FAKULTÄT FÜR MATHEMATIK UND PHYSIK
MATHEMATISCHES INSTITUT

Mengenwertige Stabilisierung mit Vergangenheitsinformationen

Diplomarbeit

von

Florian Müller

Datum: 4. Juni 2007

Aufgabenstellung / Betreuung:
Prof. Dr. Lars Grüne

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Optimale Steuerungsprobleme bilden eine wichtige Klasse von Optimierungsproblemen. Sie treten am häufigsten in technischen und wirtschaftlichen Bereichen auf und sind meist sehr komplex. Sie unterliegen in der Regel Störungen, die von außen auf das System einwirken. Trotz der enormen Fortschritte ihrer Rechenleistung in den letzten Jahren stoßen moderne Computer bei der Lösung von Steuerungsproblemen häufig an ihre Grenzen. Die Störungen sind außerdem oft so groß, dass es nicht mehr möglich ist, ein Gleichgewicht des Systems zu erreichen. In diesem Fall ist das Steuerungsproblem nicht mehr stabilisierbar. Es ist also entscheidend, effizientere Algorithmen zur Lösung der Probleme zu finden. Ein Algorithmus ist effizienter, wenn er das Problem schneller löst oder er die Auswirkung der Störung verringert. Er ist also dafür verantwortlich, dass das Steuerungsproblem für eine größere Störung stabilisierbar wird. Das Ziel dieser Arbeitist es, einen Algorithmus durch Berücksichtigung vergangener Schritte effizienter zu gestalten.

In dieser Arbeit werden gestörte, diskrete, nicht diskontierte Steuerungsprobleme über einem unendlichen Zeithorizont untersucht. Bei den untersuchten Steuerungsproblemen kann der Zustand eines Systems nicht exakt bestimmt werden. Es lässt sich nur feststellen, in welchem Zustandsbereich es sich befindet. Die Ungewissheit des Zustandes wird als Störung des Systems aufgefasst. Ein Beispiel für ein solches Problem ist das Befüllen einer Tankanlage mit 100 Liter Fassungsvermögen. Es kann nur gemessen werden, ob sich im Tank mehr als $20l$, $40l$, $60l$ oder $80l$ befinden. Das System hat also fünf mögliche Zustandsbereiche und es kann angegeben werden, wie viele Liter nachgetankt werden sollen. Bei einem Tankvorgang kann das System unterschiedlich auf verschiedene Zustände aus einem Zustandsbereich reagieren. Wenn sich das System z.B. im Zustandsbereich zwischen $20l$ und $40l$ befindet und es sollen $10l$ nachgetankt werden, ist unklar, ob das System im gleichen Zustandsbereich bleibt, oder ob es in den Zustandsbereich zwischen $40l$ und $60l$ übergeht.

Diese Art von Problemen wird auf ein Kürzeste-Wege-Problem in einem Hypergraphen zurückgeführt. Die kürzesten Wege werden dann mit Hilfe des Algorithmus von Dijkstra be-

stimmt. Ein Algorithmus, der solche gestörten Steuerungsprobleme ohne Berücksichtigung von Vergangenheitsinformationen mit dem Algorithmus von Dijkstra löst, wurde bereits von Herrn von Lossow implementiert. Er soll durch Berücksichtigung von Informationen aus vergangenen Schritten effizienter gestaltet werden.
Der graphentheoretische Ansatz zur mengenorientierten optimalen Steuerung baut auf der Diplomarbeit [1] auf. In ihr werden ungestörte Steuerungsprobleme untersucht. Einige Ideen und Notationen sind aus den Papers [2] und [3] entnommen.

Als erstes wird im zweiten Kapitel die Problemstellung ohne Vergangenheitsbezug eingeführt. Anschließend werden die Problemstellungen erläutert, die Informationen aus dem vorausgegangenen Schritt bzw. aus n vorausgegangenen Schritten zur Bestimmung der optimalen Steuerung verwenden.
Im nächsten Kapitel werden einige grundlegende Begriffe aus der Graphentheorie behandelt. Insbesondere wird erläutert, was ein kürzester Weg in einem Hypergraphen ist und wie er mit Hilfe des Algorithmus von Dijkstra berechnet werden kann.
Im vierten Kapitel werden die graphentheoretischen Ansätze für zwei Problemstellungen aus dem zweiten Kapitel dargestellt. Es wird außerdem bewiesen, dass bei der gleichen Störung des Systems die optimale Wertefunktion der Problemstellung mit Vergangenheitsbezug höchstens so groß ist, wie die ohne Vergangenheitsbezug.
Im fünften Kapitel wird zunächst auf einige Details der Implementierung eingegangen. Danach werden die verwendeten Funktionen erläutert.
Im sechten Kapitel werden die Problemstellungen ohne bzw. mit Vergangenheitsbezug anhand von zwei Beispielen numerisch verglichen.
Im letzten Kapitel wird ein Ausblick auf mögliche Verbesserungen des Ansatzes gegeben und auf Verallgemeinerungen kurz eingegangen.

Kapitel 2

Problemstellung

In diesem Kapitel wird zunächst die Problemstellung ohne Vergangenheitsbezug eingeführt. Anschließend werden die Problemstellungen, die Vergangenheitsinformationen benutzen, erläutert.

2.1 Problemstellung ohne Vergangenheitsbezug

Man betrachte das diskrete Kontrollsystem

$$x_{k+1} = f(x_k, u_k), \quad k = 0, 1, \dots \quad , \tag{2.1}$$

wobei $f : X \times U \longrightarrow X$ eine stetige Funktion ist. Der Zustandsbereich $X \subset \mathbb{R}^n$ und der Kontrollwertebereich $U \subset \mathbb{R}^m$ sind jeweils kompakt.

Das Kontrollsystem (2.1) dient als Grundlage der in dieser Arbeit behandelten gestörten, diskreten Kontrollsysteme. Es wird angenommen, dass sich der Zustand vom Kontrollsystem (2.1) nicht exakt bestimmen lässt. Nur ein Zustandsbereich, in dem es sich befindet, ist feststellbar. Dieser Sachverhalt wird als Störung des Systemzustandes modelliert.
In der Praxis kann dies durch Messungenauigkeiten zu Stande kommen, z.B. bei der Messung von Temperaturen mit einem Thermometer. Das verwendete Thermometer kann die Temperatur nur auf ein zehntel Grad genau bestimmen, aber das System reagiert unterschiedlich auf Temperaturen innerhalb eines zehntel Grades.

Für die Formulierung von solchen Störungen wird der Zustandsbereich X in zusammenhängende, disjunkte Teilbereiche X_i unterteilt. Es wird angenommen, dass die Zerlegung endlich ist. Dies ist sinnvoll, da es bei Anwendungen aus der Praxis typischerweise keine unendliche Zerlegung eines kompakten Zustandraumes gibt. Die Temperatur ist z.B. nie beliebig genau messbar oder es ist aus Platzgründen nicht möglich unendlich viele Sensoren anzubringen.

Die Anzahl der Teilbereiche beträgt $l+1$. Für diese gilt:

$$\begin{aligned} &\bigcup_{i=0}^{l} X_i = X\ , \qquad X_i \subset X\ , \qquad X_i \text{ zusammenhängend,} \\ &X_a \cap X_b = \emptyset \qquad a \neq b \qquad a, b \in \{0, \ldots l\}. \end{aligned} \tag{2.2}$$

X_i ist in dieser Arbeit immer ein Bereich der ursprünglichen Zerlegung. Die Menge aller X_i wird mit $\mathbf{X_B}$ bezeichnet.

Annahme 2.1. *Das gesteuerte System besitzt einen Gleichgewichtsbereich $X^* \subset X$ mit $X^* = \bigcup_{i \in I^*} X_i$, d.h. für alle $X_i \subset X^*$ existiert ein $u^* \in U$, so dass $f(x, u^*) \in X^*$ für alle $x \in X_i$ gilt.*

Mit Hilfe der nächsten Definitionen lässt sich das gestörte Kontrollsystem formulieren.

Definition 2.2. *Die **Zuordnungsfunktion** $\rho : X \to \mathbf{X_B}$ wird definiert als $\rho(x) = X_i$ für $x \in X_i$.*

Die Funktion gibt an, in welchem Bereich X_i aus $\mathbf{X_B}$ sich ein Zustand x befindet. Die Menge der Kontrollfolgen $\{\mathbf{u} = (u_0, u_1, \ldots), u_i \in U\ \forall i \in \mathbb{N}\}$ wird als $U^{\mathbb{N}}$ und die Menge der Teilmengenfolgen $\{\mathbf{Y} = (Y_0, Y_1, \ldots), Y_i \subset X\ \forall i \in \mathbb{N}\}$ als $X^{\mathbb{N}}$ bezeichnet.

Definition 2.3. *Eine Funktion $\beta : \left(2^X\right)^{\mathbb{N}} \times U^{\mathbb{N}} \longrightarrow X^{\mathbb{N}}$ wird definiert als*

- $\beta(\boldsymbol{Y}, \boldsymbol{u}) = (\beta_0(Y_0, u_0), \beta_1(Y_1, u_1), \ldots)$
  *Beim ersten Anwenden von β wird die **Auswahlfunktion** β_0 verwendet, beim Zweiten die Funktion β_1, usw.*
- $\beta_i(Y, u) = x, \qquad x \in Y$
  *Eine **Auswahlfunktion** $\beta_i : 2^X \times U \longrightarrow X$ wählt aus der Zustandsmenge Y für eine Steuerung u einen Zustand x aus.*

$\mathbf{B}$ *bezeichnet die Menge aller Funktionen β und $\hat{\mathbf{B}}$ die Menge aller Funktionen β_i.*

Mit diesen Definitionen erhält man das gestörte diskrete Kontrollsystem

$$X_{k+1} = F(X_k, u_k, \beta_k), \quad k = 0, 1, \ldots\ , \tag{2.3}$$

wobei

$$\begin{aligned} &F : 2^X \times U \times \hat{\mathbf{B}} \to 2^X \\ &F(Y, u, \beta_i) = \rho\big(f(\beta_i(Y, u), u)\big) \qquad Y \subset X. \end{aligned}$$

ist. Das System wechselt nicht wie das Kontrollsystem (2.1) von Zustand zu Zustand, sondern von Zustandsbereich zu Zustandsbereich.

Eine Trajektorie $\mathbf{x}(Y, \mathbf{u}, \beta) = (\mathbf{x}_k(Y, \mathbf{u}, \beta))_{k\in\mathbb{N}}$ ist durch einen Startbereich $Y \in \mathbf{X_B}$, eine Kontrollsequenz $\mathbf{u} \in U^{\mathbb{N}}$, eine Auswahlfunktion $\beta \in \mathbf{B}$ und das Kontrollsystem (2.3) festgelegt.

Definition 2.4. *Der **Einzugsbereich** des Gleichgewichts X^* ist definiert als*

$$S = \{X_0 \in \mathbf{X_B} : \exists \boldsymbol{u} \in U^{\mathbb{N}} \ \ s.d. \ \forall \beta \in \boldsymbol{B} \ \ \boldsymbol{x}_k(X_0, \mathbf{u}, \beta) \to X^* \ \ für \ \ k \to \infty\}.$$

Für jeden Zustandsbereich in S existiert eine Steuerungssequenz, die das System für jede mögliche Störung ins Gleichgewicht steuert.

Das ungestörte System hat die Kostenfunktion

$$\hat{g} : X \times U \longrightarrow \mathbb{R}_0^+,$$

wobei $\hat{g}$ stetig ist und $\hat{g}(x, u^*) = 0$ für $x \in X^*$, $\hat{g}(x, u) \geq 0$ für $x \in X^*$, $\hat{g}(x, u) > 0$ für $x \neq X^*$ gilt.
Die Ungewissheit im Zustand wirkt sich auch auf die Kosten aus. Man weiß nicht, in welchem Zustand $x_i \in X_i$ sich das System befindet. Man muss deshalb annehmen, dass es sich im ungünstigsten Zustand befindet, d.h. in dem, der die meisten Kosten verursacht. Als Kostenfunktion für das Kontrollsystem (2.3) erhält man deshalb

$$\begin{aligned} &g : 2^X \times U \longrightarrow \mathbb{R}_0^+, \\ &g(Y, u) = \sup_{y\in Y} \hat{g}(y, u) \qquad Y \subset X \end{aligned} \tag{2.4}$$

Mit Hilfe von g lässt sich das Funktional

$$J(Y, \mathbf{u}, \beta) = \sum_{k=0}^{\infty} g(\mathbf{x}_k(Y, \mathbf{u}, \beta), u_k) \ \in \mathbb{R}_0^+ \cup \{+\infty\} \qquad Y \in \mathbf{X_B} \tag{2.5}$$

definieren. Durch dieses Funktional wird die optimale Wertefunktion

$$V(Y) = \sup_{\beta\in\mathbf{B}} \inf_{\mathbf{u}\in U^{\mathbb{N}}} J(Y, \mathbf{u}, \beta) \qquad Y \in \mathbf{X_B} \tag{2.6}$$

des Optimierungsproblems definiert.
Falls Y sich nicht im Einzugsbereich S eines Gleichgewichts befindet, nimmt V den Wert unendlich an, andernfalls ist $V(Y) < \infty$. Dies gilt wegen der Endlichkeit der Zerlegung und der Beschränktheit von g auf der kompakten Menge X. Außerdem kann es nie vorkommen,

dass ein Zustandsbereich, außer dem Gleichgewichtsbereich, zweimal in einer optimalen Trajektorie benutzt wird.

V erfüllt das Bellmannsche Optimalitätsprinzip

$$V(Y) = \inf_{u \in U} \left\{ g(Y,u) + \sup_{\beta_i \in \hat{\mathbf{B}}} V(F(Y,u,\beta_i)) \right\} \qquad Y \in \mathbf{X_B}. \tag{2.7}$$

Als Nächstes wird eine Eigenschaft der optimalen Wertefunktion entlang optimaler Trajektorien aufgezeigt.

Satz 2.5. *Sei u^* die optimale Steuerung für den Zustand $Y \in \mathbf{X_B}$. Es gilt dann $\forall x \in Y$*

$$g(\rho(x), u^*) + V\Big(\rho\big(f(x,u^*)\big)\Big) \leq V(Y), \tag{2.8}$$

d.h. die optimale Wertefunktion nimmt entlang einer optimalen Trajektorie immer ab.

Beweis:
Es gilt

$$\begin{aligned} g(\rho(x), u^*) &= g(Y; u^*) && \forall x \in Y \\ V\Big(\rho\big(f(x,u^*)\big)\Big) &\leq \sup_{\beta_i \in \hat{\mathbf{B}}} V\big(F(Y,u,\beta_i)\big) && \forall x \in Y. \end{aligned}$$

Daraus folgt sofort (2.8). □

Bemerkung 2.6. *Eine Einführung in die dynamische Programmierung und optimaler Kontrollen befindet sich in [10]. Die Störung, der in den Papern [2] und [3] behandelten Kontrollsysteme, kann auf die in dieser Arbeit untersuchte Art von Störungen zurückgeführt werden. Daraus folgt die Gültigkeit des Bellmannschen Optimalitätsprinzips.*

2.2 Problemstellung mit Vergangenheitsbezug

Bisher wurde nur der Zustand x_k benutzt, um den Zustand x_{k+1} zu berechnen. In diesem Abschnitt werden zusätzlich frühere Zustände des Systems berücksichtigt.

2-stufige Problemstellung:

Zuerst wird ein 2-stufiges ungestörtes Kontrollsystem

$$z_k = \begin{pmatrix} x_{k-1} \\ x_k \end{pmatrix}^T, \qquad z_{k+1} = \begin{pmatrix} x_k \\ f_2(z_k, u_k) \end{pmatrix}^T$$

betrachtet, wobei $f_2 : \big(X \cup \{\delta\}\big) \times X \times U \to X$ eine stetige Funktion ist. Dabei werden nicht mehr einzelne Zustände, sondern Zustandsvektoren z_k verwendet. Für den Start einer Trajektorie wird das Symbol δ benötigt, da im ersten Schritt der Zustand x_{-1} unbekannt ist. Eine Trajektorie startet also mit dem Zustandsvektor $z_0 = (\delta, x_0)$.

Als Nächstes wird das gestörte, diskrete Kontrollsystem

$$Z_k = \begin{pmatrix} X_{k-1} \\ X_k \end{pmatrix}^T, \qquad Z_{k+1} = \begin{pmatrix} X_k \\ F_2(Z_k, u_k, \beta_k) \end{pmatrix}^T \tag{2.9}$$

mit

$$F_2 : (\mathbf{X_B} \cup \{\delta\}) \times \mathbf{X_B} \times U \times \hat{\mathbf{B}} \to 2^X$$

$$F_2(Z, u, \beta_i) = \begin{cases} F(Y_2, u, \beta_i) & \text{falls} \quad Y_1 = \delta \\ F(X(Z), u, \beta_i) & \text{sonst} \end{cases}, \qquad Z = \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix}^T$$

$$X(Z) = \bigcup_{u \in U_{Y_1 Y_2}, \beta_i \in \hat{\mathbf{B}}} \left(f(\beta_i(Y_1, u), u) \cap Y_2\right).$$

betrachtet, wobei in der Menge $U_{Y_i Y_j} := \{u \in U : \exists y \in Y_i \quad s.d. \quad f(y, u) \in Y_j\}$ alle Steuerungen enthalten sind, mit denen man von $Y_i \subset X$ nach $Y_j \subset X$ gelangt. F ist die Übergangsfunktion vom Kontrollsystem (2.3). In einem Schritt wechselt das System vom Zustandsbereichsvektor $Z_k = (X_{k-1}, X_k)$ zum Zustandsbereichsvektor $Z_{k+1} = (X_k, X_{k+1})$. Nur bei der Berechnung von $X(Z_k)$ in der Bestimmung von X_{k+1} gehen Informationen aus dem Schritt $k-1$ ein.
Mit Hilfe von $X(Z_k)$ werden Zustände ausgeschlossen, die das System nicht erreichen kann. Für alle $\hat{x} \in X_k \backslash X(Z_k)$ existiert kein Paar $(x_{k-1}, u) \in X_{k-1} \times U$ mit $f(x-k-1, u) = \hat{x}$.
Durch die Verwendung der zusätzlichen Information wird die Ungewissheit im System verringert. Die Störung des Ausgangszustandes nimmt ab.

