$Feas_{\text{DM}} = \{(\mathbf{y}, \mathbf{z}) \in \mathbb{N}_0^{|\mathcal{E}|} \times \{0,1\}^{|\mathcal{A}_{\text{change}}|} : (\mathbf{y}, \mathbf{z}) \text{ erfüllt (1.18), (1.19) und (1.20)}\}.$$

Dabei steht das Kürzel DM *sowohl hier als auch in den folgenden Abschnitten für* **Delay Management**.

1.3 Ein Optimierungsproblem

Im vorigen Abschnitt wurden Nebenbedingungen eingeführt, welche die Menge der zulässigen modifizieren Fahrpläne definieren. Dieser Abschnitt befasst sich mit der Aufgabe, in dieser Menge die besonders guten Fahrpläne zu identifizieren. Abschnitt 1.3.1 zunächst befasst sich mit einem Sonderfall, in dem das Auffinden eines optimalen modifizierten Fahrplans besonders einfach ist. Trotz seiner geringen Flexibilität ist dieses Problem für die später vorgestellten Lösungsverfahren von Bedeutung.

In Abschnitt 1.3.2 wird eine lineare Zielfunktion vorgestellt, welche der Bestimmung der Qualität eines modifizierten Fahrplans dient. Jener Fahrplan, welcher nach dieser Zielfunktion die beste Qualität aufweist, wird als optimale Lösung des Anschlusssicherungsproblems hervorgehoben.

1.3.1 Ein Sonderfall: Alle Anschlüsse sind festgelegt

Wenn alle Anschlüsse festgelegt sind, ist die Bestimmung eines optimalen Fahrplans besonders einfach. Es werden zwei Verfahren vorgestellt, durch die die Summe der Mindestverspätungen minimiert wird, um auf diese Weise einen modifizierten Fahrplan $\mathbf{x}$ zu erhalten, der möglichst wenig vom ursprünglichen Fahrplan $\mathbf{\Pi}$ abweicht.

Notation 1.3.1 (nach [26]) *Sei $\mathcal{A}_{\text{change}}$ als Menge von Anschlüssen gegeben. Bezeichne*

$\mathcal{A}^{\text{fix}}$ *die Menge der Anschlüsse, die gehalten werden sollen,*

$\mathcal{A}^{\text{miss}}$ *die Menge der Anschlüsse, die verpasst werden sollen,*

$\mathcal{A}^{\text{open}}$ *die Menge der Anschlüsse, für die noch nicht entschieden ist, ob sie gehalten oder verpasst werden sollen.*

Damit gilt $\mathcal{A}_{\text{change}} = \mathcal{A}^{\text{fix}} \cup \mathcal{A}^{\text{miss}} \cup \mathcal{A}^{\text{open}}$. Zur Vereinfachung der Notation setze

$$\mathcal{A}(\mathcal{A}^{\text{fix}}) = \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{drive}} \cup \mathcal{A}^{\text{fix}}.$$

Definition 1.3.2 (nach [26]) *Sei ein Ereignis-Aktivitäts-Netzwerk $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ mit Fahrzeugen F, zulässigem Fahrplan $\mathbf{\Pi}$, Pufferzeiten $\mathbf{s}$ und eine Menge von quellverspäteten Ereignissen $\mathcal{E}_{\text{del}}$ gegeben. Sei zusätzlich für jedes Ereignis $e \in \mathcal{E}$ ein Gewicht*[3] *$w_e \geq 0$ gegeben. Sei ferner $\mathcal{A}^{\text{fix}} \subseteq A$ eine Menge von zu haltenden Anschlüssen.*
Ziel des Optimierungsproblems ($\mathtt{DM}(\mathcal{A}^{\text{fix}})$) ist das Auffinden eines zulässigen modifizierten Fahrplans $\mathbf{x}$, welcher die in $\mathcal{A}^{\text{fix}}$ enthaltenen Anschlüsse aufrecht erhält, für den es also ein $\mathbf{z} \in \{0,1\}^{|\mathcal{A}_{\text{change}}|}$ gibt mit $z_a = 0$ für alle $a \in \mathcal{A}^{\text{fix}}$ und zusätzlich $(\mathbf{x} - \mathbf{\Pi}, \mathbf{z}) \in Feas_{\text{DM}}$ gilt. Dieser modifizierte Fahrplan soll dabei die folgende Größe minimieren:

$$f_{\text{DM}^{\text{fix}}} = \sum_{e \in \mathcal{E}} w_e (x_e - \Pi_e).$$

Ansatz 1 – Lineare Programmierung

In Anlehnung an Abschnitt 1.2.2 und die in Satz 1.2.17 aufgeführten Nebenbedingungen lässt sich das Problem ($\mathtt{DM}(\mathcal{A}^{\text{fix}})$) als gemischt-ganzzahliges Programm formulieren. Zunächst können allerdings die Variablen für die zu haltenden Anschlüsse festgelegt werden, das heißt $z_a = 0$ für $a \in \mathcal{A}^{\text{fix}}$. Für die übrigen Variablen z_a, $a \notin \mathcal{A}^{\text{fix}}$ ist zunächst noch kein geeigneter Wert bekannt. Die Ganzzahligkeitsbedingung $\mathbf{y} \in \mathbb{N}_0^{|\mathcal{E}|}$ ist nicht notwendig, wie später noch deutlich werden wird. Es ergibt sich das in Abbildung 1.3.1 dargestellte lineare Programm.

Es ist unter Umständen sinnvoll, anstatt der Verspätungsvariablen y_e, $e \in \mathcal{E}$ die Variablen x_e zu verwenden, also den modifizierten Fahrplan. Dies geschieht durch Ersetzen von y_e durch $x_e - \Pi_e$. Dies führt zu der Problemstellung ($\mathtt{DM}^x_{\text{LP}}(\mathcal{A}^{\text{fix}})$), dargestellt in Abbildung 1.3.1.

Der folgende Satz befasst sich mit der Aussage, dass die Nebenbedingungen der Form $y_e \in \mathbb{N}_0$ für $e \in \mathcal{E}$ nicht notwendig sind.

Satz 1.3.3 (nach [26]) *Alle Extrempunkte der für ($\mathtt{DM}_{\text{LP}}(\mathcal{A}^{\text{fix}})$) zulässigen Punktmenge sind ganzzahlig.*

Beweis: Die Koeffizientenmatrix von ($\mathtt{DM}_{\text{LP}}(\mathcal{A}^{\text{fix}})$) ist gegeben durch

[3] Gewichte können beispielsweise dazu dienen, die erwartete Anzahl der ein- oder aussteigenden Fahrgäste einzubeziehen

$$\min f_{\mathrm{DM}^{\mathrm{fix}}} = \sum_{e \in E} w_e y_e$$

so dass

$$y_e \geq d_e \quad \forall e \in \mathcal{E} \tag{1.21}$$

$$y_{e_1} - y_{e_2} \leq s_a \quad \forall a = (e_1, e_2) \in \mathcal{A}(\mathcal{A}^{\mathrm{fix}}). \tag{1.22}$$

Abbildung 1.4: Lineares Programm ($\mathtt{DM}_{\mathrm{LP}}(\mathcal{A}^{\mathrm{fix}})$)

$$\min f^x_{\mathrm{DM}^{\mathrm{fix}}} = \sum_{e \in E} w_e x_e$$

so dass

$$x_e \geq \Pi_e + d_e \quad \forall e \in \mathcal{E} \tag{1.23}$$

$$x_{e_2} - x_{e_1} \geq L_a \quad \forall a = (e_1, e_2) \in \mathcal{A}(\mathcal{A}^{\mathrm{fix}}). \tag{1.24}$$

Abbildung 1.5: Lineares Programm ($\mathtt{DM}^x_{\mathrm{LP}}(\mathcal{A}^{\mathrm{fix}})$)

$$\Phi = \begin{pmatrix} -I_{|\mathcal{E}|} \\ \Theta^\top \end{pmatrix},$$

wobei $I_{|\mathcal{E}|}$ die $|\mathcal{E}| \times |\mathcal{E}|$ Einheitsmatrix bezeichnet und Θ die Knoten-Kanten-Inzidenzmatrix von $\mathcal{N}$ ist, das heißt

$$\Theta_{e,a} = \begin{cases} -1 & \text{falls } a = (e, e') \text{ für ein } e' \in \mathcal{E}, \\ 1 & \text{falls } a = (e', e) \text{ für ein } e' \in \mathcal{E}, \\ 0 & \text{sonst.} \end{cases}$$

Θ ist eine total unimodulare Matrix, daher auch $\Theta^\top$ und auch Φ. Zusammen mit $s_a \in \mathbb{N}$ ergibt sich, dass alle Basislösungen der LP-Relaxation ganzzahlig sind, vergleiche [26], Anhang A. □

Das heißt, dass bei gemäß $\mathbf{z}$ als zu halten vorgegebenen Anschlüssen angenommen werden kann, dass die Verspätungen $\mathbf{y}$ tatsächlich in der geforderten Einheit sind, und ($\mathtt{DM}_{\mathrm{LP}}(\mathcal{A}^{\mathrm{fix}})$) – und damit ($\mathtt{DM}^x_{\mathrm{LP}}(\mathcal{A}^{\mathrm{fix}})$) – kann durch lineare Programmierung gelöst werden.

Ansatz 2 – Critical Path Method

Die *Critical Path Method* (CPM) ist eine Methode, die ihren Ursprung in der Projektplanung hat. Sie gehört zu der Klasse der *Message-passing*-Algorithmen. Zunächst wird für ein durchzuführendes Projekt ein so genanntes Projektnetzwerk – ein kreisfreier, gerichteter Graph – erstellt, dessen Kanten Teilaufgaben repräsentieren und mit einer Zeitdauer assoziiert sind. Eine von einem beliebigen Knoten ausgehende Teilaufgabe kann erst dann durchgeführt werden, wenn bereits alle zu dem Knoten hinführenden Teilaufgaben abgeschlossen sind.

Ordnet man nun die Teilaufgaben nach Voraussetzungen an – dies ist möglich, da der Graph kreisfrei ist – kann man die Aufgaben der Reihe nach abarbeiten. Damit ist sichergestellt, dass vor Erledigung einer Teilaufgabe stets alle Voraussetzungen erfüllt sind. Auf diese Weise kann für jede Aufgabe der frühest mögliche Anfangs- und Endzeitpunkt bestimmt werden und somit die Mindestdauer für die Durchführung des Projekts.

Dieser Ansatz lässt sich auch für Ereignis-Aktivitäts-Netzwerke verwenden, um bei fixierten Anschlüssen einen optimalen modifizierten Fahrplan zu erhalten. Abbildung 1.6 stellt die CPM für ein Ereignis-Aktivitäts-Netzwerk $\mathcal{N}$, Quellverspätungen $\mathbf{d}$, Pufferzeiten $\mathbf{s}$ und die Menge zu haltender Anschlüsse $\mathcal{A}^{\mathrm{fix}}$.

Für eine detailliertere Darstellung der CPM sowie den Übergang vom Ereignis-Aktivitäts-Netzwerk zum Projektnetzwerk wird auf [26], Abschnitt 7.2 verwiesen.

Bemerkung 1.3.4 (zeitminimale Lösung, nach [26]) *Besonders hervorzuheben ist an dieser Stelle noch, dass für jede Menge $\mathcal{A}^{\mathrm{fix}} \subseteq \mathcal{A}_{\mathrm{change}}$ zulässige Lösungen existieren und mit Hilfe dieser Methode die optimale,* **zeitminimale** *Lösung zu bestimmt werden kann, das heißt eine Lösung mit*

Eingabe: $\mathcal{N}$, $\mathbf{d}$, $\mathbf{s}$, $\mathcal{A}^{\text{fix}}$.
Ausgabe: Optimale Lösung für (DM($\mathcal{A}^{\text{fix}}$)).
Schritt 1. Sortiere $\mathcal{E} = \{e_1, \ldots, e_{|\mathcal{E}|}\}$ gemäß $\prec$.
Schritt 2. Für $k = 1, \ldots, |\mathcal{E}|$: $y_{e_k} = \max\{d_{e_k}, \max_{a=(e,e_k)\in\mathcal{A}(\mathcal{A}^{\text{fix}})} y_e - s_a\}$.
Schritt 3. Ausgabe: $y_e, e \in \mathcal{E}$.

Abbildung 1.6: Algorithmus: Berechnung der Optimallösung für (DM($\mathcal{A}^{\text{fix}}$)) mit Hilfe der CPM

$$y_e = \max\{d_e, \max_{a=(e',e)\in\mathcal{A}(\mathcal{A}^{\text{fix}})} y_e - s_a\}.$$

1.3.2 Anschlusssicherung als gemischt-ganzzahliges Programm

Nachdem bereits eine Zielfunktion für das Anschlusssicherungsproblem mit fixierten Anschlüssen vorgestellt wurde, ist es nur noch ein kleiner Schritt hin zu einer Zielfunktion für das allgemeine Anschlusssicherungsproblem. Zunächst einmal betrachte die Verspätung, mit der ein Fahrgast sein Ziel erreicht:

- wenn ein Anschluss verpasst wurde, wird angenommen, dass der Fahrgast den Anschlusszug in der nächsten Periode nimmt, der als pünktlich angenommen wird. Der Fahrgast hat eine Verspätung, die der Periodendauer T entspricht,
- ansonsten hat der Fahrgast die Verspätung des letzten verwendeten Zuges.

Wenn nun für jedes Ereignis $e \in \mathcal{E}$ ein Gewicht $w_e \geq 0$ vorliegt, das die ungefähre Zahl der aussteigenden Fahrgäste widerspiegelt, sowie für jeden Anschluss $a \in \mathcal{A}_{\text{change}}$ ein Gewicht w_a, das die ungefähre Anzahl der umsteigenden Fahrgäste angibt, dann kann die resultierende Gesamtverspätung abgeschätzt werden durch

$$f_{\text{DM}} = \sum_{e\in\mathcal{E}} w_e y_e + \sum_{a\in\mathcal{A}_{\text{change}}} w_a z_a T.$$

Zusammen mit den in Satz 1.2.17 formulierten Nebenbedingungen ergibt sich das gemischt-ganzzahlige Programm (DM), dargestellt in Abbildung 1.7.

$$\min f_{\text{DM}} = \sum_{e\in\mathcal{E}} w_e y_e + \sum_{a\in\mathcal{A}_{\text{change}}} w_a z_a T$$

so dass

$$\begin{aligned}
y_e &\geq d_e && \forall e \in \mathcal{E} \\
y_{e_1} - y_{e_2} &\leq s_a && \forall a = (e_1, e_2) \in \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{drive}}, \\
-M z_a + y_{e_1} - y_{e_2} &\leq s_a && \forall a = (e_1, e_2) \in \mathcal{A}_{\text{change}}, \\
z_a &\in \{0,1\} && \forall a \in \mathcal{A}_{\text{change}}.
\end{aligned}$$

Abbildung 1.7: (DM) – Anschlusssicherung als gemischt-ganzzahliges Programm

Es ist hervorzuheben, dass sowohl Zielfunktion als auch Nebenbedingungen in linearer Form gegeben sind. Dies ermöglicht den Einsatz des in Abschnitt 2.1 vorgestellten Branch-and-Bound Verfahrens zur Lösung des Problems. Zusätzlich ist erneut zu beachten, dass die Ganzzahligkeitsbedingungen $y_e \in \mathbb{N}_0$ grundsätzlich überflüssig sind, da hier wie bereits bei (DM($\mathcal{A}^{\text{fix}}$)) jede optimale Lösung mit $z_a \in \{0,1\}$ für alle $a \in \mathcal{A}_{\text{change}}$ automatisch ganzzahlige Verspätungsvariablen y_e hat.

Bedauerlicherweise können Fälle auftreten, in denen f_{DM} *nicht* die echte Gesamtverspätung aller Fahrgäste widerspiegelt, sondern lediglich eine obere Schranke, da einige Verspätungen doppelt einbezogen werden können. Eine korrekte aber gleichzeitig komplexere Modellierung, basierend auf Pfaden, die Kunden im Netzwerk wählen, wird in [26] vorgestellt. Das hier beschriebene Modell (DM) entspricht dem dort beschriebenen Modell (TDM-const).

Dort wird auch eine hinreichende Bedingung genannt, unter der die Lösungen der resultierenden Optimierungsprobleme übereinstimmen, die *Never-Meet-Property*. Diese erfordert, dass im Ereignis-Aktivitäts-Netzwerk keine zwei Verspätungen in einem Ereignis zusammentreffen dürfen. Bedauerlicherweise ist diese Bedingung in der Praxis nur *fast* erfüllt, das heißt es gibt einige wenige Ereignisse, in denen zwei Verspätungen zusammentreffen.

Zum Abschluss dieses Abschnitts soll noch eine Problemformulierung mit Hilfe von Matrizen und Vektoren vorgestellt werden.

Notation 1.3.5 ((DM) in Matrixschreibweise) *Das in Abbildung 1.7 dargestellte Programm lässt sich wegen seiner Linearität auch in Matrixschreibweise formulieren. Betrachte hierzu für* $n_{(\mathrm{wd})} = |\mathcal{A}_{\mathrm{wait}}| + |\mathcal{A}_{\mathrm{drive}}|$, $n_{(\mathrm{c})} = |\mathcal{A}_{\mathrm{change}}|$ *und* $n_{(\mathrm{e})} = |\mathcal{E}|$ *die Matrizen* $\mathbf{A}^{(\mathrm{wd})} \in \{-1,0,+1\}^{n_{(\mathrm{wd})} \times n_{(\mathrm{e})}}$ *und* $\mathbf{A}^{(\mathrm{c})} \in \{-1,0,+1\}^{n_{(\mathrm{c})} \times n_{(\mathrm{e})}}$ *sowie Vektoren* $\mathbf{b}^{(\mathrm{wd})} \in \mathbb{N}_0^{n_{(\mathrm{wd})}}$, $\mathbf{b}^{(\mathrm{c})} \in \mathbb{N}_0^{n_{(\mathrm{c})}}$, $\mathbf{c}^{(\mathrm{e})} \in \mathbb{R}^{n_{(\mathrm{e})}}$ *und* $\mathbf{c}^{(\mathrm{c})} \in \mathbb{R}^{n_{(\mathrm{c})}}$ *mit den Eigenschaften*

$$A^{(\mathrm{wd})}_{ae} = \begin{cases} +1 & \text{falls } \exists e' \in \mathcal{E} \text{ so dass } a = (e, e'), \\ -1 & \text{falls } \exists e' \in \mathcal{E} \text{ so dass } a = (e', e), \\ 0 & \text{sonst.} \end{cases} \quad \text{für } a \in \mathcal{A}_{\mathrm{wait}} \cup \mathcal{A}_{\mathrm{drive}},\ e \in \mathcal{E},$$

$$A^{(\mathrm{c})}_{ae} = \begin{cases} +1 & \text{falls } \exists e' \in \mathcal{E} \text{ so dass } a = (e, e'), \\ -1 & \text{falls } \exists e' \in \mathcal{E} \text{ so dass } a = (e', e), \\ 0 & \text{sonst.} \end{cases} \quad \text{für } a \in \mathcal{A}_{\mathrm{change}},\ e \in \mathcal{E},$$

$$\begin{aligned} b^{(\mathrm{wd})}_a &= s_a \quad \text{für } a \in \mathcal{A}_{\mathrm{wait}} \cup \mathcal{A}_{\mathrm{drive}}, \\ b^{(\mathrm{c})}_a &= s_a \quad \text{für } a \in \mathcal{A}_{\mathrm{change}}, \\ c^{(\mathrm{e})}_e &= w_e \quad \text{für } e \in \mathcal{E}, \\ c^{(\mathrm{c})}_a &= w_a T \quad \text{für } a \in \mathcal{A}_{\mathrm{change}}. \end{aligned}$$

Damit lässt sich (DM) formulieren als

$$\min {\mathbf{c}^{(\mathrm{c})}}^{\top} \cdot \mathbf{z} + {\mathbf{c}^{(\mathrm{e})}}^{\top} \cdot \mathbf{y}$$

so dass

$$\begin{pmatrix} \mathbf{0} & \mathbf{A}^{(\mathrm{wd})} \\ -\mathbf{I}_{n_{(\mathrm{c})}} \cdot M & \mathbf{A}^{(\mathrm{c})} \end{pmatrix} \begin{pmatrix} \mathbf{z} \\ \mathbf{y} \end{pmatrix} \leq \begin{pmatrix} \mathbf{b}^{(\mathrm{wd})} \\ \mathbf{b}^{(\mathrm{c})} \end{pmatrix},$$

$$z_a \in \{0,1\} \qquad \forall a \in \mathcal{A}_{\mathrm{change}},$$

$$y_e \geq d_e \qquad \forall e \in \mathcal{E}.$$

1.4 Komplexitätsbetrachtung

Dieser Abschnitt befasst sich mit der Frage, wie leicht (beziehungsweise wie schwer) die in dieser Arbeit vorgestellte Variante des Anschlusssicherungsproblems zu lösen ist. Von einer *pfadbasierten* Variante des Anschlusssicherungsproblems wurde in [9] gezeigt, dass sie unter geeigneten Bedingungen sogar dann NP-schwer ist, wenn Pufferzeiten gleich 0 sind und alle Quellverspätungen den gleichen Wert haben.

Zunächst einmal werden einige einfache Spezialfälle betrachtet, die zu der Komplexitätsklasse **P** gehören, das heißt für die eine Lösung in Polynomialzeit möglich ist.

Satz 1.4.1 (nach [26]) *Das Anschlusssicherungsproblem mit fixierten Anschlüssen (vergleiche Abschnitt 1.3.1) gehört zu der Komplexitätsklasse* **P**.

Beweis: Die in Abschnitt 1.3.1 vorgestellte Critical Path Method erfordert einen Aufwand von $O(|\mathcal{E}| \cdot \log(|\mathcal{E}|))$ für die anfängliche Sortierung der Ereignisse sowie $O(|\mathcal{E}| \cdot |\mathcal{A}|)$ für die Bestimmung der minimalen Folgeverspätungen. Der Gesamtaufwand lässt sich also durch $O(|\mathcal{E}| \cdot (\log(|\mathcal{E}|) + |\mathcal{A}|))$ beschreiben.
Das Problem ist somit in polynomial beschränkter Zeit lösbar. □

Satz 1.4.2 (nach [26]) *Die in dieser Arbeit vorgestellte Variante des Anschlusssicherungsproblems gehört zu der Komplexitätsklasse* **P**, *wenn alle Pufferzeiten gleich 0 sind und alle Quellverspätungen gleich groß sind.*

Beweis: siehe [26], Theorem 8.23. □

Satz 1.4.3 *Die in dieser Arbeit vorgestellte Variante des Anschlusssicherungsproblem gehört zu der Komplexitätsklasse* **NP**.

Beweis: Sei eine Instanz des Anschlusssicherungsproblems im Sinne von Notation 1.3.5 durch Matrizen $\mathbf{A}^{(\mathrm{wd})}$, $\mathbf{A}^{(\mathrm{c})}$, Vektoren $\mathbf{c}^{(\mathrm{c})}$, $\mathbf{c}^{(\mathrm{wd})}$, $\mathbf{b}^{(\mathrm{wd})}$ und $\mathbf{b}^{(\mathrm{c})}$ und die Periodendauer T gegeben. Betrachte zusätzlich eine Lösung $(\mathbf{z}, \mathbf{y})$ sowie eine Schranke $K \in \mathbb{R}$.

Dann lässt sich mit $O((n_{(\mathrm{c})} + n_{(\mathrm{e})}) \cdot (n_{(\mathrm{wd})} + n_{(\mathrm{c})}))$ Operationen[4] verifizieren, ob die Lösung $(\mathbf{z}, \mathbf{y})$ zulässig ist. Dann lässt sich mit $O(n_{(\mathrm{c})} + n_{(\mathrm{e})})$ Operationen der Zielfunktionswert berechnen, um dann zu vergleichen, ob er der Schranke K genügt. Diese Prüfung kann also in Polynomialzeit erfolgen, somit gehört das Anschlusssicherungsproblem zur Problemklasse **NP**. □

Es wird vermutet, dass die im Rahmen dieser Arbeit untersuchte Variante im allgemeinen Fall, das heißt mit beliebigen Pufferzeiten und Quellverspätungen, tatsächlich sogar NP-schwer ist. Allerdings wurde bisher kein NP-vollständiges Problem gefunden, das in polynomialer Zeit auf das Anschlusssicherungsproblem reduziert werden kann.

1.5 Reduktion des Ereignis-Aktivitäts-Netzwerks

Die in Abschnitt 1.2.2 eingeführten Ereignis-Aktivitäts-Netzwerke können bei realistischen Anwendungsdaten sehr groß werden, womit gleichzeitig der Problemumfang wächst. Allerdings ist für viele Überlegungen nicht das gesamte Netzwerk von Interesse, sondern nur ein kleinerer, *relevanter* Teil. Dieser Abschnitt befasst sich mit der Frage, was solch eine Relevanz ausmacht und wie bei vorgegebenen Quellverspätungen der relevante Anteil eines Ereignis-Aktivitäts-Netzwerks identifiziert werden kann, was zu einem verkleinerten Netzwerk führt. Bei den Fragestellungen zur Anschlusssicherung reicht es nun aus, dieses reduzierte Netzwerk zu betrachten.

Im Verlauf der Untersuchungen ist es durchaus möglich und auch sinnvoll, weitere Male zu reduzieren und dabei aktuelle Warten/Nichtwarten Entscheidungen einzubeziehen.

Definition 1.5.1 (Relevanz, nach [26]) *Sei ein Ereignis-Aktivitäts-Netzwerk $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ und eine Menge von Quellverspätungen $\mathcal{E}_{\mathrm{del}} \subseteq \mathcal{E}$ gegeben.*

1. *Eine Aktivität a, die kein verpasster Anschluss ist, also $a = (e_1, e_2) \in \mathcal{A}(\mathcal{A}^{\mathrm{fix}}) \cup \mathcal{A}^{\mathrm{open}}$, ist* **relevant**, *wenn bei Halten aller noch offenen Anschlüsse in einer zeitminimalen Lösung $(\mathbf{y}, \mathbf{z})$ eine Verspätung über sie übertragen wird, das heißt wenn $y_{e_1} > s_a$.*
2. *Weitere Aktivitäten – insbesondere verpasste Anschlüsse – sind* **nicht relevant**.
3. *Ein Ereignis $e \in \mathcal{E}$ ist* **relevant**, *wenn für e bei Halten aller noch offenen Anschlüsse notwendig eine Verspätung $y_e > 0$ auftritt.*
4. *Weitere Ereignisse sind* **nicht relevant**.

Bemerkung 1.5.2 *Bei einer relevanten Aktivität $a = (e_1, e_2) \in \mathcal{A}$ sind stets e_1 und e_2 relevant.*

Nun können offene, nicht relevante Anschlüsse gehalten werden, schließlich kann keine Verspätung über sie übertragen werden. Beschränkung auf ausschließlich reduzierte Ereignisse und Aktivitäten führt zu einem neuen Ereignis-Aktivitäts-Netzwerk geringerer Größe:

Definition 1.5.3 *Sei ein Ereignis-Aktivitäts-Netzwerk $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ und eine Menge von Quellverspätungen $\mathcal{E}_{\mathrm{del}} \subseteq \mathcal{E}$ gegeben. Das* **reduzierte Netzwerk** *$\mathcal{N}_{\mathrm{red}} = (\mathcal{E}_{\mathrm{red}}, \mathcal{A}_{\mathrm{red}})$ besteht ausschließlich aus den relevanten Ereignissen $\mathcal{E}_{\mathrm{red}} \subseteq \mathcal{E}$ und Aktivitäten $\mathcal{A}_{\mathrm{red}} \subseteq \mathcal{A}$ von $\mathcal{N}$.*

Beispiel 1.2 *Sei ein Ereignis-Aktivitäts-Netzwerk gemäß Tabelle 1.1 gegeben, dabei ist die Mindestdauer für eine Warteaktivität jeweils eine Minute, die Mindestdauer für eine Umsteigeaktivität jeweils drei Minuten. Mindestdauer für Fahrtaktivitäten sind durch Tabelle 1.2 vorgegeben – alle Fahrzeuge werden als gleich schnell angesehen. Abbildung 1.8 verdeutlicht, wie dieses Ereignis-Aktivitäts-Netzwerk aussehen könnte. Dabei stehen einfache Pfeile für Fahrt- und Warteaktivitäten, während Umsteigeaktivitäten durch einen Doppelpfeil gekennzeichnet sind. Jede Kante ist mit einem Wertepaar $L + s$ versehen, dabei bezeichnet L die Mindestdauer und s die für die Aktivität vor-*

[4] Diese Abschätzung ist sehr großzügig, da die Matrizen $\mathbf{A}^{(\cdot)}$ nur zwei Einträge $\neq 0$ pro Zeile haben.

Id	Fahrzeug	Bahnhof	Typ	Planzeit
e_1	1	A	dep	8:00
e_2	1	D	arr	8:11
e_3	1	D	dep	8:18
e_4	1	F	arr	8:37
e_5	1	F	dep	8:41
e_6	1	I	arr	8:55
e_7	2	B	dep	8:49
e_8	2	C	arr	8:57
e_9	2	C	dep	8:59
e_{10}	2	G	arr	9:13
e_{11}	2	G	dep	9:14
e_{12}	2	H	arr	9:22
e_{13}	2	H	dep	9:26

Id	Fahrzeug	Bahnhof	Typ	Planzeit
e_{14}	2	J	arr	9:36
e_{15}	3	I	dep	8:25
e_{16}	3	H	arr	8:29
e_{17}	3	H	dep	8:31
e_{18}	3	F	arr	8:38
e_{19}	3	F	dep	8:42
e_{20}	3	E	arr	8:48
e_{21}	3	E	dep	8:49
e_{22}	3	C	arr	8:56
e_{23}	3	C	dep	9:00
e_{24}	3	A	arr	9:09
e_{25}	4	K	dep	8:01
e_{26}	4	H	arr	8:10

Id	Fahrzeug	Bahnhof	Typ	Planzeit
e_{27}	4	H	dep	8:11
e_{28}	4	F	arr	8:17
e_{29}	4	F	dep	8:18
e_{30}	4	G	arr	8:26
e_{31}	4	G	dep	8:27
e_{32}	4	D	arr	8:38
e_{33}	4	D	dep	8:40
e_{34}	4	B	arr	8:46
e_{35}	5	D	dep	8:14
e_{36}	5	H	arr	8:38
e_{37}	5	H	dep	8:40
e_{38}	5	J	arr	8:50

Tabelle 1.1: Exemplarischer Fahrplan für ein kleines Netzwerk

Kante	A ↔ C	A↔D	B↔C	B↔D	C↔E	C↔G	D↔F	D↔G	D ↔ H
Dauer	8	9	7	6	7	14	19	10	24

Kante	E↔F	F↔G	F↔H	F↔I	G↔H	H↔I	H↔J	H↔K
Dauer	6	7	6	14	8	4	10	8

Tabelle 1.2: Mindestdauer der Fahrtaktivitäten. Hin- und Rückfahrt dauern nach Plan gleich lange.

gesehene Pufferzeit. Ereignisse sind als Kreise, Bahnhöfe als abgerundete Rechtecke dargestellt. Die Bezeichnung (Id) eines Ereignisses ist der jeweils angefügten Ellipse zu entnehmen.

Seien folgende Quellverspätungen vorgegeben: $d_{e_{16}} = 6$, $d_{e_1} = 10$.

Dann sind lediglich die Ereignisse e_1, e_2, e_3, e_4, e_5, e_6, e_9, e_{10}, e_{11}, e_{12}, e_{16}, e_{17}, e_{18}, e_{19}, e_{20}, e_{21}, e_{22}, e_{35}, e_{36}, e_{37}, e_{38} *und die Aktivitäten* (e_1, e_2), (e_2, e_3), (e_2, e_{35}), (e_3, e_4), (e_4, e_{19}), (e_5, e_6), (e_9, e_{10}), (e_{10}, e_{11}), (e_{11}, e_{12}), (e_{16}, e_{17}), (e_{17}, e_{18}), (e_{18}, e_{19}), (e_{18}, e_5), (e_{19}, e_{20}), (e_{20}, e_{21}), (e_{21}, e_{22}), (e_{22}, e_9), (e_{35}, e_{36}), (e_{36}, e_{37}), (e_{37}, e_{38}) *relevant und keine weiteren. Abbildung 1.9 zeigt das reduzierte Netzwerk* $\mathcal{N}_{\text{red}}$. *Nicht relevante Ereignisse und Aktivitäten sind ausgegraut dargestellt, die Verspätungen bei Halten aller Anschlüsse sind im Inneren jedes Ereignis-Kreises angegeben, wobei Quellverspätungen unterstrichen dargestellt sind.*

Nachdem durch dieses kleine Beispiel verdeutlicht wurde, wie sinnvoll die Reduktion sein kann, stellt sich noch die Frage, wie es möglich ist, nicht relevante Ereignisse und Aktivitäten effizient zu bestimmen. Hierbei hilft die in Definition 1.2.12 vorgestellte Ordnungsrelation.

Bemerkung 1.5.4 (nach [26]) *Sei ein Ereignis-Aktivitäts-Netzwerk* $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ *und eine Menge von Quellverspätungen* $\mathcal{E}_{\text{del}} \subseteq \mathcal{E}$ *gegeben.*

1. *(Frühe Reduktion) Da sich Verspätungen entlang von Pfaden im Netzwerk ausbreiten, ist ein Ereignis* $e \in \mathcal{E}$ *ist nicht relevant, wenn es kein Ereignis* $e' \in \mathcal{E}_{\text{del}}$ *gibt mit* $e' \preceq e$. *Damit sind insbesondere minimale Elemente der Menge* $\mathcal{E} \setminus \mathcal{E}_{\text{nonrel}}$ *entweder quellverspätet oder nicht relevant, wobei* $\mathcal{E}_{\text{nonrel}}$ *eine beliebige Auswahl nicht relevanter Ereignisse darstellt.*
2. *(Späte Reduktion) Ein Ereignis* $e \in \mathcal{E}$ *ist nicht relevant, wenn für alle Pfade* $p = e' \dots e$ *mit* $e' \in \mathcal{E}_{\text{del}}$ *gilt, dass die Verspätung von* e' *entlang des Pfades* p *durch Pufferzeiten kompensiert wird, das heißt wenn in einer zeitminimalen Lösung* $(\mathbf{y}, \mathbf{z})$ *bei Halten aller noch offenen Anschlüsse* $y_e = 0$ *gilt.*

Ein Algorithmus zur Durchführung der Reduktion ist in Abbildung 1.10 gegeben. Er baut auf dem im Zusammenhang mit der CPM in Abbildung 1.6 vorgestellten Algorithmus auf. Er ist daher auch geeignet, um eine obere Schranke für den optimalen Zielfunktionswert zu erhalten.

Für die Lösung des Optimierungsproblems (DM) reicht es aus, sich bei der Suche nach einem optimalen modifizierten Fahrplan auf die relevanten Ereignisse $\mathcal{E}_{\text{rel}}$ statt $\mathcal{E}$ und Aktivitäten $\mathcal{A}_{\text{rel}}$ statt $\mathcal{A}$ zu beschränken. Dies erleichtert im Allgemeinen die Optimierung aus zweierlei Gründen:

- Durch die Beschränkung auf weniger Ereignisse und Aktivitäten nimmt die Anzahl von Entscheidungsvariablen und Nebenbedingungen, teilweise drastisch, ab.

