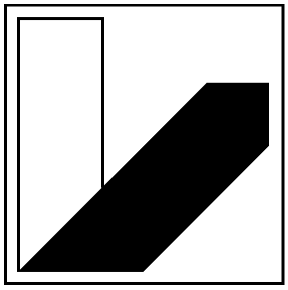


UNIVERSITÄT
BAYREUTH

FAKULTÄT FÜR MATHEMATIK UND PHYSIK
MATHEMATISCHES INSTITUT

Lehrstuhl für angewandte Mathematik

Berechnung erreichbarer Mengen mit globalen Optimierungsverfahren

Diplomarbeit

von

Michael Bodenschatz

Aufgabensteller/Betreuer:
Prof. Dr. Lars Grüne

2. Betreuer:
AOR Dr. Robert Baier

Bearbeitungszeitraum:
1. Dezember 2013 -
31. März 2014

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Erreichbare Mengen von nichtlinearen Kontrollproblemen spielen in der Anwendung eine große Rolle, da man oft nicht notwendigerweise an der optimalen Lösung eines solchen Systems interessiert ist, sondern vielmehr daran, in welchem Bereich sich eine Lösung eines betrachteten Kontrollsystems zu einem festen Zeitpunkt aufhalten kann.

Dies findet zum Beispiel bei Fehlerabschätzung von Systemen Anwendung, bei dem der Störwert als Steuerung eingeht. Auf diese Weise lässt sich abschätzen, wie sich ein Modell unter Störung von außen verhält. Auch bei der Berechnung von Wettervorhersage- und Klimamodellen, Satellitenbewegungen im Orbit und vielen anderen Bereichen wie bei Fahrerassistenzsystemen (vgl. [13]) können diese von Nutzen sein.

Anders als bei der Lösung von Optimalsteuerungsproblemen (OCP), bei denen es genügt eine Lösungstrajektorie zu untersuchen, benötigt man bei der Betrachtung von Erreichbarkeitsmengen Informationen über alle Lösungen. Wo es bei OCPs genügt die Lösungstrajektorie hinsichtlich eines Kostenfunktionals zu optimieren, hilft ein solcher Ansatz zunächst nicht bei der Berechnung von Erreichbarkeitsmengen.

Zur Berechnung dieser Mengen gibt es unterschiedliche Ansätze. Gerade bei linearen Problemen welche für konvexe Kontrollbereiche auch konvexe Erreichbarkeitsmengen liefern, ist die Auswahl an möglichen Verfahren groß. Für diese lassen sich die Mengen beispielsweise mithilfe mengenwertiger Integrationsverfahren erzeugen, bei dem die Menge über Schnitte von Halbebenen repräsentiert wird (vgl. [2]).

Für allgemeine nichtlineare Kontrollprobleme, welche in der Regel keine konvexen Erreichbarkeitsmengen liefern, ist die Auswahl aber bedeutend kleiner. Beispielsweise lassen sich solche Systeme mit einem mengenwertigen Eulerverfahren lösen, welches allerdings eine sehr hohe Laufzeit hat.

Der in dieser Arbeit verfolgte Ansatz betrachtet das Problem nicht mengenwertig, sondern verfolgt die Idee, die Menge über Abstände dieser zu Gitterpunkten im Zustandsraum zu approximieren. Statt also das Problem mengenwertig zu lösen, löst man eine Vielzahl an speziellen Optimalsteuerungsproblemen. Die Idee lässt sich wie folgt zusammenfassen:

1. Betrachte einen Bereich des Zustandsraum, welcher die Erreichbarkeitsmenge im besten Fall vollständig enthält.

2. Lege ein Gitter über den betrachteten Bereich das Zustandsraum.

3. Finde für jeden Gitterpunkt g einen Punkt x_r aus dem der Erreichbaren Menge, so dass der Abstand $\|g - x_r\|_2$ minimal ist.

Wenn man nun alle mit dieser Methode berechneten Zielpunkte x_r betrachtet, so ergibt sich eine gute Approximation der erreichbaren Menge, da für Gitterpunkte in der Erreichbarkeitsmenge $g = x_r$ gilt.

Um einen zulässigen Punkt aus der erreichbare Menge zu finden muss man das Kontrollproblem, welches üblicherweise über kontinuierliche Zeit betrachtet wird, für eine zulässige Kontrolle lösen. Die hier betrachteten System liegen dabei als Differentialgleichung vor,

weshalb man unter passenden Diskretisierungen zu bekannten Lösern greifen kann. In dieser Arbeit wird ein simples explizites Eulerverfahren verwendet um einen erreichbaren Punkt zu berechnen, da die Differentialgleichung häufig für unterschiedlichste Eingaben gelöst werden muss, prinzipiell kann man allerdings auch zu anderen Lösern greifen.

Ziel ist es allerdings nicht einen beliebigen erreichbaren Punkt zu finden, sondern den erreichbaren Punkt der den Abstand minimiert. Man löst also viele nichtlineare Optimierungsprobleme der Form

$$\begin{array}{ll} \min & \|g - x_r\| \\ \text{sd} & g \text{ ist ein gegebener Gitterpunkt} \\ & x_r \text{ ist ein erreichbarer Punkt} \end{array}$$

deren Lösungen dann die Erreichbarkeitsmenge gemäß der folgenden Skizze repräsentieren.