Eine Trajektorie $\mathbf{x}_k(Z, \mathbf{u}, \beta)$ ist definiert durch $Z \in \big(\mathbf{X_B} \cup \{\delta\}\big) \times \mathbf{X_B}$, $\mathbf{u} \in U^{\mathbb{N}}$, $\beta \in \mathbf{B}$ und das Kontrollsystem (2.9).
Man betrachte die Kostenfunktion

$$\begin{aligned} &g_2 : \big(\mathbf{X_B} \cup \{\delta\}\big) \times \mathbf{X_B} \times U \to \mathbb{R}_0^+, \\ &g_2(Z, u) = \sup_{x \in X(Z)} \hat{g}(x, u), \end{aligned} \tag{2.10}$$

wobei $X\big((\delta, X_i)\big) = X_i$ und $\hat{g}$ aus dem ungestörten Kontrollsystem ist. Mit dieser Kostenfunktion erhält man das Funktional

$$J(Z, \mathbf{u}, \beta) = \sum_{k=0}^{\infty} g_2(\mathbf{x}_k(Z, \mathbf{u}, \beta), u_k) \ \in \mathbb{R}_0^+ \cup \{+\infty\} \tag{2.11}$$

und damit die optimale Wertefunktion

$$V(Z) = \sup_{\beta \in \mathbf{B}} \inf_{\mathbf{u} \in U^{\mathbb{N}}} J(Z, \mathbf{u}, \beta). \tag{2.12}$$

V erfüllt das Bellmannsche Optimalitätsprinzip

$$V(Z) = \inf_{u \in U} \left\{ g_2(Z, u) + \sup_{\beta_i \in \hat{\mathbf{B}}} V\big(F_2(Z, u, \beta_i)\big) \right\}. \tag{2.13}$$

Es gibt für diese Problemstellung eine Entsprechung zu Satz 2.5.

Satz 2.7. *Sei u^* die optimale Steuerung für den Zustand Z. Es gilt dann $\forall x \in X(Z)$*

$$g_2(\rho(x), u^*) + V\Big(\rho\big(f(x, u^*)\big)\Big) \leq V(Y), \tag{2.14}$$

d.h. die optimale Wertefunktion nimmt entlang einer optimalen Trajektorie immer ab.

Beweis:
Es gilt

$$\begin{aligned} g_2(\rho(x), u^*) &= g_2(Z; u^*) && \forall x \in X(Z) \\ V\Big(\rho\big(f(x, u^*)\big)\Big) &\leq \sup_{\beta_i \in \hat{\mathbf{B}}} V(F_2(Z, u, \beta_i)) && \forall x \in X(Z). \end{aligned}$$

Daraus folgt sofort (2.14). □

n-stufige Problemstellung

Die Ungewissheit im Zustand wird geringer, je mehr vergangene Schritte man in die Berechnung von Z_{k+1} mit einbezieht. Für alle Zustandsbereichsvektoren $Z \in \left(\mathbf{X_B} \cup \{\delta\}\right)^n \times \mathbf{X_B}$ eines n-stufigen Systems gilt,

- dass $Z_{(i)}$ die i-te Komponente des Vektors Z ist
- und falls $Z_{(i)} = \delta$ ist, dann ist $Z_{(j)} = \delta$ für alle $j = 1, \ldots, i-1$.

Das gestörte diskrete Kontrollsystem

$$Z_k = \begin{pmatrix} X_{k-n+1} \\ X_{k-n+2} \\ \vdots \\ X_{k-1} \\ X_k \end{pmatrix}^T \qquad Z_{k+1} = \begin{pmatrix} X_{k-n+2} \\ X_{k-n+3} \\ \vdots \\ X_k \\ F_n(Z_k, u_k, \beta_k) \end{pmatrix}^T \tag{2.15}$$

mit

$$F_n : \left(\mathbf{X_B} \cup \{\delta\}\right)^n \times \mathbf{X_B} \times U \times \hat{\mathbf{B}} \to \mathbf{X_B}$$

$$F_n(Z, u, \beta_i) := \begin{cases} F\big(Z_{(n)}, u, \beta_i\big) & falls \quad Z_{(n-1)} = \delta \\ F_2\big((Z_{(n-1)}, Z_{(n)}), u, \beta_i\big) & falls \quad Z_{(n-2)} = \delta, Z_{(n-1)} \neq \delta \\ \vdots & \vdots \quad \vdots \\ F_{n-1}\big((Z_{(2)}, \ldots, Z_{(n)}), u, \beta_i\big) & falls \quad Z_{(1)} = \delta, Z_{(2)} \neq \delta \\ F\big(X(Z), u, \beta_i\big) & falls \quad Z_{(1)} \neq \delta \end{cases}$$

$$\begin{aligned} X(Z) &= \bigcup_{u \in U_{Z_{(n-1)} Z_{(n)}}, \beta_i \in \hat{\mathbf{B}}} f(\beta_i(X^{n-1}(Z), u), u) \cap Z_{(n)} \\ X^{n-1}(Z) &= \bigcup_{u \in U_{Z_{(n-2)} Z_{(n-1)}}, \beta_i \in \hat{\mathbf{B}}} f(\beta_i(X^{n-2}(Z), u), u) \cap Z_{(n-1)} \\ &\vdots \\ X^2(Z) &= \bigcup_{u \in U_{Z_{(1)} Z_{(2)}}, \beta_i \in \hat{\mathbf{B}}} f(\beta_i(X^1(Z), u), u) \cap Z_{(2)} \\ X^1(Z) &= Z_{(1)} \end{aligned}$$

hat n Stufen. F, F_2 bis F_{n-1} sind die Übergangsfunktionen der 1- bis (n-1)-stufigen gestörten diskreten Kontrollsysteme.

Eine Trajektorie $\mathbf{x}_k\big(Z, \mathbf{u}, \beta\big)$ ist definiert durch $Z \in (\mathbf{X_B} \cup \{\delta\})^n \times \mathbf{X_B}$, $\mathbf{u} \in U^{\mathbb{N}}$, $\beta \in \mathbf{B}$ und das Kontrollsystem (2.15).
Man betrachte die Kostenfunktion

$$g_n : \big(\mathbf{X_B} \cup \{\delta\}\big)^n \times \mathbf{X_B} \times U \to \mathbb{R}_0^+,$$
$$g_n(Z,u) := \begin{cases} g_{n-1}((Z_{(2)}, \ldots, Z_{(n)}), u) & falls \quad Z_{(1)} = \delta \\ \sup_{x \in X(Z)} \hat{g}(x,u) & falls \quad Z_{(1)} \neq \delta \end{cases}$$

wobei $\hat{g}$ aus dem ungestörten Kontrollsystem und g_{n-1} die Kostenfunktion des (n-1)-stufigen Problems ist. Mit dieser Kostenfunktion erhält man das Funktional

$$J(Z, \mathbf{u}, \beta) = \sum_{k=0}^{\infty} g_n(\mathbf{x}_k(Z, \mathbf{u}, \beta), u_k) \ \in \mathbb{R}_0^+ \cup \{+\infty\}$$

und damit die optimale Wertefunktion

$$V(Z) = \sup_{\beta \in \mathbf{B}} \inf_{\mathbf{u} \in U^{\mathbb{N}}} J(Z, \mathbf{u}, \beta).$$

V erfüllt das Bellmannsche Optimalitätsprinzip

$$V(Z) = \inf_{u \in U} \left\{ g_n(Z,u) + \sup_{\beta_i \in \hat{\mathbf{B}}} V\big(F_n(Z,u,\beta_i)\big) \right\}.$$

Auch für diese Problemstellung gilt, dass die optimale Wertefunktion entlang einer optimalen Trajektorie abnimmt.

Kapitel 3

Grundbegriffe aus der Graphentheorie

Für den Algorithmus zur Berechnung der optimalen Wertefunktion V werden einige Begriffe und ein Algorithmus aus der Graphentheorie benötigt. Eine Einführung in die Graphentheorie findet sich in [7].

Definition 3.1. *Ein* **Graph** G *ist eine Menge von Knoten* $V(G)$ *und Kanten* $E(G)$ *mit* $E(G) \subset V(G) \times V(G)$.

Diese Form von Graphen ist zur Berechnung der optimalen Wertefunktion V eines gestörten Kontrollsystems ungenügend. Dazu wird ein bewerteter, gerichteter Hypergraph benötigt.

Definition 3.2. *Ein* **gerichteter Hypergraph** $G = (V(G), E(G))$ *ist ein 2-Tupel bestehend aus:*

- *der nichtleeren* **Knotenmenge** $V(G)$
- *und der* **Hyperkantenmenge** $E(G) \subset V(G) \times 2^{V(G)}$.

Für eine Hyperkante $e \in E(G)$ wird folgende Notation verwendet

$$e : a \to A,$$

wobei $a \in V(G)$ der **Startknoten** und $A \subset V(G)$ die Menge der **Zielknoten** ist.
In dieser Arbeit wird eine Hyperkante $e : a \to A$, als Übergangsfunktion interpretiert. Bei Anwendung von e geht der Graph vom Startknoten s zu einem beliebigen Zielknoten $a \in A$ über. Es ist also unklar, welcher Zielknoten erreicht wird.

Definition 3.3. *Ein* **bewerteter, gerichteter Hypergraph** $G = \big(V(G), E(G)\big)$ *ist ein gerichteter Hypergraph G, in dem jeder Hyperkante* $e \in E(G)$ *eine reelle, nichtnegative Zahl w(e) zugeordnet wird. w(e) bezeichnet die Kosten bzw. die Länge der Hyperkante e.*

Definition 3.4. *Ein Weg $p(s,t,\hat{E})$ im Hypergraphen $G(V,E)$ vom Knoten s zum Knoten t ist durch eine Menge $\hat{E}$ von Hyperkanten festgelegt. Diese muss die Bedingungen*

- $\exists\, s \to S \in \hat{E}$
- $\exists\, c \to \{t\} \in \hat{E}$
- $\forall a \to A \ \in \hat{E}$ *gilt:* $\forall b \in A \ \exists\, p(b,t,\tilde{E})$ *mit* $\tilde{E} \subset \hat{E}$

erfüllen. Falls kein Weg von s nach t existiert, wird $\hat{E}$ als leere Menge definiert.

In einem Weg $p(s,t,\hat{E})$ gibt es für jeden Zielknoten einer Hyperkante aus $\hat{E}$ einen Weg nach t, der nur Hyperkanten aus der Menge $\hat{E}$ verwendet.

Definition 3.5. *Im Hypergraphen G ist eine* **Realisierung des Weges** $p(s,t,\hat{E})$ *vom Knoten s zum Knoten t definiert als*

$$p_r(s,t,\hat{E}) := \big[(s = v_0, v_1, \ldots, v_k = t), (e_0, \ldots e_{k-1})\big],$$

wobei $e_i : v_i \to A \in \hat{E}$ und $v_{i+1} \in A \ \forall i \in \{0,\ldots,k-1\}$ gelten muss.

Die Kosten bzw. die Länge dieser Realisierung sind

$$w\big(p_r(s,t,\hat{E})\big) = \sum_{j=0}^{k-1} w(e_j).$$

Sei $P_r(s,t,\hat{E})$ die Menge aller Realisierungen des Weges $p(s,t,\hat{E})$. Falls $\hat{E}$ ungleich der leeren Menge ist, beträgt die Länge des Weges $p(s,t,\hat{E})$

$$w(p(s,t,\hat{E})) = \max_{p_r \in P_r} \big\{w(p_r(s,t,\hat{E}))\big\}, \tag{3.1}$$

ansonsten gilt per Definition

$$w(p(s,t,\emptyset)) = \infty.$$

In (3.1) muss das Maximum von allen Realisierungen genommen werden, da unbekannt ist, welcher Zielknoten mit einer Hyperkante erreicht wird. Möglicherweise kann t mit einer anderen Realisierung günstiger erreicht werden, es muss aber der teuerste Fall angenommen werden.
Die Längen verschiedener Wege von s nach t können unterschiedlich sein. Es ist also wünschenswert die Länge des kürzesten Weges zu bestimmen. Diese ist

$$w_{min}(\{s,t\}) = \min_{p(s,t,\hat{E}) \in \{s,t\}} \Big\{w(p(s,t,\hat{E}))\Big\}.$$

wobei $\{s,t\}$ die Menge aller Wege von s nach t ist.

Bemerkung 3.6. *Die einzelnen Wege in der Menge $\{s,t\}$ unterscheiden sich nur in der Hyperkantenmenge $\hat{E}$. Sie sind also Variationen der Hyperkantenmenge.*

Sei S die Menge der Kanten mit dem Startknoten s. Falls ein Weg mit einer nichtleeren Hyperkantenmenge $\hat{E}$ von s nach t existiert, gilt

$$w_{min}\big(\{s,t\}\big) = \min_{e:s\to A\in S}\left\{w(e) + \max_{a\in A}\big\{w_{min}(\{a,t\})\big\}\right\}. \tag{3.2}$$

für die Länge des kürzesten Weges, ansonsten ist

$$w_{min}\big(\{s,t\}\big) = \infty. \tag{3.3}$$

Mit dem Algorithmus von Dijkstra (erstmals 1959 veröffentlicht in [9]) kann man den kürzesten Weg von allen Knoten aus V(G) zu einem Zielknoten t berechnen. Um ihn anwenden zu können muss man folgende Annahme treffen:

Annahme 3.7. *Im Graph bzw. im Hypergraphen sind die Knoten- und Kantenmenge endlich.*

Für die Lösung des Kürzesten-Wege-Problems ist bis heute noch kein effizienterer als der Algorithmus von Dijkstra gefunden worden. Mit dem normalen Algorithmus von Dijkstra kann man den kürzesten Weg in einem Graphen nach Definition 3.1 berechnen (siehe zweites Kapitel von [1]). Für einen Hypergraphen muss der Algorithmus angepasst werden.

Algorithmus 3.8 (Algorithmus von Dijkstra).

Eingabe: *Bewerteter Hypergraph $G=(V,E)$ mit Bewertungsfunktion w, Zielknoten $t\in V$*

Schritt 1: *Setze $\lambda(t)=0,\ \lambda(v)=\infty\ \forall v\in V\backslash\{t\}$, $T=V$*

Schritt 2: *Falls $T=\emptyset$, so ist der Algorithmus beendet*

Schritt 3: *Wähle einen Knoten $u\in T$ so, dass $\lambda(u)$ minimal ist unter allen Knoten in T und setze $T=T\backslash\{u\}$*

Schritt 4: *Falls $\lambda(u)=\infty$, so ist der Algorithmus beendet*

Schritt 5: *Für jede Kante $e:a\to A$ mit $u\in A$ und $A\subset V\backslash T$ setze $\lambda(a)=\min\{\lambda(a),\lambda(u)+w(e)\}$*

Schritt 6: *Gehe zu Schritt 2*

Ausgabe: *$\lambda(v)$ gibt die Länge des kürzesten Weges von v nach t an.*

Berechnet wird der kürzeste Weg von allen Knoten zum Zielknoten t. Falls die Bedingung $\lambda(u) = \infty$ in Schritt 4 erfüllt ist, kann es keine Verbesserung eines Wertes $\lambda(v)$ $v \in T$ geben. Dies erkennt man sofort, wenn man Schritt 5 betrachtet. Bei der Berechnung des Minimums gilt immer $\lambda(u) = \infty$. Deshalb kann sich $\lambda(a)$ nicht mehr verbessern und alle in T verbleibende Knoten behalten den Wert ∞.
Für alle Knoten $v \in V$, die am Ende des Algorithmus den Wert ∞ haben, gilt: Es existiert kein Weg von v nach t.
Im Algorithmus wird kein explizites Maximum über die Zielknoten einer Hyperkante gebildet. Das Maximum wird in Schritt 5 automatisch gebildet, weil für die Zielknoten jeder verwendeten Hyperkante $e : a \to A$ gilt: $\lambda(u) \geq \lambda(v) \ \ \forall v \in A$.

Die **Komplexität** des Algorithmus hängt von den verwendeten Datenstrukturen ab. In der in dieser Arbeit verwendeten Implementierung wird ein binärer Heap für die Verwaltung der Menge T verwendet. Es ergibt sich der Aufwand

$$O(|V| \log(|V|) + |E| \log(|V|) + N^2|E|).$$

Den ersten Summanden erhält man, da die Schleife von Schritt 2 bis 6 maximal $|V|$ mal durchläuft und dabei jedes mal ein Element aus T entfernt. Nach dem Entfernen muss die Heapeigenschaft von T wiederhergestellt werden. Dies hat den Aufwand $O(\log(|V|))$.
Eine Hyperkante $e : a \to A$ kann nur den Wert ihres Startknotens verringern. Falls sie ihn verringert hat, gilt $\lambda(a) = \lambda(u) + w(e)$. Eine erneute Verringerung ist mit dieser Hyperkante nicht möglich. Im gesamten Algorithmus kann sich deshalb maximal $|E|$ mal der Wert von einem Knoten ändern. Falls sich der Wert geändert hat, muss hier ebenfalls die Heapeigenschaft wiederhergestellt werden. Man erhält dadurch den zweiten Summanden.
Sei N die maximale Anzahl von Zielknoten einer Hyperkante. Der letzte Summand wird zur Überprüfung der Bedingungen in Schritt 5 benötigt. Über alle Schleifendurchgänge hinweg wird jede Kante maximal N mal angeschaut und die Überprüfung, ob A in $V \backslash T$ liegt, hat den Aufwand $O(N)$.