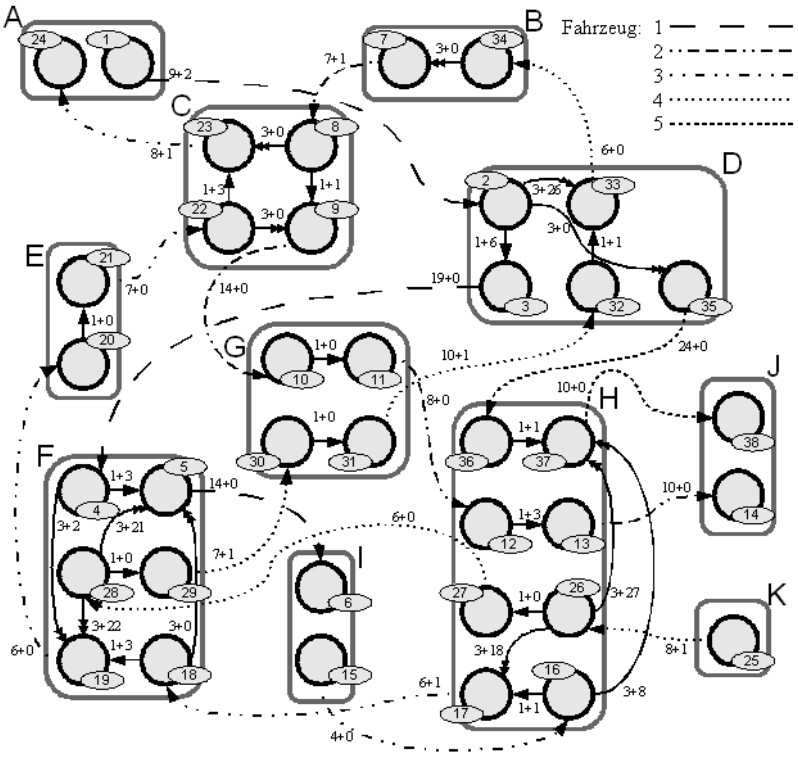


Abbildung 1.8: Ursprüngliches Ereignis-Aktivitäts-Netzwerk gemäß der Tabellen 1.1 und 1.2

- Das ursprüngliche Problem zerfällt unter Umständen in verschiedene unabhängige Teilprobleme, die getrennt voneinander, auch parallel, gelöst werden können.

Es ist also sinnvoll, das Anschlusssicherungsproblem auf folgende Weise anzugehen: Zunächst sollte das Ereignis-Aktivitäts-Netzwerk $\mathcal{N}$ erzeugt werden. Dann wird das Netzwerk reduziert, so dass es möglicherweise in $n \in \mathbb{N}$ isolierte Netzwerke $\mathcal{N}_1, \ldots, \mathcal{N}_n$ zerfällt: $\mathcal{N}_1 \cup \ldots, \cup \mathcal{N}_n = \mathcal{N}_{\text{rel}}$, dabei $\mathcal{E}_i \cap \mathcal{E}_j = \emptyset$ für $i, j = 1, \ldots, n$ mit $i \neq j$. Löse dann jedes dieser Teilprobleme, das heißt n verschiedene Instanzen von (DM), und wende die Ergebnisse auf das ursprüngliche Netzwerk an, das heißt halte alle Anschlüsse a mit $z_a = 1$, die mit $z_a = 0$ können nicht gehalten werden. Lasse dann jedes Ereignis e zum Zeitpunkt $\Pi_e + y_e$ stattfinden. Dies führt dann zu einem Gesamtverspätungswert von $\sum_{e \in \mathcal{E}} w_e y_e + \sum_{a \in \mathcal{A}_{\text{change}}} w_a z_a T$ und produziert einen zulässigen modifizierten Fahrplan.

Teilprobleme können beispielsweise durch Erweiterung des Schrittes 3 in Abbildung 1.10 identifiziert werden: Sei K die Anzahl der bereits vorliegenden isolierten Teilprobleme und $e \in \mathcal{E}$. Gilt $a \notin \mathcal{A}^{\text{miss}}$ und $y_e - s_a > 0$ für eine Aktivität $a = (e, e')$, so sind e, e' und a relevant. Setze $y_{e'} = \max(y_e - s_a, y_{e'})$. Gilt $e \in \mathcal{E}_k$ für ein $k \leq K$, so setze $\mathcal{E}_k = \mathcal{E}_k \cup \{e'\}$ und $\mathcal{A}_k = \mathcal{A}_k \cup \{a\}$. Sonst setze $K = K + 1$, $\mathcal{E}_K = \{e, e'\}$ und $\mathcal{A}_K = \{a\}$. Dabei kann es vorkommen, dass in einem Schritt jeweils zwei zuvor isolierte Teilprobleme zu einem verbunden werden, da $e \in \mathcal{E}_{e_{k_1}}, e' \in \mathcal{E}_{e_{k_2}}$ mit $e_{k_1}, e_{k_2} \leq K$ und $e_{k_1} \neq e_{k_2}$. In diesem Fall wird eine Umnummerierung der Teilprobleme erforderlich. Setze dann zusätzlich $K = K - 1$.

Das Ergebnis sind K disjunkte Zusammenhangskomponenten $\mathcal{N}_1, \ldots, \mathcal{N}_K$ des ursprünglichen Netzwerks $\mathcal{N}$, die für K unabhängige Teilprobleme stehen.

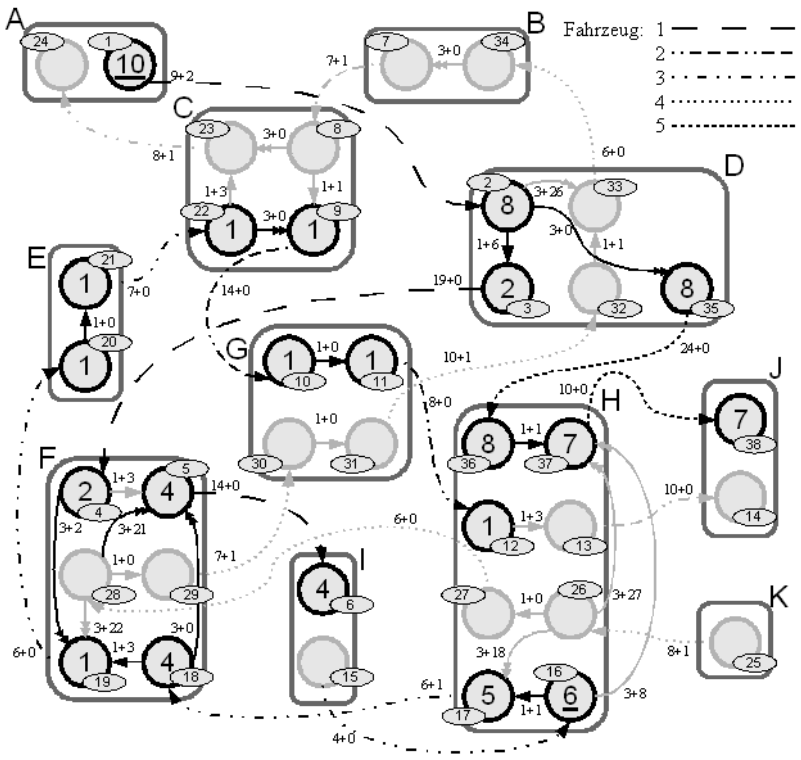


Abbildung 1.9: Reduziertes Ereignis-Aktivitäts-Netzwerk gemäß der Tabellen 1.1 und 1.2

Eingabe: $\mathcal{N}$, d_e, s_a.
Ausgabe: Relevante Ereignisse ($\mathcal{E}_{\text{rel}}$) und Anschlüsse ($\mathcal{A}_{\text{rel}}$).
Schritt 1. Sortiere $\mathcal{E} = \{e_1, \ldots, e_{|\mathcal{E}|}\}$ gemäß $\prec$. Setze $\mathcal{E}_{\text{rel}} = \emptyset$, $\mathcal{A}_{\text{rel}} = \mathcal{A}$.
Schritt 2. Setze $y_e = \max(d_e, 0)$ für alle $e \in \mathcal{E}$.
Schritt 3. Für alle $e \in \mathcal{E}$ in der sortierten Reihenfolge:

$y_e = 0$: e ist nicht relevant.
Setze $\mathcal{A}_{\text{rel}} = \mathcal{A}_{\text{rel}} \setminus \{a\}$ für alle Aktivitäten $a = (e, e')$ mit $e' \in \mathcal{E}$.

$y_e > 0$: Setze $\mathcal{E}_{\text{rel}} = \mathcal{E}_{\text{rel}} \cup \{e\}$.
Für alle $a = (e, e')$ mit $e' \in \mathcal{E}$ in sortierter Reihenfolge:

a) $a \in \mathcal{A}^{\text{miss}}$: a nicht relevant. Setze $\mathcal{A}_{\text{rel}} = \mathcal{A}_{\text{rel}} \setminus \{a\}$.

b) $a \notin \mathcal{A}^{\text{miss}}$ und $y_e - s_a > 0$: e, e' und a sind relevant. Setze $\mathcal{E} = \mathcal{E} \cup \{e, e'\}$ und $y_{e'} = \max(y_e - s_a, y_{e'})$.

c) $a \notin \mathcal{A}^{\text{miss}}$ und $y_e - s_a \leq 0$: a ist nicht relevant. Falls $a \in \mathcal{A}_{\text{change}}$ setze $\mathcal{A}^{\text{fix}} = \mathcal{A}^{\text{fix}} \cup \{a\}$ und $\mathcal{A}^{\text{open}} = \mathcal{A}^{\text{open}} \setminus \{a\}$.

Abbildung 1.10: Algorithmus: Reduktion des Ereignis-Aktivitäts-Netzwerks

2 Ganzzahlige Programmierung für das Anschlusssicherungsproblem

In Abschnitt 1.3 wurde ein gemischt-ganzzahliges Programm zur Modellierung des Anschlusssicherungsproblems vorgestellt. Dieses Kapitel befasst sich mit der Frage, wie solch ein Programm gelöst werden kann. Als ein Lösungsverfahren wird in Abschnitt 2.1 das Branch-and-Bound Verfahren eingeführt.

Aus Abschnitt 2.2 schließlich geht eine implementierungsnahe Variante des Branch-and-Bound Verfahrens hervor, welche speziell auf das Anschlusssicherungsproblem zugeschnitten ist und zusätzlich das in Abschnitt 1.5 dargestellte Reduktionsverfahren verwendet.

Das Branch-and-Bound Verfahren ist ein mächtiges Werkzeug zur Bestimmung der Optimallösung. Allerdings weist es unter Umständen eine sehr hohe Laufzeit auf. Bei zeitkritischen Anwendungen kann es sinnvoll sein, sich mit schnelleren, aber nicht notwendig exakten Verfahren zu befassen. Solche *Heuristiken* werden in Abschnitt 2.3 präsentiert.

2.1 Das Branch-and-Bound Verfahren

Für die Optimierung einer linearen Zielfunktion unter linearen, kontinuierlichen Nebenbedingungen, auch *Lineare Programmierung*[1] genannt, existieren effiziente Algorithmen wie beispielsweise das Simplexverfahren oder das Innere-Punkte-Verfahren.

Sobald es notwendig wird, dass einige Variablen ganzzahlig sind, sind diese Verfahren nicht mehr geeignet. Das wesentliche Problem bei der so genannten *gemischt-ganzzahligen Programmierung* ist, dass die Notwendigkeit, dass Variablen ganzzahlig sind, sich im Allgemeinen nicht mehr einfach durch lineare Nebenbedingungen formulieren lässt.

Verfahren zur Lösung gemischt-ganzzahliger (oder auch speziell echt ganzzahliger) Programme bauen daher üblicherweise darauf auf, die Menge der zulässigen Lösungen schrittweise zu unterteilen (englisch: *branching*) und die resultierenden Teilprobleme als Nachkommen des vorhergehenden Problems zu untersuchen. Hierdurch entsteht nach und nach ein Aufzählungsbaum, bei dem jeder Knoten ein Teilproblem repräsentiert. Je weiter man sich von der Wurzel entfernt, desto einfacher werden die Teilprobleme. Durch geschickte Bestimmung geeigneter Schranken (englisch: *bounding*), vorrangig durch Lösen von Relaxationen des Problems für untere beziehungsweise durch Heuristiken für obere Schranken, können dabei viele Teilprobleme von der genaueren Betrachtung ausgeschlossen werden, da von ihnen kein sinnvoller Beitrag zu einer guten Lösung zu erwarten ist. Dies entspricht dem Abschneiden von Zweigen im Suchbaum (englisch: *pruning*). Verschiedene derartige Verfahren werden in [22] vorgestellt.

Als ein solches enumeratives Verfahren wird hier das eigentliche Branch-and-Bound Verfahren und einige Suchstrategien näher betrachtet. Als zusätzliche weiter gehende Literatur sei auf [31] verwiesen, dort wird eine beispielorientierte Einführung gegeben. Etwas ausführlicher sind die Publikationen [14], [15], wobei die erste zusätzlich Implementierungen des Verfahrens und vorhandene Schnittstellen hervorhebt, während die zweite vordergründig die Auswirkungen verschiedener Verzweigungs- und Suchstrategien im Baum numerisch vergleicht.

Bevor jedoch das eigentliche Verfahren vorgestellt werden kann, müssen einige grundlegende Begriffe vorgestellt werden.

[1] Der Begriff *Programmierung* geht in diesem Zusammenhang auf George Dantzig, den Erfinder des Simplexverfahrens, zurück: Die Planungsprobleme des US-Militärs, für welche er seine Verfahren einsetzte, wurden als *Programme* bezeichnet, siehe hierzu [5].

2.1.1 Untere und obere Schranken

Definition 2.1.1 (Relaxation) *Sei für Mengen S, S_{R} mit $S \subseteq S_{\mathrm{R}} \subseteq \mathbb{R}^n$ und eine Bewertungsfunktion $f : \mathbb{R}^n \to \mathbb{R}$ ein Optimierungsproblem (P) gegeben durch $\min\{f(\mathbf{x}) : \mathbf{x} \in S\}$. Eine* **Relaxation** *(P_{R}) von (P) ist gegeben durch $\min\{f(\mathbf{x}) : \mathbf{x} \in S_{\mathrm{R}}\}$.*

Lemma 2.1.2 *Seien S, S_{R}, f, (P), (P_{R}) wie in Definition 2.1.1 gegeben. Sei ferner $\mathbf{x}^\star \in S$ die Optimallösung von (P), $\mathbf{x}^{\mathrm{R}} \in S_{\mathrm{R}}$ die Optimallösung von (P_{R}). Sei außerdem $\mathbf{x} \in S$ eine beliebige für (P) zulässige Lösung. Dann gilt*

1. $f(\mathbf{x}^{\mathrm{R}}) \leq f(\mathbf{x}^\star) \leq f(\mathbf{x})$,
2. *falls $\mathbf{x}^{\mathrm{R}} \in S$ so ist $\mathbf{x}^{\mathrm{R}}$ optimal für (P).*

Beweis:

1. $\mathbf{x}^\star$ ist zulässig für (P_{R}), damit folgt $f(\mathbf{x}^{\mathrm{R}}) \leq f(\mathbf{x}^\star)$ aus der Optimalität von $\mathbf{x}^{\mathrm{R}}$ für (P_{R}). Die zweite Ungleichung folgt aus der Optimalität von $\mathbf{x}^\star$ für (P).
2. Gelte $\mathbf{x}^{\mathrm{R}} \in S$. Angenommen, es gäbe ein $\mathbf{x} \in S$ mit $f(\mathbf{x}) < f(\mathbf{x}^{\mathrm{R}})$. Dann gilt insbesondere $\mathbf{x} \in S_{\mathrm{R}}$, $\mathbf{x}$ ist also zulässig für (P_{R}), somit kann $\mathbf{x}^{\mathrm{R}}$ nicht optimal für (P_{R}) gewesen sein. Es folgt, dass $\mathbf{x}^{\mathrm{R}}$ bereits optimal für (P) gewesen sein muss. □

Relaxationen führen also zu unteren Schranken für den optimalen Zielfunktionswert des Problems, während jede zulässige Lösung zu einer oberen Schranke führt. Verfahren, die ohne allzu großen Aufwand zu zulässigen Lösungen mit niedrigem Zielfunktionswert führen, bezeichnet man als *Heuristiken*.

Sei ein gemischt-ganzzahliges Programm (MIP) mit n_1 kontinuierlichen und n_2 ganzzahligen Entscheidungsvariablen gegeben in der Form

$$\begin{aligned} \min \quad & z = {\mathbf{c}^{(1)}}^\top \mathbf{x}^{(1)} + {\mathbf{c}^{(2)}}^\top \mathbf{x}^{(2)} \\ \text{so dass} \quad & \mathbf{A}^{(1)}\mathbf{x}^{(1)} \leq \mathbf{b}^{(1)}, \\ & \mathbf{A}^{(2)}\mathbf{x}^{(2)} \leq \mathbf{b}^{(2)}, \\ & \mathbf{x}^{(1)}, \mathbf{x}^{(2)} \geq 0, \\ & x_i^{(2)} \in \mathbb{N}_0 \text{ für } i = 1, \ldots, n_2. \end{aligned}$$

Die Nebenbedingungen beschränken die zulässigen Lösungen auf eine Menge $S \subseteq \mathbb{R}^{n_1} \times \mathbb{N}_0^{n_2}$, damit lässt sich das Optimierungsproblem (MIP) auch schreiben als

$$\min\left\{ {\mathbf{c}^{(1)}}^\top \mathbf{x}^{(1)} + {\mathbf{c}^{(2)}}^\top \mathbf{x}^{(2)} : \left({\mathbf{x}^{(1)}}^\top, {\mathbf{x}^{(2)}}^\top\right)^\top \in S \right\}.$$

Lässt man die Ganzzahligkeitsbedingung für $\mathbf{x}^{(2)}$ fallen, erhält man eine Menge S_R mit der Eigenschaft $S \subset S_{\mathrm{R}} \subseteq \mathbb{R}^{n_1+n_2}$. Das lineare Optimierungsproblem ($\mathsf{MIP}_{\mathrm{R}}$)

$$\min\left\{ {\mathbf{c}^{(1)}}^\top \mathbf{x}^{(1)} + {\mathbf{c}^{(2)}}^\top \mathbf{x}^{(2)} : \left({\mathbf{x}^{(1)}}^\top, {\mathbf{x}^{(2)}}^\top\right)^\top \in S_{\mathrm{R}} \right\}$$

ist eine Relaxation von (MIP), die so genannte *LP-Relaxation*.

2.1.2 Vorstellung des Branch-and-Bound Verfahrens

Sei ein gemischt-ganzzahliges Programm (MIP) wie im vorangehenden Abschnitt gegeben. Wie am Anfang dieses Abschnitts angedeutet, besteht das Branch-and-Bound Verfahren aus mehreren Phasen, die wiederholt nacheinander abgearbeitet werden.

1. (Auswahl): Aus einer Menge von Teilproblemen $\mathcal{L}$, welche zunächst nur (MIP) enthält, wird eines ausgewählt. Auf welche Weise diese Auswahl geschieht wird durch die *Suchstrategie* im Aufzählungsbaum bestimmt.

2. (Schrankenbestimmung): Löse die LP-Relaxation des ausgewählten Teilproblems, um eine untere Schranke für den optimalen Zielfunktionswert von (MIP) zu erhalten. Steht eine geeignete Heuristik zur Verfügung, verwende diese um eine obere Schranke für den optimalen Zielfunktionswert zu erhalten. Ist sie kleiner als die bisherige obere Schranke so nimmt die neue Schranke den Platz der alten ein.
3. (Bereinigung): An dieser Stelle kann es aus mehreren Gründen sinnvoll sein, das aktuelle Teilproblem zu verwerfen und zum nächsten überzugehen, das heißt wieder zu 1. zu gehen:
   a) (*pruning by infeasability*): Die Relaxation des untersuchten Teilproblems hat keine zulässige Lösung.
   b) (*pruning by bound*): Die in 2. bestimmte untere Schranke ist schlechter als die kleinste bisher bestimmte obere Schranke. Vom aktuellen Teilproblem ist somit keine bessere Lösung zu erwarten.
   c) (*pruning by optimality*): Die bei der LP-Relaxation bestimmte Lösung $\mathbf{x}^{\mathrm{R}}$ ist zulässig für (MIP). Verwende den Zielfunktionswert als neue obere Schranke für die Optimallösung von (MIP) und behalte $\mathbf{x}$ als bisher beste Lösung.
4. (Verzweigung): Wähle einen Index i aus, für den die Lösung der Relaxation gegen eine Ganzzahligkeitsbedingung von (MIP) verstößt. Erzeuge aus dem für das aktuelle Problem zulässigen Bereich zwei neue Mengen, indem der Wertebereich der i-ten Variablen einmal von oben durch $\lfloor x_i^{\mathrm{R}} \rfloor$ und einmal von unten durch $\lfloor x_i^{\mathrm{R}} \rfloor + 1$ beschränkt wird. Dies sind die zulässigen Bereiche zweier neuer Teilprobleme, die der Menge $\mathcal{L}$ hinzuzufügen sind. Welcher Index i gewählt wird, ergibt sich aus der verfolgten *Verzweigungsstrategie*.

Sofern der für (MIP) zulässige Bereich beschränkt ist, ist ein aus dieser Vorgehensweise hervorgehender Aufzählungsbaum stets endlich, wie beispielsweise in [22], Proposition 2.1 gezeigt wird. Eine kompakte Fassung des Branch-and-Bound Verfahrens als Algorithmus ist in Abbildung 2.1 dargestellt.

Eingabe: Ein gemischt-ganzzahliges Programm (MIP).
Ausgabe: Eine optimale Lösung von (MIP) oder die Aussage, dass keine solche Lösung existiert oder dass das Problem nach unten unbeschränkt ist.
Schritt 1. Setze $\mathcal{L} = \{(\mathrm{MIP})\}$, $\mathcal{X} = \emptyset$, $f^\star = \infty$.
Schritt 2. Gilt $\mathcal{L} = \emptyset$, dann terminiere. $\mathcal{X}$ enthält eine Optimallösung von (MIP), sofern eine Lösung existiert.
Schritt 3. Wähle ein Problem (P) aus $\mathcal{L}$ aus und löse die LP-Relaxation ($\mathrm{P_R}$). Ist sie unzulässig, gehe zu 2. Ist ($\mathrm{P_R}$) nach unten unbeschränkt so gilt dies auch für (P), terminiere daher. Sei ansonsten $\mathbf{x}^{\mathrm{R}}$ eine optimale Lösung von ($\mathrm{P_R}$) und f^{R} der zugehörige Zielfunktionswert.
Schritt 4. Wenn eine geeignete Heuristik zur Verfügung steht, dann bestimme damit eine für (P) zulässige Lösung $\hat{\mathbf{x}}$ mit Zielfunktionswert $\hat{f}$. Gilt $\hat{f} < f^\star$, so setze $f^\star = \hat{f}$ und $\mathcal{X} = \{\hat{\mathbf{x}}\}$.
Schritt 5. Gilt $f^{\mathrm{R}} \geq f^\star$ so gehe zu 2.
Schritt 6. Ist $\mathbf{x}^{\mathrm{R}}$ zulässig für (P) setze $f^\star = f^{\mathrm{R}}$, $\mathcal{X} = \{\mathbf{x}^{\mathrm{R}}\}$. Gehe zu 2.
Schritt 7. Wähle einen Index i aus, so dass x_i^{R} gegen eine Ganzzahligkeitsbedingung von (P) verstößt. Erzeuge zwei neue Probleme ($\mathrm{P_1}$) und ($\mathrm{P_2}$) aus (P), indem der Wertebereich von x_i einmal von oben durch $\lfloor x_i^{\mathrm{R}} \rfloor$ und einmal von unten durch $\lfloor x_i^{\mathrm{R}} \rfloor + 1$ beschränkt wird. Füge ($\mathrm{P_1}$) und ($\mathrm{P_2}$) zu $\mathcal{L}$ hinzu und gehe zu 2.

Abbildung 2.1: Algorithmus: Das Branch-and-Bound Verfahren

Das Laufzeitverhalten des Verfahrens kann entscheidend durch Modifikation der Such- und Verzweigungsstrategien beeinflusst werden. Je nach Art des zu lösenden Problems können unterschiedliche Vorgehensweisen sinnvoll sein. Dabei ist aber zu beachten, dass durch den Einsatz ausgereifter Strategien ein zusätzlicher Aufwand entstehen kann, der nicht zu vernachlässigen ist.

2.1.3 Suchstrategien

Die Suchstrategie gibt vor, welches Problem aus der Menge $\mathcal{L}$ jeweils als nächstes betrachtet werden soll. Hauptziele sind dabei einerseits die Bestimmung einer möglichst guten Lösung und andererseits die Bestätigung, dass gegebenenfalls keine bessere als die aktuell beste Lösung existiert.

Man unterteilt dabei grob in *a priori* Regeln, welche eine vorher festgelegte Auswahlregel vorgeben, und *adaptive* Regeln, welche die Auswahl von aktuell vorliegender Information, zum Beispiel über Schranken oder andere in der Menge enthaltene Probleme, abhängig machen. In dieser Arbeit werden lediglich statische a priori Regeln vorgestellt, weitere Regeln sind beispielsweise in [15] zu finden. Jede Suchregel wird durch ein grafisches Beispiel illustriert, wobei die Zahl innerhalb eines Knotens dafür steht, in welcher Iteration das entsprechende Problem bearbeitet wird. Die erste Zahl unterhalb eines Knotens steht für die aus einer Heuristik resultierende obere Schranke für den Zielfunktionswert, während die untere Zahl die aus der LP-Relaxation gewonnene untere Schranke darstellt.

Tiefensuche

Die *Tiefensuche* (englisch: *Depth-First Search*) hat das Ziel, durch Betrachtung des zuletzt zu $\mathcal{L}$ hinzugefügten Problems möglichst früh möglichst tief in den Aufzählungsbaum vorzudringen. Dies ist eine *last in, first out* Strategie, kurz LIFO. Sie hat den Vorteil, dass schnell zulässige Lösungen bestimmt werden und dadurch obere Schranken verfügbar sind. Zusätzlich hat diese Vorgehensweise den Vorteil, dass sich die Teilprobleme in jedem Schritt kaum ändern: In den meisten Fällen ändert sich lediglich eine Beschränkung des Wertebereichs für eine Variable. Außerdem verhält sich der Speicherbedarf linear zur Tiefe des Baumes. Allerdings kann es unter Umständen sehr lange dauern, bis tatsächlich Lösungen bestimmt werden, die zu guten Schranken beziehungsweise der Optimallösung führen, da diese oftmals in der Mitte des Aufzählungsbaumes liegt, während sich die Tiefensuche üblicherweise von einer Seite her annähert. Diese Strategie lässt sich durch Verwendung der Datenstruktur *Stack* für die Menge $\mathcal{L}$ realisieren.

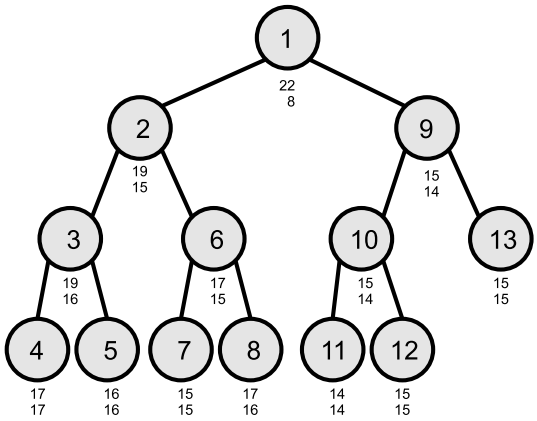


Abbildung 2.2: Beispiel für die Tiefensuche

Breitensuche

Bei der *Breitensuche* (englisch: *Breadth-First Search*) wird immer jenes Teilproblem betrachtet, welches sich am längsten in der Menge $\mathcal{L}$ befindet. Dies führt dazu, dass immer Teilprobleme mit minimalem Abstand von dem Wurzelknoten im Aufzählungsbaum untersucht werden. Der Baum wird also der Breite nach durchsucht. Zusammen mit einer guten Heuristik können auf diese Weise schnell gute Schranken für den optimalen Zielfunktionswert bestimmt werden. Auf diese Weise müssen unter Umständen nicht so viele Knoten untersucht werden. Allerdings gibt es nur relativ wenige Zusammenhänge zwischen den Teilproblemen, so dass der Übergang von einem Problem

zum nächsten aufwendiger ist als bei der Tiefensuche. Ein weiterer Nachteil ist, dass die Menge $\mathcal{L}$ unter Umständen exponentiell viele Teilprobleme aufnehmen muss. Die zugrunde liegende Strategie lässt sich durch Verwendung der Datenstruktur *Liste* beziehungsweise *Warteschlange* für die Menge $\mathcal{L}$ realisieren.

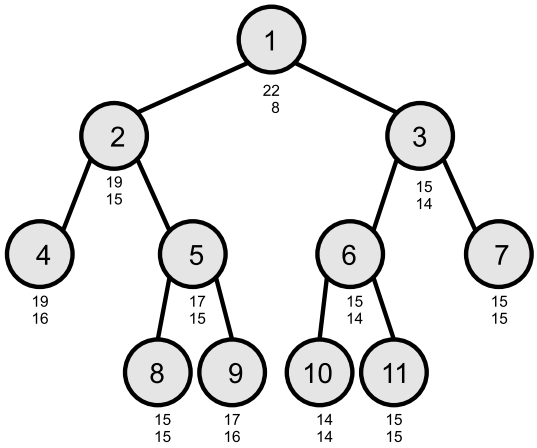


Abbildung 2.3: Beispiel für die Breitensuche

Bestensuche

Bei der Bestensuche werden die in Schritt 7 erzeugten Teilprobleme mit dem Zielfunktionswert der zuvor untersuchten LP-Relaxation verknüpft. Die Auswahlstrategie basiert nun darauf, jeweils ein Problem mit minimaler unterer Schranke auszuwählen. Dadurch, dass die unteren Schranken klein bleiben, werden auch schnell kleine obere Schranken bestimmt. Diese Suchstrategie erscheint somit besonders vielversprechend. Allerdings besteht auch hier zwischen zwei nacheinander betrachteten Teilproblemen nur wenig Zusammenhang. Auch der Speicheraufwand ist unter Umständen wieder exponentiell. Die Bestensuche lässt sich mit Hilfe einer *Prioritätenwarteschlange* für die Menge $\mathcal{L}$ umsetzen, wobei Probleme mit kleiner unterer Schranke am Anfang der Warteschlange stehen.

Der Zielfunktionswert der LP-Relaxation setzt sich aus zweierlei Komponenten zusammen: Einmal gibt es einen Beitrag, der durch bereits festgelegte Variablen entsteht. Der Beitrag dieser Variablen wird sich im Teilbaum unterhalb eines Knotens niemals ändern. Man kann diesen Beitrag daher mit einer Bewertung der bisherigen Suche identifizieren. Der verbleibende Zielfunktionswert ergibt sich anhand der noch nicht festgelegten Variablen, deren Beitrag zum Zielfunktionswert einer zulässigen Lösung mit Hilfe der LP-Relaxation nach unten abgeschätzt wird. Eine Überschätzung des Zielfunktionswertes einer Lösung mit den im betrachteten Knoten vorliegenden Variablenbeschränkungen ist nach Lemma 2.1.2 nicht möglich. Somit kann die hier beschriebene Suchstrategie insbesondere als A*-Suche aufgefasst werden, auch wenn sie in der Literatur, beispielsweise in [15], üblicherweise als *Bestensuche* (englisch: *Best-First Search*) beschrieben wird.

Es ist zu beachten, dass durch die Bestensuche im Allgemeinen höchstens genau so viele Teilprobleme betrachtet werden wie bei anderen Suchstrategien: Angenommen, es gäbe ein Teilproblem (P), das von der Bestensuche betrachtet wird, von anderen Suchstrategien jedoch nicht. Bezeichne mit $f^\star$ den optimalen Zielfunktionswert und mit f^{R} den Zielfunktionswert jener LP-Relaxation, aus welcher (P) aus Beschränkung einer Variablen resultiert – also der mit dem Teilproblem (P) verknüpfte Wert. Es muss gelten $f^{\mathrm{R}} \leq f^\star$, da (P) sonst nicht von der Bestensuche betrachtet würde. Gilt $f^{\mathrm{R}} < f^\star$ dann muss (P) auch von anderen Suchstrategien betrachtet werden, da ansonsten keine Optimalität garantiert werden kann: Schließlich könnte (P) jenes Teilproblem als Nachkomme enthalten, welches die Optimallösung als LP-Relaxation hat. Daher müssen bei der Bestensuche tatsächlich höchstens genau so viele Teilprobleme betrachtet werden wie bei anderen Suchstrategien. Ausnahmen können nur für jene Teilprobleme auftreten, welche mit einem Zielfunktionswert verknüpft sind, der dem optimalen Zielfunktionswert entspricht. Davon gibt es allerdings üblicherweise nur sehr wenige. Bei der A*-Suche bezeichnet man diese Eigenschaft allgemein

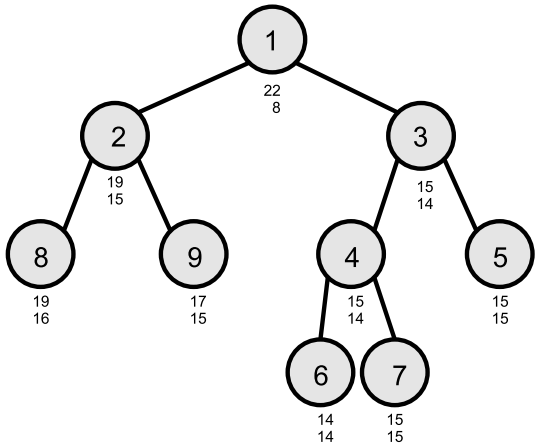


Abbildung 2.4: Beispiel für die Bestensuche (A)*

auch als *optimale Effizienz*.

Eine allgemeine Darstellung dieser Suchstrategien, also nicht nur bezogen auf Suche in einem Branch-and-Bound Aufzählungsbaum, findet man beispielsweise in [23]. Laufzeitvergleiche für die vorgestellten Suchstrategien befinden sich im Kapitel 6 dieser Arbeit.

2.1.4 Verzweigungsstrategien

Neben der Reihenfolge, in der bereits erzeugte Knoten abgearbeitet werden, kann das Verhalten des Branch-and-Bound Verfahrens auch dadurch angepasst werden, in welcher Reihenfolge die einzelnen Variablen zur Verzweigung ausgewählt werden.

Die einfachsten Strategien geben einen Kandidaten fest vor, beispielsweise indem immer der erste oder auch der letzte nicht-ganzzahlige Variablenindex gewählt wird.

Aufwendigere Strategien bewerten zunächst jede nicht-ganzzahlige Variable nach einem geeigneten Schema und wählen dann die Variable mit der besten Bewertung aus. Die Bewertung kann sich beispielsweise aus der Differenz zwischen der jeweiligen Variable in der Lösung der LP-Relaxation und der nächstliegenden ganzen Zahl (*most infeasible branching*), aus einer Abschätzung des Einflusses der ganzzahligen Rundung auf die Zielfunktion (*pseudocost branching*) oder aus der vorgezogenen Lösung von LP-Relaxationen mit Ganzzahligkeitsanforderungen (*strong branching*). Dies sind nur einige der bekannten und in der Praxis angewandten Strategien, eine detaillierte Aufstellung und Erläuterung verschiedener Verzweigungsstrategien findet man in [1].

Die Wahl der richtigen Verzweigungsstrategie hängt dabei selbstverständlich immer auch vom untersuchten Problem ab. Für die Lösung des Anschlusssicherungsproblems wurde stets jene Variable zur Verzweigung gewählt, welche dem frühsten noch nicht festgelegten Anschluss entspricht, da von diesem der größte Einfluss auf das übrige Netzwerk zu erwarten ist. Weitere Verzweigungsstrategien wurden nicht getestet.

2.2 Reduktionsbasiertes Branch-and-Bound Verfahren

Nachdem im vorangehenden Abschnitt eine allgemeine Version des Branch-and-Bound Verfahrens vorgestellt wurde, befasst sich dieser Abschnitt mit einer speziell an das Anschlusssicherungsproblem angepassten Variante. Die spezielle Struktur des vorliegenden Ereignis-Aktivitäts-Netzwerks wird dabei ausgenutzt, um mit Hilfe der in Abschnitt 1.5 vorgestellten Reduktionen wiederholt möglichst viele Variablen festlegen und dabei gewonnene obere Schranken in das Branch-and-Bound Verfahren einbeziehen zu können.

Dieses *reduktionsbasierte* Branch-and-Bound Verfahren ist in Abbildung 2.5 dargestellt. Dabei ist zu beachten, dass die LP-Relaxationen stets eine zulässige Lösung haben: Schließlich lässt sich für jede Festlegung der Anschlüsse ein zulässiger modifizierter Fahrplan erstellen. Da das Verfahren speziell auf das Anschlusssicherungsproblem ausgerichtet ist, wird die in Abschnitt 1.2.2 eingeführte Variablennotation verwendet, das heißt $\mathbf{y} \in \mathbb{N}_0^{|\mathcal{E}|}$ bezeichnet den Verspätungsvektor, während $\mathbf{z} \in \{0,1\}^{|\mathcal{A}_{\text{change}}|}$ für den Vektor der Anschlusssvariablen steht. Die Ausnutzung der Möglichkeit, dass das ursprüngliche Problem nach der ersten Reduktion in unabhängige Teilprobleme zerfällt wird dabei nicht berücksichtigt. Es ist aber denkbar, den Algorithmus für jedes Teilproblem einmal auszuführen und bei jeder Ausführung in Schritt 2 anstatt $\mathcal{N}_{\text{rel}}$ ein Subnetzwerk $\mathcal{N}_i$ zu betrachten. Als weitere Modifikationen sind auch hier verschiedene Such- und Verzweigungsstrategien denkbar. Von den vorgestellten Suchstrategien hat sich die Bestensuche als die vielversprechendste erwiesen.

Eingabe: Ein zulässiger Fahrplan $\mathbf{\Pi}$ mit Pufferzeiten $\mathbf{s}$ und Quellverspätungen $\mathbf{d}$.
Ausgabe: Eine Optimallösung für (DM).
Schritt 1. Erstelle anhand von $\mathbf{\Pi}$ und $\mathbf{s}$ ein Ereignis-Aktivitäts-Netzwerk $\mathcal{N}$ und trage Quellverspätungen $\mathbf{d}$ ein. Führe eine erste Reduktion durch.
Schritt 2. Erstelle anhand von $\mathcal{N}_{\text{rel}}$ ein gemischt-ganzzahliges Programm (MIP) gemäß Abbildung 1.7 und füge es zu $\mathcal{L}$ hinzu. Setze $f^\star = \infty$.
Schritt 3. Gilt $\mathcal{L} = \emptyset$ dann terminiere. $(\mathbf{y}^\star, \mathbf{z}^\star)$ ist die Optimallösung. Sonst: Wähle ein Problem (P) aus $\mathcal{L}$ aus.
Schritt 4. Reduziere das durch (P) repräsentierte Ereignis-Aktivitäts-Netzwerk und übertrage daraus resultierende Vereinfachungen durch Variablenfixierung auf (P).
Schritt 5. Steht durch eine Heuristik (beispielsweise durch Festlegen der offenen Anschlüsse und Bestimmen einer zeitminimalen Lösung) eine zulässige Lösung $(\hat{\mathbf{y}}, \hat{\mathbf{z}})$ mit Zielfunktionswert $\hat{f}$ zur Verfügung, so dass $\hat{f} < f^\star$ gilt, setze $f^\star = \hat{f}$, $(\mathbf{y}^\star, \mathbf{z}^\star) = (\mathbf{y}, \mathbf{z})$.
Schritt 6. Löse die LP-Relaxation von (P), erhalte $(\mathbf{y}^{\text{R}}, \mathbf{z}^{\text{R}})$ als Optimallösung und f^{R} als zugehörigen Zielfunktionswert.
Schritt 7. Falls $f^{\text{R}} \geq f^\star$ gehe zu 3.
Schritt 8. Falls $\mathbf{z}^{\text{R}}$ ganzzahlig setze $f^\star = f^{\text{R}}$, $(\mathbf{y}^\star, \mathbf{z}^\star) = (\mathbf{y}^{\text{R}}, \mathbf{z}^{\text{R}})$, gehe zu 3.
Schritt 9. Erzeuge zwei neue gemischt-ganzzahlige Programme (P_1) und (P_2) dadurch, dass die frühste noch offene Umsteigeaktivität einmal als *gehalten* und einmal als *verpasst* festgelegt wird. Füge (P_1) und (P_2) zu $\mathcal{L}$ hinzu, gehe zu 3.

Abbildung 2.5: Algorithmus: Reduktionsbasiertes Branch-and-Bound Verfahren

2.3 Heuristiken

Verfahren, die der Bestimmung einer zulässigen Lösung mit möglichst gutem Zielfunktionswert ohne allzu großen Aufwand dienen, nennt man *Heuristiken*. Dieser Abschnitt stellt zwei Arten von Heuristiken vor, die für das Anschlusssicherungsproblem geeignet sind. Ein wesentlicher Ansatzpunkt dabei ist die Tatsache, dass aus jeder Auswahl $\mathcal{A}^{\text{fix}} \subseteq \mathcal{A}_{\text{change}}$ zu fixierender Anschlüsse mit Hilfe des Reduktionsalgorithmus aus Abbildung 1.10 mit geringem Aufwand ein zulässiger modifizierter Fahrplan und damit eine zulässige Lösung erstellt werden kann. Es werden also verschiedene Auswahlstrategien für zu haltende Anschlüsse vorgestellt, aus denen sich mit Hilfe des Reduktionsalgorithmus eine zeitminimale und damit zulässige Lösung erstellen lässt.

Dabei dient die erste Art, dargestellt in Abschnitt 2.3.1, primär der Bestimmung oberer Schranken für den optimalen Zielfunktionswert, um hiermit ein Branch-and-Bound Verfahren zu beschleunigen. Wirklich gute Schranken sind

üblicherweise allerdings erst bei wiederholtem Einsatz der Heuristik im Verlauf des Verfahrens zu erwarten. Eine Ausnahme stellt die vierte Heuristik dar, welche erstaunlich gute Ergebnisse liefert.

Die zweite Art von Heuristiken werden in Abschnitt 2.3.2 vorgestellt. Die resultierende Lösung ist erfahrungsgemäß oftmals gut genug, dass sie aufgrund der geringen Laufzeit die Stelle des exakten Verfahrens einnehmen kann.

2.3.1 Fixieren von Anschlüssen nach festen Regeln

Hold all

Die einfachste zulässige Lösung ergibt sich sicherlich daraus, dass alle noch offenen Anschlüsse gehalten werden. Dieser Ansatz wird bei dem reduktionsbasierten Branch-and-Bound Verfahren aus Abschnitt 2.2 automatisch in Schritt 4 verfolgt. Bedauerlicherweise resultieren aus dieser Vorgehensweise üblicherweise zu viele Folgeverspätungen.

Integer Rounding

Ein anderer Ansatzpunkt besteht darin, die z_a-Variablen in der optimalen Lösung der LP-Relaxation des aktuellen Problems ganzzahlig zu runden: Die mit Wert < 0.5 ergeben die zu haltenden, die mit Wert ≥ 0.5 ergeben die zu verpassenden Anschlüsse. Auf diese Weise ist zu hoffen, dass im Vergleich zum optimalen Zielfunktionswert nur ein geringer Unterschied besteht.

Miss First Delayed Change

Eine weitere Heuristik basiert auf dem Vorhaben, die Pfade, entlang denen im Ereignis-Aktivitäts-Netzwerk ausgehend von einem quellverspäteten Ereignis Verspätungen übertragen werden, möglichst kurz zu halten. Auf diesem Weg wird der ursprüngliche Fahrplan möglichst wenig verändert und alle Züge fahren so pünktlich, wie es bei Auftreten der gegebenen Verspätungen eben möglich ist. Dies geschieht durch Verpassen des jeweils ersten Anschlusses entlang solch eines Pfades. Die auf diesen Anschluss folgenden Ereignisse können dann pünktlich stattfinden, sofern keine weiteren Verspätungen auftreten.

Miss First Delayed Change if Reasonable

Die zuvor genannte Heuristik lässt sich dadurch weiter verbessern, dass iterativ für jeden offenen Anschluss, über den möglicherweise eine Verspätung übertragen wird, zunächst geprüft wird, ob aus dem Verpassen dieses Anschlusses eine Verbesserung des Zielfunktionswertes zu erwarten ist. Dies ist beispielsweise dann der Fall, wenn die gewichtete Summe der dem Anschluss nachfolgenden Verspätungen die gewichtete Periodendauer übersteigt.

Etwas formeller: Sei $a \in \mathcal{A}^{\mathrm{change}}$ ein noch nicht festgelegter Anschluss, für den es keinen offenen Anschluss $a' \in \mathcal{A}^{\mathrm{change}}$ gibt mit $a' \prec a$. Bezeichne $\mathcal{E}^0$ die Menge jener Ereignisse, die auf a folgen und dessen Folgeverspätung davon abhängt, ob a gehalten wird oder nicht. Seien y_e für $e \in \mathcal{E}^0$ zunächst die Verspätungen, die ein Halten von a zur Folge hätte. Dann sollte a nur dann gehalten werden falls $\sum_{e \in \mathcal{E}^0} w_e y_e \leq w_a T$ gilt. Wiederhole diesen Vorgang, bis alle Anschlüsse festgelegt sind.

Diese Heuristik hat bei Tests außerordentlich gut abgeschnitten und die Ergebnisse weichen nur geringfügig von der jeweiligen Optimallösung ab.

2.3.2 Genetische Verfahren

Die in diesem Abschnitt behandelten *genetischen Verfahren*, auch bekannt als *genetische Algorithmen*, gehören zu der Klasse der Metaheuristiken. Sie haben sich in der Praxis oftmals bewährt, auch wenn es keine allgemein gültigen Gütegarantien gibt. Der Name dieser Klasse von Verfahren resultiert aus der Nachahmung von Prozessen, wie man sie ansonsten bei der genbasierten natürlichen Evolution beobachtet: *Auslese*, *Kreuzung* und *Mutation* bei dem

Übergang von einer Population zur Population der nächsten Generation. Dabei erhofft man sich über die Generationen hinweg eine Verbesserung des Zielfunktionswertes für das jeweils beste Individuum der Population einer Generation von Lösungen. An dieser Stelle wird nur die Grundvariante vorgestellt. Einführungen mit raffinierteren Variationen sowie viele Anwendungsbeispiele findet man beispielsweise in [10] oder [20].

Codierung, Fitness und Population

Grundlage ist eine *Codierungsfunktion*, welche eine Problemlösung als 0-1-String darstellt sowie eine *Fitnessfunktion*, welche die Qualität einer Lösung bewertet: Je höher der Fitnesswert, desto besser die Lösung. Jeder 0-1-String wird als *Chromosom* bezeichnet, jedes Zeichen als ein *Gen*[2]. Eine Ansammlung von Chromosomen wird als *Population* bezeichnet. Ausgangspunkt eines genetischen Verfahrens ist eine Anfangspopulation mit vorgegebenem Umfang, welche beispielsweise aus zufällig generierten Chromosomen oder auch besonders vielversprechend erscheinenden Kandidaten besteht.

Bei dem Anschlusssicherungsproblem bietet es sich an, eine Lösung so zu codieren, dass jeder gehaltene Anschluss auf eine 0 und jeder verpasste Anschluss auf eine 1 abgebildet wird. Auf diese Weise ist jeder String mit einer eindeutigen Warten/Nichtwarten Entscheidung verknüpft. Jedem String kann so auch eine zeitminimale Lösung und damit ein Zielfunktionswert zugeordnet werden. Aus diesem lässt sich schließlich der Wert der Fitnessfunktion definieren, wobei allerdings noch zu berücksichtigen ist, dass gute Lösungen sich durch einen *niedrigen* Zielfunktionswert auszeichnen. Sei für eine Warten/Nichtwarten Entscheidung $\mathbf{z}$, codiert durch String $\mathbf{c}$, die zugehörige zeitminimale Lösung durch $(\mathbf{y}, \mathbf{z})$ gegeben mit Zielfunktionswert $f(\mathbf{y}, \mathbf{z})$. Dann ist durch $f_{\mathrm{GA}}(\mathbf{c}) = 1/f(\mathbf{y}, \mathbf{z})$ eine geeignete Fitnessfunktion definiert.

Selektion, Kreuzung und Mutation

Bevor die Population der nächsten Generation erzeugt werden kann, müssen zunächst aus der Population der aktuellen Generation, bezeichnet durch $\mathcal{C}$, Kandidaten ausgewählt werden. Grundlage dafür ist die zuvor beschriebene Fitnessfunktion: Je besser die Fitness eines Chromosoms bewertet wird, desto wahrscheinlicher ist eine Auswahl für die Erzeugung der nächsten Generation. Die so genannte *Roulette-Wheel-Selection*-Regel weist jedem Chromosom $c \in \mathcal{C}$ eine Wahrscheinlichkeit $p_c = \frac{f_{\mathrm{GA}}(c)}{\sum_{c' \in \mathcal{C}} f_{\mathrm{GA}}(c')}$ zu. Diese Wahrscheinlichkeit entspricht einem zur Fitness proportionalen symbolischen Bereich einer Roulette-Scheibe.

c_1	00101110100	$f(c_1)$	5
c_2	11010001111	$f(c_2)$	7
c_3	01101000001	$f(c_3)$	5
c_4	00001100100	$f(c_4)$	3
c_5	11011011101	$f(c_5)$	8
c_6	11100000101	$f(c_6)$	5
c_7	00001001000	$f(c_7)$	2
c_8	00100110000	$f(c_8)$	3
c_9	00000100001	$f(c_9)$	2

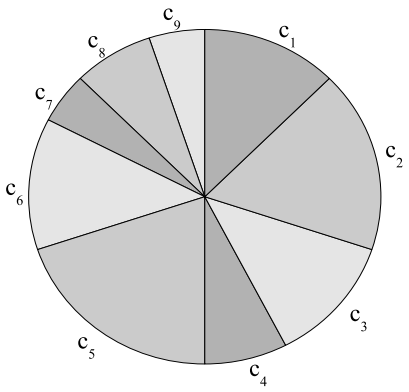


Abbildung 2.6: Neun Chromosomen mit Fitnesswerten und resultierender Roulette-Scheibe

Eine wiederholte zufällige Auswahl von Kandidaten für die nächste Generation basierend auf diesen Wahrscheinlichkeiten entspricht dann dem Rollen einer Roulettekugel, die schließlich in dem Bereich, der dem auszuwählenden

[2] Die 0-1-Codierung ist die üblichste, grundsätzlich gibt es auch andere Möglichkeiten.

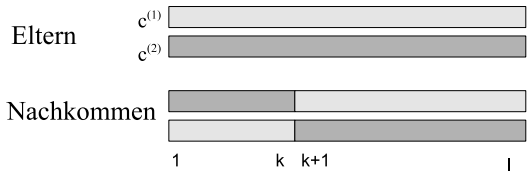


Abbildung 2.7: One-Point-Crossover

Chromosom zugeordnet ist, zum Ruhen kommt. Abbildung 2.6 zeigt ein Beispiel für solch eine Roulette-Scheibe, dabei ergibt sich die Fitness eines Chromosoms in diesem Fall aus der Anzahl der 1-Einträge.

Hat man durch wiederholte Selektion nach der betrachteten Auswahlregel eine Kandidatenmenge $\mathcal{C}^{\mathrm{cand}}$ erstellt, welche den gleichen Umfang wie die vorige Population hat, erfolgt die Rekombinationsphase: Je zwei Kandidaten $c^{(1)}$, $c^{(2)} \in \mathcal{C}^{\mathrm{cand}}$, die *Eltern*, werden mit einer zuvor festgelegter Wahrscheinlichkeit p_{co} *gekreuzt*. Findet keine Kreuzung statt werden die Kandidaten unverändert in die nächste Population aufgenommen. Die Wahl von p_{co} gibt dabei vor, wie stark sich eine Population von einer Generation zur nächsten verändern darf. Liegt p_{co} nahe bei 0, werden mit großer Wahrscheinlichkeit viele Chromosomen in der neuen Population bereits in der alten enthalten gewesen sein, während ein Wert nahe bei 1 dafür sorgt, dass die neue Population fast ausschließlich aus rekombinierten Chromosomen besteht. Allgemein hat sich ein Wert von ungefähr 0.7 als geeigneter Mittelweg bewährt.

Eingabe: Populationsgröße N, Kreuzungswahrscheinlichkeit p_{co}, Mutationswahrscheinlichkeit p_{m}, Fitnessfunktion f.
Ausgabe: Ein String, welcher einen möglichst hohen Fitnesswert hat und damit eine gute, zulässige Lösung darstellt.
Schritt 1. Erzeuge eine Ausgangspopulation $\mathcal{C}$ von N Chromosomen.
Schritt 2. Bestimme den Fitnesswert aller in $\mathcal{C}$ enthaltenen Chromosomen.
Schritt 3. Prüfe ein geeignetes Abbruchkriterium, z.B. genügende Anzahl der Generationen. Ist es erfüllt, gib den String des Chromosoms mit höchstem Fitnesswert aus und terminiere.
Schritt 4. Wähle aus $\mathcal{C}$ N Kandidaten, z.B. durch Roulette-Wheel-Selection, aus und füge sie zu einer anfangs leeren Kandidatenmenge $\mathcal{C}^{\mathrm{cand}}$. Mehrfachauswahl ist dabei erlaubt.
Schritt 5. Solange dies möglich ist wähle und entferne je zwei Kandidaten aus $\mathcal{C}^{\mathrm{cand}}$. Führe mit Wahrscheinlichkeit p_{co} eine Kreuzung dieser beiden Kandidaten durch und füge die Nachkommen zu einer anfangs leeren Menge $\mathcal{C}^{\mathrm{next}}$ hinzu. Findet keine Kreuzung statt, füge die gewählten Kandidaten unverändert zu $\mathcal{C}^{\mathrm{next}}$ hinzu. Bleibt am Ende ein einzelnes Chromosom in $\mathcal{C}^{\mathrm{cand}}$ so füge auch dieses zu $\mathcal{C}^{\mathrm{next}}$ hinzu.
Schritt 6. Für alle Chromosomen $c \in \mathcal{C}^{\mathrm{next}}$ und alle Indizes $i = 1, \ldots, |c|$:
- Ersetze c_i mit Wahrscheinlichkeit p_{m} durch $1 - c_i$.

Schritt 7. Setze $\mathcal{C} = \mathcal{C}^{\mathrm{next}}$ und gehe zu 2.

Abbildung 2.8: Genetischer Algorithmus

Bei der Kreuzung gibt es verschiedene Vorgehensweisen, an dieser Stelle wird das so genannte *One-Point-Crossover*[3] beschrieben: Ein Kreuzungsindex k wird zufällig gewählt und der Population der nächsten Generation werden zwei *Nachkommen* hinzugefügt. Diese werden durch Austausch eines Substrings der zuvor gewählten Kandidaten erzeugt. Der erste Nachkomme entsteht dann aus dem String $c_1^{(1)} \ldots c_k^{(1)} c_{k+1}^{(2)} \ldots c_\ell^{(2)}$, der zweite aus

[3] weitere übliche Kreuzungstechniken sind *Two-Point-Crossover* und *Uniform Crossover*, vergleiche [20].

dem String $c_1^{(2)} \dots c_k^{(2)} c_{k+1}^{(1)} \dots c_\ell^{(1)}$. Dabei bezeichnet ℓ die Länge des Strings. Abbildung 2.7 illustriert diese Kreuzungsregel.

Nachdem durch wiederholte Kreuzungen eine neue Population erstellt wurde, besteht die Möglichkeit, die erzeugten Chromosomen durch zufällige Mutationen zu verändern. In dieser Mutationsphase wird jedes Gen aller in der Population enthaltenen Chromosomen mit Wahrscheinlichkeit p_m umgedreht, das heißt aus einer 0 wird eine 1 und umgekehrt. Mutationen erhöhen die Dynamik des Algorithmus, da durch sie völlig neue Lösungen generiert werden können. Damit verringern sie das Risiko, dass das zu einer nur lokal optimalen Lösung gehörende Chromosom in einer Population jenes verdrängt, das eine optimale Lösung repräsentiert. Allzu große Werte für p_m können zu einem verstärkt chaotischen Verlauf des Algorithmus ohne stabile Lösungen führen, es ist daher bei der Wahl dieses Parameters Vorsicht geboten. Sinnvoll ist üblicherweise ein sehr kleiner Wert, beispielsweise 0.001.

Die vorgestellten Elemente bilden die Grundlage für einen einfachen genetischen Algorithmus. Dieser ist in Abbildung 2.8 dargestellt.

Das Verfahren kann auf einfache Weise verbessert werden, indem nämlich in jeder Generation vor Schritt 4 zunächst eine Kopie $c^\star$ des Chromosoms mit maximaler Fitness erstellt wird. Schritte 4-6 bleiben unverändert, allerdings wird nach Schritt 6 zusätzlich geprüft, ob $\mathcal{C}^\mathrm{next}$ ein mit $c^\star$ identisches Chromosom enthält. Ist dies nicht der Fall, so wird $c^\star$ zu $\mathcal{C}^\mathrm{next}$ als (N+1)-tes Chromosom hinzugenommen. Somit wird sichergestellt, dass die jeweils aktuell beste bestimmte Lösung niemals verloren gehen kann. Man bezeichnet ein derartig angepasstes Verfahren als *Elitären Genetischen Algorithmus*, vorgestellt wurde es in [6] als *Elitist Model*.

3 Kernbasierte Lernverfahren

Im Bereich der künstlichen Intelligenz sind die so genannten *maschinellen Lernverfahren* weit verbreitet. Viele dieser Lernverfahren basieren im weitesten Sinne auf dem Prinzip, dass nach der Auswertung verschiedener Trainingsbeispiele möglichst zutreffende Aussagen über zukünftige Ereignisse gemacht werden: Ein Verfahren soll also aus der Vergangenheit lernen, wie die Zukunft aussehen könnte. Abschnitt 3.1 stellt einige grundlegende Eigenschaften maschineller Lernverfahren vor. Im Anschluss befasst Abschnitt 3.2 sich intensiver mit kernbasierten Lernverfahren und was unter anderem mit ihnen machbar ist, während Abschnitt 3.3 ein konkretes Lernverfahren, die *Support Vector Machine*, vorstellt.

Weiterführende Informationen können beispielsweise [8], [21], [23], [24], [28], [30] und [34] entnommen werden. Diese Texte stellen gleichzeitig die wesentliche Grundlage für dieses Kapitel dar.

3.1 Maschinelle Lernverfahren

Ein Teilbereich der Künstlichen Intelligenz besteht darin, dass ein System – in diesem Fall häufig als *Agent* bezeichnet – ein Verhalten, beispielsweise die korrekte Reaktion auf veränderte Eingaben, erlernen soll. Ein Weg, diese Situation zu beschreiben, besteht darin, in vergangenen Ereignissen und Reaktionen ein *Muster* zu erkennen. Ziel eines Lernverfahrens ist dann, dieses Muster einerseits zu erkennen und andererseits aus ihm die korrekte oder zumindest eine möglichst gute Reaktion für zukünftige Ereignisse zu erahnen. Wie facettenreich die hierfür verwendeten Lernalgorithmen sind, lässt sich bereits daraus erahnen, dass in [4] zwischen Lernverfahren anhand von *Beispielen*, *Analogien*, *Beobachtungen*, mit Hilfe von *Brocken*[1], *Erforschung* und schließlich anhand der Auswertung von *geschriebenem Text* unterschieden wird. Diese Arbeit befasst sich primär – wie bereits im Kapitelvorspann angedeutet – mit dem Ansatz des Lernens anhand von Beispielen.

3.1.1 Externe Rückmeldungen

Ein wichtiges Element können dabei externe Rückmeldungen sein, welche das Verhalten des Agenten bewerten. Man unterscheidet dabei im Wesentlichen drei unterschiedliche grundlegende Bewertungsprinzipien, welche zu jeweils unterschiedlichen Lernalgorithmen führen:

Überwachtes Lernen: Nachdem der Agent auf eine Situation reagiert hat, wird ihm von außen mitgeteilt, welche Reaktion die richtige gewesen wäre.

Verstärkendes Lernen: Nachdem der Agent auf eine Situation reagiert hat, wird von außen nur mitgeteilt, ob seine Reaktion richtig oder falsch war.

Unüberwachtes Lernen: Der Agent erhält keinerlei Feedback von außen und kann das zu erlernende Muster lediglich aus den beobachteten Ereignissen selbst ableiten.

Je nach Lernumgebung kommen verschiedene Formen von Rückmeldungen in Frage, unter anderem auch Mischformen wie beispielsweise das halbüberwachte Lernen, bei dem das korrekte Verhalten nur auf Anfrage genannt wird. Dies kann zum Beispiel sinnvoll sein, wenn durch die Ermittlung des korrekten Verhaltens zusätzliche Kosten entstehen, welche vermieden werden können.

Beispiel 3.1 (nach [23]) *Ein junger Mann, der noch keinen Führerschein hat, möchte fahren lernen und anschließend als Taxifahrer seinen Lebensunterhalt verdienen.*

[1] Hierbei wird versucht, mehrere Arbeitsschritte zu einem einzelnen „Brocken“ zusammenzufassen, englisch: *Chunking*.

- *Zunächst, in der Fahrschulzeit, erklärt ihm der Fahrlehrer, wie er sich in verschiedenen Situationen zu verhalten hat. Es liegt* **überwachtes Lernen** *vor, und zwar in einer sehr teuren Variante, da der Fahrlehrer schließlich Geld verdienen möchte. Wegzudenken ist er allerdings, unter anderem aus rechtlichen Gründen, an dieser Stelle nicht.*
- *Nach einigen ereignisreichen Fahrstunden macht unser angehender Taxifahrer seinen Führerschein. Auch seine Taxifahrerlizenz bekommt er bald darauf. Zunächst hat er jedoch noch eine etwas ruppige Fahrweise, seine Fahrgäste werden häufig durchgeschüttelt. Da er im Anschluss in diesen Fällen selten ein gutes Trinkgeld erhält, weiß er, dass etwas nicht stimmt und er an seiner Fahrweise arbeiten muss. Ein gutes Trinkgeld hingegen ist ein Indiz dafür, dass er gut gefahren ist. Hierbei handelt es sich um* **verstärkendes Lernen:** *Gute Fahrweise wird belohnt, während schlechte Fahrweise durch Fernbleiben von Trinkgeld bestraft wird.*
- *Im folgenden Winter sieht er sich zum ersten Mal als Autofahrer mit Schnee konfrontiert. Er fährt zunächst nur sehr vorsichtig und beobachtet, wie sich das Auto unter den veränderten Bedingungen verhält – hoffentlich unfallfrei. Hierbei handelt es sich um* **unüberwachtes Lernen**, *da keine externe Rückmeldung zur Verfügung steht.*

Man sieht, dass bereits das einfache Beispiel des angehenden Taxifahrers ausreicht, um verschiedene Formen des Lernens vorzustellen.

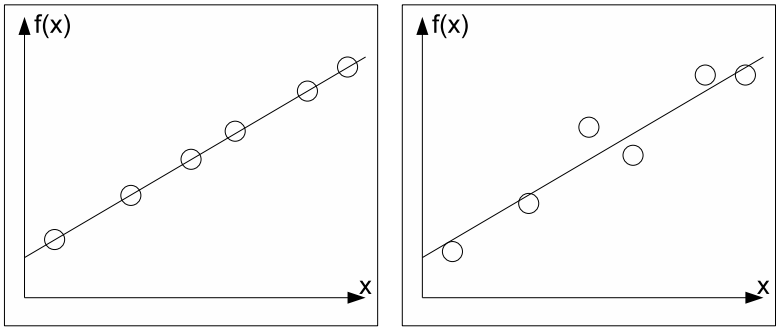
Die im weiteren Verlauf dieser Arbeit dargestellten Aspekte von Lernverfahren beziehen sich auf überwachtes Lernen: Schließlich werden im Verlauf des Lernprozesses Trainingsbeispiele inklusive der bestmöglichen Reaktionen zur Verfügung gestellt.

Unüberwachtes Lernen kann beispielsweise eingesetzt werden, um den Datenraum auf einen weniger komplexen Raum abzubilden und somit Daten vereinfacht darzustellen. Dies ist die so genannte *Principal Component Analysis*. Eine andere Anwendung ist die Identifikation von Datenmengen, die jeweils zusammen gehören – man spricht an dieser Stelle vom *Clustering*. Mehr hierzu ist beispielsweise in [30] nachzulesen.

3.1.2 Verschiedene Arten von Mustern

Sei die Menge aller möglichen Eingaben als eine Menge X gegeben, die Menge aller möglichen Reaktionen als Y. Je nach Lernumgebung können hier selbstverständlich unterschiedliche Mengen sinnvoll sein. Dabei gibt es für jedes $\mathbf{x} \in X$ ein $y \in Y$, das eine besonders gute Antwort auf die Eingabe darstellt. Was an dieser Stelle *besonders gut* heißt, hängt dabei wiederum vom betrachteten Lernproblem ab.

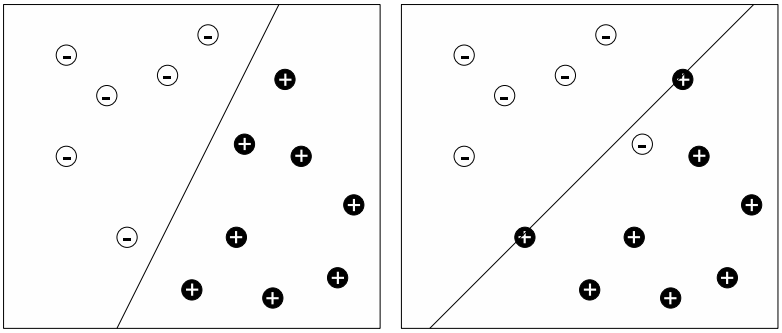
Definition 3.1.1 *Eine Funktion $f : X \to Y$ heißt* **Musterfunktion**. *Jede Musterfunktion repräsentiert die* **Hypothese**, *dass der Eingabe $\mathbf{x} \in X$ die Ausgabe $f(\mathbf{x})$ zuzuordnen ist.*



(a) exakt (b) angenähert

Abbildung 3.1: Beispiele für exakte und angenäherte lineare Regression mit $X = \mathbb{R}, Y = \mathbb{R}$.

Durch die Art der gesuchten Musterfunktion f können sich Lernverfahren grundlegend unterscheiden, da verschiedene Muster zu verschiedenen Zwecken genutzt werden können. Zwei davon werden an dieser Stelle kurz beschrieben:



(a) exakt (b) angenähert

Abbildung 3.2: Beispiele für exakte und angenäherte Klassifikation mit $X = \mathbb{R}^2, Y = \{-1; +1\}$.

Regression: Es gelte $Y \subseteq \mathbb{R}$, gesucht wird eine stetige Funktion f, welche die beobachteten Ergebnisse besonders gut, das heißt mit möglichst geringen Abweichungen, beschreibt und hoffentlich gute Vorhersagen für zukünftige Werte mach. Diese Aufgabe kann erschwert werden, wenn für Trainingsbeispiele eventuell keine exakten, sondern nur verrauschte, also leicht verfälschte Werte vorliegen. Abbildung 3.1 zeigt Beispiele für exakte und aufgrund verrauschter Beobachtungen angenäherte lineare Regression. Im linearen Fall ist das resultierende Muster darstellbar als $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$ mit $\mathbf{w} \in X, b \in \mathbb{R}$.

Klassifikation: Die Menge Y der Bezeichnungen ist diskret, beispielsweise $Y = \{-1, +1\}$. Man kann sagen, dass jedem Punkt $\mathbf{x} \in X$ eine Klasse $y \in Y$ zugewiesen wird. Gesucht wird nun eine Funktion f, welche diese Klassifikation nachvollzieht und in der Lage ist, auf ungesehene Beispiele zu verallgemeinern, ohne allzu viele Fehler zu machen. Besonders einfach ist der Fall bei linear separablen Punkten, das heißt es gibt für je zwei Klassen eine Hyperebene, die zu den jeweiligen Klassen gehörigen Punkte voneinander trennt. Auch hier kann es vorkommen, dass die Datenpunkte verrauscht sind, so dass eine exakte Klassifizierung nicht mehr möglich ist. Abbildung 3.2 zeigt Beispiele für exakte und angenäherte binäre Klassifikation mit Hilfe einer Geraden. Im linearen Fall ist die Musterfunktion darstellbar durch $f(\mathbf{x}) = \operatorname{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle + b)$ mit $\mathbf{w} \in X, b \in \mathbb{R}$.

Bei beiden Varianten werden Fälle beschrieben, in denen sich die enthaltenen Muster besonders einfach – nämlich durch einen Vektor im Datenraum X – beschreiben lassen, welchem allerdings jeweils eine andere Bedeutung beigemessen wird. Bei der Regression beschreibt er die Gerade, welche durch die Punkte dargestellt wird, während er bei der Klassifikation der zu der trennenden Hyperebene senkrechter Vektor ist. In diesen linearen Fällen ist die Mustererkennung vergleichsweise einfach.

Allerdings sind im allgemeinen Fall die Daten weder linear approximierbar noch linear separabel, sogar mit exakten, unverrauschten Daten. In diesem Fall sind sowohl zur Musterbestimmung als auch zur Musterdarstellung weitere Hilfsmittel nötig. Ein Ansatz hierfür wird in Abschnitt 3.2 vorgestellt.

3.1.3 Empirische Risikominimierung

Bei der Untersuchung verschiedener Musterfunktionen treten früher oder später für ein Wertepaar $(\mathbf{x}, y)$ Abweichungen zwischen der vorgeschlagenen Reaktion $f(\mathbf{x})$ und der tatsächlich besten Reaktion y auf, also ein *Fehler*. Die Indikation eines solchen Fehlers lässt sich mit Hilfe einer *Verlustfunktion* umsetzen:

Notation 3.1.2 (nach [21]) *Sei eine Eingabemenge X sowie eine Menge möglicher Reaktionen Y gegeben. Eine* **Verlustfunktion** $L : Y \times Y \to \mathbb{R}_0^+$ *beschreibt bei Betrachtung der Musterfunktion f den Fehler $L(f(\mathbf{x}), y)$, der für ein Punktpaar $(\mathbf{x}, y) \in X, Y$ gemacht wird.*

Bemerkung 3.1.3 (nach [21]) *Bei Klassifikation wird üblicherweise der* **0/1-Verlust**

$$L(f(\mathbf{x}), y) = \begin{cases} 0 & \textit{falls } f(\mathbf{x}) = y \\ 1 & \textit{sonst,} \end{cases}$$

verwendet, während bei Regression üblicherweise der **quadratische Verlust** *eingesetzt wird:*

$$L(f(\mathbf{x}), y) = (f(\mathbf{x}) - y)^2.$$

Die Aufgabe eines Lernalgorithmus besteht nun darin, anhand einer Menge von Trainingsbeispielen $\{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(\ell)}, y^{(\ell)})\} \subset X \times Y$, die als eine Stichprobe unabhängig nach der unbekannten Wahrscheinlichkeitsverteilung $P(\mathbf{x}, y)$ gezogen wurden, eine Funktion f zu bestimmen. Dieses f sollte in der Lage sein, ein in den Daten vorhandenes Muster zu erkennen und idealerweise das erkannte Muster auf weitere Beispiele, die nach der selben Verteilung $P(\mathbf{x}, y)$ bestimmt wurden, anzuwenden. Die beste mögliche Funktion f minimiert dabei den erwarten Fehler:

Notation 3.1.4 (nach [21]) *Sei eine gemeinsame Wahrscheinlichkeitsverteilung $P(\mathbf{x}, y)$ über $X \times Y$ und eine geeignete Verlustfunktion L gegeben. Der* **erwartete Fehler** *bei Auswahl einer Musterfunktion f*

$$R(f) = \int_{X \times Y} L(f(\mathbf{x}), y) dP(\mathbf{x}, y), \tag{3.1}$$

wird auch als **Risiko** *bezeichnet.*

Da die Verteilung $P(\mathbf{x}, y)$ allerdings nicht bekannt ist, kann die Bestimmung der Musterfunktion f nicht durch direkte Minimierung des Ausdrucks $R(f)$ erfolgen. Vielmehr ist man darauf angewiesen, ein f zu bestimmen, das der optimalen Lösung möglichst nahe kommt, wobei die vorliegenden Trainingsbeispiele Aufschluss darüber geben sollen, was für Eigenschaften die Verteilung hat. Hierbei soll das auf einer konkreten Stichprobe basierende *empirische Risiko* helfen.