Zur Veranschaulichung des Algorithmus wird ein Beispiel behandelt.

Beispiel 3.9. *Man betrachte folgenden Graphen.*

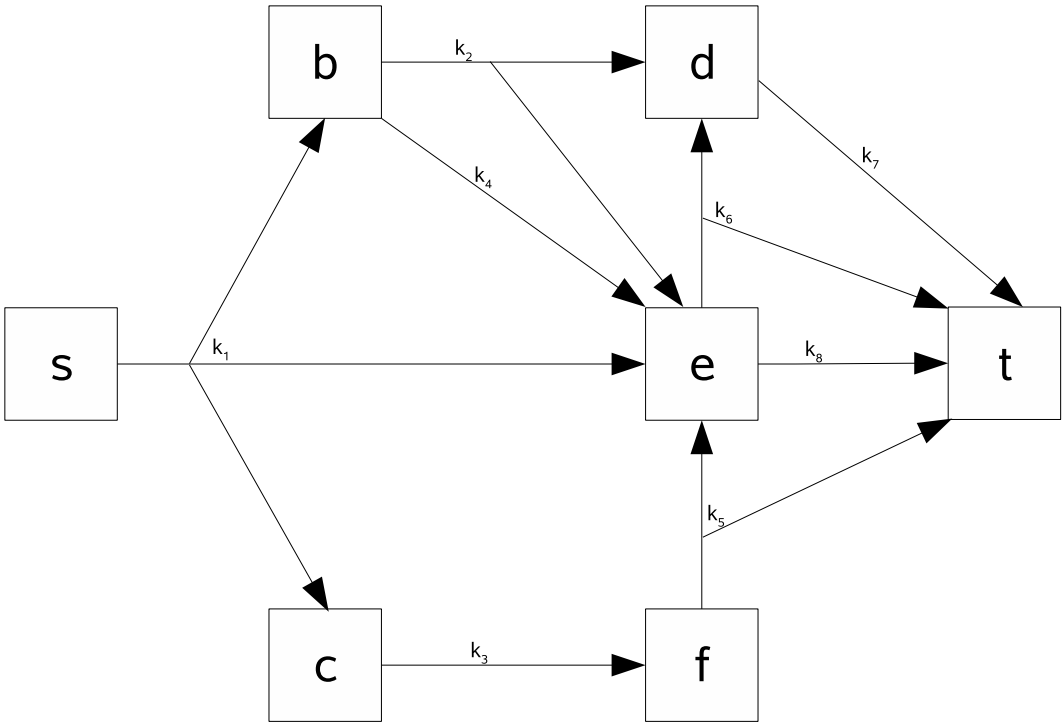

Abbildung 3.1: Hypergraph aus Beispiel 3.9

die Kanten haben folgende Längen

Kante k	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8
w(k)	40	5	15	15	10	30	20	40

Schritt 1: Man erhält:

Knoten v	s	b	c	d	e	f	t
$\lambda(v)$	∞	∞	∞	∞	∞	∞	0

$T = \{s, b, c, d, e, f, t\}$

Schritt 3: Wähle $u = t$, setze $T = T \backslash \{t\}$, $T = \{a, b, c, d, e, f\}$

Schritt 5 + 6: Die Kanten k_5, k_6, k_7 und k_8 zeigen auf z. Die Kanten k_7 und k_8 erfüllen die Bedingungen in Schritt 5. Man erhält

Knoten v	s	b	c	d	e	f	t
$\lambda(v)$	∞	∞	∞	20	40	∞	0

Gehe zu Schritt 2.

In den nächsten Schritten erhält man

Knoten v	s	b	c	d	e	f	t	
$\lambda(v)$	∞	∞	∞	20	40	∞	0	$T = \{s, b, c, e, f\}$

Knoten v	s	b	c	d	e	f	t	
$\lambda(v)$	∞	45	∞	20	40	50	0	$T = \{s, b, c, f\}$

Knoten v	s	b	c	d	e	f	t	
$\lambda(v)$	∞	45	∞	20	40	50	0	$T = \{s, c, f\}$

Knoten v	s	b	c	d	e	f	t	
$\lambda(v)$	∞	45	65	20	40	50	0	$T = \{s, c\}$

Knoten v	s	b	c	d	e	f	t	
$\lambda(v)$	105	45	65	20	40	50	0	$T = \{s\}$

Schritt 3: Wähle $u = s$, setze $T = T \backslash \{s\}$, $T = \emptyset$

Schritt 4: Da keine Kanten auf s zeigen, kann es keine Verringerung eines Wertes geben

Schritt 5: Gehe zu Schritt 2

Schritt 2: Ende des Algorithmus

Kapitel 4

Graphentheoretischer Ansatz

Der Algorithmus von Dijkstra kann nur auf endliche Hypergraphen angewendet werden. Deshalb wird im Folgenden angenommen, dass die Steuerungsmenge endlich ist. Dies ist notwendig, damit nur endlich viele Hyperkanten in einem Hypergraphen existieren.
Von jetzt an wird nur noch die Problemstellung ohne Vergangenheitsbezug und die 2-stufige Problemstellung betrachtet. Die Ergebnisse sind auf die n-stufige Problemstellung übertragbar. Die Rechenzeit zum Aufbau des Hypergraphen und zur Lösung des Kürzesten-Wege-Problems wächst mit der Anzahl der Stufen extrem schnell an.

4.1 Ansatz für die Problemstellung ohne Vergangenheitsbezug

Das Kontrollsystem (2.3) kann als gerichteter Hypergraph $G(V, E)$ mit

$$\begin{aligned} V &= \{X_i : \ i = 0, 1, \ldots, l\} \\ E &= \{X_i \to A_u^i : \ i = 0, 1, \ldots, l, \ u \in U\} \end{aligned} \tag{4.1}$$

dargestellt werden, wobei

$$A_u^i = \{X_j \in \mathbf{X_B} : \ \exists x \in \overline{X_i} \ \text{ mit } f(x, u) \in X_j\}$$

ist. Es existiert also für jedes Paar $(X_i, u) \in \mathbf{X_B} \times U$ genau eine Hyperkante. Die Länge einer Hyperkante ist definiert als

$$w(X_i \to A_u^i) = \max_{x \in \overline{X_i}} \hat{g}(x, u),$$

wobei $\hat{g}$ die Kostenfunktion des ungestörten Kontrollsystems ist.

Die Länge des kürzesten Weges von X_i nach X^* entspricht dem Wert der optimalen Wertefunktion V. Dies erkennt man, wenn man

$$V(X_i) = \inf_{u \in U} \left\{ g(X_i, u) + \sup_{\beta_i \in \hat{\mathbf{B}}} V\big(F(X_i, u, \beta_i)\big) \right\} \qquad X_i \in \mathbf{X_B}$$

und

$$w_{min}\big(\{X_i, X^*\}\big) = \min_{e: X_i \to A_U^i \in S} \left\{ w(e) + \max_{X_j \in A_u^i} \big\{ w_{min}(\{X_j, X^*\}) \big\} \right\}$$

vergleicht, wobei alle Kanten mit dem Startknoten X_i in der Menge S enthalten sind.
Wenn eine Funktion β_i einen Punkt nicht mehr aus X_i sondern aus $\overline{X_i}$ auswählt, kann das Supremum durch ein Maximum ersetzt werden. Wegen der Endlichkeit von U kann das Inifimum ebenfalls durch ein Minimum ersetzt werden.
Für jede Steuerung $u \in U$ existiert die Kante $e : X_i \to A_u^i \in S$ und für jede Kante $e \in S$ existiert die Steuerung $u \in U$. Für diese Paare gilt

$$w(e) = w(X_i \to A_u^i) = \max_{x \in \overline{X_i}} \hat{g}(x, u) = \sup_{x \in X_i} \hat{g}(x, u) = g(X_i, u).$$

und

$$\bigcup_{\beta_i \in \hat{\mathbf{B}}} F(X_i, u, \beta_i) = A_u^i$$

Durch Induktion, ausgehend von X^*, folgt daraus die Identität.

Zur Veranschaulichung folgt ein einfaches Beispiel.

Beispiel 4.1. *Man betrachte das Kontrollsystem*

$$x_{k+1} = x_k + \frac{1}{4} u_k x_k \tag{4.2}$$

mit der Kostenfunktion

$$\hat{g}(x, u) = \frac{1}{4} x,$$

wobei $x \in X = [0\ ;\ 1]$ *und* $u \in \{-1; 1\}$ *gilt. Der Bereich* X *wird in die Intervalle* $a = [0\ ;\ 0,5]$ *und* $b =]0,5\ ;\ 1]$ *unterteilt.*

Bei diesem Beispiel kann passieren, dass das System (1.3) den Bereich X verlässt. Es wird angenommen, dass das System nicht mehr konvergieren kann, wenn es den Bereich X verlassen hat, d.h. dieser Schritt hat die Kosten ∞. Daraus folgt, dass die Kosten der Hyperkante $X_i \to A_u^i$ unendlich sind, falls $\exists\, x \in X_i$ mit $f(x, u) \notin X$. Solche Hyperkanten werden nie für einen kürzesten Weg verwendet. Sie können also vom Algorithmus von Dijkstra ignoriert werden, d.h. sie werden bei der Erstellung des Hypergraphen überhaupt nicht gespeichert.

Für das Beispiel erhält man folgende Hyperkanten

- $e_1 : a \to \{a\}$ für $u = -1$, $w(e_1) = 0,125$
- $e_2 : a \to \{a, b\}$ für $u = 1$, $w(e_2) = 0,125$
- $e_3 : b \to \{a, b\}$ für $u = -1$, $w(e_3) = 0,25$

und den Hypergraphen

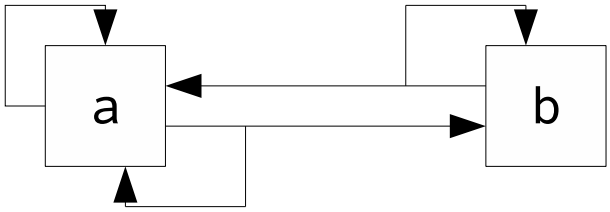

Abbildung 4.1: Hypergraph ohne Vergangenheitsbezug für Beispiel 4.1

Das System kann den Bereich X vom Knoten b mit der Steuerung $u = 1$ verlassen. Deshalb wird die Hyperkante für diesen Knoten mit dieser Steuerung ignoriert.
Die Länge des kürzesten Weges von b nach a ist somit ∞.

4.2 Ansatz für die Problemstellung mit Vergangenheitsbezug

Das 2-stufige Kontrollsystem (2.9) kann als gerichteter Hypergraph $G(V, E)$ mit

$$\begin{aligned} V &= \{X_i X_j : \ X_i \in \mathbf{X_B} \cup \{\delta\}; \ \ X_j \in \mathbf{X_B}\} \\ E &= \{X_i X_j \to A_u^{ij} : \ u \in U; \ \ X_i X_j \in V\} \end{aligned} \tag{4.3}$$

dargestellt werden, wobei

$$A_u^{ij} = \left\{ X_j X_k \in V : \ \exists x \in \overline{X(Z)} \ \text{ mit } f(x,u) \in X_k \ \text{ und } Z = (X_i, X_j) \right\}$$

ist und

$$\overline{X(Z)} = \bigcup_{u \in U_{\overline{X_i X_j}}, \beta_i \in \hat{\mathbf{B}}} \left(f(\beta_i(\overline{X_i}, u), u) \ \cap \ \overline{X_j} \right).$$

Man erhält für jede Steuerung und jeden Knoten eine Hyperkante.
Als Länge einer Hyperkante wird

$$w(X_i X_j \to A_u^{ij}) = \max_{x \in \overline{X(Z)}} \hat{g}(x, u),$$

definiert.

Durch Vergleich von (2.13) und (3.2) erkennt man, analog zur Problemstellung ohne Vergangenheitsbezug, dass die Länge des kürzesten Weges von X_iX_j nach X_kX^*, dem Wert der optimalen Wertefunktion V für den Zustand X_iX_j entspricht. Die entscheidenden Argumente sind hier ebenfalls die Endlichkeit von U und die Abgeschlossenheit der Bereiche X_i. Der Wert der optimalen Wertefunktion kann also mit dem Algorithmus von Dijkstra bestimmt werden.

Eine Steuerung $u \in U$ wird bei der Berechnung von $\overline{X(Z)}$ nur berücksichtigt, falls man mit dieser Steuerung X nicht verlassen kann. Eine Steuerung, mit der man den Bereich X verlässt, ist uninteressant, da diese nie für einen kürzesten Weg verwendet wird. Man erhält

$$U_{\overline{X_iX_j}} := \{u \in U : \exists x \in X_i \;\; s.d. \;\; f(x,u) \in X_j; \;\; \nexists x \in X_i \;\; s.d. \;\; f(x,u) \notin X\}$$

Zur Veranschaulichung wird Beispiel 4.1 betrachtet.

Knoten Z	δa	δb	aa	ab	ba	bb
$\overline{X(Z)}$	$[0\ ;\ 0,5]$	$[0,5\ ;\ 1]$	$[0\ ;\ 0,5]$	$[0,5\ ;\ 0,625]$	$[0,375\ ;\ 0,5]$	$[0,5\ ;\ 0,75]$

Das System kann X vom Bereich b aus mit der Steuerung $u = 1$ verlassen. Es wird also keine Kante für diesen Knoten mit dieser Steuerung gespeichert werden. Diese Steuerung wird bei der Berechnung von $X(bb)$ ebenfalls nicht berücksichtigt.

Man erhält folgende Hyperkanten

- $e_1 : \delta a \to \{aa\}$ für $u = -1$, $w(e_1) = 0,125$
- $e_2 : \delta a \to \{aa, ab\}$ für $u = 1$, $w(e_2) = 0,125$
- $e_3 : \delta b \to \{ba, bb\}$ für $u = -1$, $w(e_3) = 0,25$
- $e_4 : aa \to \{aa\}$ für $u = -1$, $w(e_4) = 0,125$
- $e_5 : aa \to \{aa, ab\}$ für $u = 1$, $w(e_5) = 0,125$
- $e_6 : ab \to \{ba\}$ für $u = -1$, $w(e_4) = 0,15625$
- $e_7 : ab \to \{bb\}$ für $u = 1$, $w(e_5) = 0,15625$
- $e_8 : ba \to \{aa\}$ für $u = -1$, $w(e_4) = 0,125$
- $e_9 : ba \to \{aa, ab\}$ für $u = 1$, $w(e_5) = 0,125$
- $e_{10} : bb \to \{ba, bb\}$ für $u = -1$, $w(e_4) = 0,1875$
- $e_{11} : bb \to \{bb\}$ für $u = 1$, $w(e_5) = 0,1875$

und den Hypergraphen

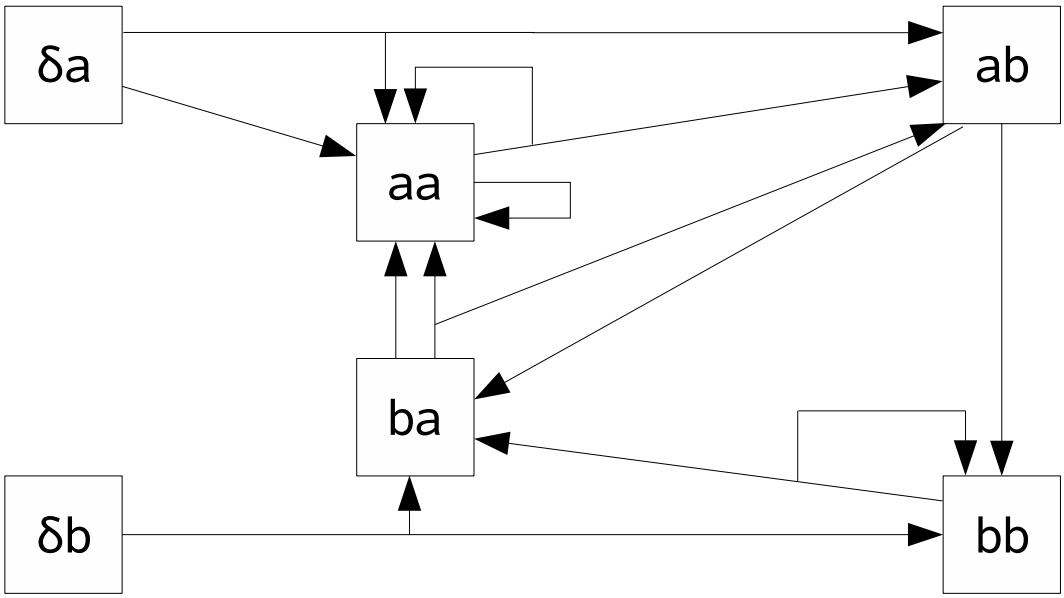

Abbildung 4.2: Hypergraph mit Vergangenheitsbezug für Beispiel 4.1

Die Längen der kürzesten Wege sind

Knoten Z	δa	δb	aa	ab	ba	bb
$\lambda(Z)$	0	∞	0	$0,15625$	0	∞

Man erhält ein besseres Ergebnis als bei der Problemstellung ohne Vergangenheitsbezug. Als nächstes wird untersucht, ob dies immer der Fall ist. Dazu wird das folgende Lemma benötigt.