Notation 3.1.5 (nach [21]) *Sei eine Stichprobe von ℓ Paaren aus Eingaben und korrekten Reaktionen, also $S = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(\ell)}, y^{(\ell)})\}$, sowie eine geeignete Verlustfunktion L gegeben. Das* **empirische Risiko** *ist gegeben durch den Mittelwert der einzelnen Fehler:*

$$R_{\text{emp}}(f) = \frac{1}{\ell} \sum_{i=1}^{\ell} L(f(\mathbf{x}^{(i)}), y^{(i)}). \tag{3.2}$$

Der Ansatz, eine Funktion f zu bestimmen, welche $R_{\text{emp}}(f)$ minimiert, wird als **empirische Risikominimierung** bezeichnet.

Durch geeignete Anforderungen an f ist durch eine Verallgemeinerung des Gesetzes der großen Zahlen gewährleistet, dass die Differenz von $R_{\text{emp}}(f)$ und $R(f)$ für $\ell \to \infty$ gegen 0 geht, also wenn viele Trainingsbeispiele verwendet werden. Wie viele Trainingsbeispiele letztendlich nötig sind, hängt unter anderem davon ab, welcher Lernalgorithmus eingesetzt wird. Da die Bereitstellung eines Trainingsbeispiels üblicherweise sehr aufwendig ist, besteht großes Interesse an Lernalgorithmen, welche bereits für moderat große Werte von ℓ zu akzeptablen Ergebnissen führen.

3.1.4 Hindernisse bei der Musterbestimmung

Die Güte eines Lernalgorithmus lässt sich unter anderem dadurch beschreiben, ob folgende Probleme zufriedenstellend umgangen werden, welche üblicherweise die Mustererkennung erschweren.

Komplexität (englisch: *Complexity*): Um für realitätsnahe Anwendungen von Bedeutung zu sein, müssen Algorithmen zur Mustererkennung in der Lage sein, mit großen Datenmengen umgehen zu können. Daher ist die Komplexität dieser Algorithmen von großer Bedeutung: Wenn das Laufzeitverhalten eines Algorithmus beispielsweise exponentielle Komplexität hat, kann er nur für kleine Datenmenge mit akzeptabler Ressourcennutzung eingesetzt werden. Algorithmen, dessen Laufzeit und Speicherbedarf durch Polynome von nicht

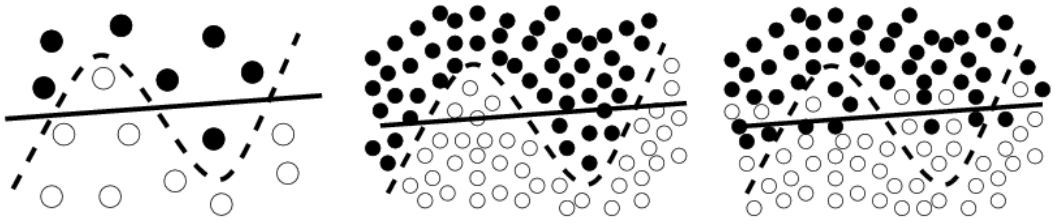
allzu hohem Grad beschränkt werden können, werden als **effizient** bezeichnet. Dabei wird zwischen dem Aufwand zur Bestimmung einer Musterfunktion, der *Bestimmungseffizienz*, sowie dem Aufwand zur Bewertung zukünftiger Daten, der *Auswerte-Effizienz*, unterschieden.

Rauschen (englisch: *Noise*): Die vorliegenden Daten sind unter Umständen nicht exakt, beispielsweise durch Mess- oder Rundungsfehler. Es kann trotzdem noch möglich sein, ein Muster zu erkennen, wenn der durch Rauschen entstandene Fehler nicht allzu groß ist. Dabei wird in Kauf genommen, dass das Muster nicht für alle Trainingsdaten exakt ist. Algorithmen, die in der Lage sind, annähernd korrekte Muster trotz verrauschter Daten zu erkennen, werden als **robust** bezeichnet.

Überanpassung (englisch: *Overfitting*): Zieht man eine Klasse $\mathcal{F}$ von Musterfunktionen mit hoher Komplexität in Betracht, kann es vorkommen, dass ein Lernalgorithmus durch geeignete Trainingsdaten eine Musterfunktion bestimmt, welche sehr gut an diese Daten angepasst ist, für andere Daten allerdings sehr schlechte Vorhersagen macht. Es wäre an dieser Stelle sinnvoll, ein Muster in Form einer einfacheren Funktion zu bestimmen, welche die Trainingsdaten nicht ganz so gut beschreibt, dafür aber für andere Daten zuverlässigere Vorhersagen machen kann. Algorithmen, die in der Lage sind, diese Überanpassung zu umgehen, werden als **stabil** bezeichnet.

Einen guten Mustererkennungsalgorithmus erkennt man also daran, dass er sowohl *effizient* als auch *robust* und *stabil* ist.

Beispiel 3.2 (nach [21]) *Abbildung 3.3 zeigt, wie Überanpassung und Rauschen gemeinsam zu Problemen führen können. Die gestrichelte Klassifizierungshypothese in Abbildung 3.3(a) ist komplexer als die durchgezogene, weist allerdings auch weniger Trainingsfehler auf. Sinnvoll können für die geringe Anzahl von Trainingsbeispielen beide sein. Erst durch eine Erhöhung der Anzahl von Trainingsbeispielen wird deutlich, welche Hypothese besser geeignet ist: In dem in Abbildung 3.3(b) Fall ist die gestrichelte Hypothese zu wählen, da die durchgezogene zu wenig an die vorliegenden Daten angepasst ist (Unteranpassung, englisch:* Underfitting*). In dem in Abbildung 3.3(c) hingegen ist die durchgezogene Hypothese zu wählen und ein geringer Trainingsfehler wird in Kauf genommen, da hier die gestrichelte Hypothese zu stark an die ursprünglich wenigen Daten angepasst wäre (Überanpassung).*



(a) korrekte Hypothese noch unklar (b) gestrichelte Hypothese korrekt (c) durchgezogene Hypothese korrekt

Abbildung 3.3: Das Overfitting-Dilemma (Grafik aus [21])

Üblicherweise sollte zur Vermeidung von Überanpassung eine einfache Hypothese einer komplizierteren vorgezogen werden, wenn beide in der Lage ist, ein beobachtetes Phänomen ähnlich gut zu beschreiben. Dieses Prinzip wird auch als *Ockhams Rasiermesser* (englisch: *Occam's Razor*) bezeichnet.

Dieser Ansatz wird bei der *strukturellen Risikominimierung* (englisch: *Structural Risk Minimization*) beachtet. Zunächst sei angemerkt, dass verschiedene Musterfunktionen sich dadurch voneinander abgrenzen lassen, wie komplex sie sind. Beispielsweise ist eine Musterfunktion, die auf einem Polynom vierten Grades basiert, komplexer als eine, die allein auf linearen Begebenheiten basiert. Fasst man verschiedene mögliche Musterfunktionen zu einer Menge $\mathcal{F}$ zusammen, so ist dabei unter anderem von Interesse, wie komplex die darstellbaren Muster sind. Dies wird dann durch die Komplexität der Klasse $\mathcal{F}$ ausgedrückt. Die Komplexität verschiedener Funktionsklassen lässt sich in Form der so genannten VC-Dimension h formalisieren, welche auf Vapnik und Chervonenkis zurück geht, vergleiche hierzu [34]. Bei Klassifikationsaufgaben gibt die VC-Dimension beispielsweise an, wie viele beliebige mit ± 1 gekennzeichnete Beispiele durch eine Funktion aus $\mathcal{F}$ in positive und negative Beispiele trennbar sind.

Bei der strukturellen Risikominimierung wird nun eine Folge von Funktionsklassen $\mathcal{F}_1 \subset \ldots \subset \mathcal{F}_k$ mit nichtfallender VC-Dimension betrachtet. Für diese Funktionsklassen werden jeweils für die gleichen Trainingsdaten Musterfunktionen $f_1, \ldots, f_k$ bestimmt, welche jeweils das durch 3.2 gegebene empirische Risiko minimieren. Dann wird jene Musterfunktion f_i und die zugehörigen Funktionsklasse $\mathcal{F}_i$ ausgewählt, welche noch zu bestimmende obere Schranken für den zu erwartenden Generalisierungsfehler minimiert. Eine solche Schranke wird beispielsweise für binäre Klassifizierung durch den folgenden Satz angegeben.

Satz 3.1.6 (nach [21]) *Sei für einen Raum X und die Menge $Y = \{\pm 1\}$ eine beliebige gemeinsame Wahrscheinlichkeitsverteilung $P(\mathbf{x}, y)$ gegeben. Sei ferner h die endliche VC-Dimension einer Funktionsklasse $\mathcal{F}$ und sei R_{emp} nach (3.2) mit $0/1$-Verlust gegeben. Für $\delta > 0$, $f \in \mathcal{F}$ und $\ell > h$ gilt dann die Ungleichung*

$$R(f) \leq R_{\mathrm{emp}}(f) + \sqrt{\frac{h(\ln \frac{2\ell}{h} + 1) - \ln \frac{\delta}{4}}{\ell}} \tag{3.3}$$

für den erwarteten Fehler R mit Wahrscheinlichkeit $1 - \delta$.

Beweis: siehe [34], Abschnitt 3.7. □

Ungleichung (3.3) stellt somit eine obere Schranke für den zu erwartenden Fehler dar. Dabei kann der erste Term auf der rechten Seite für Funktionen mit niedriger VC-Dimension groß sein, da durch Verwendung einer unflexiblen Funktion f viele Trainingsfehler in Kauf genommen werden müssen. Allerdings stellt die Verwendung einer Funktionsklasse mit niedriger VC-Dimension sicher, dass der Konfidenzterm unter der Wurzel in (3.3) klein ist. Bei dem Einsatz von Funktionsklassen hoher VC-Dimension kehrt sich die Situation um: Der empirische Fehler nimmt ab, weil die Funktion besser auf die einzelnen Trainingspunkte eingehen kann, während der Konfidenzterm größer wird und daher die Aussagen über einen kleinen Generalisierungsfehler an Substanz verlieren – das Risiko einer Überanpassung nimmt zu. Ziel der strukturellen Risikominimierung ist daher, einen möglichst guten Kompromiss aus empirischem Risiko und Aussagen über die Zuverlässigkeit der Hypothese zu finden. Abbildung 3.4 verdeutlicht diesen Zusammenhang schematisch.

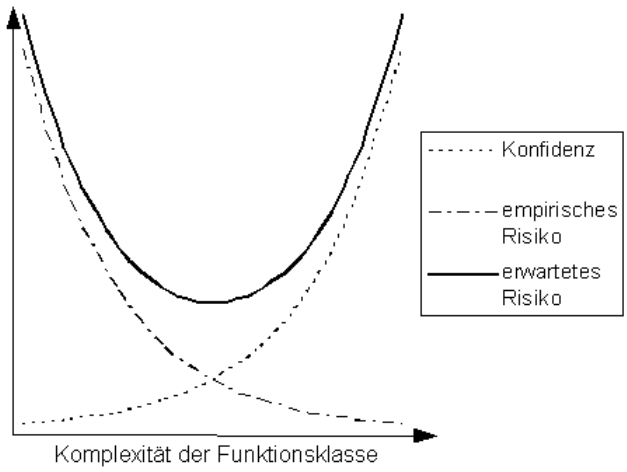


Abbildung 3.4: Obere Schranken für den erwarteten Fehler (nach [21])

Der Ansatz, die erwartete Genauigkeit eines bestimmten Musters an einen Konfidenzparameter δ zu binden, wird als *wahrscheinlich annähernd korrektes* (englisch: *Probably Approximately Correct*, kurz PAC) Lernen bezeichnet. Diese Herangehensweise zur Abschätzung des Lernerfolgs ist notwendig, da die bestimmte Musterfunktion – und damit ihre Genauigkeit – von der Zufälligkeit der ausgewählten Stichprobe $\{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(\ell)}, y^{(\ell)})\} \subset X \times Y$ von Trainingsdaten abhängt. Schließlich wird auch nach beliebig vielen zufällig ausgewählten Trainingsbeispielen ein gewisses Restrisiko bleiben, dass ein identifiziertes Muster lediglich auf zufälligen Strukturen basiert.

Bei der Minimierung der rechten Seite von Ungleichung 3.3 gibt es nach [34] im Wesentlichen zwei Herangehensweisen:

- Fixiere den Konfidenzterm durch geeignete Vorgabe einer Funktionsklasse $\mathcal{F}$ und minimiere dann das empirische Risiko. Diese Herangehensweise wird durch *Neuronale Netzwerke* verfolgt.
- Fixiere eine obere Schranke für das empirische Risiko (diese kann auch 0 sein) und minimiere dann den Konfidenzterm. Diese Herangehensweise wird durch *Support Vector Machines* verfolgt, welche in Abschnitt 3.3 betrachtet werden.

3.1.5 Bedeutung von Vorwissen

Für einige Lernprobleme kann bereits vorhandenes Hintergrundwissen genutzt werden, so dass der Lernprozess deutlich beschleunigt werden kann. Ergebnis ist hierbei stets eine Hypothese, die sich aus dem Hintergrundwissen ableiten lässt und dabei konform mit den Trainingsbeispielen und den zugehörigen Beobachtungen ist. Je nach Art des Hintergrundwissens werden verschiedene Lernverfahren betrachtet.

Erklärungsbasiertes Lernen (EBL): Aus dem Hintergrundwissen wird zunächst eine Hypothese abgeleitet, welche die Trainingsbeispiele korrekt erklärt. Durch Generalisierung wird aus dieser Hypothese eine allgemeinere Hypothese erzeugt, die so allgemein ist, dass sie idealerweise in der Lage ist, weitere noch unbekannte Beispiele erklären zu können. Die Klassifizierungen der Beispiele ergeben sich dann aus ihrer Beschreibung und der zuvor aus dem Hintergrundwissen abgeleiteten Hypothese.

Relevanzbasiertes Lernen (RBL): Aus vorhandenem Hintergrundwissen geht hervor, dass nur ein begrenzter Anteil der mit einem Beispiel verbundenen Informationen relevant ist. Ziel ist die Bestimmung einer Hypothese anhand der Beschreibung vorliegender Trainingsbeispiele inklusive ihrer Klassifizierungen, während die relevanten Informationen durch das Hintergrundwissen vorgegeben werden. Die Klassifizierungen weiterer Beispiele ergibt sich dann aus der erstellten Hypothese sowie den Beschreibungen der Beispiele.

Wissensbasiertes Lernen (KBIL): Ziel ist die Bestimmung einer Hypothese, die in der Lage ist, kombiniert mit vorhandenem Hintergrundwissen aus den Beschreibung eines Beispiels das korrekte Ergebnis zu bestimmen[2]. Dabei werden einfache Hypothesen bevorzugt.

Hierbei findet vielfach die **Induktive Logikprogrammierung (ILP)** Anwendung. Dieses Verfahren setzt Prädikatenlogik erster Stufe für die Formulierung von Hypothesen ein, welche mehr Ausdruckskraft hat als die für die zuvor genannten Verfahren üblicherweise eingesetzte attributbasierte Sprache. Hierbei wird das Vorwissen primär zu zwei Zwecken genutzt: Einerseits wird der Suchraum zu verkleinert, indem Hypothesen, die im Widerspruch zu dem Vorwissen stehen, ausgeschlossen werden. Andererseits wird durch Vorwissen die Möglichkeit gegeben, Beobachtungen durch einfachere Hypothesen zu bestimmen – welche ihrerseits leichter zu bestimmen sind.

Für einen vertiefenden Einblick sei an dieser Stelle erneut auf [23] verwiesen.

3.1.6 Abgrenzung: Lernverfahren in dieser Arbeit

Kapitel 4 befasst sich detailliert damit, wie maschinelle Lernverfahren zur Untersuchung des in Kapitel 1 vorgestellten Problems eingesetzt werden können. Als Abschluss des vorliegenden Unterabschnitts wird hervorgehoben, welche der hier vorgestellten Punkte für diese Arbeit von besonderer Bedeutung sind.

Es wird mit Hilfe von **Beispielen** in Form von beispielhaften Quellverspätungen gelernt. Dabei werden die korrekten Ergebnisse für diese Lernbeispiele durch die in Kapitel 2 vorgestellten Lösungsverfahren vorgegeben. Für jeden Quellverspätungs-Datensatz werden jeweils die optimalen zu haltenden beziehungsweise zu verpassenden Anschlüsse vorgegeben. Somit liegt **überwachtes Lernen** vor. Ziel des Lernens ist, bei vorgegebenen Quellverspätungen für jeden Anschluss die Entscheidung zu treffen, ob der Anschlusszug warten soll oder nicht. Gesucht ist daher eine **Klassifikation** der Anschlüsse.

[2] Die Abkürzung *KBIL* resultiert aus der englischen Bezeichnung *Knowledge-Based Inductive Learning*.

Für manche Kombinationen von Quellverspätungen ist die zu lösende Instanz des Anschlusssicherungsproblems so schwierig, dass der Lösungsprozess abgebrochen werden muss. Somit ist es möglich, dass nur eine suboptimale Lösung vorliegt und damit für einige Anschlüsse die vorgegebene Klassifizierung nicht korrekt ist. Dies ist eine Form von **Rauschen**.

Vorliegendes Hintergrundwissen über das untersuchte Problem, beispielsweise auf welche Weise sich Folgeverspätungen im Ereignis-Aktivitäts-Netzwerk ausbreiten, beeinflusst die Wahl der Funktionsklassen für potentielle Musterfunktionen, also der zur Verfügung stehenden Hypothesen. Es liegt daher **relevanzbasiertes Lernen** vor. Dabei werden unterschiedliche Funktionsklassen eingesetzt, um **Überanpassung** zu vermeiden und dadurch den **Generalisierungsfehler** zu verringern.

3.2 Kernfunktionen für Lernverfahren

Ein wesentliches Problem vieler Lernalgorithmen ist, dass sie lediglich in der Lage sind, auf linearen Gegebenheiten basierende Muster zu erkennen. Bei nichtlinearen Mustern sind sie machtlos und können eine bestenfalls mäßige Leistung erzielen. Auch treten Probleme auf, wenn der Begriff der Linearität für den Eingaberaum nicht anwendbar ist, beispielsweise bei der Mustererkennung in Texten oder auf diskreten Datenstrukturen wie Bäumen oder Graphen.

3.2.1 Merkmalsräume und Kerne

Eine Möglichkeit dem abzuhelfen besteht darin, für jeden Datenpunkt bestimmte *Merkmale* zu identifizieren. Dies kann durch eine Abbildung ϕ vom ursprünglichen Datenraum *X* in einen so genannten *Merkmalsraum* F geschehen. F sollte dabei ein linearer Raum sein, typischerweise ein Hilbert-Raum. Dann wird nicht mehr in *X* nach einem linearen Muster in den ℓ Trainingspunkten einer Menge $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\ell)}\} \subset X$ gesucht, sondern in deren Bildern $\phi(\mathbf{x}^{(1)}), \ldots, \phi(\mathbf{x}^{(\ell)})$.

Dies ist im Wesentlichen der Ansatzpunkt für *kernbasierte* Lernverfahren. Ein besonders interessanter Aspekt an ihnen ist, dass durch die Verwendung einer *Kernabbildung* $\kappa : X \times X \to \mathbb{R}$ die Merkmale der Trainingsdaten gar nicht explizit berechnet werden müssen. Vielmehr sind die Lernalgorithmen so aufgebaut, dass sie in der Lage sind, ausschließlich mit Skalarprodukten von Abbildungen je zweier Datenpunkte aus *X* zu arbeiten.

Definition 3.2.1 *Sei ein Datenraum X gegeben. Aus einzelnen Datenpunkten $\mathbf{x} \in X$ lassen sich Merkmale extrahieren, welche sich durch eine geeignete Abbildung ϕ in einem Raum F darstellen lassen. F heißt dann* **Merkmalsraum** *(engl.* feature space*) und die Abbildung $\phi : X \to F$ heißt* **Merkmalsabbildung** *(engl.* feature mapping*).*

Definition 3.2.2 (nach [30]) *Ein* **Kern** *ist eine Funktion κ, welche für alle $\mathbf{x}^{(1)}, \mathbf{x}^{(2)} \in X$ die Eigenschaft*

$$\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \left\langle \phi(\mathbf{x}^{(1)}), \phi(\mathbf{x}^{(2)}) \right\rangle$$

hat, wobei ϕ eine Abbildung von X in einen (mit einem inneren Produkt versehenen) Merkmalsraum F ist mit

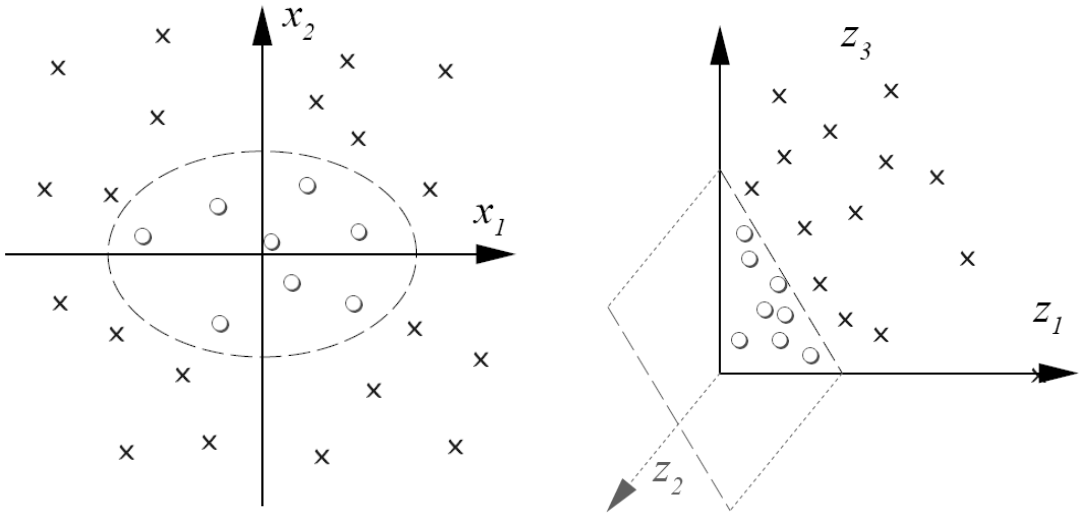
$$\phi : \mathbf{x} \mapsto \phi(\mathbf{x}) \in F.$$

Die Eigenschaften des Merkmalsraumes können dabei deutlich von denen des Datenraumes abweichen. Oftmals weist er eine sehr viel höhere – in einigen Fällen sogar unendliche – Dimension auf. Die Existenz einer Kernfunktion κ bietet die Möglichkeit, das Skalarprodukt für die Bilder zweier Datenpunkte $\mathbf{x}^{(1)}$ und $\mathbf{x}^{(2)}$ implizit zu berechnen, ohne dass die Bilder $\phi(\mathbf{x}^{(1)})$ und $\phi(\mathbf{x}^{(2)})$ jemals explizit bestimmt werden müssen. Dies ist insbesondere dann von Vorteil, wenn der Merkmalsraum eine hohe Dimension aufweist.

Beispiel 3.3 (nach [29]) *Abbildung 3.5(a) zeigt einen Fall mit $X = \mathbb{R}^2$, in dem die Datenpunkte durch den Rand einer Ellipse um den Ursprung klassifiziert werden können. Klassifizierungen mit Hilfe linearer Ansätze sind nicht sinnvoll. Betrachtet man hingegen die Punkte in einem Merkmalsraum $F = \mathbb{R}^3$ der Monome zweiten Grades, in den sie mit Hilfe einer Merkmalsabbildung*

$$\phi : \mathbb{R}^2 \to \mathbb{R}^3, \quad (x_1, x_2) \mapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \tag{3.4}$$

eingebettet werden, so lassen sie sich die beiden Punktklassen ohne weiteres durch eine Hyperebene trennen.



(a) Ursprüngliche Daten mit elliptischer Klassifikationsgrenze (b) Bilder der Daten, im Merkmalsraum linear separierbar

Abbildung 3.5: Zweidimensionales Klassifikationsbeispiel (aus [29])

Das vorgestellte Beispiel 3.3 zeigt, wie die Merkmalsabbildung ϕ bereits in Räumen mit geringer Dimension nützlich ist. Dabei ist zu beachten, dass es in dem genannten Beispiel ausreicht, den von der ersten und dritten Komponente des Merkmalsvektors $\phi(\mathbf{x})$ aufgespannten Raum als Merkmalsraum zu betrachten, da die Bilder mit unterschiedlicher Kennzeichnung bereits in der z_1-z_3-Ebene linear separierbar sind. Die zweite Komponente wurde dennoch angeführt, um eine einfache Repräsentation der durch die Merkmalsabbildung induzierten Kernabbildung $\kappa(\cdot,\cdot) = \langle\phi(\cdot),\phi(\cdot)\rangle$ bereitzustellen:

$$\begin{aligned}\kappa : \kappa(\mathbf{x}^{(1)},\mathbf{x}^{(2)}) &= \langle\phi(\mathbf{x}^{(1)}),\phi(\mathbf{x}^{(2)})\rangle = ({x_1^{(1)}}^2,\sqrt{2}x_1^{(1)}x_2^{(1)},{x_2^{(1)}}^2)({x_1^{(2)}}^2,\sqrt{2}x_1^{(2)}x_2^{(2)},{x_2^{(2)}}^2)^\top \\ &= \left((x_1^{(1)},x_2^{(1)})(x_1^{(2)},x_2^{(2)})^\top\right)^2 = \langle\mathbf{x}^{(1)},\mathbf{x}^{(2)}\rangle^2.\end{aligned}$$

Das Skalarprodukt im Merkmalsraum ist somit gleich dem Quadrat des Skalarproduktes im Datenraum, was die Berechnung des Kerns sehr einfach macht, ohne den Merkmalsraum tatsächlich näher betrachten zu müssen. Dieser Vorteil tritt bei einem Beispiel mit solch geringer Dimension zwar weniger in Vordergrund, gewinnt allerdings bei raffinierteren Kernkonstruktionen an Bedeutung, wie in Abschnitt 3.2.2 beschrieben wird. Insbesondere existieren lineare Mustererkennungsalgorithmen, welche nicht direkt mit den Bildern der vorgegebenen Daten arbeiten sondern lediglich mit den Skalarprodukten von Paaren der Bilder.

Nach Definition 3.2.2 ist klar, dass durch jede Merkmalsabbildung $\phi : X \to F$ mit zugehörigem Merkmalsraum F und Skalarprodukt $\langle\cdot,\cdot\rangle_F : F \times F \to \mathbb{R}$ eine Kernabbildung $\kappa : X \times X \to \mathbb{R}$ definiert wird. Interessanterweise ist diese Aussage auch umkehrbar, sofern man geeignete Anforderungen an den Kern κ stellt. Für die Formulierung dieser Anforderungen sind allerdings noch einige Begriffe notwendig.

Notation 3.2.3 (nach [30]) *Sei für ein $\ell \in \mathbb{N}$ eine Punktmenge $S = \{\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(\ell)}\} \subset X$ gegeben. Die für einen Kern κ aus der Beschränkung auf die Menge S resultierende Matrix $\mathbf{K} \in \mathbb{R}^{\ell\times\ell}$, gegeben durch*

$$k_{ij} = \kappa(\mathbf{x}^{(i)},\mathbf{x}^{(j)}) \quad \forall i,j = 1,\ldots,\ell,$$

wird als **Kernmatrix** *bezeichnet.*

Lemma 3.2.4 (nach [30]) *Sei κ der durch eine Merkmalsabbildung $\phi : X \to F$ implizit definierte Kern. Dann ist für eine beliebige endliche Teilmenge $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\ell)}\} \subset X$ die resultierende Kernmatrix $\mathbf{K}$ symmetrisch und positiv semidefinit.*

Beweis: **Symmetrie:** Seien $i, j \in \{1, \ldots, \ell\}$ beliebig. Dann gilt:

$$k_{ij} = \kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle = \langle \phi(\mathbf{x}^{(j)}), \phi(\mathbf{x}^{(i)}) \rangle = \kappa(\mathbf{x}^{(j)}, \mathbf{x}^{(i)}) = k_{ji},$$

die Symmetrie der Kernmatrix ergibt sich somit aus der Symmetrie des Skalarproduktes in F.

Positive Semidefinitheit: Sei $\mathbf{v} \in \mathbb{R}^\ell$ ein beliebiger Vektor. Dann gilt:

$$\begin{aligned} \mathbf{v}^\top \mathbf{K} \mathbf{v} &= \sum_{i,j=1}^{\ell} v_i v_j k_{ij} = \sum_{i,j=1}^{\ell} v_i v_j \langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle \\ &= \left\langle \sum_{i=1}^{\ell} v_i \phi(\mathbf{x}^{(i)}), \sum_{j=1}^{\ell} v_j \phi(\mathbf{x}^{(j)}) \right\rangle = \left\| \sum_{i=1}^{\ell} v_i \phi(\mathbf{x}^{(i)}) \right\|^2 \geq 0, \end{aligned}$$

also ist die Kernmatrix $\mathbf{K}$ tatsächlich positiv semidefinit.

□

Definition 3.2.5 (nach [30]) *Eine Funktion $\kappa : X \times X \to \mathbb{R}$ heißt* **endlich positiv semidefinit**, *wenn sie eine symmetrische Funktion ist, für welche die aus Beschränkung auf beliebige endliche Teilmengen des Raumes X resultierenden Matrizen positiv semidefinit sind.*

Die endliche positive Definitheit eines Kerns κ ist die zentrale Anforderung für die Existenz einer entsprechenden Merkmalsabbildung. Nun lässt sich die in Definition 3.2.2 vorgestellte implizite Definition der Kernabbildung durch eine Merkmalsabbildung ϕ umkehren.

Satz 3.2.6 (nach [30]) *Für eine Funktion $\kappa : X \times X \to \mathbb{R}$, die entweder stetig ist oder einen abzählbaren Wertebereich hat, existiert eine Merkmalsabbildung ϕ in einen Hilbert-Raum F mit*

$$\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \langle \phi(\mathbf{x}^{(1)}), \phi(\mathbf{x}^{(2)}) \rangle$$

für alle $\mathbf{x}^{(1)}, \mathbf{x}^{(2)} \in X$ genau dann, wenn κ endlich positiv semidefinit ist.

Beweis: siehe [30], Theorem 3.11. □

Bemerkung 3.2.7 (nach [30]) *Bei vorgegebenem Kern κ wird solch ein Raum F_κ, für den eine geeignete Merkmalsabbildung $\phi : X \to F_\kappa$ mit $\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \langle \phi(\mathbf{x}^{(1)}, \phi(\mathbf{x}^{(2)}) \rangle$ existiert, als* **reproduzierender Kern-Hilbert-Raum** *(englisch:* **Reproducing Kernel Hilbert Space**, *kurz* **RKHS***) bezeichnet.*

Vielfach findet sich eine andere Formulierung der Anforderungen an eine Funktion κ für die Existenz eines zugrunde liegenden Merkmalsraumes. Diese beruht auf einer Aussage, die bereits im Jahr 1909 in [19] gemacht wurde:

Satz 3.2.8 (Theorem von Mercer, nach [30], [21], [34]) *Sei X eine kompakte Teilmenge des $\mathbb{R}^n$. Angenommen κ ist eine stetige symmetrische Funktion so dass der Integraloperator $T_\kappa : L_2(X) \to L_2(X)$*

$$(T_\kappa f)(\cdot) = \int_X \kappa(\cdot, \mathbf{x}) f(\mathbf{x})\, d\mathbf{x}$$

positiv ist, das heißt

$$\int_{X \times X} \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) f(\mathbf{x}^{(1)}) f(\mathbf{x}^{(2)})\, d\mathbf{x}^{(1)}\, d\mathbf{x}^{(2)} \geq 0$$

für alle $f \in L_2(X)$. Dann kann $\kappa(\mathbf{x}, \mathbf{z})$ in eine auf $X \times X$ gleichmäßig konvergente Reihe aus Funktionstermen ϕ_j expandiert werden, das heißt

$$\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \sum_{j=1}^{\infty} \phi_j(\mathbf{x}^{(1)}) \phi_j(\mathbf{x}^{(2)}).$$

Beweis: (nach [30]) Zunächst folgt aus der Positivität des Intgraloperators die endliche Semidefinitheit der Funktion κ: Angenommen, es gäbe eine endliche Matrix $\mathbf{K}$, welche aus einer Beschränkung auf die Punkte $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\ell)} \in X$ resultiert, welche nicht positiv semidefinit ist. Sei also $\alpha \in \mathbb{R}^\ell$ ein Vektor mit der Eigenschaft

$$\boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\alpha} = \sum_{i,j=1}^{\ell} \kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \alpha_i \alpha_j = \epsilon < 0$$

und betrachte

$$f_\sigma(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i \frac{1}{(2\pi\sigma)^{d/2}} \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}^{(i)}\|^2}{2\sigma^2}\right) \in L_2(X),$$

wobei d die Dimension des Raumes X ist. Es gilt

$$\lim_{\sigma \to 0} \int_{X \times X} \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) f_\sigma(\mathbf{x}^{(1)}) f_\sigma(\mathbf{x}^{(2)})\, d\mathbf{x}^{(1)}\, d\mathbf{x}^{(2)} = \epsilon,$$

aber das heißt dass das Integral für ein $\sigma > 0$ kleiner als 0 ist, was im Widerspruch zu der Positivität des Integraloperators steht. Die Matrix $\mathbf{K}$ muss also positiv semidefinit sein.

□

An dieser Stelle sei ausdrücklich darauf hingewiesen, dass man einen Kern κ sogar dann verwenden kann, wenn weder die zugehörige Merkmalsabbildung ϕ noch der Merkmalsraum F_κ bekannt ist! Die Abbildung muss lediglich den in Satz 3.2.6 beziehungsweise Satz 3.2.8 genannten Bedingungen genügen. Zu dem selben Kern können sogar verschiedene Merkmalsabbildungen gehören:

Beispiel 3.4 (nach [30]) *Die Merkmalsabbildung* $\tilde{\phi}$

$$\tilde{\phi} : \mathbb{R}^2 \to \mathbb{R}^4, \quad (x_1, x_2) \mapsto (x_1^2, x_1 x_2, x_2 x_1, x_2^2)$$

führt zu dem selben Kern κ *wie die in Beispiel 3.3 vorgestellte Funktion* ϕ*:*

$$\begin{aligned}
\langle \tilde{\phi}(\mathbf{x}^{(1)}), \tilde{\phi}(\mathbf{x}^{(2)}) \rangle &= (x_1^{(1)^2}, x_1^{(1)} x_2^{(1)}, x_2^{(1)} x_1^{(1)}, x_2^{(1)^2})(x_1^{(2)^2}, x_1^{(2)} x_2^{(2)}, x_2^{(2)} x_1^{(2)}, x_2^{(2)^2})^\top \\
&= x_1^{(1)^2} x_1^{(2)^2} + x_1^{(1)} x_2^{(1)} x_1^{(2)} x_2^{(2)} + x_2^{(1)} x_1^{(1)} x_2^{(2)} x_1^{(2)} + x_2^{(1)^2} x_2^{(2)^2} \\
&= (x_1^{(1)} x_1^{(2)} + x_2^{(1)} x_2^{(2)})^2 = \langle \mathbf{x}^{(1)}, \mathbf{x}^{(2)} \rangle^2 = \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}).
\end{aligned}$$

Dieser Effekt beruht darauf, dass die zweite und dritte Dimension in dem durch $\tilde{\phi}$ *definierten Merkmalsraum stets gleich sind und die dritte Dimension keine zusätzlichen Merkmale enthält. Der dreidimensionale Merkmarlsraum, der mit der Abbildung* ϕ *zusammenhängt, wird also in einen vierdimensionalen Merkmalsraum eingebettet, wobei eine Dimension redundant ist. Allgemein kann dieser Effekt immer dann auftreten, wenn Merkmale redundant codiert werden, beispielsweise als Vielfache anderer Merkmale.*

3.2.2 Konstruktion von Kernen

Es lassen sich je nach untersuchter Situation beliebige Merkmalsabbildungen und damit implizit eine Kernabbildung definieren. Die Frage ist dabei aber stets, wie solch eine Abbildung ϕ aussehen kann, damit die Merkmale derart sind, dass in den Daten enthaltene Muster linear beschrieben werden können. Dabei sollten stets die in Abschnitt 3.1.4 angesprochenen Hindernisse der Komplexität, des Rauschens und der Überanpassung berücksichtigt werden. Die Frage es dabei aber stets, wie solch eine Abbildung ϕ aussehen kann, damit die Merkmale derart sind, dass in den Daten enthaltene Muster linear beschrieben werden können. Dabei sollten stets die in Abschnitt 3.1.4 angesprochenen Hindernisse der Komplexität, des Rauschens und der Überanpassung berücksichtigt werden.

Der einfachste Fall liegt sicherlich bei dem **konstanten Kern** $\kappa : \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = c$ für alle $\mathbf{x}^{(1)}, \mathbf{x}^{(2)} \in X$ für ein $c \in \mathbb{R}$ vor. Dieser Kern führt auf eine eindimensionale konstante Merkmalsabbildung $\phi : \phi(\mathbf{x}) = \sqrt{c}$ für alle $\mathbf{x} \in X$. So einfach dieser Kern ist, so nutzlos ist er leider im Allgemeinen auch.

Ein nicht ganz so einfacher, aber trotzdem noch kanonischer Fall liegt vor, wenn *X* selbst schon ein Hilbert-Raum ist und als Merkmalsabbildung die Identität auf *X* verwendet wird. Dann ist der **lineare Kern** das auf *X* definierte Skalarprodukt, also $\phi : \phi(\mathbf{x}) = \mathbf{x}$ für alle $\mathbf{x} \in X$ und $\kappa : \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \langle \mathbf{x}^{(1)}, \mathbf{x}^{(2)} \rangle$ für alle $\mathbf{x}^{(1)}, \mathbf{x}^{(2)} \in X$. Diesen Kern kann man dadurch modifizieren, dass man nicht alle Komponenten von einem Datenpunkt $\mathbf{x} \in X$ betrachtet, sondern als Merkmalsabbildung die Projektion in einen Hilbert-Raum von geringerer Dimension betrachtet.

Es besteht auch die Möglichkeit, bereits bekannte Kerne anzupassen, ein Beispiel hierfür ist die Normierung eines Kerns κ. Um den **normierten Kern** zu erhalten, geht man zu einem Merkmalsraum über, in dem alle Merkmalsvektoren die Länge 1 haben:

$$\hat{\kappa}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \left\langle \frac{\phi(\mathbf{x}^{(1)})}{\|\phi(\mathbf{x}^{(1)})\|}, \frac{\phi(\mathbf{x}^{(2)})}{\|\phi(\mathbf{x}^{(2)})\|} \right\rangle = \frac{\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)})}{\sqrt{\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(1)})\kappa(\mathbf{x}^{(2)}, \mathbf{x}^{(2)})}}.$$

Auch können bereits bekannte Kerne auf verschiedene Weisen kombiniert werden, um neue Kerne zu erhalten. Grundlage dafür sind die im folgenden Satz angeführten Operationen, unter denen die Menge der Kernfunktionen abgeschlossen ist:

Satz 3.2.9 (nach [30]) *Betrachte Kerne κ_1, κ_2 über $X \times X$, $X \subseteq \mathbb{R}^n$, einen Wert $a \in \mathbb{R}^+$, eine reellwertige, auf X definierte Funktion $f(\cdot)$, eine Abbildung $\phi : X \to \mathbb{R}^N$, einen Kern κ_3 über $\mathbb{R}^N \times \mathbb{R}^N$ sowie eine positiv symmetrisch semidefinite $n \times n-$Matrix $\mathbf{B}$.*

Dann sind die folgenden Abbildungen ebenfalls Kerne:

1. $\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \kappa_1(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) + \kappa_2(\mathbf{x}^{(1)}, \mathbf{x}^{(2)})$,
2. $\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = a\kappa_1(\mathbf{x}^{(1)}, \mathbf{x}^{(2)})$,
3. $\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \kappa_1(\mathbf{x}^{(1)}, \mathbf{x}^{(2)})\kappa_2(\mathbf{x}^{(1)}, \mathbf{x}^{(2)})$,
4. $\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = f(\mathbf{x}^{(1)})f(\mathbf{x}^{(2)})$,
5. $\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \kappa_3(\phi(\mathbf{x}^{(1)}), \phi(\mathbf{x}^{(2)}))$,
6. $\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = {\mathbf{x}^{(1)}}^\top \mathbf{B}\mathbf{x}^{(2)}$.

Beweis: siehe [30], Proposition 3.22. □

Mit Hilfe dieser Operationen lassen sich beliebig komplexe Kernfunktionen konstruieren. Dabei spielen einige Konstruktionen eine besondere Rolle, da die resultierenden Kerne häufig erfolgreich für Mustererkennungsaufgaben eingesetzt werden.

Satz 3.2.10 (nach [30]) *Sei κ_1 ein Kern über $X \times X$, p ein Polynom mit nichtnegativen Koeffizienten und $\sigma > 0$. Dann sind auch die folgenden Abbildungen Kerne:*

1. $\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = p(\kappa_1(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}))$,
2. $\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \exp(\kappa_1(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}))$,
3. $\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \exp(-\|\mathbf{x}^{(1)} - \mathbf{x}^{(2)}\|^2/(2\sigma^2))$.

Beweis: siehe [30], Proposition 3.24. □

Die Beobachtung, dass Polynome von Kernen ebenfalls Kerne sind, sofern sie keine negativen Koeffizienten haben, ist für viele Kerne von großer Bedeutung. Insbesondere lassen sich auf dieser Grundlage viele komplexere Kerne erstellen, welche für die unterschiedlichsten Situationen maßgeschneidert sind. Ein einfaches Beispiel für solch einen **Polynomkern** wurde bereits in Abschnitt 3.2.1 angeführt. Für praktische Anwendungen werden Polynomkerne häufig in der Form

$$\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \left(\langle \mathbf{x}^{(1)}, \mathbf{x}^{(2)} \rangle + R\right)^d \text{ mit } R \geq 0, d \in \mathbb{N} \tag{3.5}$$

eingesetzt. Dabei lässt sich durch den Parameter R indirekt das Gewicht der Merkmale je nach Monomgrad steuern: Größere Werte von R führen zu einer stärkeren Gewichtung von Monomen kleinen Grades, während durch ein kleines R mehr Gewicht auf die Monome höheren Grades entfallen.

Auch der dritte in Satz 3.2.10 angeführte Kern wird häufig für die implizite Definition von Merkmalsabbildungen verwendet, er wird auch **Gaußkern** genannt. In diesem Fall ist der Merkmalsraum tatsächlich unendlich-dimensional.

Zusätzlich werden in der Praxis gelegentlich **sigmoidale Kerne** eingesetzt, welche die Form

$$\kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \tanh\left(v\langle \mathbf{x}^{(1)}, \mathbf{x}^{(2)}\rangle + R\right) \text{ mit } v, R \in \mathbb{R} \tag{3.6}$$

haben. Sie können als zweischichtige Neuronale Netzwerke aufgefasst werden, bei denen die jeweiligen Neuronen eine sigmoidale Aktivierungsfunktion haben. Kerne dieser Art haben allerdings den Nachteil, dass sie nur für einige Werte von v und R endlich positiv semidefinit sind. Sigmoidale Kerne und insbesondere der Zusammenhang mit neuronalen Netzwerken werden in [34] näher beleuchtet.

Eine ausführliche Vorstellung weiterer komplexer Kerne findet man beispielsweise in [30]. Dort werden diverse Möglichkeiten beschrieben, wie gerade für diskrete Strukturen – beispielsweise Graphen, Bäume oder Strings – geeignete Kerne erstellt werden können. Mit der Thematik der Kerne für diskrete Strukturen befasst sich zusätzlich [11] sehr intensiv.

Als Abschluss dieses Abschnitts folgt ein Negativbeispiel, das eine Grenze des mit Hilfe von Kernfunktionen beziehungsweise maschinellen Lernverfahren im Allgemeinen Machbaren verdeutlichen soll.

Beispiel 3.5 (nach [30]) *Angenommen, man kennt von sieben Freunden $i = 1, \ldots, 7$ sowohl die Telefonnummern t_i als auch die Kreditkartennummern c_i. Aus diesen Daten lässt sich ein Polynom p sechsten Grades in einer Variablen erstellen, welches die Werte korrekt interpoliert, das heißt $p(t_i) = c_i$ für $i = 1, \ldots, 7$. Allerdings ist wohl kaum damit zu rechnen, dass diese Regression für Telefon- und Kreditkartennummern anderer Menschen zuverlässige Werte liefert – auch dies ist eine Form von Überanpassung. Der Merkmalsraum der Monome mit maximalem Grad 6 ist daher nicht geeignet.*

Es ist zu hoffen, dass es keinen Merkmalsraum gibt, in dem für Beispiel 3.5 eine zuverlässige Verallgemeinerung möglich ist, und dass somit tatsächlich *keine* echten Zusammenhänge zwischen Telefon- und Kreditkartennummern bestehen. Das Problem ist (glücklicherweise) *schlecht gestellt*, da die Ausgangsdaten nicht stetig von den Eingangsdaten abhängen.

3.3 Support Vector Machines

Bereits in Abschnitt 3.1 wurde beschrieben, wie eine Klassifikation von Punktmengen durch eine trennende Hyperebene vorgenommen werden kann. Dieser Abschnitt befasst sich mit der Problemstellung, wie eine trennende Hyperebene in einem Hilbert-Raum bestimmt werden kann, von welcher eine besonders gute Generalisierung zu erwarten ist. Wie diese so genannten *Hyperebenen mit maximalem Rand* aussehen können wird in Abschnitt 3.3.1 beschrieben. Die Bestimmung solcher Hyperebenen führt schließlich zu einem in Abschnitt 3.3.2 konvexen Optimierungsproblem mit quadratischer Zielfunktion und linearen Nebenbedingungen. Dabei ergibt sich, dass bei einer Optimallösung nur wenige der dualen Variablen des Problems ungleich Null sind. Ein zusätzlicher Vorteil der resultierenden Berechnungen ist, dass sie lediglich die Berechnung des Skalarproduktes von Punktpaaren im Hilbert-Raum benötigen, aber nicht mehr die Punkte selbst. Dies stellt den Punkt dar, an dem die im Abschnitt 3.2 vorgestellten Kerne ansetzen. Hiermit befasst sich der Abschnitt 3.3.3.

Diese Arbeit befasst sich ausschließlich mit dem Fall, dass die Trainingspunkte im Merkmalsraum linear getrennt werden. Dies führt zu der so genannten *Hard Margin Support Vector Machine*. Lässt man zu, dass Bilder der Trainingspunkte durch lineare Trennung falsch klassifiziert werden – beispielsweise weil sie nicht linear trennbar sind oder aber um einen größeren Rand zu erhalten – führt dies zu der *Soft Margin Support Vector Machine*. Die Theorie und der nachfolgend vorgestellte Algorithmus muss dafür geringfügig modifiziert werden; dies wird beispielsweise in [30] vorgestellt.

3.3.1 Hyperebenen mit maximalem Rand

Sei ein Hilbert-Raum F gegeben. Eine Hyperebene in F lässt sich mit geeigneten Parametern $\mathbf{w} \in F$, $b \in \mathbb{R}$ darstellen als die Menge aller Punkte $\mathbf{x} \in F$ mit der Eigenschaft

$$\langle \mathbf{w}, \mathbf{x}\rangle + b = 0. \tag{3.7}$$

Für jede Hyperebene gibt es durch Umskalierung unendlich viele Kombinationen von Parametern $\mathbf{w}, b$. Es gibt zwei grundlegende Ansätze, die Anzahl der Freiheitsgrade durch Wahl einer eindeutigen Skalierung zu reduzieren, wobei in dieser Arbeit die zweite Variante als *kanonische Parameterwahl* angesehen wird:

- Der Vektor $\mathbf{w}$ wird als normiert, das heißt $\|\mathbf{w}\| = 1$, vorausgesetzt. Dieser Ansatz wird beispielsweise in [30] verfolgt.
- Für eine vorgegebene Menge von Punkten $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\ell)} \in F$, von denen keiner direkt auf der Hyperebene liegt[3], werden $\mathbf{w}$ und b so gewählt, dass

$$\min_{i=1,\ldots,\ell} |\langle \mathbf{w}, \mathbf{x}^{(i)} \rangle + b| = 1 \tag{3.8}$$

  gilt. Auf diese Weise hat der am nächsten an der Hyperebene liegende Punkt $\mathbf{x}^{(i)}$, gemessen rechtwinklig zur Hyperebene, den Abstand $\frac{1}{\|\mathbf{w}\|}$. Dieser Ansatz wird in [30] ebenfalls vorgestellt, in [21], [28] und [34] wird er ausschließlich verwendet.

Dieser kleinste Abstand $\frac{1}{\|\mathbf{w}\|}$ eines Punktes $\mathbf{x}^{(i)}$ von der Hyperebene wird als **Rand** (englisch: **Margin**) bezeichnet.

Definition 3.3.1 (nach [21]) *Sei für einen Hilbert-Raum F eine Menge von Beispielen $(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(\ell)}, y^{(\ell)})$ mit $\mathbf{x}^{(i)} \in F$ und $y^{(i)} \in \{\pm 1\}$ für $i = 1, \ldots, \ell$ gegeben. Eine* **trennende Hyperebene** *ist gegeben durch ein Parameterpaar $\mathbf{w} \in F$, $b \in \mathbb{R}$ mit der Eigenschaft*

$$y_i \cdot (\langle \mathbf{w}, \mathbf{x}^{(i)} \rangle + b) \geq 1 \qquad \forall i = 1, \ldots, \ell. \tag{3.9}$$

Eine **Hyperebene mit maximalem Rand** *liegt vor, wenn es bei kanonischer Parameterwahl keinen Vektor $\tilde{\mathbf{w}} \in F$ gibt mit $\|\tilde{\mathbf{w}}\| < \|\mathbf{w}\|$.*

Trennende Hyperebenen können gebildet werden, sobald es Beispiele mit sowohl positiven als auch negativen Beispielen gibt, das heißt $\bigcup_{i=1,\ldots,\ell} y^{(i)} = \{\pm 1\}$ – ansonsten gäbe es schließlich nichts zu trennen. Dieser Fall wird üblicherweise vorausgesetzt, da das Lernproblem im Fall ausschließlich positiver oder negativer Beispiele wenig interessant ist. Wenn nun sowohl positive als auch negative Beispiele vorliegen, gibt es bei jeder trennenden Hyperebene mit maximalem Rand mindestens zwei Punkte, die jeweils genau den Abstand $\frac{1}{\|\mathbf{w}\|}$ von der Hyperebene haben.

Definition 3.3.2 *Sei mit den Vorgaben aus Definition 3.3.1 eine trennende Hyperebene mit maximalem Rand gegeben. Vektoren $\mathbf{x}^{(i)}, i \in \{1, \ldots, \ell\}$ mit der Eigenschaft $\|\langle \mathbf{w}, \mathbf{x}^{(i)} \rangle + b\| = 1$ heißen* **Stützvektoren** *(englisch:* **Support Vectors***).*

Die Bezeichnung *Stützvektor* ergibt sich daher, dass der Punkt $\mathbf{x}^{(i)}$ auf dem Rand der konvexen Hülle $C_{y^{(i)}} \subseteq F$ aller Punkte mit der Bezeichnung $y^{(i)}$ liegt und somit eine *stützende Hyperebene* (englisch: *Supporting Hyperplane*) durch ihn an $C_{y^{(i)}}$ angelegt werden kann. Abbildung 3.6 veranschaulicht, wie eine trennende Hyperebene mit maximalem Rand sowie die zugehörigen Stützvektoren aussehen können. Die gezeigten stützenden Hyperebenen an die Mengen C_{-1} beziehungsweise C_{+1} sind in diesem Fall parallel und haben den Abstand $\frac{2}{\|\mathbf{w}\|}$. Für jede trennende Hyperebene mit maximalem Rand existieren zwei derartige parallele stützende Hyperebenen. Offensichlich ist aber nicht jede Hyperebene, die eine der Mengen $C_{y^{(i)}}, i = \pm 1$ stützt, parallel zu einer trennenden Hyperebene.

3.3.2 Konvexe Optimierung

Für eine gegebene Trainingsmenge $\{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(\ell)}, y^{(\ell)})\} \subset F \times \{\pm 1\}$ lässt sich die Suche nach einer trennenden Hyperebene mit maximalem Rand $\frac{1}{\|\mathbf{w}\|}$ als quadratisches Programm $\mathsf{QP}_{\text{primal}}$ auffassen. Dieses ist in Abbildung 3.7 dargestellt. Da die Zielfunktion (3.10) strikt konvex ist und die Nebenbedingungen (3.11) konvex sind, ist ein lokales Minimum auch gleichzeitig eindeutiges globales Minimum.

[3] Sollte ein Punkt auf der Hyperebene liegen so können für ihn ohnehin keine Klassifikationsaussagen auf Basis der Hyperebene gemacht werden, weswegen er sinnvollerweise ignoriert wird.

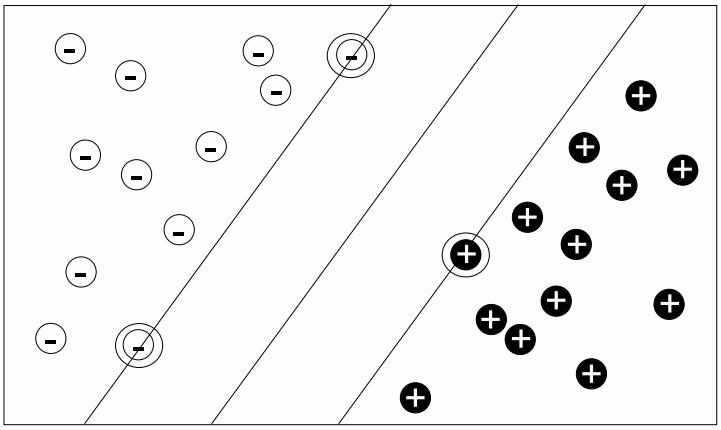


Abbildung 3.6: Eine trennende Hyperebene mit maximalem Rand. Stützvektoren sind eingekreist.

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \tag{3.10}$$

$$\text{so dass} \quad y^{(i)}(\langle \mathbf{x}^{(i)}, \mathbf{w}\rangle + b) \geq 1 \quad \forall i = 1, \ldots, \ell. \tag{3.11}$$

Abbildung 3.7: Quadratisches Programm $\mathsf{QP}_{\mathrm{primal}}$

Durch die Hinzunahme von Lagrange-Multiplikatoren $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_\ell) \in \mathbb{R}^\ell$ mit $\alpha_i \geq 0, i = 1, \ldots, \ell$ lässt sich die zugehörige Lagrange-Funktion aufstellen:

$$\mathfrak{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{\ell} \alpha_i (y^{(i)}(\langle \mathbf{x}^{(i)}, \mathbf{w}\rangle + b) - 1). \tag{3.12}$$

$\mathfrak{L}$ ist bezüglich der primalen Variablen $\mathbf{w}$ und b zu minimieren und bezüglich der dualen Variablen $\boldsymbol{\alpha}$ zu maximieren. Eine notwendige Bedingung für solch einen Sattelpunkt ist, dass die partiellen Ableitungen von $\mathfrak{L}$ nach den primalen Variablen 0 sind. Damit ergibt sich

$$0 = \frac{\partial \mathfrak{L}(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial b} = -\sum_{i=1}^{\ell} \alpha_i y^{(i)} \tag{3.13}$$

und

$$0 = \frac{\partial \mathfrak{L}(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{\ell} \alpha_i y^{(i)} \mathbf{x}^{(i)}, \tag{3.14}$$

also gilt in einem Sattelpunkt von $\mathfrak{L}$ notwendig $\sum_{i=1}^{\ell} \alpha_i \, y^{(i)} = 0$ und $\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y^{(i)} \mathbf{x}^{(i)}$.

Setzt man diese Bedingungen in (3.12) ein, so erhält man

$$\begin{aligned}\mathfrak{L}(\mathbf{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2}\langle\sum_{i=1}^{\ell}\alpha_i y^{(i)}\mathbf{x}^{(i)}, \sum_{i=1}^{\ell}\alpha_i y^{(i)}\mathbf{x}^{(i)}\rangle - \sum_{i=1}^{\ell}\alpha_i(y^{(i)}(\langle\mathbf{x}^{(i)}, \sum_{j=1}^{\ell}\alpha_j y^{(j)}\mathbf{x}^{(j)}\rangle + b) - 1) \\ &= -\frac{1}{2}\sum_{i,j=1}^{\ell}\alpha_i\alpha_j y^{(i)}y^{(j)}\langle\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\rangle - b\underbrace{\sum_{i=1}^{\ell}\alpha_i y^{(i)}}_{=0} + \sum_{i=1}^{\ell}\alpha_i \\ &= \sum_{i=1}^{\ell}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{\ell}\alpha_i\alpha_j y^{(i)}y^{(j)}\langle\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\rangle. \end{aligned} \tag{3.15}$$

Notation 3.3.3 *Zur Vereinfachung betrachte eine Matrix* $\mathbf{Q} \in \mathbb{R}^{\ell\times\ell}$ *mit den Einträgen*

$$q_{ij} = y^{(i)}y^{(j)}\langle\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\rangle \qquad \forall i, j = 1, \ldots, \ell.$$

Damit ergibt sich das zu $\mathsf{QP}_{\mathrm{primal}}$ duale quadratische Programm $\mathsf{QP}_{\mathrm{dual}}$, dargestellt in Abbildung 3.8. Die Zielfunktion (3.16) ist konvex, vergleiche Lemma 3.3.5, die Nebenbedingungen (3.17) und (3.18) sind linear. Damit ist $\mathsf{QP}_{\mathrm{dual}}$ durch konvexe Programmierung lösbar und ein lokales Optimum ist notwendig auch global optimal, wenn auch nicht mehr notwendig eindeutig.

$$\min_{\boldsymbol{\alpha}} -\boldsymbol{\alpha}^\top\mathbf{1} + \frac{1}{2}\boldsymbol{\alpha}^\top\mathbf{Q}\boldsymbol{\alpha} \tag{3.16}$$

so dass

$$\alpha_i \geq 0 \quad \forall i = 1, \ldots, \ell, \tag{3.17}$$

$$\sum_{i=1}^{\ell}\alpha_i y^{(i)} = 0. \tag{3.18}$$

Abbildung 3.8: Quadratisches Programm $\mathsf{QP}_{\mathrm{dual}}$

Sei nun $\boldsymbol{\alpha}^\star \in \mathbb{R}^\ell$ eine optimale Lösung von $\mathsf{QP}_{\mathrm{dual}}$. Nach den Karush-Kuhn-Tucker-Komplementaritätsbedingungen[4] gilt notwendig

$$\alpha_i^\star(y^{(i)}(\langle\mathbf{x}^{(i)}, \mathbf{w}\rangle + b^\star) - 1) = 0 \quad \forall i = 1, \ldots, \ell. \tag{3.19}$$

Aus dieser Gleichung lässt sich der Wert von $b^\star$ folgendermaßen bestimmen: Wähle ein beliebiges $i \in \{1, \ldots, \ell\}$ mit $\alpha_i > 0$. Dann gilt

$$y^{(i)}(\langle\mathbf{x}^{(i)}, \mathbf{w}\rangle + b^\star) - 1 = 0 \Leftrightarrow \langle\mathbf{x}^{(i)}, \sum_{j=1}^{\ell}\alpha_j y^{(j)}\mathbf{x}^{(j)}\rangle + b^\star = y^{(i)} \text{ da } y^{(i)} \in \{\pm 1\} \tag{3.20}$$

und damit

$$b^\star = y^{(i)} - \sum_{j=1}^{\ell}\alpha_j y^{(j)}\langle\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\rangle. \tag{3.21}$$

Aber aus Gleichung 3.19 lässt sich noch eine weitere Tatsache ablesen: Für ein $i \in \{1, \ldots, \ell\}$ kann $\alpha_i^\star \neq 0$ nur dann gelten, wenn Ungleichung (3.9) mit Gleichheit erfüllt ist, was gleichbedeutend damit ist, dass $\mathbf{x}^{(i)}$ ein Stützvektor ist. Üblicherweise gibt es nur vergleichsweise wenige Stützvektoren, so dass Summierungen mit $\boldsymbol{\alpha}$-Koeffizienten entsprechend wenig Aufwand erfordern: Es muss ja lediglich über die Menge $\mathrm{sv} \subseteq \{1, \ldots, \ell\}$ der Stützvektor-

[4] siehe hierzu beispielsweise [2], Abschnitt 5.5.3

Indizes summiert werden. Damit lässt sich die Klassifikator-Funktion, die das eigentliche Ziel des Lernverfahrens ist, folgendermaßen beschreiben:

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i \in \mathrm{sv}} \alpha_i y^{(i)} \langle \mathbf{x}, \mathbf{x}^{(i)} \rangle + b\right). \tag{3.22}$$

Die einzelnen Schritte werden in Abbildung 3.9 noch einmal als Algorithmus zusammengefasst. Durch die große Bedeutung der Stützvektoren bei der Definition der Hyperebene wird er als **Support Vector Machine** bezeichnet. Dadurch, dass Verstöße bei der Trennung der Punkte mit Rand nicht zulässig sind, wird der Rand als **fest** angesehen. Dies führt zu dem ergänzend vorangestellten Attribut **Hard Margin**.

Eingabe: Trainingsmenge $\{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(\ell)}, y^{(\ell)})\} \subset F \times \{\pm 1\}$.
Ausgabe: Klassifikator-Funktion $f(\cdot)$, welche einem Punkt $\mathbf{x} \in F$ eine Bezeichnung $y \in \{\pm 1\}$ zuordnet.
Schritt 1. Löse das quadratische Optimierungsproblem $\mathrm{QP}_{\mathrm{dual}}$ und erhalte optimales $\boldsymbol{\alpha}^\star \in \mathbb{R}^\ell$.
Schritt 2. Bestimme die Menge $\mathrm{sv} = \{i \in \{1, \ldots, \ell\} | \alpha_i^\star > 0\}$.
Schritt 3. Setze $\mathbf{w}^\star = \sum_{i \in \mathrm{sv}} \alpha_i^\star y^{(i)} \mathbf{x}^{(i)}$.
Schritt 4. Setze für ein beliebiges $i \in \mathrm{sv}$: $b^\star = y^{(i)} - \langle \mathbf{x}^{(i)}, \mathbf{w}^\star \rangle$.
Schritt 5. Ausgabe: $f(\cdot) = \operatorname{sgn}(\langle \mathbf{w}^\star, \cdot \rangle + b^\star)$.

Abbildung 3.9: Algorithmus: Hard Margin SVM

3.3.3 Kombination mit Kernfunktionen

Bisher wurde in diesem Abschnitt lediglich beschrieben, wie man eine lineare Klassifikator-Funktion in einem Hilbert-Raum erhalten kann. Es ist wünschenswert, diese Erkenntnisse verallgemeinern zu können. An diesem Punkt kommen die in Abschnitt 3.2 vorgestellten Kernfunktionen ins Spiel. Zur Erinnerung: Für einen Eingaberaum X und eine endlich positiv semidefinite Kernfunktion $\kappa : X \times X \to \mathbb{R}$ existiert ein Hilbert-Raum F und eine Merkmalsfunktion $\phi : X \to F$ so dass $\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle$ gilt.

Nun kommt eine bedeutende Eigenschaft der Support Vector Machine zum Tragen: Für die Bestimmung der optimalen Parameter $\boldsymbol{\alpha}^\star, b^\star$ werden gar nicht die Punkte im Hilbert-Raum selbst benötigt, sondern lediglich die Skalarprodukte von Punktpaaren. Auch die Klassifikator-Funktion (3.22) benötigt lediglich die Skalarprodukte der Stützvektoren mit dem zu klassifizierenden Punkt.

Notation 3.3.4 *Zur Vereinfachung betrachte auch hier eine Matrix $\mathbf{Q}^\kappa \in \mathbb{R}^{\ell \times \ell}$ mit den Einträgen*

$$q_{ij}^\kappa = y^{(i)} y^{(j)} \kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \qquad \forall i, j = 1, \ldots, \ell.$$

In Anlehnung an ($\mathrm{QP}_{\mathrm{dual}}$) lässt sich nun ein quadratisches Programm ($\mathrm{QP}_{\mathrm{dual}}^\kappa$) erstellen, dargestellt in Abbildung 3.10, indem man das Skalarprodukt $\langle \cdot, \cdot \rangle$ in (3.16) durch die Kernfunktion $\kappa(\cdot, \cdot)$ ersetzt.

Auch hier gilt: Die Zielfunktion (3.23) ist konvex, siehe hierzu Lemma 3.3.5, und die Nebenbedingungen (3.25) und (3.25) linear, also ist auch $\mathrm{QP}_{\mathrm{dual}}^\kappa$ durch konvexe Programmierung lösbar und ein lokales Optimum ist auch hier notwendig global optimal, auch wenn es nicht eindeutig sein muss.

Lemma 3.3.5 (nach [30]) *Die Matrix $\mathbf{Q}^\kappa$ ist positiv semidefinit.*

$$\min_{\boldsymbol{\alpha}} -\boldsymbol{\alpha}^\top \mathbf{1} + \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{Q}^\kappa \boldsymbol{\alpha} \tag{3.23}$$

so dass

$$\alpha_i \geq 0 \quad \forall i = 1, \ldots, \ell, \tag{3.24}$$

$$\sum_{i=1}^{\ell} \alpha_i y^{(i)} = 0. \tag{3.25}$$

Abbildung 3.10: Quadratisches Programm $\mathtt{QP}^{\kappa}_{\mathrm{dual}}$

Beweis: Sei $\mathbf{v} \in \mathbb{R}^\ell$ beliebig. Dann gilt:

$$\begin{aligned}\mathbf{v}^\top \mathbf{Q}^\kappa \mathbf{v} &= \sum_{i,j=1}^{\ell} v_i v_j y^{(i)} y^{(j)} \kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sum_{i,j=1}^{\ell} v_i v_j y^{(i)} y^{(j)} \langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle \\ &= \left\langle \sum_{i=1}^{\ell} v_i y^{(i)} \phi(\mathbf{x}^{(i)}), \sum_{j=1}^{\ell} v_j y^{(j)} \phi(\mathbf{x}^{(j)}) \right\rangle = \left\| \sum_{i=1}^{\ell} v_i y^{(i)} \phi(\mathbf{x}^{(i)}) \right\|^2 \geq 0.\end{aligned}$$

Also ist $\mathbf{Q}^\kappa$ positiv semidefinit, woraus insbesondere folgt, dass die Zielfunktionen (3.16) und (3.23) konvex sind. □

Sei nun eine Trainingsmenge $\{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(\ell)}, y^{(\ell)})\} \subset X \times \{\pm 1\}$ gegeben. Wir haben eine Abbildung $\phi : X \to F$ gegeben und wissen, wie in F eine trennende Hyperebene mit maximalem Rand bestimmt werden kann. Was liegt da näher, als die Bilder der Beispiele, also $\phi(\mathbf{x}^{(1)}), \ldots, \phi(\mathbf{x}^{(\ell)})$, als Eingangsdaten für den in Abbildung 3.9 dargestellten Algorithmus zu verwenden? Der modifizierte Algorithmus wird in Abbildung 3.11 dargestellt. Dabei wird auf eine explizite Berechnung des Richtungsvektors $\mathbf{w} \in F$ verzichtet, da die Punkte im Merkmalsraum nicht explizit benötigt werden und somit die Merkmalsabbildung ϕ nicht explizit angewendet werden muss.

Eingabe: Trainingsmenge $\{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(\ell)}, y^{(\ell)})\} \subset X \times \{\pm 1\}$ und Kern $\kappa : X \times X \to \mathbb{R}$.
Ausgabe: Klassifikator-Funktion $f(\cdot)$, welche einem Punkt $\mathbf{x} \in X$ eine Bezeichnung $y \in \{\pm 1\}$ zuordnet.
Schritt 1. Löse das quadratische Optimierungsproblem $\mathtt{QP}^{\kappa}_{\mathrm{dual}}$ und erhalte optimales $\boldsymbol{\alpha}^\star \in \mathbb{R}^\ell$.
Schritt 2. Bestimme die Menge $\mathrm{sv} = \{i \in \{1, \ldots, \ell\} | \alpha_i^\star > 0\}$.
Schritt 3. Setze für ein beliebiges $i \in \mathrm{sv}$: $b^\star = y^{(i)} - \sum_{j \in \mathrm{sv}} \alpha_j^\star y^{(j)} \kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$.
Schritt 4. Ausgabe: $f(\cdot) = \mathrm{sgn}\left(\sum_{i \in \mathrm{sv}} \alpha_i^\star y^{(i)} \kappa(\mathbf{x}^{(i)}, \cdot) + b^\star\right)$.

Abbildung 3.11: Algorithmus: Kernbasierte Hard Margin SVM

Auf diese Weise können auch nichtlineare Muster bestimmt werden. Insbesondere ist es möglich, für beliebige Datenräume X Musterfunktionen zu erhalten, solange man nur in der Lage ist, eine geeignete Kernfunktion anzugeben. Dabei ist allerdings stets zu beachten, dass die Berechnung der Kernfunktion vergleichsweise effizient sein sollte.

3.3.4 Zuverlässigkeit der Hard Margin SVM

Nachdem Algorithmen zur Bestimmung einer Klassifikator-Funktion vorgestellt wurden, soll an dieser Stelle genauer betrachtet werden, wie zuverlässig die Klassifikation jeweils ist. Dabei wird einerseits auf ein in Abschnitt

3.1.4 vorgestelltes Resultat zurückgegriffen und andererseits eine weitere Fehlerschranke vorgestellt, die auf kernbasiertes Lernen ausgerichtet ist. Der Unterschied zwischen den beiden Ansätzen basiert auf einer abweichenden Auslegung des Komplexitätsbegriffs der Funktionsklasse $\mathcal{F}$, aus der eine Musterfunktion f ausgewählt werden kann.

1. Ansatz: VC-Dimension

Bereits in Satz 3.1.6 wurde eine Schranke vorgestellt, die eine Abschätzung für den Generalisierungsfehler bei statistischem Lernen im Allgemeinen ermöglicht. Zur Erinnerung: Sei $f \in \mathcal{F}$ eine aus der empirischen Risikominimierung mit ℓ Trainingsbeispielen resultierende Musterfunktion, wobei die Funktionsklasse $\mathcal{F}$ die VC-Dimension h hat. Für den erwarteten Fehler gilt mit Wahrscheinlichkeit $1-\delta$, dabei $\delta \in (0,1)$, die Abschätzung

$$\int_{X\times Y} L(f(\mathbf{x}),y)dP(\mathbf{x},y) \leq \frac{1}{\ell}\sum_{i=1}^{\ell} L(f(\mathbf{x}^{(i)}),y^{(i)}) + \sqrt{\frac{h(\ln\frac{2\ell}{h}+1)-\ln\frac{\delta}{4}}{\ell}}. \tag{3.26}$$

Trennende Hyperebenen in einem N-dimensionalen Raum F haben stets die VC-Dimension $N+1$, das heißt dass $N+1$ geeignet angeordnete Punkte $\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(N+1)}$ in einem N-dimensionalen Raum bei jeder Zuordnung von Bezeichnungen $y^{(1)},\ldots,y^{(N+1)} \in \{\pm 1\}$ durch eine Hyperebene korrekt getrennt werden können, $N+2$ Punkte hingegen nicht.