Lemma 4.2. *Sei $G_1(V_1, E_1)$ der repräsentierende Graph der Problemstellung ohne Vergangenheitsbezug und $G_2(V_2, E_2)$ der der 2-stufigen Problemstellung. Sei $\hat{E}_1$ die Hyperkantenmenge eines beliebigen Weges nach X^* in $G_1(V_1, E_1)$ und $e_1 : X_i \to A_u^i$ eine beliebige Hyperkante aus $\hat{E}_1$. Dann existiert für alle $X_jX_i \in V_{X_i}$ eine Kante $e_2 : X_jX_i \to A_u^{ji} \in E_2$ mit*

- $A_u^{ji} \neq \emptyset$,
- $\widetilde{A_u^{ji}} \subseteq A_u^i$,
- $w(e_2) \leq w(e_1)$,

wobei $V_{X_i} := \{X_jX_i \in V_2 : \exists\, Weg\ von\ X_jX_i\ nach\ X_kX^*\}$ ist
und $\widetilde{A_u^{ji}} := \{X_k \in V_1 : X_iX_k \in A_u^{ji}\}$.

Beweis:
Aus $X_jX_i \in V_{X_i}$ folgt $\overline{X(Z)} \neq \emptyset$ für $Z = (X_j, X_i)$.
$\Longrightarrow$ Die Kante $e_2 : X_jX_i \rightarrow A_u^{ji} \in E_2$ existiert mit $A_u^{ji} \neq \emptyset$.
$\widetilde{A_u^{ji}} \subseteq A_u^i$ und $w(e_2) \leq w(e_1)$ folgt aus $\overline{X(Z)} \subseteq \overline{X_i}$ und der Definition der Hyperkanten. $\square$

Satz 4.3. *Sei $G_1(V_1, E_1)$ der repräsentierende Hypergraph der Problemstellung ohne Vergangenheitsbezug und $G_2(V_2, E_2)$ der der 2-stufigen Problemstellung. Es gilt dann*

$$w_{min}\big(\{X_jX_i\,; X_kX^*\}\big) \leq w_{min}\big(\{X_i\,; X^*\}\big) \qquad \forall X_jX_i \in V_{X_i}.$$

Beweis:
Die Idee ist, den Algorithmus von Dijkstra abwechselnd auf G_1 und G_2 anzuwenden, die Werte der Knoten zu vergleichen und induktiv den Satz zu beweisen.
Zuerst werden alle $\lambda(v)$ initialisiert. Man erhält für G_1 das $\lambda(X^*) = 0$ und für G_2 das $\lambda(X_kX^*) = 0$ für alle $X_k \in \mathbf{X_B}$. Alle anderen Knoten bekommen den Wert ∞ zugewiesen.

Induktionsanfang: $n = 1$

Mit Hilfe von Dijkstra wird der Knoten X_i von G_1 bestimmt, der am nächsten zu X^* liegt. Bei der letzten Verringerung von $\lambda(X_i)$ wurde die Kante $e_1 : X_i \rightarrow A_{u_1}^i$ verwendet, wobei $A_{u_1}^i = \{X^*\}$ und $\lambda(X^*) = 0$ ist. Es gilt dann $\lambda(X_i) = w(e_1)$.
Der Algorithmus wird jetzt solange auf G_2 angewendet, bis er die Länge des kürzesten Weges zu allen Knoten $X_jX_i \in V_{X_i}$ bestimmt hat.

Behauptung: $\lambda(X_jX_i) \leq \lambda(X_i) \qquad \forall X_jX_i \in V_{X_i}$

Nach Lemma 4.2 existiert für alle $X_jX_i \in V_{X_i}$ die Kante $e_2 : X_jX_i \rightarrow A_{u_1}^{ji}$ mit $A_{u_1}^{ji} \neq \emptyset$, $\widetilde{A_{u_1}^{ji}} \subseteq A_{u_1}^i$ und $w(e_2) \leq w(e_1)$.
Aus $A_{u_1}^{ji} \neq \emptyset$, $\widetilde{A_{u_1}^{ji}} \subseteq A_{u_1}^i$ folgt $A_{u_1}^{ji} = \{X_iX^*\}$ und es gilt $\lambda(X_iX^*) = 0$. Der Algorithmus hat die Kante e_2 verwendet, bevor er die Länge des kürzesten Weges von X_jX_i ermittelte, da für den Zielknoten X_iX^* der kürzeste Weg bereits bestimmt wurde.
Deswegen gilt $\lambda(X_jX_i) \leq 0 + w(e_2)$.
Aus $w(e_2) \leq w(e_1)$ folgt nun die Behauptung.

Induktionsschritt: $n \rightarrow n + 1$

Der Algorithmus von Dijkstra hat die Länge des kürzesten Weges zu n Knoten im Hypergraphen G_1 ermittelt. Diese Knoten sind in der Menge N zusammengefasst. Für G_2 ist die Länge des kürzesten Weges zu X_kX^* für die Knoten X_jX_i mit $X_i \in N$ und $X_jX_i \in V_{X_i}$

bestimmt. Für diese Knoten gilt $\lambda(X_jX_i) \leq \lambda(X_i)$.
Für den Induktionsschritt wird der Algorithmus auf G_1 angewendet, um die Länge des kürzesten Weges für den $(n+1)$-ten Knoten zu ermitteln. Es gilt $\lambda(X_{n+1}) \geq \lambda(X_i)$ für alle $X_i \in N$. Bei der letzten Verringerung des Wertes von $\lambda(X_{n+1})$ wurde die Kante $e_3 : X_i \rightarrow A^i_{u_2}$ verwendet. Es gilt

$$\lambda(X_{n+1}) = w(e_3) + \max_{Y \in A^{n+1}_{u_2}} \lambda(Y). \tag{4.4}$$

Der Algorithmus von Dijkstra wird solange auf G_2 angewendet, bis er die Länge des kürzesten Weges für alle Knoten $X_jX_{n+1} \in V_{X_{n+1}}$ nach X_kX^* bestimmt hat.

Behauptung: $\lambda(X_jX_{n+1}) \leq \lambda(X_{n+1}) \qquad \forall X_jX_{n+1} \in V_{X_{n+1}}$

Nach Lemma 4.2 existiert für alle $X_jX_{n+1} \in V_{X_{n+1}}$ die Kante $e_4 : X_jX_{n+1} \rightarrow A^{jn+1}_{u_2}$ mit $A^{jn+1}_{u_2} \neq \emptyset$, $\widetilde{A^{jn+1}_{u_2}} \subseteq A^{n+1}_{u_2}$ und $w(e_4) \leq w(e_3)$.
Aus der Induktionsvoraussetzung folgt, dass für alle Knoten in $A^{jn+1}_{u_2}$ die Länge des kürzesten Weges bereits ermittelt wurde. Der Algorithmus hat deshalb, bevor er den kürzesten Weg gefunden hat, überprüft, ob er mit der Kante e_4 den Wert von $\lambda(X_jX_{n+1})$ verringern kann. Daraus folgt

$$\lambda(X_jX_{n+1}) \leq w(e_4) + \max_{Y_1Y_2 \in A^{jn+1}_{u_2}} \lambda(Y_1Y_2) \tag{4.5}$$

Aus der Induktionsvoraussetzung und $\widetilde{A^{jn+1}_{u_2}} \subseteq A^{n+1}_{u_2}$ folgt

$$\max_{Y_1Y_2 \in A^{jn+1}_{u_2}} \lambda(Y_1Y_2) \leq \max_{Y \in A^{n+1}_{u_2}} \lambda(Y). \tag{4.6}$$

Mit $w(e_4) \leq w(e_3)$, (4.4), (4.5) und (4.6) folgt die Behauptung.

Der Fall der Nichtexistenz des Weges von X_i nach X^* in G_1 muss nicht behandelt werden, da in diesem Fall $\lambda(X_i) = \infty$ ist und für alle $X_jX_i \in V_{X_i}$ gilt, dass $\lambda(X_jX_i) < \infty$ ist. Die Ungleichung ist also immer erfüllt. □

Aufgrund der Identität von w_{min} und V, lässt sich dieses Ergebnis auf die optimale Wertefunktion übertragen.

Korollar 4.4. *Sei V_1 die optimale Wertefunktion der Problemstellung ohne Vergangenheitsbezug und V_2 die der 2-stufigen Problemstellung mit Vergangenheitsbezug. Es gilt dann*

$$V_2\big((X_j, X_i)\big) \leq V_1(X_i) \qquad \forall (X_j, X_i) \in \big(\mathbf{X_B} \cup \{\delta\}\big) \times \mathbf{X_B} \;\; \textit{mit} \;\; V_2\big((X_j, X_i)\big) < \infty.$$

Bemerkung 4.5. *Sei E_a^A die Menge aller Kanten mit dem Startknoten a und den Zielknoten A. Falls für einen Hypergraphen mit endlich vielen Knoten die Mächtigkeit jeder Menge E_a^A endlich ist, so besitzt dieser nur endlich viele Hyperkanten. Diese Eigenschaft ist garantiert, wenn alle Hyperkanten aus E_a^A zu einer mit der Länge*

$$w(a \to A) = \inf_{e \in E_a^A} w(e).$$

vereinigt werden. Durch das Zusammenfassen bleibt die Länge des Kürzesten Weges unverändert. Die Voraussetzung der Endlichkeit von U kann also aufgehoben werden. Um den Hypergraphen am Computer erstellen zu können, muss aber wieder eine endliche Menge von Testpunkten bzw. Teststeuerungen verwendet werden.

Kapitel 5

Implementierung

5.1 Allgemeines

Für die Problemstellung ohne Vergangenheitsbezug wird ein Programm vonHerrn von Lossows verwendet. Es baut auf der Implementierung aus [1] für ein ungestörtes Kontrollsystem auf. Das Programm für die 2-stufige Problemstellung benutzt als Grundlage dieses Programm.
In diesem Kapitel wird mit $G_1(V_1, E_1)$ der Hypergraph der Problemstellung ohne Vergangenheitsbezug und mit $G_2(V_2, E_2)$ der der Problemstellung mit Vergangenheitsbezug bezeichnet. Eine Beschreibung der einzelnen Dateien befindet sich im Anhang A.

Datenstruktur der Hypergraphen

Die Hypergraphen werden als verkettete Listen gespeichert. Bei jedem Knoten sind die Hyperkanten vermerkt, die auf diesen Knoten zeigen. Andererseits sind bei den Hyperkanten der Startknoten, die Zielknoten und das Gewicht der Hyperkante gespeichert. Dies reicht aus, um den Algorithmus von Dijkstra anzuwenden.

Implementierung der Zellenverwaltung

Für Zellen der Zellenverwaltung werden Rechtecke bzw. Hyperquader in höheren Dimensionen verwendet. Zur Verwaltung der Zellen wird der Ansatz aus [5] zu Grunde gelegt. Diese Datenstruktur ist speichersparend und schnell, da die einzelnen Zellen in einem binären Baum verwaltet werden. Die Implementierung im Rahmen dieser Arbeit beruht auf einer Implementierung von Herrn Prof. Grüne, die sich in [4] findet und für den graphentheoretischen Ansatz erweitert wurde.

Grundlegende Programmstruktur

Beide Programme haben zwei entscheidende Komponenten. Eine dient der Erstellung des Hypergraphen für das Kontrollsystem, die andere der Bestimmung des kürzesten Weges in diesem Hypergraphen. Beide Programme besitzen folgenden grundlegenden Aufbau:

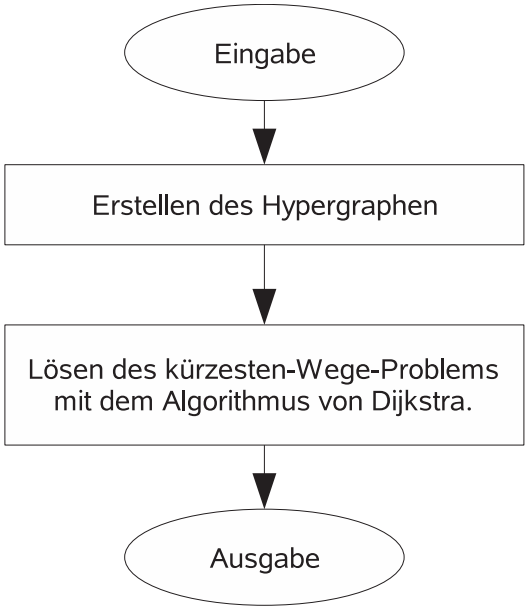

Abbildung 5.1: Grundstruktur der Programme

Implementierung des Differentialgleichungslösers

Ein zeitlich diskretes System kann als Diskretisierung einer Differentialgleichung gegeben sein. Um solche Systeme lösen zu können, ist ein Differentialgleichungslöser notwendig. Es wird ein adaptives, eingebettetes Runge-Kutta-Verfahren der Ordnung (4,5) verwendet, wie in Abschnitt 2.7 von [6] beschrieben.

Testpunkte

Bei beiden Problemstellungen wird zur Bestimmung der Länge einer Kante ein Maximum über einem abgeschlossenen Bereich gebildet. Dafür müssen unendlich viele Punkte aus dem Bereich ausgewertet werden. Ein Computerprogramm kann jedoch in endlicher Zeit nur endlich viele Punkte auswerten. Es werden deshalb Testpunktmengen $TP(X_j)$ für $X_j \in V_1$ bzw. $TP(X_iX_j)$ für $X_iX_j \in V_2$ eingeführt. Als Bewertung einer Kante $e_1 : X_j \to A_u^j \in E_1$

bzw. $e_2 : X_iX_j \to A_u^{ij} \in E_2$ erhält man dann

$$w(e_1) = \max_{x \in TP(X_j)} \hat{g}(x,u) \quad \text{bzw.} \quad w(e_2) = \max_{x \in TP(X_iX_j)} \hat{g}(x,u).$$

Die Testpunktmengen sind ebenfalls notwendig, um für alle Kanten die Mengen A_u^j bzw. A_u^{ij} zu bestimmen. Es kann nur für endlich viele Testpunkte überprüft werden, ob ein $x \in \overline{X_j}$ mit $f(x,u) \in X_k$ bzw. ein $x \in \overline{X(X_iX_j)}$ mit $f(x,u) \in X_k$ existiert und sich deshalb der Knoten X_k in A_u^i bzw. der Knoten X_jX_k in A_u^{ij} befindet.
Im Programm wird die Testpunktmenge $TP(X_iX_j)$ abhängig von der Menge $TP(X_i)$ bestimmt. Es gilt:

$$TP(X_iX_j) = \begin{cases} TP(X_j) & falls \quad X_i = \delta \\ TP2(X_i) \cap X_j & sonst \end{cases}$$

wobei

$$TP2(X_i) = \{x \in X : \exists tp \in TP(X_i) \;\; \exists\, u \in U \text{ mit } \;\; f(tp,u) = x\}.$$

In der Praxis liefert der Algorithmus wegen der Stetigkeit auch bei wenigen Testpunkten gute Ergebnisse. Dies gilt vor allem für die Problemstellung ohne Vergangenheitsbezug, wenn Randpunkte der Bereiche X_i als Testpunkte verwendet werden.

5.2 Knotenverwaltung und Knotenzugriff

Ein wesentlicher Faktor für die Geschwindigkeit und den Speicherbedarf der Programme stellt die Knotenverwaltung und die Art des Zugriffs auf einen einzelnen Knoten dar.

5.2.1 Problemstellung ohne Vergangenheitsbezug

Für jeden Knoten existiert für jede Steuerung $u \in U$ eine Kante, die den Knoten als Startknoten verwendet, d.h. alle Knoten des Hypergraphen werden benötigt. Es werden deshalb zu Beginn des Programms alle Knoten initialisiert und im Knotenarray *graph–>hk* gespeichert. Der Zugriff auf die Knoten erfolgt über die Zellenverwaltung. Für jede Zelle des Gitters existiert eine Struktur *cube*. In dieser Struktur wird unter anderem die Adresse des Knotens gespeichert, der diese Zelle repräsentiert. Über diese Adresse kann gezielt auf einen Knoten zugegriffen werden.

5.2.2 Problemstellung mit Vergangenheitsbezug

Für die Problemstellung mit Vergangenheitsbezug ist es nicht sinnvoll, alle Knoten zu speichern. Es würde dafür zuviel Speicherplatz verbraucht werden, da in der Praxis nur ein Bruchteil der Knoten benötigt wird. Bei einer etwas feineren Zerlegung von X gilt für sehr viele Knoten $X_iX_j \in V_2$, dass $\overline{X\big((X_i, X_j)\big)} = \emptyset$ ist. Für diese Knoten existieren keine Hyperkanten, die sie als Start- oder Zielknoten benutzen. Sie werden für die Berechnung des kürzesten Weges nicht benötigt.
Im folgenden Ansatz werden nur die Knoten im Knotenarray *graph–>hk* gespeichert, die als Start- oder Zielknoten einer Hyperkante verwendet werden. Zu Beginn des Programms werden die Knoten δX_i mit $X_i \in \mathbf{X_B}$ initialisiert und in *graph–>hk* gespeichert. Sie dienen alle als Startknoten für Hyperkanten. Die restlichen Knoten werden bei ihrer erstmaligen Verwendung in der Erstellung des Hypergraphen initialisiert und an das Knotenarray angehängt.