Stellt man an die Hyperebene nach Definition 3.3.1 zusätzlich die Anforderung, dass sie die Trainingspunkte mit Rand $\frac{1}{\|\mathbf{w}\|} \leq \Lambda$ für ein $\Lambda > 0$ trennt, so ist die VC-Dimension beschränkt durch

$$h \leq \Lambda^2 R^2 + 1. \tag{3.27}$$

Dabei ist R der Radius einer beliebigen Kugel, die alle Trainingspunkte enthält.

Durch eine Vergrößerung des Randes lässt sich also die obere Schranke für die VC-Dimension und damit letztendlich für den erwarteten Fehler verringern, sofern die Trainingspunkte immer noch fehlerfrei mit Rand getrennt werden.

Weitere Überlegungen zu VC-Dimension und zugehörigen Schranken für den Generalisierungsfehler für die Hard Margin SVM findet man beispielsweise in [28] oder [34].

2. Ansatz: Rademacher-Komplexität

Eine etwas andere Herangehensweise wird in [30] geschildert: Hier wird für eine Funktion f aus einer Klasse $\mathcal{F}$ von Funktionen untersucht, wie gut sie zufällig generierte Daten klassifizieren kann. Die Bezeichnungen der Datenpunkte werden dabei als ± 1-wertige Zufallsvariablen, die so genannten *Rademacher-Variablen*, angesehen. Die danach benannte *Rademacher-Theorie* wird an dieser Stelle nicht vertieft, sondern es wird lediglich eine aus der Theorie resultierende Schranke für den erwarteten Fehler gegeben. Details sind beispielsweise unter der genannten Quelle zu finden.

Satz 3.3.6 (nach [30]) *Sei $\delta \in (0,1)$ gegeben. Betrachte eine Trainingsmenge von ℓ gemäß einer (unbekannten) Wahrscheinlichkeit $P(\mathbf{x},y)$ zufällig bestimmten Beispielen $\{(\mathbf{x}^{(1)},y^{(1)}),\ldots,(\mathbf{x}^{(\ell)},y^{(\ell)})\} \subset X\times Y$. Angenommen, die Punkte sind in einem durch ein κ implizit definierten Merkmalsraum F durch eine Hyperebene mit maximalem Rand $\gamma^\star = \frac{1}{\|\mathbf{w}\|}$ in kanonischer Form trennbar, bestimmt durch die Hard Margin Support Vector Machine, welche zusätzlich die Klassifikator-Funktion f liefert. Sei ferner $\mathbf{K}$ die zu der Punktmenge gehörige Kernmatrix. Dann gilt für den erwarteten Fehler die Abschätzung*

$$\int_{X\times Y} L(f(\mathbf{x}),y)dP(\mathbf{x},y) \leq \frac{4}{\ell\gamma^\star}\sqrt{\mathrm{tr}(\mathbf{K})} + 3\sqrt{\frac{\ln(2/\delta)}{2\ell}} \tag{3.28}$$

mit Wahrscheinlichkeit $1-\delta$.

Beweis: siehe [30], Theorem 4.17 und Theorem 7.22. □

Auch hier fällt auf, dass ein größerer Rand $\gamma^\star$ zu einer kleineren Fehlerschranke führt. Zusätzlich scheint eine Vergrößerung der Stichprobe unmittelbar zu einer Verringerung der Schranke zu führen, dies muss aber nicht notwendig zutreffen, da sich hierdurch einerseits die Spur der Kernmatrix um die quadrierten Normen der hinzugenommenen Werte vergrößert und andererseits die neuen Punkte zu einer Verkleinerung des Randes führen können.

4 Anwendung kernbasierter Lernverfahren auf das Anschlusssicherungsproblem

Abschnitt 4.1 befasst sich damit, einen Zusammenhang zwischen dem in Kapitel 1 vorgestellten Anschlusssicherungsproblem und dem in Kapitel 3 eingeführten Begriff der Klassifizierung herzustellen. Dies führt zu dem bisher noch wenig erforschten Gebiet der *multidimensionalen Klassifizierung*. Die Untersuchung des resultierenden Klassifikationsproblems mit dem in Abschnitt 3.3 vorgestellten Verfahren erfordert eine Kernfunktion κ. Abschnitt 4.2 befasst sich damit, was für Kerne dafür in Frage kommen. In Abschnitt 4.3 wird schließlich angesprochen, inwiefern einige der in Abschnitt 2.3.1 vorgestellten zusätzlich Heuristiken eingesetzt werden können, um den Zielfunktionswert weiter zu verbessern.

Dabei ist zu beachten, dass die Notation aus dem Bereich der Anschlusssicherung an die der kernbasierten Lernverfahren angepasst wird: Verspätungsdatensätze, die den Punkten im Datenraum X entsprechen, werden mit $\mathbf{x}$ beziehungsweise $\mathbf{x}^{(i)}$ bezeichnet, skalare Punktbezeichnungen mit y beziehungsweise $y^{(i)}$ und vektorwertige Punktbezeichnungen mit $\mathbf{y}$ beziehungsweise $\mathbf{y}^{(i)}$.

4.1 Multidimensionale Klassifizierung

In Abschnitt 1.3.1 wurde bereits angeführt, dass das Anschlusssicherungsproblem besonders einfach ist, wenn alle Anschlüsse festgelegt sind. In diesem Fall ist die Berechnung einer zeitminimalen Lösung effizient möglich. Ein Ansatz für eine neuartige Heuristik ist die Idee, Methoden des maschinellen Lernens anzuwenden. Dazu sind einzelne Verspätungsdatensätze in geeigneter Darstellung als Punkte $\mathbf{x}$ im Raum der möglichen Verspätungsdatensätze X anzusehen sowie jedem Anschluss eine ± 1-wertige Variable zuzuordnen, wobei $+1$ beispielsweise dem Halten und -1 dem Verpassen eines Anschlusses entspricht. Eine Instanz des Anschlusssicherungsproblems wird dann durch den Punkt $\mathbf{x}$, eine vollständige Warten/Nichtwarten Entscheidung durch einen Vektor $\mathbf{y} \in \{\pm 1\}^C$ identifiziert, wobei $C = |\mathcal{A}_{\text{change}}|$ die Anzahl der Anschlüsse bezeichnet. Jede einzelne Instanz lässt sich somit als eine Zusammenfassung von C parallelen Klassifizierungsproblemen mit Datenpunkt $\mathbf{x}$ und Bezeichnung y_i für $i = 1, \ldots, C$ ansehen.

Nun stellt sich die Frage, inwiefern die in Kapitel 3 angesprochenen Lernverfahren für solch ein Klassifizierungsproblem anwendbar sind: Schließlich muss jetzt nicht mehr nur in *einer* Dimension klassifiziert werden sondern in C Dimensionen, wobei jede einzelne Klassifikationsentscheidungen die anderen beeinflussen kann, da ja das Verpassen eines Anschlusses dazu führt, dass der Anschlusszug sich weniger (beziehungsweise gar nicht) verspätet und somit das Verpassen nachfolgender, vom Anschlusszug ausgehender Anschlüsse nur selten sinnvoll ist.

Lernverfahren für vektorwertige Bezeichnungen $\mathbf{y}$ sind bisher kaum verbreitet, aber es gibt erste Ansätze:

- In [33] werden beispielsweise multidimensionale Bayes'sche Netzwerkklassifikatoren vorgestellt, welche beim Lernen auch unmittelbare Abhängigkeiten zwischen den einzelnen Bezeichnungen berücksichtigen.
- Zur Durchführung von multidimensionaler Regression wird in [30] vorgeschlagen, die Lernaufgabe für einen n-dimensionalen Bezeichnungsvektor $\mathbf{y} \in \mathbb{R}^n$ in n unabhängig voneinander betrachtete Lernaufgaben zu zerlegen, also für jede Dimension eine herkömmliche Regression durchzuführen. So geht man zwar das Risiko ein, Informationen über die Zusammenhänge zwischen den einzelnen Bezeichnungskomponenten zu verlieren, ist aber in der Lage, das eigentliche Problem der multidimensionalen Regression zu umgehen.

Der zweite Ansatz soll nun auf das Problem der multidimensionalen Klassifikation übertragen werden: Anstatt in einem Schritt zu lernen, welcher Bezeichnungsvektor $\mathbf{y} \in \{\pm 1\}^C$ zu einem Punkt $\mathbf{x} \in X$ gehört, werden C einzelne Lernprobleme zur Bestimmung der jeweiligen Vektorkomponenten y_i für $i = 1, \ldots, C$ betrachtet. Dann können

für ℓ exemplarische Verspätungsdatensätze $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\ell)}$ die optimalen Bezeichnungsvektoren $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(\ell)}$ durch Lösen des zugehörigen in Abbildung 1.7 vorgestellten gemischt-ganzzahligen Programms (DM) bestimmt werden, beispielsweise durch das vorgestellte Branch-and-Bound Verfahren, und als Trainingsdaten für die C Lernprobleme verwendet werden. Der Zeitaufwand für diesen Lösungsprozess ist allerdings erheblich und auch das Lösen der Lernprobleme kann je nach gewähltem Lernverfahren viel Zeit in Anspruch nehmen.

Es ist anzumerken, dass nicht jedes Trainingsbeispiel für jeden Anschluss von Bedeutung ist, sondern nur für jene Anschlüsse, die bei dem jeweiligen Trainingsbeispiel relevant sind. Für alle anderen Verspätungsdatensätze kann der Anschluss schließlich trivialerweise gehalten werden. Dies hat zweierlei Folgen:

- Für einzelne Anschlüsse kann es unter Umständen schwierig sein, ausreichend viele Trainingsbeispiele zu finden, in denen sie relevant sind.
- Die Lernaufgaben für die einzelnen Anschlüsse werden leichter, weil nicht mehr alle Trainingsbeispiele in den Lernprozess einbezogen werden müssen, sondern nur noch jene, in denen der jeweilige Anschluss relevant ist.

Es können zwar durchaus für jeden Anschluss alle Trainingsbeispiele in den Lernprozess einbezogen werden – schließlich könnte auf diese Weise unter Umständen festgestellt werden, dass einige scheinbare Muster nur zufällige Phänomene waren. Allerdings steht dieser Nutzen in keinem Verhältnis zu dem damit verbundenen erhöhten Lernaufwand.

Bei der multidimensionaler Klassifizierung ist außerdem zu berücksichtigen, dass das Auftreten mindestens einer fehlerhaften Klassifizierung mit steigender Länge des Bezeichnungsvektors zunimmt. Für das Anschlusssicherungsproblem heißt das konkret: Je mehr Anschlüsse vorliegen beziehungsweise für einen zu untersuchenden Verspätungsdatensatz relevant sind, desto größer ist das Risiko, dass für mindestens einen Anschluss die falsche Warten/Nichtwarten Entscheidung getroffen wird.

4.2 Betrachtung verschiedener Kerne

Nachdem in Abschnitt 3.3 die kernbasierte Hard Margin Support Vector Machine als ein Lernverfahren für Klassifikationsaufgaben vorgestellt wurde, soll sie auch tatsächlich für das Erlernen guter Warten/Nichtwarten Entscheidungen eingesetzt werden. Dafür ist zunächst noch zu entscheiden, welche Merkmalsabbildung ϕ und damit welcher Merkmalsraum F und welche Kernfunktion κ zu wählen ist. Es werden verschiedene Ansätze vorgestellt und diskutiert.

4.2.1 Quellverspätungskern

Die einfachste Variante besteht sicherlich darin, einen Verspätungsdatensatz $\mathbf{x} \in X$ auf einen Vektor abzubilden, der eine Komponente für jedes Ereignis des Ereignis-Aktivitäts-Netzwerks enthält, wobei für ein Ereignis $e_i \in \mathcal{E}$ die i-te Vektorkomponente der Quellverspätung des Ereignisses e_i entspricht. Die resultierenden Vektoren $\phi(\mathbf{x})$ sind dann schwach besetzt, da die Anzahl der Nicht-Null-Einträge der Anzahl der Quellverspätungen entspricht.

Da sowohl Quell- als auch Folgeverspätungen linear in die Zielfunktion des Programms (DM) eingehen, ist zu hoffen, dass die Verspätungsvektoren im $|\mathcal{E}|$-dimensionalen Merkmalsraum tatsächlich für jeden Anschluss linear in Warten- und Nichtwarten Entscheidungen einzuteilen sind. Allerdings ist der Kern möglicherweise etwas zu einfach, um tatsächlich gute Ergebnisse zu erzielen. Schließlich ist er nur für jene Paare von Verspätungsdatensätzen ungleich Null, in denen jeweils das selbe Ereignis quellverspätet ist.

4.2.2 Verspätungsausbreitungskern

Ein etwas komplexerer Ansatz basiert darauf, auch jene Folgeverspätungen zu berücksichtigen, die bei gegebenen Quellverspätungen ohnehin unvermeidbar sind, das heißt jene Verspätungen, die über Fahrt- und Warteaktivitäten bei bestmöglicher Ausnutzung der Pufferzeiten auftreten. Man berücksichtigt also, wie weit sich die Verspätungen

im Ereignis-Aktivitäts-Netzwerk ausbreiten. Dieser Ansatz wird dadurch motiviert, dass jede dieser Folgeverspätungen auf nachfolgende Anschlüsse eine sehr ähnliche[1] Auswirkung hat, wenn nur sie auftreten würde und nicht die eigentliche Quellverspätung.

Auch in diesem Fall ist der Merkmalsraum $|\mathcal{E}|$-dimensional und die Merkmalsvektoren sind schwach besetzt, wenn auch nicht mehr ganz so schwach wie bei dem eben vorgestellten Quellverspätungskern. Auf diese Weise werden Eigenschaften des Netzwerks in die Kernfunktion integriert, was auf ein Ergebnis hoffen lässt, dass mehr Aussagekraft hat als jenes, welches aus der Verwendung des Quellverspätungskerns resultiert.

Ebenso wie der Quellverspätungskern erfolgt die Auswertung des Kernes durch explizite Berechnung des Skalarproduktes im Merkmalsraum, was jedoch aufgrund der schwachen Besetzung der Merkmalsvektoren effizient möglich ist.

4.2.3 All-Pfad-Verspätungskern

Ein weiterer Ansatz stellt jene Verspätungen in den Vordergrund, welche entlang von Pfaden im Ereignis-Aktivitäts-Netzwerk weitergegeben werden. Dabei wird für jeden Pfad die gewichtete Verspätung betrachtet, die bei Halten aller Anschlüsse bei dem letzten Ereignis auftreten würde.

Sei P die Menge aller Pfade im Netzwerk, $p \in P$ ein beliebiger Pfad. Bezeichne für einen Verspätungsdatensatz $\mathbf{x} \in X$ mit $f_p(\mathbf{x}) \in \mathbb{N}_0$ die Verspätung am Ende des Pfades $p \in P$. Definiere dann die Merkmalsabbildung ϕ durch die Eigenschaft $\phi(\mathbf{x})_p = w_e \cdot f_p(\mathbf{x})$, wobei $e \in \mathcal{E}$ das Ereignis am Ende des Pfades p ist. Der Merkmalsraum F hat somit die Dimension $|P|$. Da es im Ereignis-Aktivitäts-Netzwerk sehr viele Pfade gibt, ist diese Dimension sehr hoch: Schließlich enthält bereits ein linienförmiger Graph mit $n \in \mathbb{N}$ Kanten $\frac{n(n-1)}{2}$ Pfade und die Anzahl kann sich mit zunehmender Anzahl von Verzweigungen drastisch erhöhen.

Im Gegensatz zu den bisher vorgestellten Kernen ist hier der Berechnungsaufwand erheblich, auch wenn es stellenweise möglich ist, Pfade mit gemeinsamen Teilabschnitten zusammenzufassen, sofern die übertragene Verspätung gleich ist. Dafür werden durch diesen Kern wichtige Informationen über die Struktur des Netzwerkes an den Lernalgorithmus übermittelt, welche bei den bisher geschilderten Ansätzen ganz oder zumindest zum größten Teil verloren gingen.

4.2.4 Zusammengesetzte Kerne

Anhand der in Abschnitt 3.2.2 vorgestellten Techniken zur Konstruktion von Kernen besteht durchaus die Möglichkeit, die bisher vorgestellten Kerne zu erweitern und beispielsweise eine polynomiale Variante des Quellverspätungs- oder Verspätungsausbreitungskerns zu betrachten. Erste Versuche mit praktischen Daten haben allerdings zu keinen befriedigenden Ergebnissen geführt so dass dieser Ansatz nicht weiter verfolgt wurde. Es ist aber denkbar, dass die in Kapitel 6 vorgestellten Ergebnisse durch geeignete Kernkonstruktionen noch weiter zu verbessern sind, ohne die Auswertungseffizienz allzu sehr zu beeinträchtigen. Es ist allerdings zu beachten, dass jede Modifikation der Kernfunktion einen Neustart des Lernprozesses erfordert und somit sehr aufwendige Konsequenzen hat.

4.3 Nachbesserung durch Heuristiken

Bisher wurde davon ausgegangen, dass die Ergebnisse der multidimensionalen Klassifikation eines Verspätungsdatensatzes exakt umgesetzt wurden, das heißt jeder mit -1 klassifizierte Anschluss wird verpasst und jeder mit $+1$ klassifizierte Anschluss wird gehalten. Manchmal sind aber Verbesserungen an diesen Entscheidungen leicht zu erkennen: Beispielsweise macht es keinen Sinn, einen Anschluss zu verpassen, wenn die Verspätung des Ankunftsereignisses $e \in \mathcal{E}$, von dem die entsprechende Umsteigeaktivität $a \in \mathcal{A}_{\text{change}}$ ausgeht, durch die Pufferzeit s_a kompensiert werden kann.

[1] Man kann sich überlegen, dass die Wirkung gleich ist, wenn die in Abschnitt 1.3.2 angesprochene Never-Meet-Property erfüllt ist.

Die Warten/Nichtwarten Entscheidungen können also durch den Einsatz von Heuristiken nachgebessert werden. Die oben beschriebene Vorgehensweise entspricht beispielsweise der Heuristik *Miss First Delayed Change*, wobei mit $+1$ klassifizierte Anschlüsse stets gehalten werden: Höchstens jene Anschlüsse werden verpasst, über die tatsächlich eine Verspätung übertragen wird. Entsprechend ist eine Anwendung der Heuristik *Miss First Delayed Change if Reasonable* denkbar und zeigt für Versuchsdaten sehr gute Ergebnisse, wie in Kapitel 6 noch zu sehen sein wird.

Aufgrund der hohen Qualität der jeweils bestimmten Lösungen werden die wesentlichen Schritte des heuristikverbesserten multidimensionalen Klassifikators noch einmal kurz zusammengefasst:

Vorbereitung: Bestimme für eine Trainingsmenge je einen Klassifikator $f_a(\cdot)$ für jeden Anschluss $a \in \mathcal{A}_{\mathrm{change}}$ mit Hilfe der Hard Margin SVM. Sortiere die Anschlüsse in $\mathcal{A}_{\mathrm{change}}$ nach der partiellen Ordnung $\prec$.

Klassifizierung: Sei ein Testverspätungsdatensatz gegeben in Form eines Verspätungsvektors $\mathbf{x}$.
Für alle $a \in \mathcal{A}_{\mathrm{change}}$ in der sortierten Reihenfolge:

- Falls $f_a(\mathbf{x}) = +1$: Halte den Anschluss a.
- Falls $f_a(\mathbf{x}) = -1$: Prüfe mit Hilfe der Heuristik *Miss First Delayed Change if Reasonable*, ob ein Halten von a sinnvoll erscheint und halte ihn nur dann.

Selbstverständlich kann man auch für jeden Verspätungsdatensatz, sofern die Ressourcen ausreichen, alle Varianten betrachten und schließlich die jeweils beste wählen.

5 Implementierung

Dieses Kapitel stellt einige Aspekte vor, welche bei der Implementierung der in dieser Arbeit vorgestellten Überlegungen eine Rolle gespielt haben.

Zunächst wird in der grundlegende Systementwurf in Abschnitt 5.1 vorgestellt. Dabei ist zu beachten, dass die Implementierung in der Programmiersprache C++ vorgenommen wurde. Bei diesem Systementwurf wurden einige Entwurfsmuster verwendet. Daher wird das Konzept der Entwurfsmuster, welche in der modernen Softwareentwicklung eine immer wichtigerere Rolle einnehmen, in Abschnitt 5.2 vorgestellt. Dieser richtet sich vorrangig nach [7], welches als Standardwerk zu diesem Thema angesehen wird.

Die Implementierung verwendet außerdem Schnittstellen zu externer Software, welche einige der angesprochenen Teilaufgaben sehr effizient lösen. Diese werden in Abschnitt 5.3 vorgestellt.

5.1 Grundstruktur des Systems

Zur Untersuchung des Anschlusssicherungsproblems wurde ein aus mehreren Modulen bestehendes System entwickelt. An dieser Stelle wird nur das Hauptmodul *DisKon_cpp* detailliert vorgestellt, weitere Module zur Unterstützung der Funktionalität sind:

- *storing:* Implementierung einfacher Datenstrukturen zur Speicherung von Daten: Liste, Vektor, Prioritätenwarteschlange und Stack.
- *sorting:* Implementierung von Sortieralgorithmen, insbesondere Merge-Sort, da dieser besonders gut mit sequentiellen Listen arbeiten kann.
- *OptimizerInterface:* Deklaration einer einheitlichen Schnittstelle für die Anbindung externer Optimierungssoftware. Derzeit wurde lediglich eine Schnittstelle für GLPK implementiert, eine Anbindung von Xpress ist angedacht.
- *GenAlgo:* Implementierung der Metaheuristik *Genetisches Verfahren*.

Außerdem gehört zu dem Programm noch eine Benutzerschnittstelle, die wahlweise über Menüs oder über Kommandozeile angesteuert werden kann.

Die zu dem Hauptmodul DisKon_cpp gehörenden Klassen und die wichtigsten Beziehungen können dem in Abbildung 5.1 gezeigten UML-Diagramm entnommen werden. Um die Lesbarkeit zu erhöhen wurde hierbei auf Details verzichtet. Die dargestellten Klassen und ihr Zweck sollen an dieser Stelle kurz beschrieben werden.

`tcTimetable`: Implementiert das durch einen Fahrplan definierte Ereignis-Aktivitäts-Netzwerk mit allen Bahnhöfen und Fahrzeugen. Die meisten Klassen des Moduls greifen auf die Attribute dieser Klasse zu, um mit dem Fahrplan arbeiten zu können.

`tcDelayGenerator`: Mit dieser Klasse lassen sich Verspätungsdatensätze für einen vorhandenen Fahrplan erzeugen.

`tcEvent`: Ereignisse im Ereignis-Aktivitäts-Netzwerk werden durch diese Klasse modelliert.

`tcActivity`: Aktivitäten im Ereignis-Aktivitäts-Netzwerk werden durch diese Klasse modelliert.

`tcStation`: dient der Modellierung von Bahnhöfen.

`tcVehicle`: dient der Modellierung von Fahrzeugen.

`tcNetworkReducer`: Bereitstellung von Operationen zur Reduktion eines Ereignis-Aktivitäts-Netzwerk.

`tcConnSolver`: Aufbau einer internen Repräsentation eines zu einer Instanz von (`DM`) gehörigen gemischt-

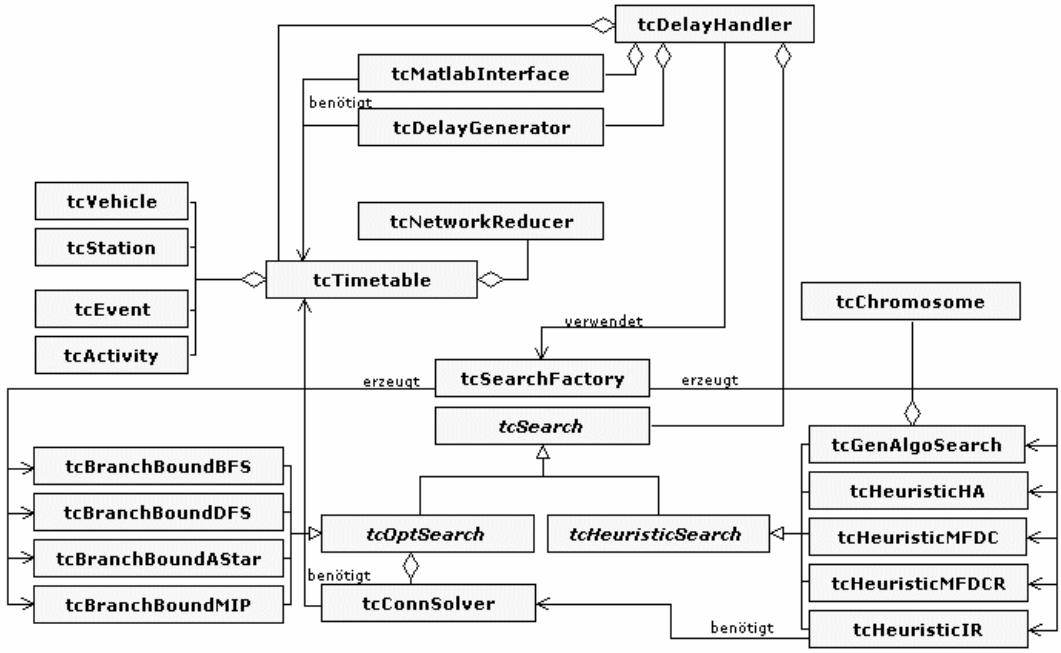


Abbildung 5.1: UML-Diagramm: Das Subsystem DisKon_cpp

ganzzahligen Programms mit Hilfe von externer Optimierungssoftware unter der Verwendung der Bibliothek *OptmizerInterface*.

`tcSearch`: Abstrakte Oberklasse, beschreibt die Suche nach möglichst guten Lösungen für eine Instanz von (`DM`).

`tcOptSearch`: Abstrakte Unterklasse von `tcSearch`. Ist Oberklasse für Klassen, die verschiedene Branch-and-Bound Suchregeln umsetzen, um die Optimallösung einer Instanz von (`DM`) zu bestimmen.

`tcBranchBoundAStar`: Unterklasse von `tcOptSearch`, implementiert die A*-Suche.

`tcBranchBoundBFS`: Unterklasse von `tcOptSearch`, implementiert die Breitensuche.

`tcBranchBoundDFS`: Unterklasse von `tcOptSearch`, implementiert die Tiefensuche.

`tcBranchBoundMIP`: Unterklasse von `tcOptSearch`, die die ganzzahlige Lösung der Instanz von (`DM`) an die externe Optimierungssoftware delegiert.

`tcHeuristicSearch`: Abstrakte Unterklasse von `tcSearch`. Ist Oberklasse für Klassen, die Heuristiken für eine Instanz von (`DM`) umsetzen.

`tcHeuristicHA`: Unterklasse von `tcHeuristicSearch`, implementiert die Heuristik *Hold All*.

`tcHeuristicIR`: Unterklasse von `tcHeuristicSearch`, implementiert die Heuristik *Integer Rounding*.

`tcHeuristicMFDC`: Unterklasse von `tcHeuristicSearch`, implementiert die Heuristik *Miss First Delayed Change*.

`tcHeuristicMFDCR`: Unterklasse von `tcHeuristicSearch`, implementiert die Heuristik *Miss First Delayed Change if Reasonable*.

`tcGenAlgoSearch`: Unterklasse von `tcHeuristicSearch`, implementiert die Metaheuristik *Genetisches Verfahren* mit Hilfe des Moduls *GenAlgo*.

`tcChromosome`: Klasse zur Implementierung eines für das Anschlusssicherungsproblem zugeschnittene Chromosom für das in `tcGenAlgoSearch` implementierte Genetische Verfahren.

`tcSearchFactory`: Klasse zum Erzeugen von Unterklassen von `tcSearch`, also konkreten Suchverfahren.

`tcMatlabInterface`: Schnittstelle zu der externen Software MATLAB für Datenverwaltung und Ansteuerung der Kernbasierten Lernverfahren.

`tcDelayHandler`: Fassadenklasse für externe Anwender, um die Anzahl der für den Einsatz benötigten Schnittstellenklassen zu reduzieren.

5.2 Verwendete Entwurfsmuster

Viele Programmieransätze tauchen in verschiedenen Programmen immer wieder auf. Oft handelt es sich dabei um einfache Ideen, die schnell umgesetzt sind – leider ist die Umsetzung häufig konkret auf das jeweils vorliegende Problem zugeschnitten. Dies hat zur Folge, dass ohne übergeordnete Strukturierung aufgrund unzureichender Wiederverwendbarkeit fast die gleiche Lösung immer wieder neu angepasst und umgesetzt werden muss. Möglicherweise erfordert dann auch jede Änderung der Implementierung, dass die vorhandene Lösung für das Teilproblem erneut angepasst werden muss, weil sie sich als nicht flexibel genug erwiesen hat.

An dieser Stelle setzt das Konzept der *Entwurfsmuster*[1] (englisch: *Design Patterns*) an: Ein Entwurfsmuster stellt dabei eine Schablone dar, wie ein vorliegendes Teilproblem mit Hilfe bewährter Programmiertechniken flexibel und wiederverwendbar umgesetzt werden kann. Manchmal geht es dabei lediglich um eine konsequente und disziplinierte Nutzung objektorientierter Ansätze, welche vielfach erprobt sind und sich immer wieder als nützlich erwiesen haben.

Auch wenn es nicht immer einfach ist, einen Zugang zu diesen Mustern zu finden, hilft eine hohe Musterdichte bei der Strukturierung eines Programms, erleichtert die Wartung und Erweiterung und erhöht die Lesbarkeit. Auch ist ein Programm, welches Entwurfsmuster sinnvoll verwendet, für Entwurfsmusterkundige leichter nachvollziehbar, Entwurfsmuster stellen quasi das Vokabular einer neuen Sprache dar.

Dabei wird zwischen drei grundlegenden Arten von Entwurfsmustern unterschieden, welche wiederum jeweils *klassenbasierte* und *objektbasierte* Muster unterteilt werden können:

Erzeugungsmuster befassen sich mit dem Erzeugungsprozess neuer Objekte. Welche Objekte erzeugt werden sollen und wie sie konfiguriert werden sollen, wird häufig erst zur Laufzeit entschieden – wobei das verwendete Erzeugungsmuster unterstützend wirken soll. Dabei verwenden *klassenbasierte* Erzeugungsmuster Vererbung, um Objekte verschiedener Klassen zu erzeugen. *Objektbasierte* Erzeugungsmuster hingegen delegieren die Erzeugung neuer Objekte an ein geeignetes Objekt.

Strukturmuster helfen dabei, den strukturellen Aufbau verschiedener Klassen und Objekte sinnvoll zu koordinieren. Dies kann geschehen, indem durch Vererbung Schnittstellen und Implementierungen zusammenzuführen; in diesem Fall spricht man von *klassenbasierten* Strukturmustern. Einen anderen Ansatzpunkt stellen *objektbasierte* Strukturmuster dar, welche Objekte auf raffinierte Weise zusammenführen, um neue Funktionalitäten zu erhalten.

Verhaltensmuster schließlich unterstützen eine sinnvolle Zusammenarbeit verschiedener Klassen und Objekte sowie die Aufteilung von Zuständigkeiten. Dabei wird das Verhalten bei *klassenbasierten* Verhaltensmustern durch Vererbung unter verschiedenen Klassen aufgeteilt. Bei *objektbasierten* Verhaltensmustern wird diese Aufteilung durch Komposition erzielt.

Bei der Implementierung der Verfahren zur Untersuchung des Anschlusssicherungsproblems mit Hilfe von Kernbasierten Lernverfahren kommen einige dieser Entwurfsmuster zum Einsatz. Diese werden an dieser Stelle kurz vorgestellt, auch der Einsatz dieser Muster soll demonstriert werden. Für eine Auflistung und ausführliche Vorstellung diverser üblicher Entwurfsmuster sei an dieser Stelle auf [7] verwiesen. Aus dieser Quelle stammen auch die verwendeten Abbildungen zur Illustration der jeweiligen Entwurfsmuster, welche nach der Modellierungssprache *Object Modeling Language* entworfen wurden.

[1] Der Begriff des *Musters* hat in diesem Abschnitt eine andere Bedeutung als in Kapitel 3.

5.2.1 Iterator

Es kommt immer wieder vor, dass man sequentiell auf die einzelnen Objekte eines zusammengesetzten Objektes, beispielsweise einer Liste, zugreifen möchte. Dabei kann es vorkommen, dass gar nicht bekannt ist, wie das zusammengesetzte Objekt, das so genannte *Aggregat* aufgebaut ist. Stehen verschiedene Implementierungen, beispielsweise einfach oder doppelt verkettete Liste, zur Verfügung, so steht möglicherweise erst zur Laufzeit fest, welche Implementierung gewählt wurde und damit wie auf die einzelnen Objekte zugegriffen werden kann.

Hier setzt das objektbasierte Verhaltensmuster *Iterator* an: Nicht die `Aggregat`-Objekte selber kümmern sich darum, wie die enthaltenen Objekte traversiert werden, sondern es werden ihnen `Iterator`-Objekte zugeordnet, die diese Aufgabe übernehmen. Dabei können die enthaltenen Objekte auch gleichzeitig von mehreren Iterator-Objekten durchlaufen werden.

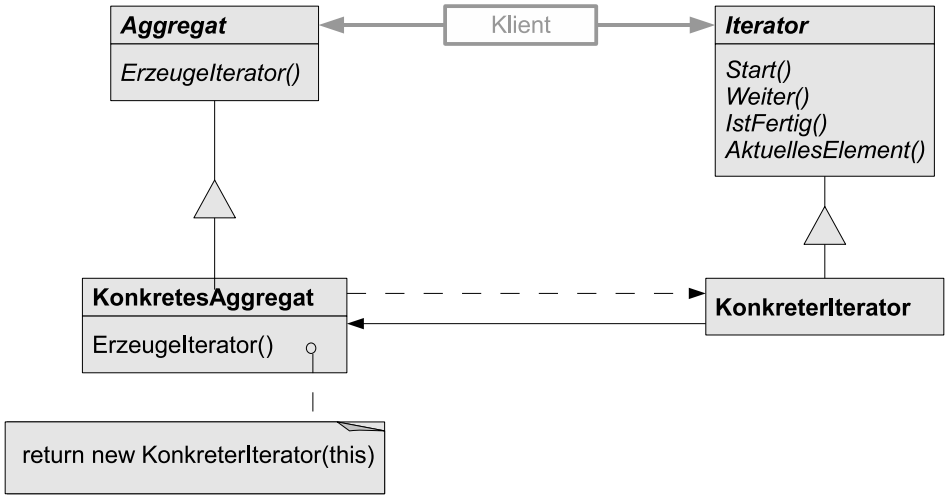


Abbildung 5.2: Strukturdiagramm für das Entwurfsmuster Iterator

Abbildung 5.2 zeigt ein Strukturdiagramm der Klassen und Beziehungen, die für das Iterator-Muster von Bedeutung sind: Der Klient selbst kennt lediglich die Klassen `Iterator`, welche eine Schnittstelle zum Zugriff auf und zur Traversierung von Elementen definiert, und `Aggregat`, welche eine Schnittstelle zum Erzeugen eines Objekts der Klasse Iterator definiert. Darüber hinaus gibt es die Klasse `KonkreterIterator`, welche die Schnittstelle der Klasse Iterator implementiert und die aktuelle Position während der Traversierung des Aggregats verwaltet. Außerdem gehört auch die Klasse `KonkretesAggregat` dazu, welche die Operation zum Erzeugen eines konkreten Iterators durch Rückgabe eines Objektes der passenden KonkreterIterator-Klasse implementiert. Schließlich ist anzumerken, dass auf die abstrakte Aggregats-Klasse verzichtet werden kann, wenn sichergestellt ist, dass nur eine KonkretesAggregat-Klasse verwendet wird. Hierbei wird allerdings die Erweiterbarkeit des Entwurfs eingeschränkt.

Die Datenstruktur *Liste* ist ein besonders typisches Beispiel für die Verwendung des Entwurfsmusters Iterator: In einer abstrakten Listenklasse werden zunächst alle Methoden zusammengefasst, die für die Nutzung der Datenstruktur notwendig sind. Außerdem wird die angesprochene abstrakte Methode zur Erzeugung eines Iterators hinzugefügt. Von dieser Klasse erbt dann eine konkrete Implementierung der Datenstruktur, welche die Methode zur Erzeugung eines für sie geeigneten Listeniterators implementiert. Die Objekte der Listeniterator-Klasse sind in der Lage, die bei der Erzeugung an sie übergebene Liste nach und nach zu durchlaufen (Methode `Weiter()`), zu prüfen ob das Ende der Liste erreicht wurde (Methode `IstFertig()`), das aktuelle Listenelement zurückzugeben (Methode `AktuellesElement()`) sowie den Listendurchlauf erneut zu starten (Methode `Start()`).

Bei der Umsetzung von Fahrplänen wurden immer wieder Listen eingesetzt, um beispielsweise Mengen von Bahnhöfen, Fahrzeugen, Ereignissen und Aktivitäten zu repräsentieren. Bei dem Zugriff auf diese Daten kamen daher auch immer wieder Iteratoren zum Einsatz.

5.2.2 Schablonenmethode

Oftmals kann ein Programmteil zur Bearbeitung einer Aufgabe auf verschiedene Arten implementiert werden, beispielsweise durch verschiedene Arten von Entscheidungsregeln. Dabei ist der grobe Programmablauf oftmals sehr ähnlich, die Variationen findet man oft nur in einzelnen Details.

Der Ansatzpunkt für das klassenbasierte Verhaltensmuster *Schablonenmethode* (englisch: *Template Method*) ist der, dass der grobe Algorithmus wie ein Programmschablone als Methode in einer Basisklasse definiert wird, während einzelne Elemente des Algorithmus an Unterklassen delegiert wird, das heißt es werden abstrakte Methoden aufgerufen, die in Unterklassen überschrieben werden.

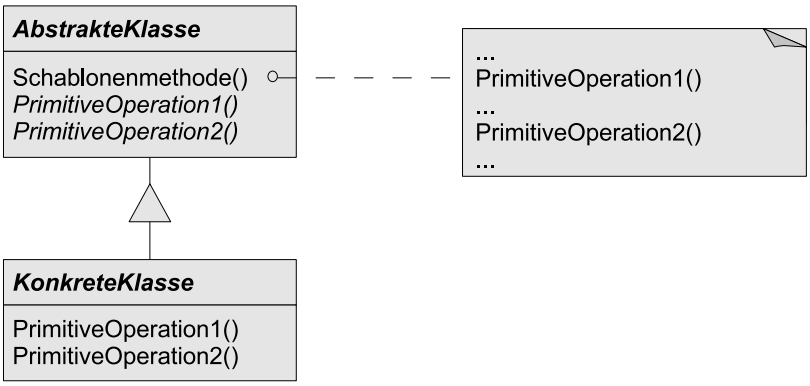


Abbildung 5.3: Strukturdiagramm für das Entwurfsmuster Schablonenmethode

Abbildung 5.3 zeigt ein Strukturdiagramm der Klassen und Beziehungen, die für das Entwurfsmuster Schablonenmethode von Bedeutung sind: In `AbstrakteKlasse` wird der eigentliche Algorithmus als `Schablonenmethode()` definiert. Diese verwendet die abstrakten primitiven Methoden `PrimitiveOperation1()` und `PrimitiveOperation2()` als einzelne Algorithmusschritte. Letztere werden allerdings nicht in AbstrakteKlasse definiert, sondern erst in der Unterklasse `KonkreteKlasse`. Auf diese Weise wird die Grobstruktur des Algorithmus durch `AbstrakteKlasse` und das detaillierte Verhalten durch `KonkreteKlasse` vorgegeben.

Damit stellt das Muster grundlegende Konzepte der Vererbung in den Vordergrund: Schritte, die für alle Ausprägungen eines Algorithmus gleich sind, werden von der Basisklasse übernommen, während abweichende Details von den konkreten Unterklassen implementiert werden. Auf diese Weise werden Codewiederholungen vermieden, während durch die abstrakten Methoden von AbstrakteKlasse genaue Einschubpunkte für erweiternde Unterklassen vorgegeben wird.

Bei der Implementierung von Strategien für die Suche nach einer möglichst guten Lösung für das Anschlusssicherungsproblem wurde das Entwurfsmuster Schablonenmethode eingesetzt, indem die abstrakte Klasse `tcSearch` eine Methode `solveWrapper()` zur Verfügung stellt. In dieser wird beispielsweise eine Stoppuhr gestartet, welche im Verlauf der Suche nach der Optimallösung für die Prüfung eines Abbruchkriteriums verwendet werden kann. Schließlich wird die abstrakte Methode `solve()` aufgerufen, welche die eigentliche Suche nach der Lösung umsetzt, welche in geeigneten Unterklassen implementiert wird. Wenn nun durch ein Kommando von außen die Suche nach einer Lösung gestartet werden soll, genügt ein Aufruf der Methode `solveWrapper()`. Zusätzlich stellt die Klasse `tcSearch` weitere Methode zur Verfügung, beispielsweise eine Prüfmethode, ob eine Zeitschranke bei der Suche nach einer Lösung verletzt wurde.

5.2.3 Fabrikmethode

Manchmal kann es vorkommen, dass zur Laufzeit neue Objekte erzeugt werden müssen, wobei zum Zeitpunkt des Bindens noch nicht feststeht, um welche Objekte es sich handelt – lediglich eine möglicherweise abstrakte

Oberklasse ist bekannt.

Eine Möglichkeit, diesem Problem zu begegnen, stellt das klassenbasierte Erzeugungsmuster *Fabrikmethode* (englisch: *Factory Method*) dar: Hier wird durch geeignete Vererbungshierachien sichergestellt, dass dynamisch ein Objekt der gewünschten Klasse erzeugt wird.

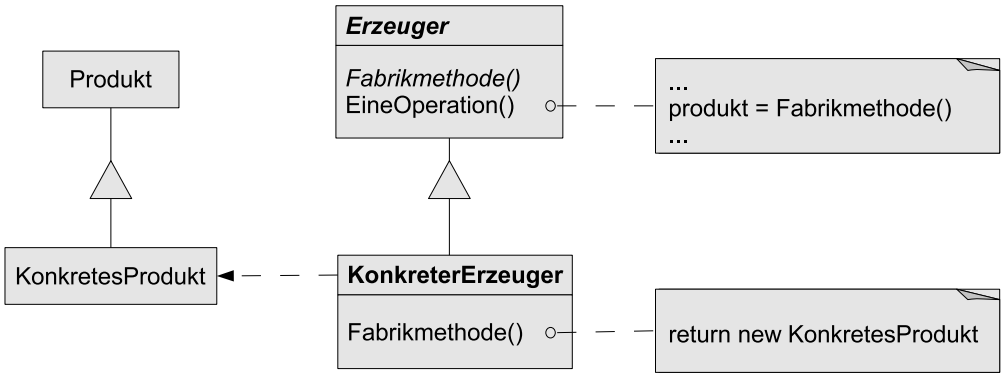


Abbildung 5.4: Strukturdiagramm für das Entwurfsmuster Fabrikmethode

Abbildung 5.4 zeigt ein Strukturdiagramm der Klassen und Beziehungen, die für das Entwurfsmuster Fabrikmethode von Bedeutung sind: Auf der einen Seite gibt die abstrakte Klasse `Produkt` vor, zu welcher Oberklasse die durch die Fabrikmethode erzeugbaren Objekte gehören und `KonkretesProdukt` stellt dar, welches Objekt erzeugt werden kann. Auf der anderen Seite steht die eigentliche Anwendung, die abstrakte Klasse `Erzeuger`. Sie deklariert die `Fabrikmethode()`, die ein Objekt der Klasse `Produkt` zurückgibt. Unter Umständen kann auch die Erzeugerklasse selbst die `Fabrikmethode()` aufrufen, um ein `Produkt`-Objekt zu erzeugen. Außerdem gehört noch die Klasse `KonkreterErzeuger` dazu, welche die `Fabrikmethode()` überschreibt und damit ein Objekt der Klasse `KonkretesProdukt` zurückgibt.

Ein Beispiel für den Einsatz des Entwurfsmuster Fabrikmethode ist das in Abschnitt 5.2.1 vorgestellte Iterator-Muster: Die Methode `ErzeugeIterator()` übernimmt die Rolle der Fabrikmethode, um bei Bedarf ein an die zugrunde liegende Datenstruktur angepasstes `Iterator`-Objekt erzeugen zu können.

Bei der Erzeugung von Objekten zur Suche nach guten beziehungsweise optimalen Lösungen für das Anschlusssicherungsproblem wurde eine Variation des Entwurfsmusters Fabrikmethode verwendet: Hierbei stellt eine Klasse `tcSearchFactory` mit `createSearch(char* )` eine statische Methode zur Erzeugung von `tcSearch`-Objekten zur Verfügung. Dabei gibt der Parameter an, von welcher Unterklasse von `tcSearch` das zu erzeugende Objekt sein soll. Die Klasse `tcSorterFactory` nimmt hierbei gleichzeitig die Rolle von `Erzeuger` und `KonkreterErzeuger` ein, da die Bestimmung der Klasse des zu erzeugenden Objektes durch den Parameter vorgegeben wird. Da die Objekterzeugung die einzige Aufgabe der Klasse `tcSearchFactory` ist, wurde `createSearch(char* )` als Klassenmethode angelegt.

5.2.4 Fassade

Oftmals können die verschiedenen Funktionalitäten eines Subsystems nur dann vernünftig genutzt werden, wenn tatsächlich alle oder zumindest viele Klassen eingebunden werden. Daher wäre es wünschenswert, eine einzelne Schnittstelle zu verwenden anstatt auf die Schnittstellen der vielen einzelnen Klassen angewiesen zu sein.

Dies ist der Ansatzpunkt für das Objektbasierte Strukturmuster *Fassade* (englisch: *Facade*): Einem Subsystem wird eine Fassadenklasse hinzugefügt, welche diese einheitliche Schnittstelle realisiert. Zur Laufzeit wird ein Objekt dieser Fassadenklasse erzeugt. Anfragen an die Objekte im Subsystem werden nun nicht mehr direkt an die Objekte selber sondern zunächst an das Fassadenobjekt gesendet, welches die notwendigen weiteren Schritte einleitet. Die Subsystemklassen werden auf diese Weise gewissermaßen gekapselt: Man muss sie und ihre Schnittstellen nicht mehr kennen, da die Fassadenklasse ja alle benötigten Elemente bereitstellt. In Einzelfällen kann es jedoch immer

noch sinnvoll sein, auf die Subsystemklassen direkt zuzugreifen um auf diese Weise hochgradig spezialisierte Operationen vorzunehmen. Abbildung 5.5 zeigt ein Strukturdiagramm des Entwurfsmusters Fassade, wobei hier keine expliziten Klassen und Beziehungen gegeben sind.

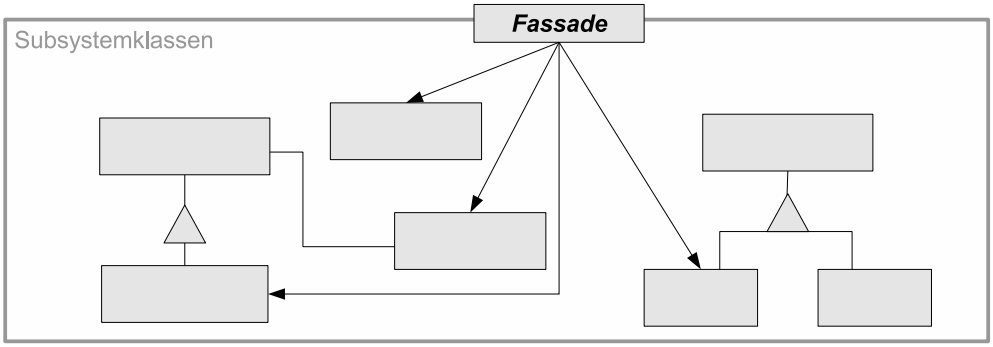


Abbildung 5.5: Strukturdiagramm für das Entwurfsmuster Fassade

Bei der Implementierung der Algorithmen zum Anschlusssicherungsproblem übernimmt die Klasse `tcDelayHandler` die Rolle der Fassadenklasse: In ihr laufen alle Fäden der Subsystemklassen zusammen. Bei Hinzunahme einer geeigneten Benutzerschnittstelle lässt sich allein mit Hilfe dieser Klasse das gesamte Subystem einsetzen.

5.3 Anbindung externer Software

Abschnitt 5.3.1 gibt daher einen kurzen Einblick in das GNU Linear Programming Kit, welches zur Lösung von LP-Relaxationen eingesetzt wird. Abschnitt 5.3.2 stellt einige Elemente des Softwaresystems MATLAB vor, das bei der Verwaltung der Verspätungsdatensätze von unschätzbarem Wert ist. Schließlich wird in Abschnitt 5.3.3 das Softwarepaket SVM^{light} vorgestellt, welches eine effiziente Implementierung der in Abschnitt 3.3 vorgestellten Support Vector Machine darstellt. Für die Verwendung dieser Software aus MATLAB heraus wird zusätzlich das Paket MEX-SVM verwendet, welches ebenfalls in Abschnitt 5.3.3 vorgestellt wird.

5.3.1 GNU Linear Programming Kit (GLPK)

Bei dem GNU Linear Programming Kit handelt es sich um ein in ANSI C geschriebenes Softwarepaket für die Lösung von Problemen aus den Bereichen der hochdimensionalen linearen sowie der gemischt-ganzzahligen Programmierung. Das GLPK ist freie Software, welche unter [16] kostenlos heruntergeladen werden kann.

Die Software kann wahlweise durch Bibliotheksfunktionen mit Hilfe einer Anwendungsprogramm-Schnittstelle (englisch: Application Programming Interface, kurz *API*) in eigene Programme eingebunden werden oder auch mit Hilfe des Programms `glpsol` von der Kommandozeile aus genutzt werden. Dabei können Dateien zur Beschreibung der zu lösenden Modelle in den Formaten MPS[2] oder CPLEX LP oder auch in dem GNU-MathProg-Language-Syntax vorliegen.

Zur Lösung von Problemen aus dem Bereich der linearen Programmierung steht einerseits eine Innere-Punkte-Methode und andererseits ein revidiertes Simplexverfahren zur Verfügung. Durch entsprechende Funktionsaufrufe kann man bei Nutzung der API sicherstellen, dass die gewünschte Lösungsroutine aufgerufen wird. Bei gemischt-ganzzahligen Programmen ist man allerdings ausschließlich auf das revidierte Simplexverfahren angewiesen. Über die Wahl des Solvers hinaus hat man durch eine Vielzahl von Kommandos zur Manipulation eines Modells sowie der Vorgehensweise bei der Lösung des Programms weitreichende Möglichkeiten, ein Problem auf verschiedenen

[2] Ein weit verbreitetes Format für mathematische Programmierung, Abkürzung für *mathematical programming system*.

Wegen anzugehen. Modelle können beispielsweise wie bei Einsatz des Programms `glpsol` aus Dateien eingelesen werden, Eintrag für Eintrag sukzessiv aufgebaut werden oder auch durch Vorgabe einer Nebenbedingungsmatrix definiert werden.

Die vollständige API wird in [17] ausführlich beschrieben, worin auch Informationen zum `glpsol` enthalten sind. Informationen zur Modellierungssprache MathProg kann man aus [18] entnehmen.

5.3.2 MATLAB

Damit eine Support Vector Machine bei der Klassifizierung von Testdatensätzen einigermaßen zuverlässige Ergebnisse liefert, müssen zunächst hinreichend viele Trainingsdatensätze bearbeitet werden, aus welchen die SVM lernen kann, siehe hierzu auch 3.3.4. Hierbei müssen auch nach Abschluss des Lernprozesses sowohl Informationen über die Datenpunkte als auch die korrekten Bezeichnungen aller Trainingsdatensätze für die Klassifikation weiterer Testdaten abrufbar sein.

Man ist daher auf eine strukturierte Datenverwaltung angewiesen. Aus diesem Grund wurde die kommerzielle Software MATLAB, kurz für MATrix LABoratory, hinzugezogen: Mit ihr lassen sich sowohl Verspätungsdaten als auch korrekte Bezeichnungen mit Hilfe von schwach besetzten Vektoren darstellen und komprimiert abspeichern. Zusätzlich hat man auf diese Weise die Möglichkeit, die Daten mit dem breiten Funktionsspektrum von MATLAB zu untersuchen. Für viele Bereiche des wissenschaftlichen Rechnens stellt MATLAB ein sehr nützliches, wenn nicht sogar unverzichtbares Werkzeug dar. Eine Übersicht über den Leistungsumfang sowie der Erweiterungen in Form von so genannten *Toolboxes* findet man in der Onlinedokumentation [32].

Abbildung 5.6 zeigt beispielsweise, welche Anschlüsse für welchen Verspätungsdatensatz verpasst werden. Dabei steht jede Zeile für einen von 2000 Verspätungsdatensätzen und jede Spalte für einen von 3499 Anschlüssen. Es ist sehr gut erkennbar, dass für einige Anschlüsse sehr viel öfter ein Verpassen sinnvoll ist als für andere. Insgesamt wurde bei diesem Auszug 7196 Mal vorgeschlagen, einen Anschluss zu verpassen. Man sieht: Die zusätzliche Anbindung von MATLAB erfordert zwar zusätzlichen Aufwand, dieser rentiert sich allerdings sehr schnell, wenn man die erfassten Daten untersuchen, beispielsweise visualisieren möchte.

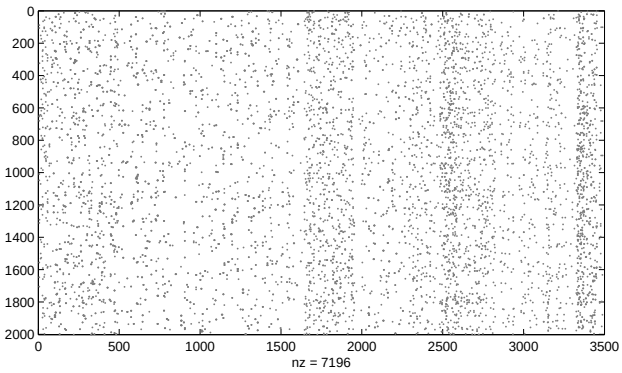


Abbildung 5.6: Darstellung der in den jeweiligen Optimallösungen verpassten Anschlüsse

Zusätzlich ist MATLAB in der Lage, verschiedene Optimierungsprobleme zu lösen. So kann man also die in Abschnitt 3.3 vorgestellte Support Vector Machine mit Hilfe von MATLAB implementieren. Bei Verwendung einer großen Anzahl von Trainingsbeispielen ist dieser Ansatz ohne weitere Hilfsmittel nicht praktikabel, was in Abschnitt 5.3.3 näher betrachtet wird.

Eine Hürde stellt jedoch noch die Anbindung der MATLAB-Routinen an das in C++ geschriebene Hauptprogramm dar. Doch auch hierfür existieren Schnittstellen: Man kann von einen C-Programm aus eine *MATLAB-Engine* erzeugen, welche alle Fähigkeiten eines vollwertigen MATLAB-Clients hat. Allerdings wird diese nun nicht mehr

unmittelbar durch Nutzereingaben, sondern durch das aufrufende C-Programm gesteuert. Durch Bibliotheksfunktionen ist es nun möglich, Variablenwerte zwischen C-Programm und dem Workspace der MATLAB-Engine hin- und herzukopieren oder auch MATLAB-Kommandos aufzurufen. Bei der Übersetzung des C-Programms sind dabei einige zusätzliche Details zu beachten. Wie eine konkrete Problemlösung aussieht, hängt dabei auch stets von dem eingesetzten Compiler, der verwendeten MATLAB Version sowie dem Betriebssystem ab. Die hier dargestellten Lösungen beziehen sich auf die Übersetzung mit Hilfe des kostenlos verfügbaren Compilers `gcc`, MATLAB in der Version 2007a und das kostenlos verfügbare Betriebssystem Linux.

- Für den Zugriff auf den Workspace der MATLAB-Engine werden spezielle Datentypen und Funktionen benötigt. Beispielsweise kann man die Funktion
  `int engEvalString(Engine* ep, char* s)`
  verwenden, um den Strings `s` im dem durch den Zeiger `ep` repräsentierten Workspace als MATLAB-Kommando auszuführen. Diese Funktionen und die Datentypen für den Zugriff auf den Workspace sind in der Header-Datei `engine.h` definiert. Im Übersetzungsprozess muss dem Compiler mitgeteilt werden, wo sich diese Header-Datei befindet. Dies erreicht man bei Verwendung des `gcc` durch eine zusätzliche Kommandozeilenoption, diese kann beispielsweise so aussehen:
  `-I/usr/local/matlabr2007a/extern/include/`
- Zusätzlich müssen dem Linker mitgeteilt werden, wo er die bereits übersetzen Binärdateien in Form von statischen Bibliotheken findet. Bisher kennt der Compiler nämlich lediglich die Prototypen der Funktionen, allerdings noch keine konkreten Definitionen. Hierfür müssen beim Binden der Binärdateien, also dem Erstellen der ausführbaren Datei, einerseits wieder die Pfade angegeben werden, an denen die Bibliotheken zu finden sind. Andererseits müssen die benötigten Bibliotheken[3] selbst angegeben werden. Auch dies geschieht durch `gcc`-Kommandozeilenoptionen:
  `-L/usr/local/matlab2007a/bin/glnx86/ -leng`
- Zusätzlich benötigt das ausführbare Programm Informationen darüber, wo bestimmte gemeinsam genutzte Objektdateien (englisch: *Shared Objects*) zu finden sind, sie haben unter Linux die Endung `.so`. Auch diese Informationen können bei Nutzung des `gcc` mit Hilfe der Kommandozeile übermittelt werden:
  `-Wl,-rpath,/usr/local/matlab2007a/bin/glnx86/`

Für Programme, die in ANSI C erstellt wurden, übernimmt das bei MATLAB mitgelieferte Compilerskript `mex` diese Aufgaben, bei Programmen in C++ müssen diese Details von Hand in den Übersetzungsprozess eingepflegt werden.

Weitere Details zur Schnittstelle zwischen C und MATLAB findet man unter [32], Abschnitt *External Interfaces*.

5.3.3 SVM^{light} und MEX-SVM

Mit steigender Anzahl ℓ von Trainingsbeispielen steigt offensichtlich die Komplexität der zu lösenden Lernaufgabe. Schließlich hat der gesuchte Lösungsvektor $\boldsymbol{\alpha}$ die Länge ℓ und die in Abschnitt 3.3 eingeführten Matrizen $\mathbf{Q}$ und $\mathbf{Q}^{\kappa}$ haben sogar ℓ^2 Einträge. Selbst wenn davon jeweils nur sehr wenige Einträge ungleich Null sind, stellt dies bei herkömmlichen Lösungsverfahren für quadratische Optimierungsprobleme häufig ein unüberwindbares Hindernis in Bezug auf Zeit- und Speicherbedarf, da sie nicht darauf ausgelegt sind, mit schwach besetzten Matrizen zu arbeiten. Vorlage für diesen Abschnitt ist der Artikel [13], welchem auch weitergehende Details zu entnehmen sind.

Ein Ansatzpunkt besteht aus einer Dekompositionsstrategie, mit deren Hilfe das ursprünglich hochdimensionale Optimierungsproblem in eine Reihe von kleineren, einfach zu lösenden Probleme zerlegt werden kann. Hierbei wird in jedem Schritt die Menge der dualen Entscheidungsvariablen $\alpha_1, \ldots, \alpha_\ell$ in einen aktiven und einen inaktiven Teil zerlegt. Die inaktiven Variablen werden fixiert, während die aktiven Variablen frei bleiben. Dies führt zu einem einfacheren, ebenfalls quadratischen Optimierungsproblem, welches zu lösen ist. Die resultierende Lösung bleibt zulässig für das ursprüngliche Problem, da man es als eine Relaxation des einfacheren Problems ansehen

[3] Das Übersetzungsskript `mex` verwendet standardmäßig die Bibliotheken `-leng -lmx -lmex`, von denen sich für die eingesetzten Funktionen allerdings nur die Option `-leng` als notwendig herausgestellt hat.

kann. Durch eine geschickte Unterteilung in aktive und inaktive Variablen, beispielsweise durch Betrachtung geeigneter Hilfsprobleme für die Bestimmung geeigneter Richtungen des steilsten Abstiegs, kann in jedem Schritt eine Verbesserung des Zielfunktionswerts erzielt werden. Dabei kann gleichzeitig durch Betrachtung der Karush-Kuhn-Tucker Bedingungen, welche bei konvexer Optimierung notwendig und hinreichend für ein Optimum sind, in jedem Schritt geprüft werden, ob man bereits einen optimalen Lösungsvektor $\boldsymbol{\alpha}^\star$ gefunden hat.

Die beschriebenen Techniken sind nur ein Teil des Leistungsumfangs, den das für wissenschaftliche Zwecke frei verfügbare Softwarepaket SVM^{light} zu bieten hat. Mit seiner Hilfe ist man in der Lage, auch Probleme bei Verwendung von sehr vielen Trainingsbeispielen sogar dann effizient zu lösen, wenn nur mäßig viel Speicher bereitsteht. Dabei werden gleichzeitig verschiedene Abschätzungen für den Generalisierungsfehler mitgeliefert, welche an dieser Stelle allerdings nicht näher beschrieben werden.

Die eigentliche SVM^{light} Software und Informationen zur Verwendung sowie Beispiele findet man unter [12]. Die Software besteht aus einem Lernmodul (`svm_learn`) und einem Klassifikationsmodul (`svm_classify`), welche beide primär über die Kommandozeile gesteuert werden. Dabei bestehen über eine Vielzahl von Parametern detaillierte Konfigurationsmöglichkeiten: So lässt sich beispielsweise angeben, wie groß die aktive Menge jeweils sein darf oder wie viel Cache für Auswertungen der Kernfunktion zur Verfügung steht. Als Kernfunktionen werden standardmäßig lineare, polynomiale und sigmoidale Kerne sowie der Gauss-Kern angeboten, es besteht aber auch die Möglichkeit, durch Erweiterung einer enthaltenen Schnittstelle eine eigene Kernfunktion zu definieren. Da der Quelltext ebenfalls öffentlich zugänglich ist besteht außerdem die Möglichkeit, die Software als Bibliothek einzubinden und die Lern- und Klassifikationsfunktionen aus eigenen Programmen heraus aufzurufen.

Die Eingabe der Trainings- und Testdaten erfolgt über Textdateien, auch Ergebnisse des Lernprozesses und der Klassifikation werden über Dateien ausgegeben. Diese Vorgehensweise ist für die meisten Anwendungen vorteilhaft und praktisch.

Für die Untersuchungen im Rahmen dieser Arbeit wurde allerdings eine andere Herangehensweise gewählt. Wie bereits in Abschnitt 5.3.2 erwähnt, sind die Trainingsdaten für das in dieser Arbeit untersuchte Problem bereits mit Hilfe von MATLAB erfasst und in komprimierter Form gespeichert. Daher wurde die Software SVM^{light} über eine MATLAB-Schnittstelle namens MEX-SVM verwendet. Dabei entsprechen die über die zugehörigen m-Files zur Verfügung gestellten Funktionen `svm_learn` und `svm_classify` den gleichnamigen Programmen aus SVM^{light}. Allerdings erfolgt der Datenaustausch nun nicht mehr über Dateien, sondern über geeignete Strukturvariablen. Optionen, die dem Programm bisher über die Kommandozeile mitgegeben wurden, können nun als Parameter-String übergeben werden. Dies hat den entscheidenden Vorteil, dass keine weiteren Dateien erzeugt werden müssen. Wenn schließlich Ergebnisse gespeichert werden sollen, so kann hierfür die von MATLAB bereitgestellte komprimierte Datenrepräsentation genutzt werden. Details zum Einsatz von MEX-SVM sowie Informationen zur Installation findet man unter [3].

6 Ergebnisse

Dieses Kapitel ist der Untersuchung des Laufzeitverhaltens der in den vorangehenden Kapiteln beschriebenen Verfahren und der Qualität der jeweils bestimmten Lösungen für das in Abbildung 1.7 dargestellte Programm (DM) gewidmet. Dabei wurde für Fahrplanangaben und Verspätungen sekundengenau gerechnet, auch wenn zur Veranschaulichung gelegentlich eine minutengenaue Darstellung gewählt wurde.

Grundlage ist dabei ein Auszug aus Originalfahrplänen der Deutschen Bahn AG für ein Gebiet in der Harzregion für einen Tag. Daran sind 127 Bahnhöfe und 707 Fahrzeuge beteiligt. Dies führt zu einem Ereignis-Aktivitäts-Netzwerk mit 9757 Ereignissen und 4773 Fahrt- und 4277 Warteaktivitäten. Eine Umsteigemöglichkeit wird einbezogen, wenn mindestens 3 Minuten und höchstens 30 Minuten zwischen Ankunft des ersten und Abfahrt des zweiten Zuges liegen, also $L_a \geq 180$ und $s_a \leq 1620$ für $a \in \mathcal{A}_{\text{change}}$. Außerdem wird ein Umsteigevorgang, der zu einer unmittelbaren Rückfahrt führen würde, nicht in Betracht gezogen. Dies führt zu 3499 Umsteigeaktivitäten. Als Periodendauer für den Fahrplantakt wird eine Stunde gewählt, also $P = 3600$.

Da keine Daten über tatsächliche Fahrgäste und Umsteiger vorliegen, werden Einheitsgewichte verwendet, das heißt $w_e = 1.0$ und $w_a = 1.0$ für alle $e \in \mathcal{E}$ und $a \in \mathcal{A}_{\text{change}}$. Da außerdem keine Pufferzeiten für Fahrt- und Warteaktivitäten vorliegen, gilt zusätzlich $L_a = 0$ für $a \in \mathcal{A}_{\text{wait}}$ sowie $s_a = 0$ für $a \in \mathcal{A}_{\text{drive}}$. Für Umsteigeaktivitäten gilt einheitlich $L_a = 180$. Diese Einschränkungen tragen bedauerlicherweise dazu bei, dass das verwendete Modell weniger realitätsnah ist. Trotzdem ist zu hoffen, dass die Berechnungen auch für Datensätze, die realistischer sind, gute Ergebnisse liefern.

Für dieses Netzwerk wurden drei Mengen von Verspätungsdatensätzen betrachtet:

$\mathcal{D}^{\star}_{\text{train}}$: 10000 Datensätze mit bis zu zehn zufällig bestimmten Quellverspätungen von bis zu 30 Minuten pro Datensatz. Diese Menge ist die Grundlage für den Vergleich der verschiedenen Branch-and-Bound Suchstrategien.

$\mathcal{D}_{\text{train}}$: 200000 Datensätze mit bis zu zehn zufällig bestimmten Quellverspätungen von bis zu 30 Minuten pro Datensatz. $\mathcal{D}_{\text{train}}$ wurde mit den durch die Heuristik-verbesserte-Bestensuche bestimmten Bezeichungen als Trainingsmenge für die Hard Margin Support Vector Machine verwendet, dabei gilt $\mathcal{D}^{\star}_{\text{train}} \subset \mathcal{D}_{\text{train}}$. Diese Menge ist außerdem die Grundlage für den Vergleich der Bestensuche mit verschiedenen Heuristiken.

$\mathcal{D}_{\text{test}}$: 10000 Datensätze mit bis zu 15 zufällig bestimmten Quellverspätungen von bis zu 35 Minuten pro Datensatz. Datensätze, welche zu reduzierten Netzwerken mit weniger als 20 relevanten Anschlüssen führten, wurden dabei herausgefiltert. Diese Menge ist die Grundlage für den Vergleich der Bestensuche mit verschiedenen Heuristiken, insbesondere jener, die aus der Nutzung der Hard Margin SVM hervorgehen.

6.1 Vergleich der Branch-and-Bound Suchstrategien

In Abschnitt 2.1.3 wurden verschiedene Suchstrategien für das Branch-and-Bound Verfahren vorgestellt. Um deren Leistungsfähigkeit zu vergleichen wurde das Anschlusssicherungsproblem mit den verschiedenen Suchstrategien für die in $\mathcal{D}^{\star}_{\text{train}}$ enthaltenen Verspätungsdatensätze gelöst.

Die Ergebnisse werden in den Tabellen 6.1 und 6.2 zusammengefasst. Dabei wurde die Suche nach der Optimallösung nach maximal 60 beziehungsweise 300 Sekunden abgebrochen und der bisher beste gefundene Wert verwendet. Jeder Tabelleneintrag stellt dabei die durchschnittlichen Ergebnisse mehrerer Datensätze dar.

Die Tabellen sind jeweils folgendermaßen zu verstehen: Die erste Spalte (#) gibt an, welche Datensätze zu einer Kategorie zusammengefasst wurden, dabei steht die Angabe 21-40 beispielsweise für jene Datensätze, in denen 21-40 Anschlüsse relevant sind. Die zweite Spalte (N) gibt an, wie viele Datensätze zu dieser Kategorie gehören,

		Laufzeit in Sekunden					durchschn. Zielfunktionswert nach max. $60s$				
#	N	DFS	BFS	A*	A*+H	MIP	DFS	BFS	A*	A*+H	MIP
0	1213	0.0	0.0	0.0	0.0	0.0	101.9	101.9	101.9	101.9	101.9
1-10	4351	0.0	0.0	0.0	0.0	0.0	351.2	351.2	351.2	351.2	351.2
11-20	1858	0.1	0.1	0.0	0.5	0.2	688.2	688.2	688.2	688.2	688.2
21-40	1607	4.7	3.1	1.1	5.8	1.0	1033.3	1038.9	1030.0	1028.1	1028.1
41-60	629	36.3	20.3	7.9	19.3	5.2	1700.6	1703.1	1463.3	1417.2	1417.2
61-80	228	57.4	37.9	18.8	32.9	13.2	2956.8	2731.1	1985.6	1694.9	1694.9
>80	114	59.8	51.6	30.1	42.8	26.1	4685.2	4904.6	3162.9	1928.4	1928.4

Tabelle 6.1: Durchschnittswerte Laufzeit und Zielfunktionswert nach max. 60 Sekunden, Datensatzmenge $\mathcal{D}^\star_{\text{train}}$

		Laufzeit in Sekunden					durchschn. Zielfunktionswert nach max. $300s$				
#	N	DFS	BFS	A*	A*+H	MIP	DFS	BFS	A*	A*+H	MIP
0	1213	0.0	0.0	0.0	0.0	0.0	101.9	101.9	101.9	101.9	101.9
1-10	4351	0.0	0.0	0.0	0.0	0.0	351.2	351.2	351.2	351.2	351.2
11-20	1858	0.2	0.2	0.1	0.5	0.2	688.2	688.2	688.2	688.2	688.2
21-40	1607	9.0	7.1	2.1	8.0	1.2	1029.5	1032.3	1028.7	1028.1	1028.1
41-60	629	103.9	57.7	18.4	43.6	10.8	1535.1	1537.3	1431.6	1417.2	1417.2
61-80	228	244.8	139.5	48.1	95.0	36.4	2520.1	2312.3	1835.6	1697.7	1694.9
>80	114	289.4	217.2	94.9	146.0	88.8	4282.5	4160.6	2387.6	1930.3	1928.4

Tabelle 6.2: Durchschnittswerte Laufzeit und Zielfunktionswert nach max. 300 Sekunden, Datensatzmenge $\mathcal{D}^\star_{\text{train}}$

darauf folgen die eigentlichen Ergebnisse: Es wurde je einmal die *Tiefensuche* (DFS), *Breitensuche* (BFS) und *Bestensuche* ohne Einsatz von Heuristiken (A*) sowie die *Bestensuche* mit Einsatz der Heuristiken *Hold All*, *Integer Rounding* und *Miss First Delayed Change if Reasonable* in jedem Iterationsschritt (A*+H) als Suchstrategie eingesetzt. Außerdem wurde zum Vergleich das jeweilige Problem mit Hilfe des *MIP-Solvers des Softwarepakets GLPK* gelöst (MIP).