Der Zugriff auf einen Knoten wird dadurch komplizierter. Beim Anhängen bzw. beim Reservieren des Speicherplatzes für einen neuen Knoten kann sich die Adresse von *graph–>hk* und somit die Adressen der einzelnen Knoten ändern. Man muss deshalb über die Position des Knotens im Knotenarray und nicht über seine Adresse auf ihn zugreifen.
Zur Bestimmung der Position eines Knotens in *graph–>hk* wird die Zellenverwaltung benutzt. Wie bei der Problemstellung ohne Vergangenheitsbezug, gibt es für jede Zelle des Gitters eine Struktur *cube*. In ihr ist unter anderem der Index *graph_index* und das Array *nachfolger* gespeichert.
In *graph_index* ist vermerkt, welchen Bereich X_i von X die Zelle repräsentiert. Mit *zugriff(graph_index)* kann direkt auf diese Zelle zugegriffen werden. Jeder Eintrag in *nachfolger* besteht aus den *int*-Werten *pos* und *nachfolger_index*. In *pos* wird die Position des Knotens im Knotenarray gespeichert, auf welchen der *nachfolger_index* verweist. Um einen Eintrag

in *nachfolger* schneller zu finden, sind alle nach dem *nachfolger_index* sortiert.
Wird die Position des Knotens X_iX_j im Knotenarray gesucht, muss man die Zelle *zugriff[i]* betrachten und in dessen *nachfolger* den Eintrag suchen, bei dem der *nachfolger_index* gleich j ist. In *pos* dieses Eintrages ist die Position des Knotens in *graph–>hk* gespeichert.
Bei den Hyperkanten speichert man gleich die Position der Start- und Zielknoten im Knotenarray und nicht wie bei der Problemstellung ohne Vergangenheitsbezug die Adressen der Knoten.

Dieser Ansatz braucht etwas mehr Prozessorzeit, dafür aber deutlich weniger Speicherplatz als der Ansatz der Problemstellung ohne Vergangenheitsbezug. Die zusätzliche Prozessorzeit entsteht durch den sortierten Aufbau von *nachfolger* und durch das Suchen nach Einträgen in *nachfolger*. Bei mehrdimensionalen Beispielen oder bei feineren Zerlegungen ist jedoch der vorhandene Arbeitsspeicher der entscheidende Engpass. Sobald der Arbeitsspeicher für das Programm nicht mehr ausreicht, müssen Daten auf die Festplatte ausgelagert werden. Die Auslagerung der Daten verbraucht aber deutlich mehr Zeit, als für den Aufbau und das Suchen in *nachfolger* benötigt wird. In den Kapiteln 1.6 und 6.3.11 in [11] werden die Zugriffszeiten auf unterschiedliche Speichermedien und die Benutzung von virtuellem Speicherplatz erläutert.

5.3 Das Erstellen der Hyperkanten

Als erstes wird die **Problemstellung ohne Vergangenheitsbezug** betrachtet. Die Ausgangslage ist der Graph $G_1(V_1, E_1)$, wobei $V_1 = \{X_i \in \mathbf{X_B}\}$ und $E_1 = \emptyset$.

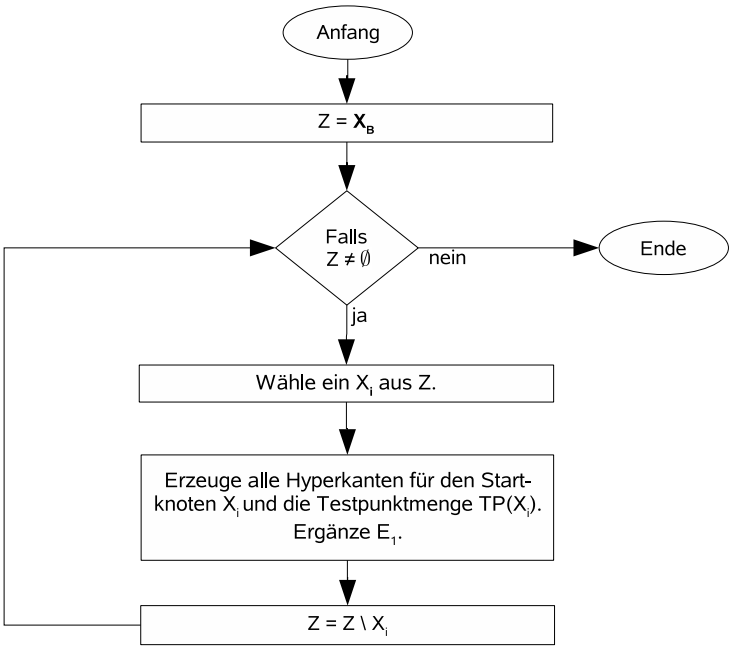

Abbildung 5.2: Flussdiagramm der Hyperkantenerstellung für die Problemstellung ohne Vergangenheitsbezug

Für jeden Knoten wird für jede Steuerung eine Hyperkante erzeugt. Diese wird zur Hyperkantenmenge E_1 hinzugefügt. $G_1(V_1, E_1)$ ist komplett, nachdem alle Bereiche abgearbeitet worden sind. Anschließend kann mit dem Algorithmus von Dijkstra der kürzeste Weg berechnet werden.

Die Ausgangslage für die **Problemstellung mit Vergangenheitsbezug** ist der Hypergraph $G_2(V_2, E_2)$, wobei $V_2 = (\delta X_i : X_i \in \mathbf{X_B})$ ist und $E_2 = \emptyset$.
Für den Knoten X_iX_j wird keine Hyperkante erzeugt, falls $TP2(X_i) \cap X_j = \emptyset$. Andernfalls wird für jede Steuerung eine Hyperkante $e : a \rightarrow A$ erstellt und zu E_2 hinzugefügt. Außerdem wird die Knotenmenge V_2 durch die Menge $\{X_iX_j \in A : X_iX_j \notin V_2\}$ ergänzt.
In der Implementierung werden die Mengen $TP(X_iX_j) = TP2(X_i) \cap X_j$ in dem Array *punkteliste* der Zelle *zugriff(j)* gespeichert. Nachdem der Bereich X_i abgearbeitet worden ist, werden die Einträge in *punktliste* nicht mehr benötigt. Sie können für den nächsten abzuarbeitenden Bereich überschrieben werden.

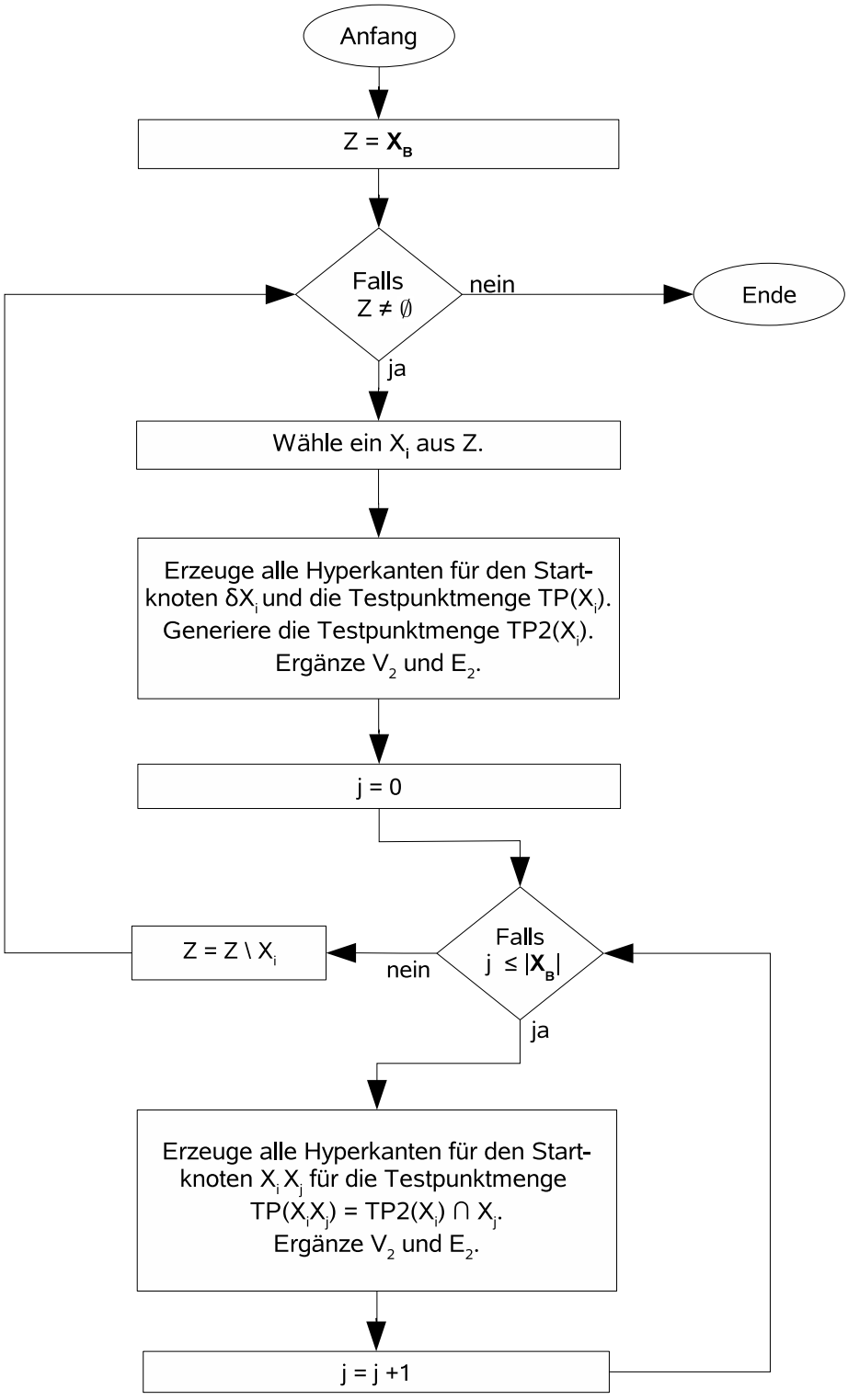

Abbildung 5.3: Flussdiagramm der Hyperkantenerstellung für die Problemstellung mit Vergangenheitsbezug

5.4 Dokumentation der einzelnen Funktionen

Zuerst werden alle Funktionen betrachtet, die für die Problemstellung ohne und mit Vergangenheitsbezug benötigt werden. Dabei werden zunächst die Funktionen aus der Implementierung aus [4] beschrieben.

- `cube* get_zelle(void);`

  Gibt die Adresse der aktuellen Zelle zurück bzw. NULL, falls der interne Zeiger nicht gesetzt wurde.

- `cube* get_x_zelle(real *x);`

  Gibt die Adresse der Zelle zurück, in der der Punkt x liegt bzw. NULL, falls x nicht in Ω liegt (Ω entspricht der Menge X).

Die Dokumentation der folgenden Funktionen ist aus dem Anhang von [4] übernommen.

- `double make_grid(double *xu, double *xo, int dim, int r);`

  Erzeugt eine Zellenüberdeckung im $\mathbb{R}^{dim}$ auf der rechteckigen Menge Ω mit "linker unterer" Ecke xu und "rechter oberer" Ecke xo. In jeder Koordinatenrichtung werden r Zellen erzeugt, also $r \cdot dim$ Zellen insgesamt. Falls r keine Zweierpotenz ist wird automatisch abgerundet. Als Rückgabewert gibt die Funktion den Durchmesser der Zellen (in der 2-Norm) zurück. Jede Zelle wird mit Status=2 vorbelegt.

  Beispiel (2d):

  ```
  double k, xu[2], xo[2]; xu[0] = -1; xu[1] = -1; xo[0] = 1; xo[1] =
  1; k=make_grid(xu,xo,2,4);
  ```

- `int write_grid(char *filename);`

  Schreibt die aktuelle Zellenüberdeckung in die Datei mit Namen `filename`. Für jede Zelle wird eine Zeile mit linker unterer und rechter oberer Ecke sowie dem Status ausgegeben.

  Beispiel:

  ```
  write_grid("test.grd");
  ```

- `void refine_grid(void);`

  Verfeinert jede Zelle des Gitters mit Status=2. Hierbei wird immer in eine Koordinatenrichtung verfeinert, wobei sich die Richtungen zyklisch abwechseln, in 3d also x - y - z - x - y - z - ...

- `int first_cell(void);`

  Setzt den internen Zeiger auf die erste Zelle der Überdeckung. Gibt -1 zurück, falls noch kein Gitter erzeugt wurde, sonst 0. Die Zelle, auf die der interne Zeiger zeigt, wird im Folgenden als aktuelle Zelle bezeichnet.

- `int next_cell(void);`

  Setzt den internen Zeiger auf die nächste Zelle der Überdeckung. Gibt eine fortlaufende positive Nummer (Index der Zelle) zurück und -1, falls die letzte Zelle der Überdeckung bereits erreicht war.

  Beispiel für `first_cell` und `next_cell`: Schleife über alle Zellen

  ```
  int index; ... index = first_cell(); do { ... /* Hier kann die
  aktuelle Zelle bearbeitet werden */ ... index = next_cell(); } while
  (index!=-1);
  ```

- `void get_corner(int i, double *x);`

  Gibt den i-ten Eckpunkt der aktuellen Zelle (siehe `first_cell`) aus. Der Index i muss zwischen 0 und $2^{dim} - 1$ liegen, die Variable x muss entsprechend der Dimension deklariert sein.

  Beispiel (2d):

  ```
  double x[2];
  int i; ... for (i=0; i<4; i++) { get_corner(i, x);
  printf("Eckpunkt %d: %f %f\n", i, x[0], x[1]);
  }
  ```

- `void get_testpoint(int i, int tpnum, double *x);`

  Gibt den i-ten Testpunkt (von insgesamt $tpnum$) in der aktuellen Zelle aus. Der Index i muss zwischen 0 und $tpnum-1$ liegen, die Variable x muss entsprechend der Dimension deklariert sein. Die Testpunktanzahl muss $\geq 2^{dim}$ sein und sollte von der Form m^{dim} für ein $m \in \mathbb{N}$ sein, falls dies nicht der Fall ist, wird automatisch intern abgerundet. Die Testpunkte werden gleichmäßig in der Zelle platziert; dabei sind die Eckpunkte immer Testpunkte.

  Beispiel (2d):

```
double x[2]; int i,j; ... j = 9; for(i=0; i<j; i++) {
get_testpoint(i,j,x);
printf("Testpunkt %d: %f %f\n", i, x[0], x[1]);
}
```

- `int get_status(void);`

  Gibt den Status der aktuellen Zelle zurück bzw. -1, falls der interne Zeiger nicht gesetzt wurde.

- `int set_status(int status);`

  Setzt den Status der aktuellen Zelle auf *status*; dieser Wert muss dabei ≥ 0 sein. Gibt bei Erfolg 0 zurück, sonst -1.

- `int get_x_status(double *x);`

  Gibt den Status der Zelle zurück, in der der Punkt x liegt bzw. -1, falls x nicht in Ω liegt oder keine Überdeckung erzeugt wurde.

- `int set_x_status(double *x, int status);`

  Setzt den Status der Zelle, in der x liegt auf *status*; dieser Wert muss dabei ≥ 0 sein. Gibt bei Erfolg 0 zurück, sonst -1.

Die folgenden Funktionen werden ebenfalls für beide Problemstellungen verwendet. Wegen der unterschiedlichen Problemstellungen und Knotenverwaltungen existieren jedoch teils erhebliche Unterschiede in der Implementierung der Funktionen. Die Aufgaben der Funktionen sind jedoch identisch.

- `hgraph* neuer_graph(int anz);`

  Legt einen neuen Hypergraph *hgraph* mit *anz* Knoten an. In ihm existieren noch keine Hyperkanten.

- `hkante* neue_kante(hknoten *start, double gewicht);`

  Initialisiert eine neue Hyperkante *hkante* mit der Länge *gewicht* und dem Startknoten *start*. Die Zielknoten müssen im Programm nach dem Aufruf der Funktion noch bei der Hyperkante gespeichert werden.

- `void fuege_hkante_ein(hgraph *graph, hkante *hypkante);`

  Fügt die Hyperkante *hypkante* in den Hypergraphen *graph* ein. Bei der Problemstellung ohne Vergangenheitsbezug wird *graph* nicht mit übergeben.

- `void dijkstra(hgraph *graph, int anz_D, hknoten **D);`

  Berechnet mit dem Algorithmus von Dijkstra die kürzesten Wege für den Hypergraphen *graph* zu den *anz_D* Zielknoten, die in D gespeichert sind.

- `void f(double *x, double *y, double *u);`

  Berechnet den Wert der Übergangsfunktion für den Zustand x und die Steuerung u. Der neue Zustand des Systems wird in y gespeichert. Diese Funktion muss für jedes Problem vom Benutzer zur Verfügung gestellt werden.

- `void g(double *x, double *u);`

  Berechnet die Kosten, die beim Zustand x für die Steuerung u entstehen. Diese Funktion muss für jedes Problem vom Benutzer zur Verfügung gestellt werden.

- `void erzeuge_hkanten_ungewissheit_in_zelle(hgraph *graph, int dim_x, int anzahl_u, double **U, int tp, void f(...), double g(...), cube** zugriff);`

  Erstellt die Hyperkanten des Hypergraphen *graph* für das Kontrollsystem mit der Übergangsfunktion f und der Kostenfunktion g. Die Hyperkanten werden für die *anzahl_u* Steuerungen, die in U gespeichert sind und für tp Testpunkte pro Knoten erzeugt. *dim_x* gibt die Dimension von x an. Das Adressenarray $zugriff$ wird nur für die Problemstellung mit Vergangenheitsbezug benötigt. Mit Hilfe von $zugriff$ kann man direkt auf die einzelnen Zellen zugreifen, z.B. mit $zugriff[i]$ erhält man die Adresse der Zelle, die den Bereich X_i repräsentiert.