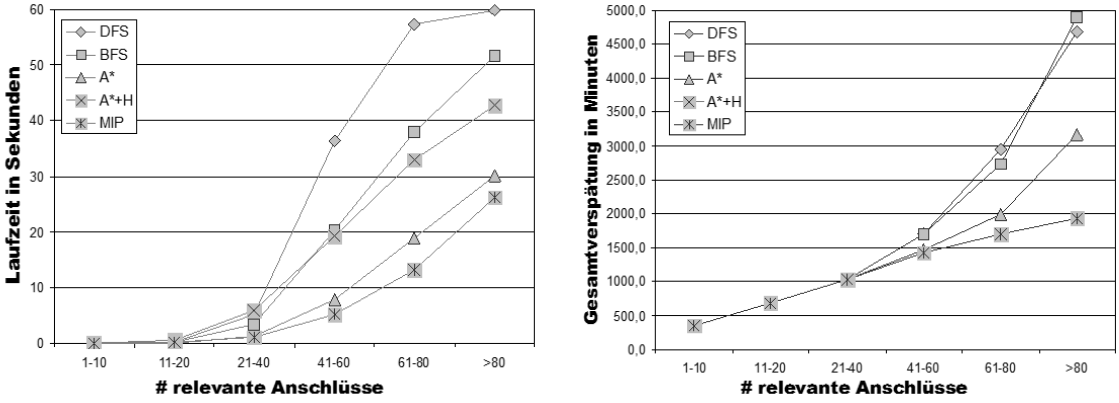
Aus den Tabellen lässt sich ablesen, dass im Mittel der MIP-Solver effizienter war als die untersuchten Suchstrategien, wobei der Unterschied zur Bestensuche sehr gering ausfällt. Um so schlechter schneiden allerdings Tiefen- und Breitensuche ab, sowohl in Bezug auf Laufzeit als auch auf Zielfunktionswert. Zusätzlich fällt auf, dass der MIP-Solver sowohl bei 60 als auch bei 300 Sekunden Laufzeitschranke die gleichen Ergebnisse liefert. Allerdings war nicht bei allen Datensätzen nach 60 beziehungsweise 300 Sekunden klar, dass die gefundene Lösung optimal ist. In Einzelfällen, insgesamt 22 mal, konnte die einfache Bestensuche innerhalb von 300 Sekunden die Optimalität einer gefundenen Lösung zu garantieren, während der MIP-Solver dazu innerhalb der gesetzten Zeitschranke nicht in der Lage war. Wie häufig dies für die jeweiligen Klassen von Verspätungsdatensätzen aufgetreten ist, kann Tabelle 6.3 entnommen werden.

Außerdem wird deutlich, dass die Verbesserung der Bestensuche durch Heuristiken in jedem Iterationsschritt zwar seinen Preis in Form der höheren Laufzeit hat, welcher aber angesichts der besseren Zielfunktionswerte gerechtfertigt scheint: In den Tabellen 6.1 und 6.2 ist kaum noch ein Unterschied zwischen der Lösung des professionellen MIP-Solvers und der durch Heuristiken verbesserten Variante der Bestensuche erkennbar.

Grafische Veranschaulichungen der Tabellen finden sich in den Abbildungen 6.1 und 6.2. Hier wird die Qualität der durch die Bestensuche mit Heuristik-Unterstützung vorgeschlagenen Lösungen noch einmal besonders deutlich, da

# rel. Anschlüsse	1-10	11-20	21-40	41-60	61-80	>80
abs. Häufigkeit	0	0	0	4	7	11
rel. Häufigkeit	0	0	0	0.6%	3.1%	9.6%

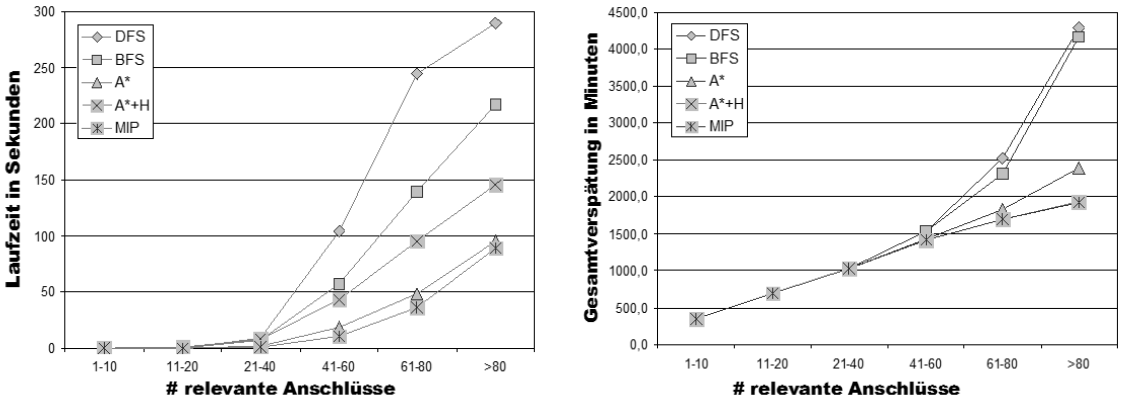
Tabelle 6.3: Häufigkeit, mit der A*+H für $\mathcal{D}^\star_{\text{train}}$ besser ist als MIP



(a) Laufzeit in Sekunden

(b) Zielfunktionswert (Gesamtverspätung in Minuten)

Abbildung 6.1: Visualisierung der Tabelle 6.1: Laufzeit und Zielfunktionswert nach max. 60 Sekunden



(a) Laufzeit in Sekunden

(b) Zielfunktionswert (Gesamtverspätung in Minuten)

Abbildung 6.2: Visualisierung der Tabelle 6.2: Laufzeit und Zielfunktionswert nach max. 300 Sekunden

kein Unterschied zwischen ihnen und den Lösungen des MIP-Solvers erkennbar ist.

6.2 Vergleich verschiedener Heuristiken für Trainingsdaten

In diesem Abschnitt werden die Ergebnisse ausgewählter Heuristiken für die Datensatzmenge $\mathcal{D}_{\text{train}}$ mit denen der Heuristik-verbesserten Bestensuche verglichen. Die Beschränkung auf die Ergebnisse der in Abschnitt 2.3.1 vorgestellten Heuristik *Miss First Delayed Change if Reasonable* (MFDCR) und das in Abschnitt 2.3.2 vorgestellte *Genetische Verfahren* (GA) ist sinnvoll, da die Ergebnisse der Heuristik *Hold All* minderwertig sind und die Heuristik *Miss First Delayed Change if Reasonable* auf *Miss First Delayed Change* (MFDC) aufbaut, welche ebenfalls schlechtere Ergebnisse liefert.

Tabelle 6.4 zeigt also die durchschnittlichen Ergebnisse für Laufzeit und Zielfunktionswert der einzelnen Verfahren, wobei die Menge $\mathcal{D}_{\text{train}}$ diesmal in etwas feinere Klassen eingeteilt wurde.

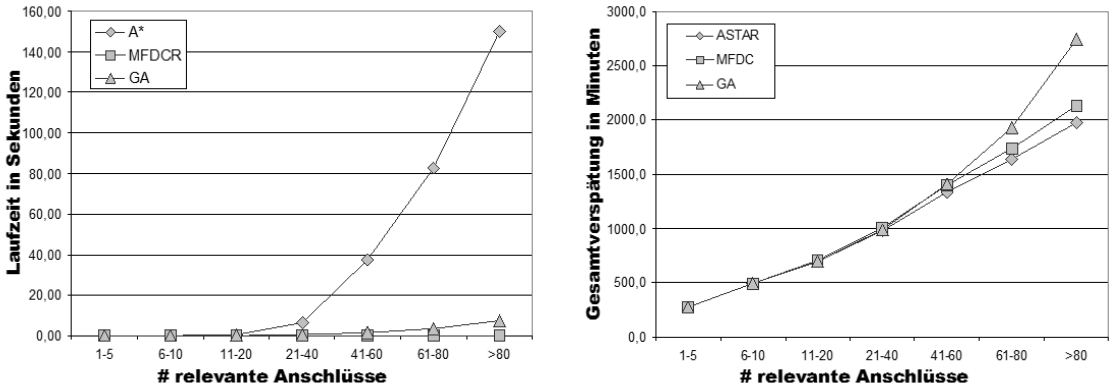
Es ist zu beachten, dass hierbei die Heuristik-verbesserte Bestensuche (A*+H), ein eigentlich exaktes Verfahren, nach maximal 300 Sekunden die Suche nach der Optimallösung aufgegeben hat, wenn nach jener Zeitspanne noch

		Laufzeit in Sekunden			durchschn. Z-Wert nach max. 300s		
#	N	A*+H	MFDCR	GA	A*+H	MFDCR	GA
0	21834	0.0	0.0	0.0	102.0	102.0	102.0
1-5	50241	0.0	0.0	0.1	278.2	278.4	278.2
6-10	35900	0.1	0.0	0.1	494.5	496.5	494.5
11-20	44918	0.6	0.0	0.2	694.7	703.1	694.8
21-40	34408	6.5	0.0	0.6	981.7	1008.2	986.5
41-60	9002	37.4	0.0	1.6	1333.5	1395.1	1411.1
61-80	2613	82.8	0.0	3.6	1635.0	1736.2	1933.4
>80	1084	150.1	0.0	7.2	1974.1	2127.3	2740.0

Tabelle 6.4: Durchschnittswerte Laufzeit und Zielfunktionswert nach max. 300 Sekunden für Bestensuche und zwei Heuristiken, Datensatzmenge $\mathcal{D}_{\mathrm{train}}$

keine Optimalität garantiert werden konnte.

Das Genetische Verfahren wurde mit Populationsgröße 100, Kreuzungswahrscheinlichkeit 0.7, Mutationswahrscheinlichkeit 0.001 über jeweils 100 Generationen mit One-Point-Crossover durchgeführt. Außerdem wurde sichergestellt, dass das jeweils beste Individuum einer Generation stets in die nächste Generation übernommen wird. Für größere Populationen und höhere Generationsanzahl sind bessere Ergebnisse sehr wahrscheinlich, allerdings würde dies auch die Laufzeit erhöhen. Auch sind Verbesserungen durch andere Werte für Kreuzungs- und Mutationswahrscheinlichkeit sowie andere Kreuzungstechniken möglich. Diese Verbesserungsansätze wurden allerdings im Rahmen dieser Arbeit nicht näher untersucht.



(a) Laufzeit in Sekunden

(b) Zielfunktionswert (Gesamtverspätung in Minuten)

Abbildung 6.3: Visualisierung der Tabelle 6.4: Laufzeit und Zielfunktionswert nach max. 300 Sekunden

Die Ergebnisse für Laufzeit und Zielfunktionswert werden in Abbildung 6.3 graphisch dargestellt.

Daraus wird ersichtlich, dass die Heuristik MFDCR durchaus mit der Bestensuche konkurrieren kann, und das bei extrem kurzer Laufzeit. Das Genetische Verfahren schneidet nicht ganz so gut ab, sowohl Laufzeit als auch Zielfunktionswerte sind schlechter als bei MFDCR. Allerdings können sich die Ergebnisse angesichts der Tatsache, dass dieses Verfahren nicht speziell für das Anschlusssicherungsproblem ausgelegt ist, immer noch sehen lassen: In akzeptabler Zeit werden auch für moderat komplexe Verspätungsszenarien noch Ergebnisse bestimmt, welche nicht allzu weit von der Optimallösung entfernt sind. Außerdem ist nicht zu vergessen, dass für das Genetische Verfahren bei höheren Werten für Populations- und Generationszahl bessere Ergebnisse zu erwarten sind. Auch ist denkbar, die Lösung der Heuristik MFDCR in die Anfangspopulation eines Genetischen Verfahrens aufzunehmen, woraus man auf noch bessere Ergebnisse hoffen kann.

6.3 Untersuchung der aus Lernverfahren resultierenden Heuristiken

Wie bereits im Kapitelvorspann angesprochen wurde die Menge $\mathcal{D}_{\text{train}}$ zusammen mit den zugehörigen Lösungen des Heuristik-verbesserten Branch-and-Bound Verfahrens mit Bestensuche als Trainingsmenge einer Hard Margin SVM verwendet.

Insgesamt wurden für jeden Anschluss drei Klassifikatoren erzeugt, indem zwei verschiedene Kerne eingesetzt wurden und eine geeignete Auswahl von Trainingsbeispielen gemacht wurde:

K1: Quellverspätungskern, alle Beispiele als Trainingsmenge,

K2: Quellverspätungskern, Trainingsmenge enthält nur jene Beispiele, in denen der jeweils betrachtete Anschluss relevant ist,

K3: Verspätungsausbreitungskern, Trainingsmenge enthält nur jene Beispiele, in denen der jeweils betrachtete Anschluss relevant ist.

Der Verspätungsausbreitungskern, der alle Beispiele in den Lernprozess einbezieht, wurde nicht weiter untersucht, da sich hier der Lernaufwand als zu hoch erwiesen hat: Pro Anschluss wurde in den meisten Fällen mehr als eine Stunde Rechenzeit benötigt, was bei insgesamt 3499 Anschlüssen einen nicht zu rechtfertigenden Aufwand darstellt.

		durchschnittlicher Zielfunktionswert nach max. 300s								
#	N	A*+H	MFDCR	GA	K1	K2	K3	K1+H	K2+H	K3+H
20-29	2188	1138.7	1161.2	1140.6	1285.9	1713.6	1312.8	1161.1	1158.2	1298.6
30-39	1941	1417.8	1455.5	1432.0	1627.4	2293.8	1627.6	1455.3	1448.5	1609.1
40-49	1548	1609.0	1663.5	1666.0	1905.5	2857.2	1835.2	1662.6	1655.3	1815.4
50-59	1293	1831.5	1910.0	1970.6	2207.4	3399.6	2089.6	1907.8	1892.1	2055.8
60-69	906	2024.6	2103.8	2283.1	2492.0	3972.6	2310.6	2102.9	2090.6	2271.5
70-79	734	2196.4	2304.9	2596.9	2749.1	4509.9	2506.4	2304.1	2281.7	2459.3
80-89	537	2317.2	2450.7	2889.8	2934.9	5043.0	2623.5	2448.1	2418.4	2577.5
90-99	339	2474.9	2618.3	3205.0	3167.3	5591.1	2804.2	2616.0	2581.2	2740.2
100-119	329	2628.4	2808.2	3637.3	3490.8	6339.8	2982.4	2803.0	2749.8	2901.4
>119	185	3035.6	3271.0	4592.1	4136.6	7814.5	3346.1	3262.9	3177.0	3292.3

Tabelle 6.5: Durchschnittswerte Zielfunktionswert nach max. 300 Sekunden für Datensatzmenge $\mathcal{D}_{\text{test}}$

Tabelle 6.5 zeigt die Zielfunktionswerte, welche verschiedene Lösungsprozeduren für die Datensätze der Menge $\mathcal{D}_{\text{test}}$ errechnet haben. Neu eingeführt sind die Kürzel für die *kernbasierten multidimensionalen Klassifikatoren* K1, K2 und K3, außerdem deren in Abschnitt 4.3 vorgestellte Verbesserung durch die Heuristik *Miss First Delayed Change if Reasonable*, gekennzeichnet durch K1+H, K2+H und K3+H.

Aus Tabelle 6.5 wird ersichtlich, dass die kernbasierten Ansätzen allein betrachtet nicht unbedingt sinnvoll sind, ausgenommen vielleicht K3. Sieht man sich die in den jeweiligen Szenarien – hier nicht aufgeführten – vorgeschlagenen Warten/Nichtwarten Entscheidungen an, wird auch schnell deutlich, dass die minderwertigen Zielfunktionswerte häufig darauf basieren, dass Anschlüsse verpasst werden, über die ohnehin keine Verspätung transferiert würde. Derartige Fehlentscheidungen lassen sich leicht durch Heuristiken vom Typ MFDC oder MFDCR beheben, und tatsächlich liefern die kernbasierten Klassifikatoren in Kombination mit MFDCR hervorragende Ergebnisse, die durchaus mit denen konkurrieren können, die von dem Branch-and-Bound Verfahren innerhalb der Laufzeit-

# rel. Anschlüsse	20-29	30-39	40-49	50-59	60-69	70-79	80-89	90-99	100-119	> 119
abs. Häufigkeit	0	0	0	1	3	6	6	7	11	17
rel. Häufigkeit	0	0	0	0.08%	0.33%	0.82%	1.12%	2.06%	3.34%	9.19%

Tabelle 6.6: Häufigkeit, mit der K2+H für $\mathcal{D}_{\text{test}}$ besser ist als A+H*

schranke von 300 Sekunden bestimmt wurden. In insgesamt 51 Fällen, dargestellt in Tabelle 6.6, lieferte der Ansatz K2+H, also der Quellverspätungskern mit relevanten Trainingsbeispielen mit Nachbesserung durch MFDCR, sogar bessere Lösungen als die Bestensuche innerhalb der Laufzeitschranke.

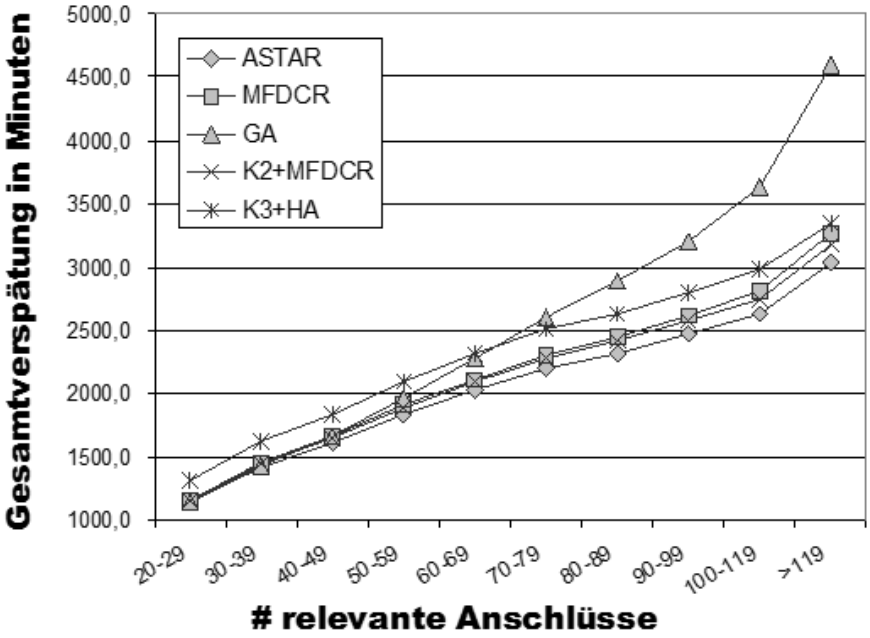


Abbildung 6.4: Visualisierung ausgewählter Spalten von Tabelle 6.5: Zielfunktionswert nach max. 300 Sekunden

Einige ausgewählte Spalten von Tabelle 6.5 werden in Abbildung 6.4 graphisch dargestellt, und zwar einmal die Bestensuche als Referenzlösung, die Heuristik MFDCR, das Genetische Verfahren sowie die jeweils beste Variante des kernbasierten multidimensionalen Klassifikators mit und ohne Nachbesserung durch die Heuristik MFDCR. Es wird deutlich, welch hohe Qualität die Kombination als kernbasierten Lernverfahren und der Heuristik MFDCR aufweist: Kombiniert sind die Ansätze leistungsfähiger als einzeln betrachtet, außerdem sind sie in der Lage, alle weiteren untersuchten Heuristiken auszustechen.

6.4 Schlussbetrachtung

In dieser Arbeit wurden mehrere bekannte, teilweise sehr unterschiedliche Lösungsansätze vorgestellt und für die Anwendung auf ein realitätsnahes Problem angepasst. So wurden verschiedene Werkzeuge für die Lösung des Anschlusssicherungsproblem bereitgestellt. Die Untersuchung dieser Ideen war eine interessante Herausforderung, die vor allem dazu einladen soll, herkömmliche Lösungsverfahren für eigene Problemstellungen anzupassen und verschiedenartige Herangehensweisen auszuprobieren und vor allem zu kombinieren, auch wenn dies nicht immer unmittelbar zu Erfolgen führt.

Als wichtigste Ergebnisse dieser Arbeit sollen an dieser Stelle folgende Punkte noch einmal hervorgehoben werden:

- Durch die Kombination von Bestensuche, Reduktionsoperationen und Branch-and-Bound Verfahren wurde ein Algorithmus vorgestellt, welcher in einigen Fällen mit dem MIP-Solver des Softwarepakets GLPK konkurrieren kann.
- Genetische Verfahren liefern auch für das Anschlusssicherungsproblem schnell akzeptable Lösungen. Allerdings existieren spezialisierte Heuristiken, die mit weniger Aufwand bessere Ergebnisse erzielen.

- Mit *Miss First Delayed Change if Reasonable* wurde eine Heuristik vorgestellt, die in kürzester Zeit und mit geringem Aufwand sehr gute Lösungen für das Anschlusssicherungsproblem liefert.
- Durch den Einsatz von kernbasierten Lernverfahren ist es gelungen, in Kombination mit dieser Heuristik noch bessere Ergebnisse zu erzielen, die in Einzelfällen sogar besser sind als die von der Bestensuche bestimmten Lösungen.

Diese Ergebnisse sind es, die Mut machen, verschiedene Ansätze, die zunächst einmal scheinbar nicht mit einander verwandt sind, zusammenzuführen und dadurch immer mächtigere Werkzeuge für die Lösung verschiedenartiger Probleme zu erhalten.

Interessant ist dabei vor allem die Frage, ob und inwiefern kernbasierte Lernfervahren für andere ganzzahlige Optimierungsprobleme eingesetzt werden können.

A Verwendete Notation

Bevor konkrete Angaben zur Notation gemacht werden, sollen einige Anmerkungen vorweg genommen werden. Vektoren und Matrizen sind grundsätzlich **fett** dargestellt, dies gilt auch für griechische Buchstaben. Zusätzlich wurden für Vektoren Kleinbuchstaben und für Matrizen Großbuchstaben verwendet, wobei der Fahrplanvektor $\mathbf{\Pi}$ eine Ausnahme darstellt. Einzelne Einträge sind durch Kleinbuchstaben mit Subskript-Index gekennzeichnet.

Es ist zusätzlich zu beachten, dass einige Bezeichnungen vereinzelt mit unterschiedlicher Bedeutung auftreten. Allerdings wurde darauf geachtet, dass die korrekte Bedeutung aus dem jeweiligen Kontext ersichtlich wird

A.1 Anschlusssicherung

V Menge der Bahnhöfe

E Menge der Direktfahrkanten zwischen Bahnhöfen

F Menge der Fahrzeuge

V^g Menge der Bahnhöfen, die von Fahrzeug $g \in F$ angefahren werden

E^g Menge der Direktfahrkanten, die von Fahrzeug $g \in F$ verwendet werden

$\Pi_{\mathrm{arr}_g^v}$ Fahrplanangabe: Geplante Ankunftszeit des Fahrzeugs $g \in F$ am Bahnhof $v \in V$

$\Pi_{\mathrm{dep}_g^v}$ Fahrplanangabe: Geplante Abfahrtszeit des Fahrzeugs $g \in F$ vom Bahnhof $v \in V$

$\mathbf{\Pi}$ Fahrplan als Vektor

L_g^v Mindeswartezeit des Fahrzeugs $g \in F$ in Bahnhof $v \in V$

L_g^{uv} Mindestfahrtzeit des Fahrzeugs $g \in F$ von u nach v für Bahnhöfe $u, v \in V$

L_{gh}^v Mindestdauer, die für den Umsteigevorgang von Fahrzeug $g \in F$ nach $h \in F$ in Bahnhof $v \in V$ eingeplant ist

$\mathbf{L}$ Mindestdauerangaben als Vektor

s_g^v Pufferzeit für den Wartevorgang des Fahrzeugs $g \in F$ in Bahnhof $v \in V$

s_g^{uv} Pufferzeit für die Fahrt eines Fahrzeugs $g \in F$ von u nach v für Bahnhöfe $u, v \in V$

s_{gh}^v Pufferzeit für den Umsteigevorgang von Fahrzeug $g \in F$ nach $h \in F$ in Bahnhof $v \in V$

$\mathbf{s}$ Pufferzeitangaben als Vektor

d_g^v Quellverspätung der Ankunft des Fahrzeugs $g \in F$ im Bahnhof $v \in V$

$\mathbf{d}$ Quellverspätungen als Vektor

$\mathcal{E}_{\mathrm{del}}$ Menge der quellverspäteten Ankünfte

$\mathbb{N}$ Menge der natürlichen Zahlen: $\{1, 2, 3, \ldots\}$

$\mathbb{N}_0$ $\{0, 1, 2, 3, \ldots\}$

$x_{\mathrm{arr}_g^v}$ tatsächlicher Zeitpunkt der Ankunft des Fahrzeugs $g \in F$ im Bahnhof $v \in V$

$x_{\mathrm{dep}_g^v}$ tatsächlicher Zeitpunkt der Abfahrt des Fahrzeugs $g \in F$ von Bahnhof $v \in V$

$\mathbf{x}$ modifizierter Fahrplan als Vektor

z_{ghv} Entscheidungsvariable, ob der Anschluss von Fahrzeug $g \in F$ nach $h \in F$ in Bahnhof $v \in V$ gehalten werden soll

$\mathbf{z}$ Entscheidungsvariablen als Vektor

$y_{\mathrm{arr}_g^v}$ Zeiteinheiten, um die sich die Ankunft des Fahrzeugs $g \in F$ im Bahnhof $v \in V$ verzögert

$y_{\mathrm{dep}_g^v}$ Zeiteinheiten, um die sich die Abfahrt des Fahrzeugs $g \in F$ von Bahnhof $v \in V$ verzögert

$\mathbf{y}$ Verspätungen als Vektor

M (hinreichend große) Konstante zur Kompensation von Verspätungen

$\mathcal{E}_{\mathrm{arr}}$ Menge der Ankunftsereignisse

$\mathcal{E}_{\mathrm{dep}}$ Menge der Abfahrtsereignisse

$\mathcal{E}$ Menge aller Ereignisse eines Ereignis-Aktivitäts-Netzwerks

Π_e geplanter Zeitpunkt für Eintreten des Ereignisses $e \in \mathcal{E}$

$\mathcal{A}_{\mathrm{drive}}$ Menge der Fahrtaktivitäten

$\mathcal{A}_{\mathrm{wait}}$ Menge der Warteaktivitäten

$\mathcal{A}_{\mathrm{change}}$ Menge der Umsteigeaktivitäten

$\mathcal{A}$ Menge aller Aktivitäten eines Ereignis-Aktivitäts-

Netzwerks

L_a Mindestdauer der Aktivität $a \in \mathcal{A}$

s_a Puffezeit für die Aktivität $a \in \mathcal{A}$

$\mathcal{N}$ Ereignis-Aktivitäts-Netzwerk

$Feas_{\mathrm{DM}}$ Menge der für das Anschlusssicherungsproblem zulässigen Lösungen

$\mathcal{A}^{\mathrm{fix}}$ Menge von festgelegten Anschlüssen

$\mathcal{A}^{\mathrm{miss}}$ Menge von verpassten Anschlüssen

$\mathcal{A}^{\mathrm{open}}$ Menge von offenen Anschlüssen

$\mathcal{A}(\mathcal{A}^{\mathrm{fix}})$ $\mathcal{A}_{\mathrm{wait}} \cup \mathcal{A}_{\mathrm{drive}} \cup \mathcal{A}^{\mathrm{fix}}$

T Periodendauer im periodischen Fahrplan

A.2 Ganzzahlige Programmierung

(P) Optimierungsproblem

(P_{R}) Relaxation von (P)

f Bewertungsfunktion

S Menge der zulässigen Lösungen

S_{R} Menge der zulässigen Lösungen nach Abschwächung von Restriktionen

$\mathbf{x}$ (beliebige) Lösung für Problem (P)

$\mathbf{x}^{\mathrm{R}}$ Optimallösung für Problem (P_{R})

$\mathbf{x}^\star$ Optimallösung für Problem (P)

$\hat{\mathbf{x}}$ von einer Heuristik vorgeschlagene Lösung für Problem (P)

$\mathcal{L}$ Menge von gemischt-ganzzahligen Programmen

f_{GA} Fitnessfunktion

$\mathcal{C}$ aktuelle Population für Genetischen Algorithmus

$\mathcal{C}^{\mathrm{cand}}$ Kandidatenmenge für die Population der nächsten Generation eines Genetischen Algorithmus

A.3 Maschinelle Lernverfahren

f Musterfunktion

X Daten- beziehungsweise Eingaberaum

Y Menge der möglichen Bezeichnungen

$\mathbf{x}$ Datenpunkt aus X

y korrekte Bezeichung eines Datenpunktes

$\mathbf{w}$ Richtungsvektor im Datenraum (ab Abschnitt 3.2: im Merkmalsraum)

b Translationsparameter für Hyperebene

L Verlustfunktion

ℓ Anzahl von Trainingsbeispielen

$\mathbf{x}^{(i)}$ i-ter Datenpunkt

$y^{(i)}$ korrekte Bezeichung des i-ten Datenpunktes

$R(f)$ erwarteter Fehler für Funktion f

$R_{\mathrm{emp}}(f)$ empirisches Risiko für Funktion f

$\mathcal{F}$ Klasse von Funktionen

h VC-Dimension einer Klasse $\mathcal{F}$

F Merkmalsraum

ϕ Merkmalsabbildung

κ Kernfunktion

$\mathbf{K}$ Kernmatrix

$\boldsymbol{\alpha}$ Vektor von Lagrange-Multiplikatoren

$\mathfrak{L}$ Lagrange-Funktion

$\mathbf{Q}$ Kernmatrix, in welche die korrekten Bezeichnungen einbezogen wurden

$\boldsymbol{\alpha}^\star$ optimale Lagrange-Multiplikatoren

$b^\star$ optimaler Translationsparameter für Hyperebene

sv Menge von Stützvektoren

Literaturverzeichnis

[1] ACHTERBERG, TOBIAS, THORSTEN KOCH und ALEXANDER MARTIN: *Branching Rules Revisited*. Oper. Res. Lett., 33(1):42–54, 2005.

[2] BOYD, STEVEN und LIEVEN VANDENBERGHE: *Convex Optimization*. Cambridge University Press, Cambridge, 2004.

[3] BRIGGS, TOM: *MATLAB/MEX Interface to SVM^{light}*.
Verfügbar unter `http://webspace.ship.edu/thbrig/mexsvm/`.

[4] BUCHANAN, BRUCE G.: *Some Approaches to Knowledge Acquisition*. In: MITCHELL, TOM M., JAIME G. CARBONELL und RYSZARD S. MICHALSKI (Herausgeber): *Machine Learning. A Guide to current Research*. Kluwer Academic Publishers, 1986.

[5] DANTZIG, GEORGE B. und MUKUND N. THAPA: *Linear programming 1: Introduction*. Springer, New York [u.a.], 1997.

[6] DE JONG, KENNETH A.: *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Doktorarbeit, University of Michigan, Ann Arbor, 1975.

[7] GAMMA, ERICH, RICHARD HELM, RALPH JOHNSON und JOHN VLISSIDES: *Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, München, 2004.

[8] GÄRTNER, THOMAS, JOHN W. LLOYD und PETER A. FLACH: *Kernels and Distances for Structured Data*. Machine Learning, 57(3), 2004.

[9] GATTO, MICHAEL, RIKO JACOB, LEON PEETERS und ANITA SCHÖBEL: *The Computational Complexity of Delay Management*. Technischer Bericht 456, ETH Zurich, 2005.

[10] GOLDBERG, DAVID E.: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Longman Publishing, Boston, 1989.

[11] HAUSSLER, DAVID: *Convolution Kernels on Discrete Stuctures*. Technischer Bericht 10, Department of Computer Science, University of California at Santa Cruz, 1999.

[12] JOACHIMS, THORSTEN: *SVM^{light} Support Vector Machine*.
Verfügbar unter `http://svmlight.joachims.org/`.

[13] JOACHIMS, THORSTEN: *Making large-Scale SVM Learning Practical*. In: SCHÖLKOPF, B., C. BURGES und A. SMOLA (Herausgeber): *Advances in Kernel Methods – Support Vector Learning*. MIT-Press, 1999.

[14] LINDEROTH, JEFFREY T. und TED RALPHS: *Noncommercial Software for Mixed-Integer Linear Programming*. In: KARLOF, J.K. (Herausgeber): *Integer Programming: Theory and Practice*, Band 3 der Reihe *Operations Research Series*, Kapitel 10. CRC Press, 2006.

[15] LINDEROTH, JEFFREY T. und MARTIN W.P. SAVELSBERGH: *A Computational Study of Search Strategies for Mixed Integer Programming*. INFORMS J. on Computing, 11(2):173–187, 1999.

[16] MAKHORIN, ANDREW: *GLPK (GNU Linear Programming Kit)*.
Verfügbar unter `http://www.gnu.org/software/glpk/`.

[17] MAKHORIN, ANDREW: *GNU Linear Programming Kit Reference Manual*. Moscow Aviation Institute, Moskau, 2007.

[18] MAKHORIN, ANDREW: *Modeling Language GNU MathProg*. Moscow Aviation Institute, Moskau, 2007. Für GLPK Version 4.16.

[19] MERCER, JAMES: *Functions of Positive and Negative Type, and their Connection with the Theory of Integral Equations*. Philosophical Transactions of the Royal Society of London, Series A 209:415–446, 1909.

[20] MITCHELL, MELANIE: *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, 1998.

[21] MÜLLER, KLAUS-ROBERT, SEBASTIAN MIKA, GUNNAR RÄTSCH, KOJI TSUDA und BERNHARD SCHÖLKOPF: *An Introduction to Kernel-Based Learning Algorithms*. IEEE Transactions on Neural Networks, 12(2):181–201, 2001.

[22] NEMHAUSER, GEORGE L. und LAURENCE A. WOLSEY: *Integer and Combinatorial Optimization*. Wiley Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, New York [u.a.], 1988.

[23] RUSSELL, STUART J. und PETER NORVIG: *Künstliche Intelligenz: Ein moderner Ansatz*. Pearson Education Deutschland, München, 2. Auflage, 2004.

[24] SCHABACK, ROBERT und HOLGER WENDLAND: *Kernel techniques: From machine learning to meshless methods*. Acta Numerica, 15:543–639, 2006.

[25] SCHÖBEL, ANITA: *Persönliche Kommunikation*, 2004-2008.

[26] SCHÖBEL, ANITA: *Optimization in Public Transportation*. Optimization and Its Applications. Springer, New York [u.a.], 2006.

[27] SCHÖBEL, ANITA und ANDREAS GINKEL: *The Bicriteria Delay Management Problem*. Transportation Science, 4(4):527–538, 2007.

[28] SCHÖLKOPF, BERNHARD: *Support Vector Learning*. Doktorarbeit, Technische Universität Berlin, 1997.

[29] SCHÖLKOPF, BERNHARD und ALEXANDER J. SMOLA: *Learning with Kernels*. MIT Press, Cambridge, 2001.

[30] SHAWE-TAYLOR, JOHN und NELLO CRISTIANINI: *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[31] SIERKSMA, GERARD: *Linear and Integer Programming*. Pure and Applied Mathematics. Marcel Dekker, New York, Basel, 2. Auflage, 2002.

[32] THE MATHWORKS: *Documentation for MathWorks Products*. Verfügbar unter `http://www.mathworks.com/access/helpdesk/help/helpdesk.html`.

[33] VAN DER GAAG, LINDA C. und PETER R. DE WAAL: *Multi-dimensional Bayesian Network Classifiers*. Technischer Bericht 056, Department of Information and Computer Sciences, Utrecht University, 2006.

[34] VAPNIK, VLADIMIR N.: *The Nature of Statistical Learning Theory*. Springer, New York [u.a.], 1995.