- `void ausgabe_wertefunktion(int dim, char *dname, hgraph *graph, cube **zugriff);`

  Speichert für die Problemstellung ohne Vergangenheitsbezug in der Datei *dname* die optimale Wertefunktion V. Dabei wird für jede Zelle eine Zeile mit der linken oberen Ecke, der rechten unteren Ecke und dem Wert der Wertefunktion auf der Zelle ausgegeben.
  Für die Problemstellung mit Vergangenheitsbezug kann einer Zelle X_i kein eindeutiger Wert zugeordnet werden, da jeder Knoten $X_jX_i \in V_2$ einen anderen Wert haben kann. Es wird deshalb für die Zelle X_i

  $$\min_{X_jX_i \in V_2} \lambda(X_jX_i)$$

  in *min_dname*,

  $$\max_{X_jX_i \in V_{X_i}} \lambda(X_jX_i)$$

in *max_dname* und

$$V(\delta X_i)$$

in *dleta_dname* anstelle von V als Wert ausgegeben.
Diese verkürzten Wertefunktionen können für die graphische Darstellung der optimalen Wertefunktion verwendet werden. Mit ihnen können beide Problemstellungen besser verglichen werden.
Bei der Problemstellung ohne Vergangenheitsbezug wird *graph* und $zugriff$ nicht mit übergeben.

- `void ausgabe_trajektorie(hgraph *graph, int k,char *dname, double *x, int dim_x, anzahl_u, double **U, void f(...),double g(...));`

  Speichert in der Datei *dname* die Trajektorie mit dem Startwert x für eine optimale Steuerungssequenz, die sich aus der optimalen Wertefunktion ergibt. Die Trajektorie wird maximal für k Schritte berechnet. Dabei wird für jeden Schritt der aktuelle Punkt und der Wert der optimalen Wertefunktion in diesem Punkt in dname in einer Zeile gespeichert.

- `double* speicheranfordern(int i);`

  Reserviert den Speicherplatz für ein Array mit *double*-Einträgen der Länge i.

- `double** speicheranfordern_m(int i);`

  Reserviert für ein zweidimensionales Array mit *double*-Einträgen den Speicherplatz für die i Zeiger der äußeren Dimension des Arrays.

Die nächste Funktion wird nur für die Problemstellung ohne Vergangenheitsbezug benötigt.

- `void verbinde_gitter_graph(hgraph *graph);`

  Speichert in jeder Zelle des Gitters die Adresse des Knotens, der von der Zelle repräsentiert wird.

Als letztes werden die Funktionen betrachtet, die nur bei der Problemstellung mit Vergangenheitsbezug verwendet werden.

- `void erstindizierung(void);`

  Speichert für jede Zelle den Index *graph_index*, der den Bereich spezifiziert, den die Zelle repräsentiert.

- `void verbinde_gitter_zugriff(cube **zugriff);`

  Belegt das Array $zugriff$ mit den Adressen der Zellen. In $zugriff[i]$ wird die Adresse der Zelle gespeichert, die den Bereich X_i vertritt.
  Mit dieser Funktion wird außerdem noch der Speicherplatz für das Array $nachfolger$ und die *punkteliste* in jeder Zelle reserviert. Die *punkteliste* dient zum Speichern der Testpunktmengen.

- `int get_max_index(void);`

  Gibt die Anzahl der Zellen des Gitters zurück.

- `void knotenanhaengen(hgraph *graph);`

  Hängt einen Knoten an *graph* an und initialisiert diesen.

- `int knotensuche(hgraph *graph, cube **zugriff, int vor, int nach, int ein);`

  Sucht den Knoten$X_{vor}X_{nach}$ im Knotenarray *graph–>hk*. Die Position des Knotens im Knotenarray wird zurückgegeben. Falls der Knoten nicht gefunden wird, bestimmt der Parameter *ein* das Verhalten der Funktion. Für den Wert 1 wird ein neuer Knoten an das Knotenarray angehängt und die Position des eingefügten Knotens zurückgegeben. Für alle anderen Werte von *ein* ist der Rückgabewert der Funktion -1.

- `int ergaenze_D(hgraph *graph, cube *einfuegen, hknoten **D, int index_D, cube **zugriff);`

  Der Bereich X_j, den die Zelle *einfuegen* repräsentiert, soll zum Gleichgewichtsbereich hinzugefügt werden. Durch Betrachtung des Status der Zelle wird überprüft, ob der Bereich bereits hinzugefügt worden ist. Wenn nicht, werden im Array D, vom Index *index_D* an, alle Knoten der Form X_iX_j aus *graph–>hk* gespeichert. Außerdem wird im Status der Zelle vermerkt, dass der Bereich abgearbeitet ist. Die Funktion gibt den nächsten unbenutzten Index von D zurück.

Kapitel 6

Numerische Untersuchung von Beispielen

In diesem Abschnitt werden zwei Beispiele betrachtet. Die ungestörten Kontrollsysteme, die als Grundlage für die gestörten dienen, wurden bereits in [1] untersucht. V_1 bezeichne die optimale Wertefunktion der Problemstellung ohne Vergangenheitsbezug. Zur Darstellung der optimalen Wertefunktion V_2 der Problemstellung mit Vergangenheitsbezug werden drei Funktionen eingeführt

$$\begin{aligned} V_\delta(X_i) &:= V_2\big((\delta, X_i)\big) && X_i \in \mathbf{X_B} \\ V_{\min}(X_i) &:= \min_{i\in\{0,\dots,l\}} V_2\big((X_j, X_i)\big) && X_i \in \mathbf{X_B} \\ V_{\max}(X_i) &:= \max_{(X_j,X_i)\in V_{X_i}} V_2\big((X_j, X_i)\big) && X_i \in \mathbf{X_B}, \end{aligned}$$

mit $V_{X_i} := \Big\{(X_i, X_j) \in \mathbf{X_B} \cup \{\delta\} \times \mathbf{X_B} \text{ mit } V_2\big((X_j, X_i)\big) < \infty\Big\}$.
Normalerweise ist beim Start einer Trajektorie der vorherige Zustand unbekannt. Es ist also sinnvoll, zum Vergleich mit V_1 V_δ zu nehmen. $V_{\min}$ und $V_{\max}$ dienen dazu, ein Gefühl für die die Werte von V_2 zu bekommen und Satz 4.2 zu veranschaulichen.
Die Berechnung der optimalen Wertefunktionen ist hinsichtlich der Feinheit des Gitters keine Approximation. Es werden nur die einzelnen Zustände durch Testpunkte und die Steuerungsmenge durch Teststeuerungen approximiert.

6.1 Ein einfaches ein-dimensionales Beispiel

Das ungestörte System

$$x_{k+1} = x_k + (1-a)u_k x_k, \quad k = 0, 1, \dots, \tag{6.1}$$

mit der Kostenfunktion

$$g(x, u) = (1-a)x,$$

wobei $x \in X = [0,1], u_k \in U = [-1,1]$ und $a \in (0,1)$ ein fester Parameter ist, dient als Grundlage des gestörten Kontrollsystems.
Da $g(x,u)$ nicht von u abhängt, ist es am besten, das ungestörte System möglichst schnell ins Gleichgewicht bei 0 zu steuern. Dies erreicht man durch die konstante Kontrollfolge $\mathbf{u} = (-1,-1,-1,\ldots)$.
Mit dieser Kontrollfolge erhält man

$$x_{k+1} = x_k + (1-a)(-1)x_k = x_k - x_k + ax_k = ax_k$$

und induktiv

$$x_k = a^k x_0.$$

Somit gilt

$$V(x) = \sum_{k=0}^{\infty} g(x_k, -1) = \sum_{k=0}^{\infty} (1-a)a^k x = x(1-a)\sum_{k=0}^{\infty} a^k = x(1-a)\frac{1}{1-a} = x$$

Für die Berechnungen werden 10 Testpunkte pro Zelle verwendet, U wird durch 11 äquidistante Punkte approximiert und als Parameter a wird 0.8 gewählt. Der Bereich wird immer in ein gleichmäßiges Gitter zerlegt. Bei allen Abbildungen zu diesem Beispiel wird zum Vergleich in schwarz die optimale Wertefunktion des ungestörten Systems eingezeichnet.

In den nächsten drei Abbildungen wird $V_{\min}$ (rot) und $V_{\max}$ (blau) verglichen.

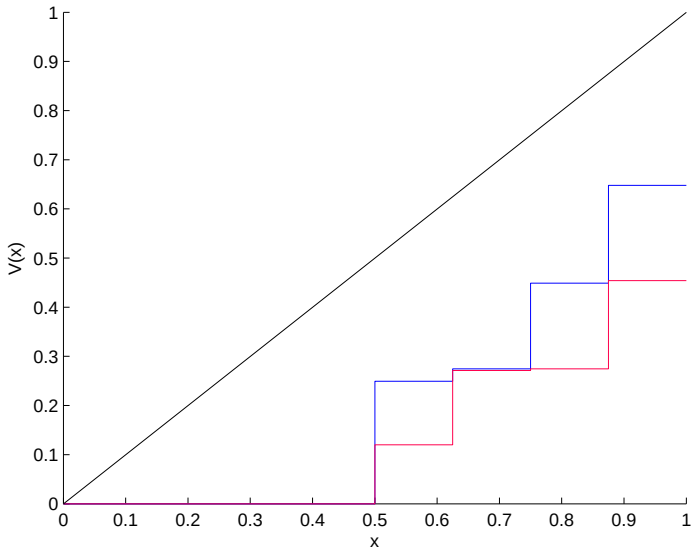

Abbildung 6.1: Wertefunktionen für ein Gitter mit 8 Zellen

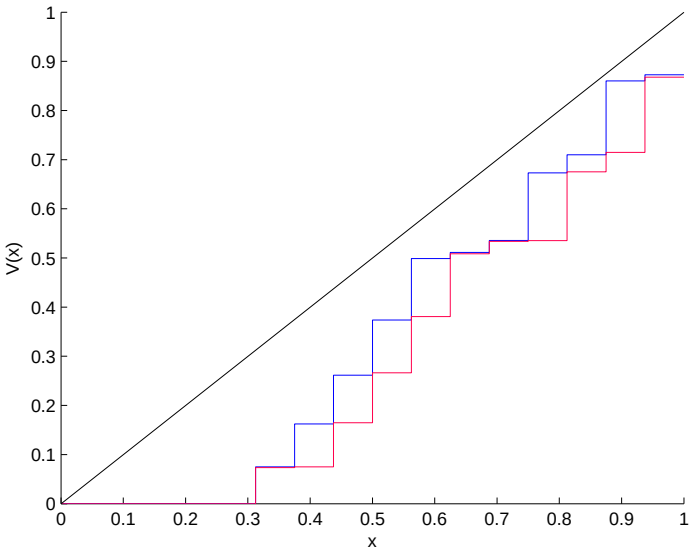

Abbildung 6.2: Wertefunktionen für ein Gitter mit 16 Zellen

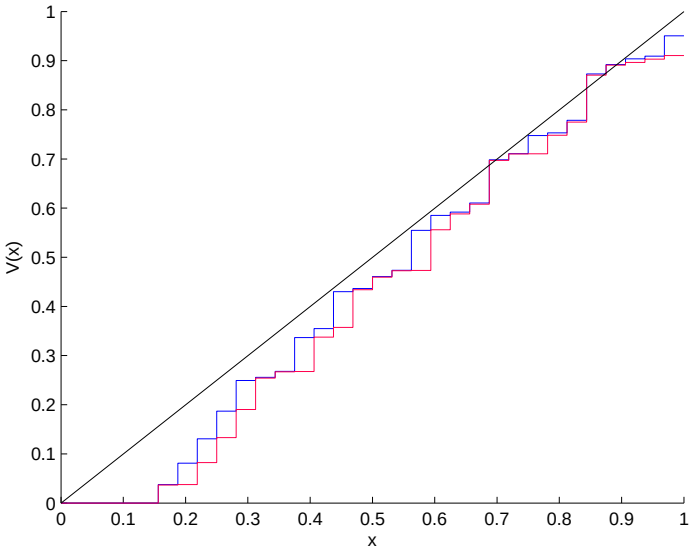

Abbildung 6.3: Wertefunktionen für ein Gitter mit 32 Zellen

Die Funktion V_δ stimmt in diesem Beispiel für alle untersuchten Zerlegungen mit $V_{\max}$ überein.

Damit man sinnvolle Ergebnisse erhält, ist es notwendig den Gleichgewichtsbereich groß genug zu wählen. Ist er zu klein, gilt fast immer, dass $V_1(X_i) = \infty$ bzw. $V_2\big((X_i, X_j)\big) = \infty$ ist. Für V_1 muss man mindesten 5 und für V_2 mindesten 4 Zellen als Gleichgewichtsbereich wählen. In diesem Punkt ist die Problemstellung mit Vergangenheitsbezug der anderen überlegen.
Für die Zerlegung mit 8 Zellen hat V_1 außerhalb des Gleichgewichtsbereichs den Wert unendlich. Für V_2 erhält man bereits bei einer gröberen Zerlegung ein nutzbares Ergebnis.

In den nächsten Abbildungen wird V_1 (rot) mit V_δ (blau) verglichen, wobei für den Gleichgewichtsbereich immer 5 Zellen gewählt wurden.

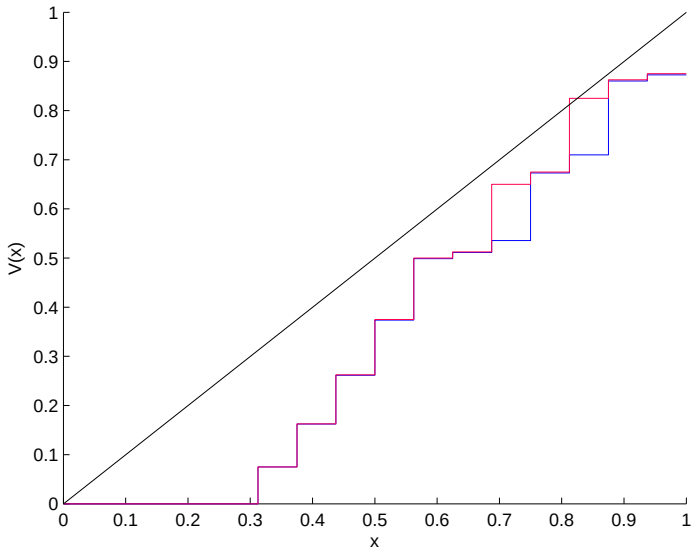

Abbildung 6.4: Wertefunktionen für ein Gitter mit 16 Zellen

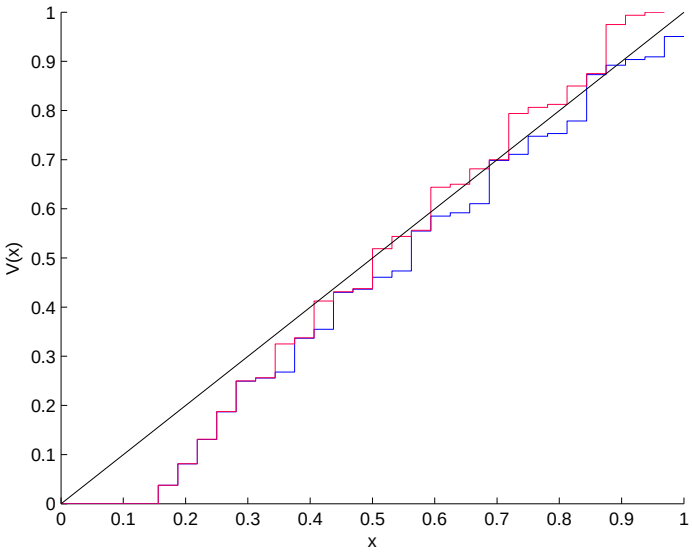

Abbildung 6.5: Wertefunktionen für ein Gitter mit 32 Zellen

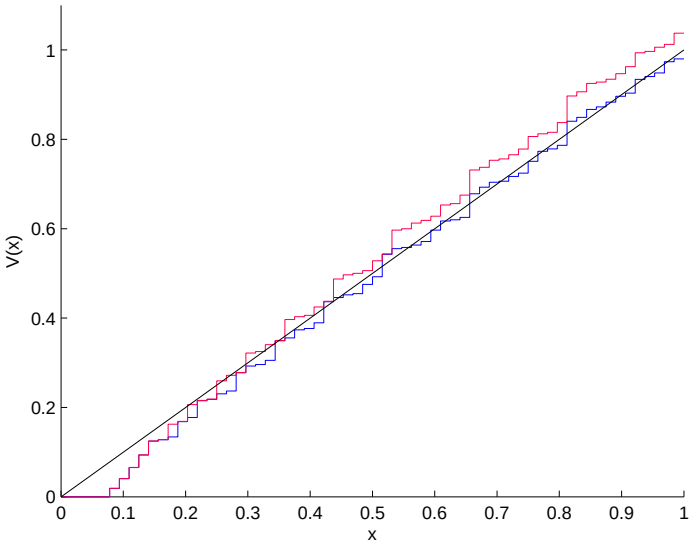

Abbildung 6.6: Wertefunktionen für ein Gitter mit 64 Zellen

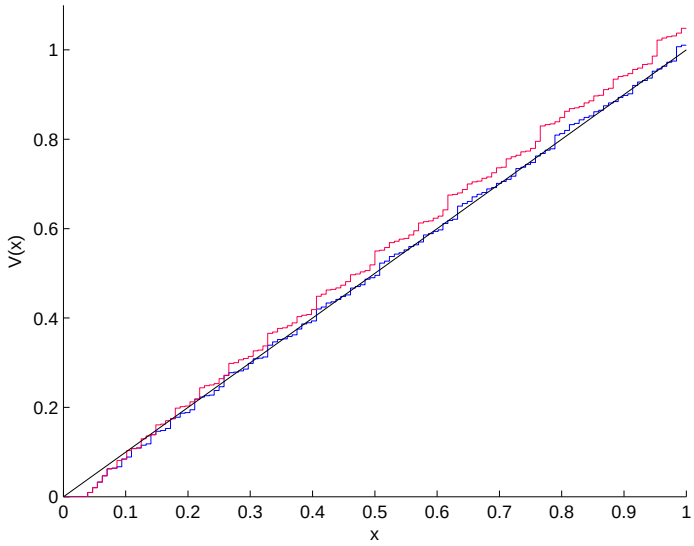

Abbildung 6.7: Wertefunktionen für ein Gitter mit 128 Zellen

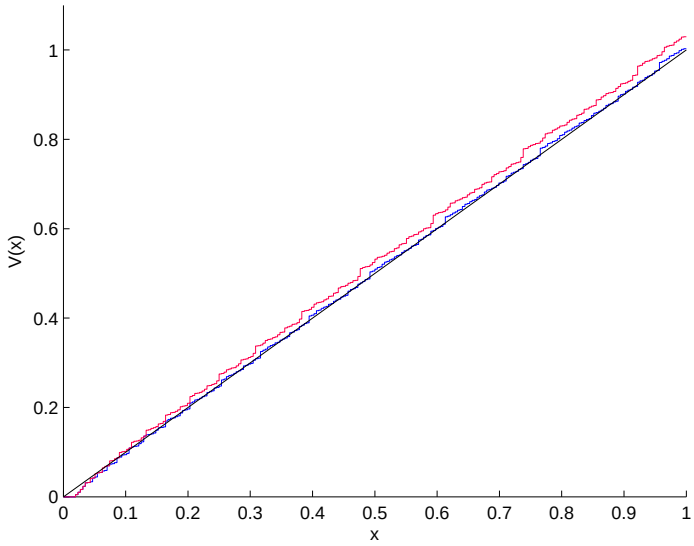

Abbildung 6.8: Wertefunktionen für ein Gitter mit 256 Zellen

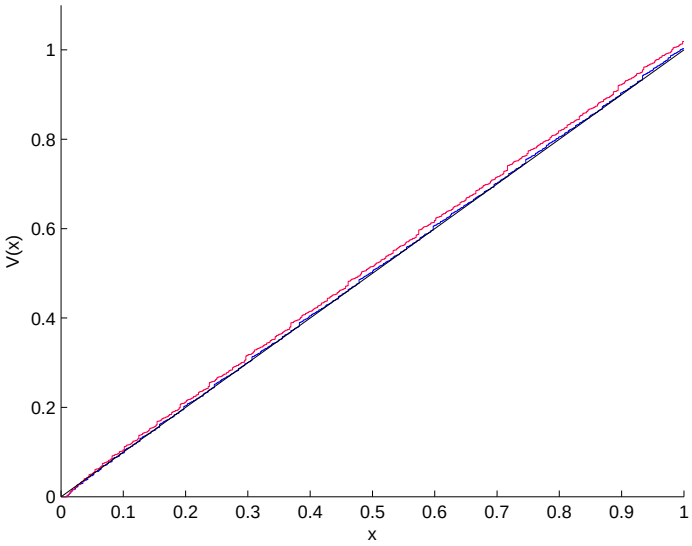

Abbildung 6.9: Wertefunktionen für ein Gitter mit 512 Zellen

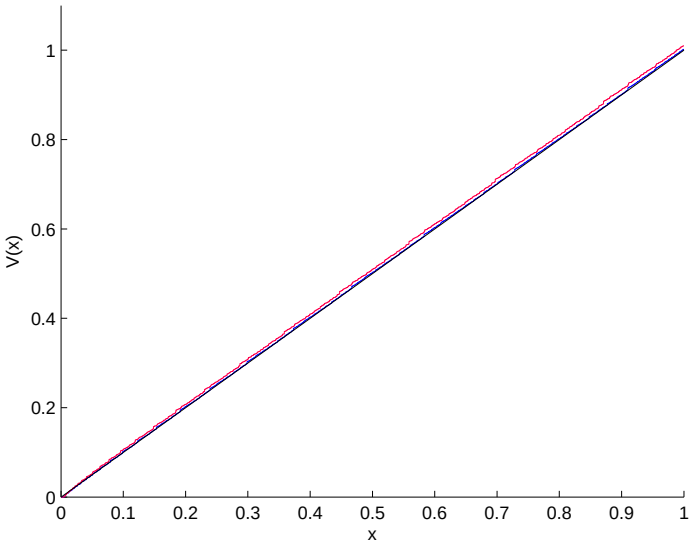

Abbildung 6.10: Wertefunktionen für ein Gitter mit 1024 Zellen

In allen Abbildungen liegt V_1 immer oberhalb von V_δ. Je feiner das Gitter gewählt wird, desto geringer wird die Störung des Systems. Man sieht deshalb, dass sich V_1 und V_δ mit zunehmender Feinheit immer weiter an die optimale Wertefunktion V des ungestörten Systems annähern. Durch die Ungewissheit im System müssten V_1 und V_δ durchgehend größere Werte als V haben. Dies ist nicht der Fall, da die gestörten Systeme einen Gleichgewichtsbereich und nicht, wie das ungestörte System, nur einen Gleichgewichtspunkt besitzen. Im Gleichgewichtsbereich bekommen, bei der Berechnung der optimalen Wertefunktion, alle Zustände den Wert 0 zugewiesen. Deshalb befinden sich V_1 und V_δ anfangs unterhalb von V. Erst mit der Zeit nehmen sie größere Werte als V an.

6.2 Das Pendelmodell

Als nächstes Beispiel wird ein Pendelmodell angeführt, mit dem der ungestörten Fall z.B. in [1] und [8] untersucht wurde. Ein gestörte Version des Pendelmodells wurde beispielsweise in [2] und [3] betrachtet.
Zuerst wird das ungestörte System beschrieben. Ein Wagen, auf dem ein umgedrehtes starres Pendel befestigt ist, kann beschleunigt oder abgebremst werden. Dies stellt die Kontrolle des Systems dar. Das Ziel ist es, durch günstiges Beschleunigen und Abbremsen das Pendel aufzurichten und in der Senkrechten zu halten. Daraus ergibt sich folgendes Kontrollsystem:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{\frac{g}{l}\sin(x_1) - \frac{1}{2}m_r x_2^2 \sin(2x_1) - \frac{m_r}{ml}\cos(x_1)u}{\frac{4}{3} - m_r\cos^2(x_1)} \end{aligned} \tag{6.2}$$

Hierbei stellt M die Masse des Wagens, m die Masse des Pendels, $m_r = m/(m+M)$ das Massenverhältnis, l die Entfernung des Schwerpunktes des Pendels vom Wagen und g=Erdanziehungskraft dar. x_1 entspricht dem Winkel des Pendels zur Senkrechten und x_2 der Winkelgeschwindigkeit. Die Position und Geschwindigkeit des Wagens wird hier nicht betrachtet.
Als Kostenfunktion wird

$$q(x,u) = \frac{1}{2}(0.1x_1^2 + 0.05x_2^2 + 0.01u^2) \tag{6.3}$$

gewählt.
Als Parameter werden $M = 8$, $m = 2$, $l = 0.5$ und $g = 9.8$ verwendet.
Durch

$$f(x,u) = \phi^T(x;u) \tag{6.4}$$

erhält man ein zeitdiskretes Kontrollsystem, dabei bezeichnet $\phi^t(x;u)$ die Lösung von (6.2) mit Anfangswert x und konstanter Steuerung u zur Zeit t.
Die Kostenfunktion $g(x,u)$ ist durch

$$g(x,u) = \int_0^T q(\phi^t(x;u),u)dt \tag{6.5}$$

gegeben.
Dieses System ist die Grundlage des im Folgenden untersuchten gestörten, diskreten Kontrollsystems.

Das System wird für die Parameter $T = 0.1$, $X = [-8, 8] \times [-10, 10]$ und
$U = \{-64, -56, \ldots, -8, 0, 8, \ldots, 56, 64\}$ untersucht. Für alle Berechnungen werden 9 Testpunkte pro Zelle verwendet.

Die Farben in den nachfolgenden Abbildungen stellen den Wert der berechneten Wertefunktion dar. Die Werte steigen von Blau zu Rot an. Falls V den Wert unendlich hat, wird die Zelle weiß dargestellt.
Bei gleichmäßiger Verfeinerung des Gitters erhält man für V_1 und V_δ folgende Approximationen.

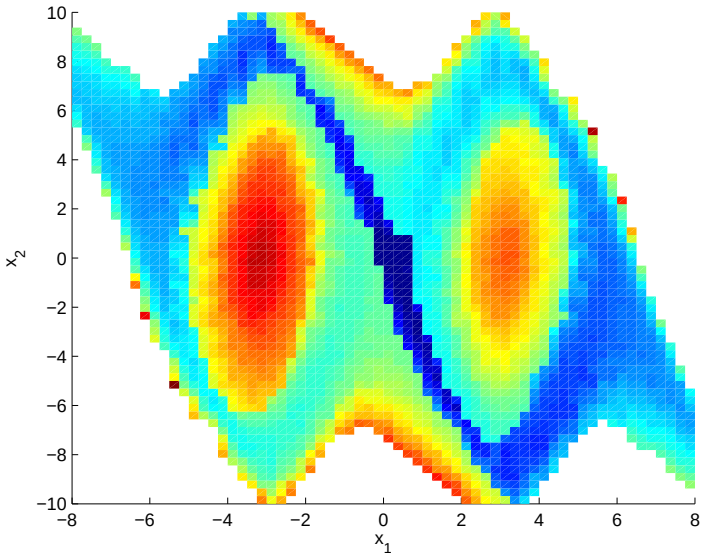

Abbildung 6.11: V_δ für 64 mal 64 Zellen

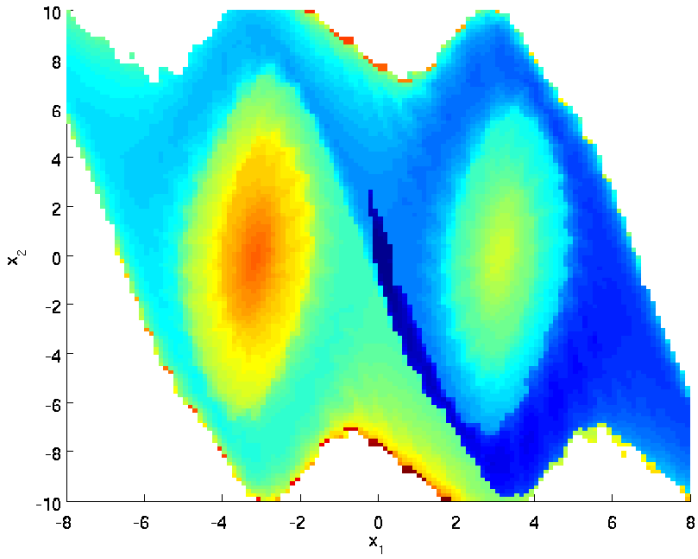

Abbildung 6.12: V_1 für 128 mal 128 Zellen

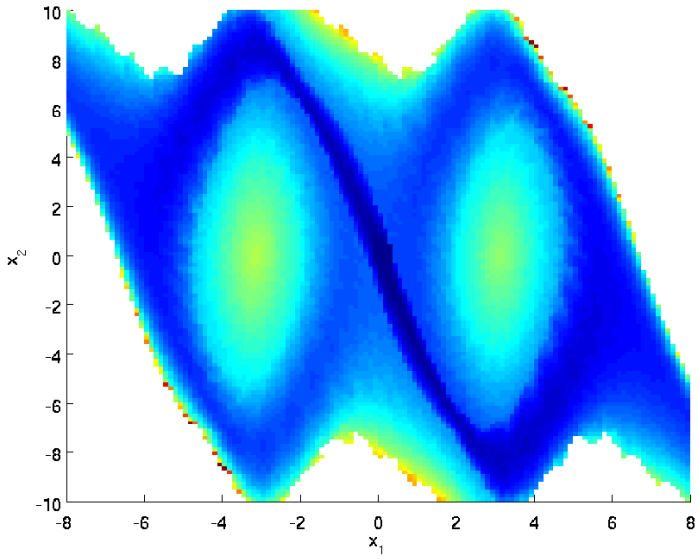

Abbildung 6.13: V_δ für 128 mal 128 Zellen

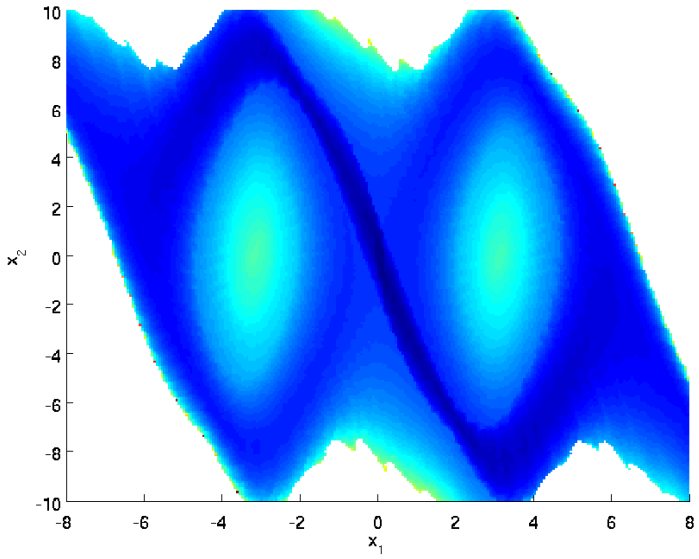

Abbildung 6.14: V_1 für 256 mal 256 Zellen

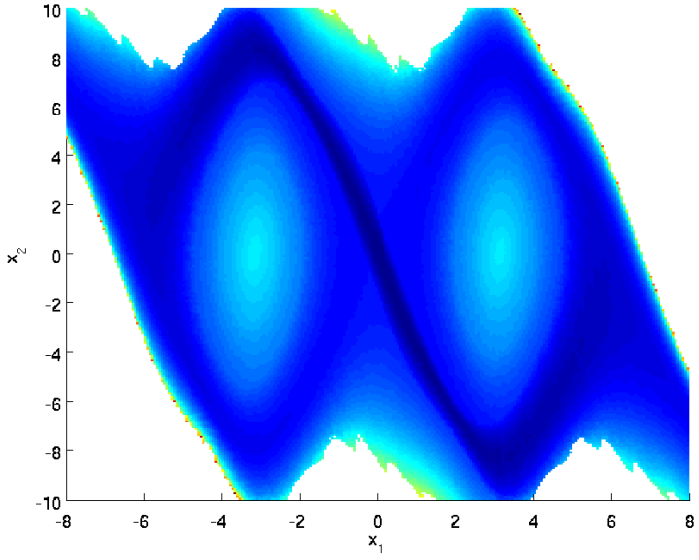

Abbildung 6.15: V_δ für 256 mal 256 Zellen

Für gröbere Zerlegungen nimmt die optimale Wertefunktion außerhalb des Gleichgewichtsbereichs für fast alle Zellen den Wert unendlich an. Deshalb werden für sie keine Abbildungen präsentiert. Für die Problemstellung mit Vergangenheitsbezug erhält man schon für eine gröbere Zerlegung als für die ohne Vergangenheitsbezug ein sinnvolles Ergebnis. Das System ist bereits für eine größere Störung des Ausgangswertes stabilisierbar.

Als Nächstes werden Trajektorien für das Pendelmodell betrachtet. Eine Trajektorie die in x startet wird durch

$$\begin{aligned} x &= x_0 \\ x_{i+1} &= f(x_i, u_i) \end{aligned}$$

bestimmt, wobei u_i die Steuerung ist, für die der Ausdruck

$$g(x_i, u) + V_1\big(\rho(f(x_i, u))\big)$$

bei der Problemstellung ohne Vergangenheitsbezug, bzw.

$$g(x_i, u) + V_2\Big(\big(\rho(x_i), \rho(f(x_i, u))\big)\Big)$$

bei der Problemstellung mit Vergangenheitsbezug seinen minimalen Wert annimmt.

Auf der linken Seite werden jeweils die Trajektorien für die Startwerte $\begin{pmatrix} 3,1 \\ 0,1 \end{pmatrix}$ und $\begin{pmatrix} -3 \\ 0,1 \end{pmatrix}$ abgebildet. Auf der rechten Seite wird der Verlauf der optimalen Wertefunktion entlang der Trajektorie mit dem Startwert $\begin{pmatrix} 3,1 \\ 0,1 \end{pmatrix}$dargestellt. Als Erstes wird die Problemstellung ohne Vergangenheitsbezug betrachtet.

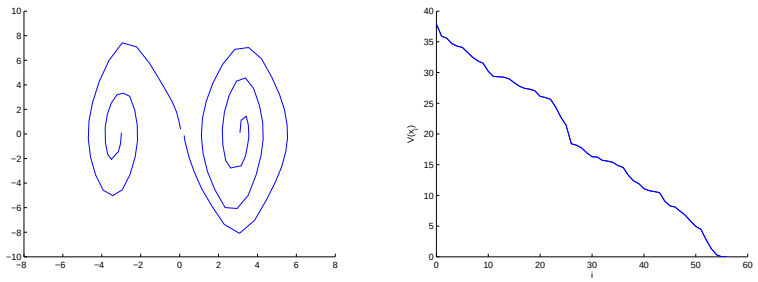

Abbildung 6.16: Trajektorien für ein Gitter mit 128 mal 128 Zellen

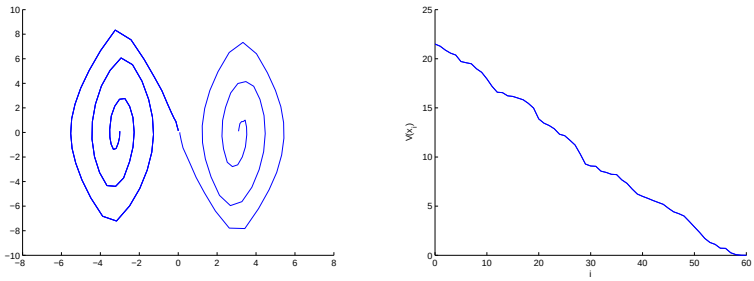

Abbildung 6.17: Trajektorien für ein Gitter mit 256 mal 256 Zellen

Als Nächstes werden die Trajektorien für die Problemstellung mit Vergangenheitsbezug betrachtet.

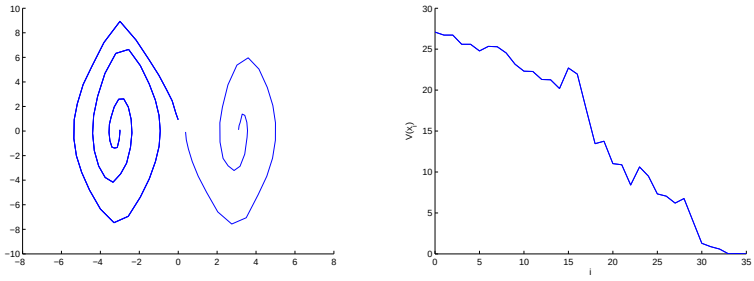

Abbildung 6.18: Trajektorien für ein Gitter mit 64 mal 64 Zellen

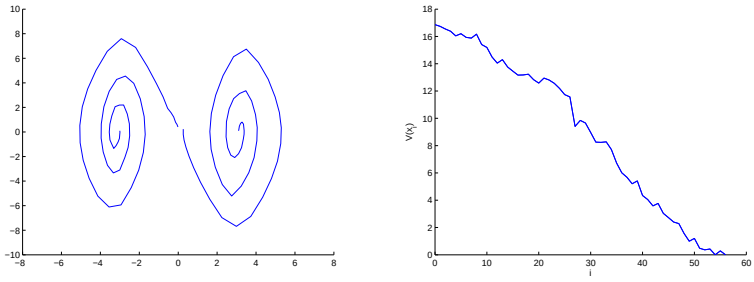

Abbildung 6.19: Trajektorien für ein Gitter mit 128 mal 128 Zellen

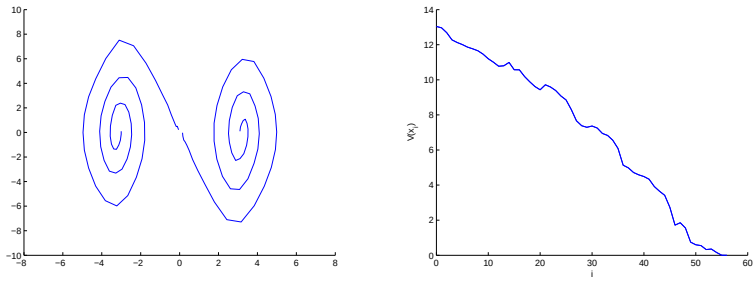

Abbildung 6.20: Trajektorien für ein Gitter mit 256 mal 256 Zellen

Wenn man den Verlauf der Wertefunktion entlang der Trajektorien betrachtet, fällt auf, dass bei der Problemstellung mit Vergangenheitsbezug der Wert manchmal ansteigt. Nach der Theorie sollte dies nicht der Fall sein.
Es passiert, dass die optimale Steuerung u^* aus der Berechnung von V für die Bestimmung der Trajektorie nicht genommen wird, da man mit ihr einen Zustand erreichen würde, der den Wert unendlich hat. Es wird deshalb eine andere Steuerung verwendet und für diese ist es nicht mehr gegeben, dass die optimale Wertefunktion abnimmt.
Dies kommt durch die Approximation der Zelle durch Testpunkte zu Stande. Die optimale Wertefunktion wurde nur für die Testpunkte berechnet. Die einzelnen Zustände der Trajektorien stimmen aber so gut wie nie mit diesen Testpunkten überein.
Bei der Problemstellung ohne Vergangenheitsbezug sind die Testpunkte so gewählt, dass viele auf dem Rand und auf den Ecken der Zellen liegen. Deshalb kommen Steigerungen der Werte nur äußerst selten vor.
Die für die Berechnung von V_2 gewählten Testpunkte liegen meist nicht auf dem Rand und auf den Ecken von $X(Z)$. Daher steigen die Werte entlang der Trajektorie öfter an.

6.3 Knotenanzahl und Rechenzeiten

Zuerst wird die Anzahl von möglichen Knoten für beide Problemstellungen betrachtet.
Bei der Berechnung von V_1 werden alle Knoten benötigt. Für ein Gitter mit n Zellen hat der gespeicherte Graph von der Problemstellung ohne Vergangenheitsbezug n Knoten.
Bei der Berechnung von V_2 werden nicht alle Knoten gespeichert. Andernfalls hätte der Graph $n(n+1)$ Knoten. Bei der Problemstellung mit Vergangenheitsbezug wird meist nur ein Bruchteil der möglichen Knoten verwendet.

Zunächst wird die Anzahl der Knoten vom Beispiel aus Kapitel 6.1 betrachtet.

n	8	16	32	64	128	256	512	1024
$n(n+1)$	72	272	1056	4160	16512	65792	262656	1049600
Anzahl der verwendeten Knoten	34	91	277	895	2294	5095	10696	21897
verwendete Knoten in %	47,2	33,5	26,2	21,5	13,9	7,8	4,2	2,1

In der folgenden Tabelle wird das Pendelmodell untersucht.

n	32^2	64^2	128^2	256^2
$n(n+1)$	$1,05 * 10^6$	$1,68 * 10^7$	$2,68 * 10^8$	$4,29 * 10^9$
Anzahl der verwendeten Knoten	$9,53 * 10^3$	$5,61 * 10^4$	$3,62 * 10^5$	$2,52 * 10^6$
verwendete Knoten in %	0,91	0,33	0,14	0,06

Man erkennt, dass es sinnvoll ist, nur die benötigten Knoten für die Problemstellung mit Vergangenheitsbezug zu speichern. Es kann dadurch einiges am Speicherbedarf des Programms eingespart werden. Die Anzahl der Hyperkanten ist abhängig von der Anzahl der verwendeten Knoten. Falls ein Knoten überhaupt als Start- oder Zielknoten verwendet wird, so dient er als Startknoten von $|U|$ Hyperkanten.

Im Folgenden werden die Rechenzeiten des Programms untersucht. Dazu wurde ein Computer mit 4 GB Arbeitsspeicher und zwei AMD Opteron 254 (2,8 Ghz, 1 MB Cache) Prozessoren verwendet. Für die Berechnung wurde jedoch nur ein Prozessor benutzt.
T bezeichne die Rechenzeit für das gesamte Programm. In der nächsten Tabelle sind die Rechenzeiten des ein-dimensionalen Beispiels in Sekunden dargestellt.

n	8	16	32	64	128	256	512	1024
$T(V_1)$	0.0016	0.0037	0.0084	0.0205	0.0429	0.0975	0.2069	0.4167
$T(V_2)$	0.0156	0.0415	0.0963	0.2355	0.4630	0.7001	1.2743	2.4044

Für das Pendelmodell sind Rechenzeiten ebenfalls in Sekunden dargestellt.

n	64^2	128^2	256^2	512^2
$T(V_1)$	17.7	70.6	286.1	1176.1
$T(V_2)$	265.8	1154.6	5344.7	

Der Arbeitsspeicher ist bei der Berechnung von V_2 für die Zerlegung in 256^2 Zellen fast komplett benutzt worden. Es ist deshalb nicht sinnvoll die Wertefunktion für feinere Zerlegungen zu berechnen, da sich die Programmlaufzeit überproportional verlängern würde. Wie erwartet, benötigt bei einer gleich feinen Zerlegung das Programm für die Problemstellung mit Vergangenheitsbezug deutlich mehr Rechenzeit.
Die Programme verbrauchen die meiste Zeit für das Erstellen der Hypergraphen, da dort für jeden Testpunkt und jede Steuerung Funktionen ausgewertet bzw. Differentialgleichungen gelöst werden müssen.

6.4 Zusammenfassung

Der größte Vorteil der Problemstellung mit Vergangenheitsbezug ist, dass diese bereits für eine gröbere Zerlegung stabilisierbar ist. Dies ist vor allem dann entscheidend, wenn es aus technischen oder physikalischen Gründen nicht möglich ist, den Zustand des Systems genauer zu bestimmen. Bei einer gleich feinen Zerlegung ist die optimale Wertefunktion für die Problemstellung mit Vergangenheitsbezug höchstens so groß wie für die ohne Vergangenheitsbezug. Das System kann also im ungünstigstem Fall mit geringeren Kosten ins Gleichgewicht gesteuert werden.
Als Nachteile sind die höheren Rechenzeiten und die nicht vorhandene Monotonie der Wertefunktion entlang der Trajektorien aufzuführen.

Kapitel 7

Ausblick

Welche Möglichkeiten gibt es die Effizienz des Algorithmus zu steigern?

Nahe liegend ist es, den n-stufigen Ansatz zu verwenden. Dabei wird nicht nur ein vergangener, sondern gleich n vergangene Schritte zur Lösung des Steuerungsproblems mit einbezogen (siehe Kapitel 2.2). Es muss untersucht werden, ob der erhöhte Rechenaufwand und der größere Speicherbedarf für die zu erwartenden besseren Ergebnisse gerechtfertigt sind.

Eine andere Möglichkeit ist, bei der Berechnung des neuen Zustandsbereichs X_{k+1} zusätzlich die Steuerung u_{k-1} zu berücksichtigen, mit der der Zustandsbereich X_k erreicht wurde. Für die 2-stufige Problemstellung werden die Zustände des Kontrollsystems zu $Z_k = \begin{pmatrix} X_{k-1} \\ u_{k-1} \\ X_k \end{pmatrix}^T$ erweitert. Die Berechnung der Menge $X(Z)$ ändert sich dann folgendermaßen.

$$X(Z) = \bigcup_{\beta_i \in \hat{\mathbf{B}}} (f(\beta_i(X_{k-1}, u_{k-1}), u_{k-1}) \cap X_k)$$

Die Menge $X(Z)$ wird also durch die Berücksichtigung von u_{k-1} kleiner. Die Störung des Ausgangswertes nimmt ab. Es ist zu erwarten, dass das Steuerungsproblem für eine größere Störung stabilisierbar ist.

Es kann ebenfalls untersucht werden, ob man durch eine andere Wahl von Testpunkten und Teststeuerungen bessere Ergebnisse erzielen kann. Für die Teststeuerungen wäre es am günstigsten sie so zu wählen, dass man alle möglichen Hyperkanten erfasst (siehe Bemerkung 4.4).
Ein wesentlicher Punkt ist eine geschicktere Wahl der Testpunkte. Dadurch würde vermutlich vermieden, dass die optimale Wertefunktion entlang der Trajektorie öfters ansteigt. Eine Möglichkeit wäre, die Testpunktmengen $TP2(X_i)$ durch zusätzliche Testpunkte zu ergänzen.

Diese erhält man z.B. durch die Betrachtung der Zustandsbereichsmenge

$$X_{TP2}(X_i) = \{X_j \in \mathbf{X_B} : \exists x \in TP2(X_i) \text{ mit } x \in X_j\}.$$

Für jeweils zwei benachbarte Bereiche X_j und X_k aus $X_{TP2}(X_i)$ werden geeignete Testpunkteaus der Schnittmenge $X_j \cap X_k$ gewählt und zur Testpunktmenge $TP2(X_i)$ hinzugefügt. Wegen der Stetigkeit von f ist zu erwarten, dass es $X_j \cap X_k$ Punkte gibt, die in den Bereichen $X\big((X_i, X_j)\big)$ und $X\big((X_i, X_k)\big)$ liegen. Dann befinden sich mehr Testpunkte auf dem Rand der Mengen $X(Z)$. Dadurch ist ist ein besseres Ergebnis zu erwarten.

Anhang A

Inhalt der CD

In der Datei `da.pdf` ist diese Diplomarbeit im pdf-Format gespeichert.
Auf der CD ist in den folgenden Dateien das Programm für die Lösung der Problemstellung ohne Vergangenheitsbezug gespeichert. Sie sind im Ordner `ohne_Vergangenheitsbezug` abgelegt.

`makefile`	Makefile zum Kompilieren der Programme
`defs.h`	Header-Datei mit grundlegenden Definitionen, die von der Gitterverwaltung verwendet werden
`grid.h`	Header-Datei für die Gitterverwaltung und für alle Funktionen, die auf das Gitter zugreifen.
`grid.c`	Implementierung der Funktionen aus `grid.h`
`hgraph.h`	Header-Datei für das Erstellen des Hypergraphen und das Lösen des Kürzesten-Wege-Problems
`hgraph.c`	Implementierung der Funktionen aus `hgraph.h`
`dopri5.h`	Header-Datei für den Differentialgleichungslöser
`dopri5.c`	Implementierung des Differentialgleichungslösers
`system1.c`	Beispielsystem aus Kapitel 6.1
`pendel.c`	Beispielsystem aus Kapitel 6.2

In den folgenden Dateien ist das Programm für die 2-stufige Problemstellung gespeichert. Diese befinden sich im Ordner `mit_Vergangenheitsbezug`.

`makefile`	Makefile zum Kompilieren der Programme
`defsmit.h`	Header-Datei mit grundlegenden Definitionen, die von der Gitterverwaltung verwendet werden
`gridmit.h`	Header-Datei für die Gitterverwaltung und für alle Funktionen, die auf das Gitter zugreifen.
`gridmit.c`	Implementierung der Funktionen aus `grid.h`

`hgraphmit.h`	Header-Datei für das Erstellen des Hypergraphen und das Lösen des Kürzesten-Wege-Problems
`hgraphmit.c`	Implementierung der Funktionen aus `hgraph.h`
`dopri5.h`	Header-Datei für den Differentialgleichungslöser
`dopri5.c`	Implementierung des Differentialgleichungslösers
`system1mit.c`	Beispielsystem aus Kapitel 6.1
`pendelmit.c`	Beispielsystem aus Kapitel 6.2

Mit `system1.c`, `pendel.c`, `system1mit.c` und `pendelmit.c` werden die Programme zur Berechnung der optimalen Wertefunktion und zur Berechnung von Trajektorien für die jeweiligen Beispiele aufgerufen. Die Ausgabe erfolgt jeweils in dafür angelegte Dateien.

Im Ordner `matlab` sind die zur graphischen Darstellung verwendeten Matlab-Funktionen abgelegt.

`gridplot1d.m`	Plottet die Wertefunktionen für ein-dimensionale Beispiele
`gridplot1dvergleich.m`	Plottet zwei Wertefunktionen für ein-dimensionale Beispiele
`gridplot2d.m`	Plottet V_1, $V_{\min}$, $V_{\max}$ oder V_δ für zwei-dimensionale Beispiele
`trajplot2d2.m`	Plottet zwei Trajektorien für zwei-dimensionale Beispiele
`trajplotV2d.m`	Plottet die Wertefunktion eines zwei-dimensionalen Beispiels entlang einer Trajektorie

Anhang B

Notation

In der folgenden Übersicht sind einige wichtige Bezeichnungen und Notationen vermerkt, die im Laufe der vorliegenden Arbeit regelmäßig erscheinen.

BEZEICHNUNG	BEDEUTUNG
$\|E\|$, wobei E eine Menge ist	Mächtigkeit von E
$\overline{D}, D \subset \mathbb{R}^n$	der Abschluss von D
Landau-Symbol O: $f \in O(g)$	$\exists\, c \in \mathbb{R}$ mit $f(n) \leq g(n)$ für fast alle $n \in \mathbb{N}$
$e : a \to A$	Hyperkante vom Startkonten a zu der Zielknotenmemge A

Literaturverzeichnis

[1] M. von Lossow, *Mengenorientierte optimale Steuerung und Fehlerschätzung* ; Diplomarbeit 2005;
`http://www.uni-bayreuth.de/~lgruene/diplom/marcusvon_lossow.pdf`.

[2] L. Grüne, O. Junge, *Global optimal control of perturbed systems*; Journal of Optimization Theory and Applications; wird in Ausgabe 136 im Jahr 2008 veröffentlicht;
`http://www.uni-bayreuth.de/departments/math/~lgruene/publ/robsetstab.html`.

[3] L. Grüne, O. Junge, *Approximately optimal nonlinear stabilization with preservation of the Lyapunov function property*; noch nicht veröffentlicht;
`http://www.math.uni-bayreuth.de/ lgruene/publ/approxlyap.html`.

[4] L. Grüne, *Numerik Dynamischer Systeme*; Vorlesungsskript;
`http://www.uni-bayreuth.de/departments/math/~lgruene/numdyn0304/`.

[5] M. Dellnitz, A. Hohmann, *A subdivision algorithm for the computation of unstable manifolds and global attractors*; Numerische Mathematik 75 (3) (1997), 293-317.

[6] L. Grüne, *Numerische Mathematik II: Differentialgleichungen*; Vorlesungsskript;
`http://www.uni-bayreuth.de/departments/math/~lgruene/numerik03/`.

[7] J. Clark, D. Holton, *Graphentheorie*; Spektrum Akademischer Verlag, Heidelberg, 1994.

[8] L. Grüne, O. Junge, *A set oriented approach to optimal feedback stabilization*; Systems & Control Letters, 54 (2) (2005), 169-180.

[9] E. Dijkstra, *A note on two problems in connexion with graphs*; Numerische Mathematik 1 (1959), 269-271.

[10] D. Bertsekas, *Dynamic Programming and Optimal Control* (3. Auflage); Athena Scientific, Belmont, 2005.

[11] H. Gumm, M.Sommer, *Einführung in die Informatik* (7. Auflage); Oldenbourg, 2006.

ERKLÄRUNG

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Bayreuth, den 4. Juni 2007 .....

Florian Müller