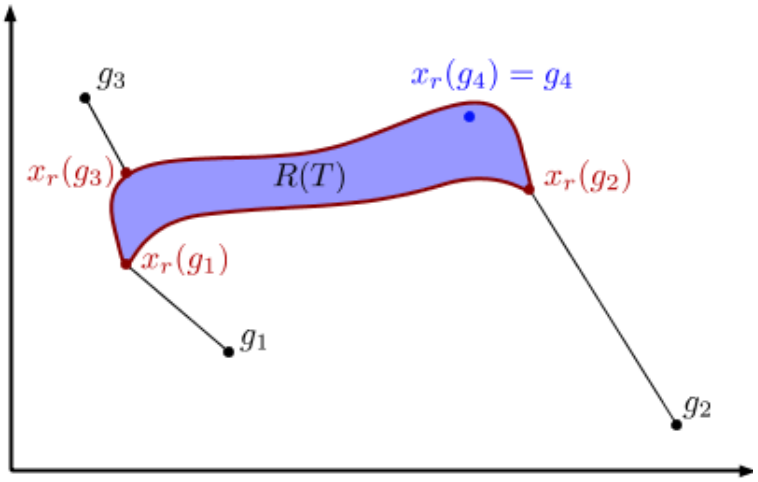


Abbildung 1.1.: Illustration erreichbare Menge approximiert durch minimierte Distanzfunktionen

Zur Lösung dieser Kontrollprobleme setzte man bisher auf lokale Optimierer, welche den Vorteil haben, dass sie oft sehr schnell Lösungen liefern, für eine große Klasse an Problemen verwendet werden können und man eine große Auswahl unterschiedlichster Algorithmen hat. Allerdings liefern diese lediglich lokale Lösungen, welche zwar in einer Umgebung optimal sind, allerdings nicht notwendigerweise bezüglich des gesamten Suchraums.

Es kann also für Probleme mit vielen lokalen Minima der Fall auftreten, das große Teile der erreichbaren Menge nicht berechnet werden. An dieser Stelle ist es natürlich von Interesse, wie gut hier globale Optimierer funktionieren, wie angesichts der NP-Schwere von globalen Optimierungsproblemen deren Laufzeit ist und ob diese ohne weiteres für diese Art von Problemen eingesetzt werden können. Ferner stellt sich die Frage, wie man mögliche Ergebnisse bezüglich Laufzeit und auch Qualität im Rahmen der durch die Optimierer bereitgestellten Optionen verbessern kann.

1.2. Gliederung

Zunächst ist eine formal sinnvolle Definition des eigentlichen Kontrollproblems und der Erreichbarkeitsmenge mitsamt deren zeitlicher und räumlicher Diskretisierung nötig gefolgt von dem daraus resultierenden Basisalgorithmus zur Berechnung der approximierten Erreichbarkeitsmengen.

Anschließend wird näher auf die adaptive Variante eingegangen, welche für einen Großteil der numerischen Experimente verwendet wurde um die Laufzeit zu verringern. Dabei wird auch grob auf auftretende Probleme und deren Lösungen eingegangen.

Im darauf folgenden Kapitel werden die genutzten globalen Optimierer vorgestellt. Dies umfasst neben deren grundlegender Funktionsweise auch möglicher Anpassungen um Ergebnisse zu verbessern.

Danach wird etwas auf die eigentliche Implementierung eingegangen, welche Konzepte hierfür verwendet wurden und wie der allgemeine Programmaufbau ist.

Bei den Numerischen Resultaten werden zunächst die Ergebnisse der Optimierer mit der bisher verwendeten lokalen Variante verglichen und sowohl Stärken als auch Schwächen aufgezeigt. Jeweils wird versucht die Ergebnisse im Rahmen der durch die Optimierer gegebenen Optionen weiter zu verbessern. Anschließend werden die globalen Optimierer miteinander verglichen, sowohl unter qualitativen Gesichtspunkten, also auch im Hinblick auf deren Geschwindigkeit.

Abschließend wird ein Fazit gezogen und es wird näher darauf eingegangen, in wie weit sich globale Optimierer für Probleme dieser oder ähnlicher Art eignen.

Im Anhang befinden sich abschließen Informationen und Voraussetzungen zum Kompilieren des Programmcodes.

2. Problemstellung und Algorithmus

2.1. Problemstellung

Zunächst ist eine Definition des eigentlichen Problems nötig. Diese orientiert sich stark an jener aus [5], was ebenfalls für die zeitliche Diskretisierung gilt.

Definition 1: (nichtlineares Kontrollsystem)
Ein nichtlineares zeitinvariantes Kontrollproblem in kontinuierlicher Zeit $t \in \mathbb{R}$ mit Zustand $x \in \mathbb{R}^n$ und Dimension $n \in \mathbb{N}$, Anfangszeit t_0, Anfangszustand $x_0 \in X_0 \subset \mathbb{R}^n$ und Menge an zulässigen Kontrollwerten $U \subset \mathbb{R}^m$ mit Dimension $m \in \mathbb{N}$ ist gegeben durch

$$\begin{aligned} \tfrac{d}{dt}x(t) = \dot{x}(t) &= f(x(t), u(t)) \\ u &\in L^\infty(\mathbb{R}, U) \\ x(t_0) &= x_0 \in X_0 \end{aligned}$$

wobei $f : \mathbb{R}^n \times U \to \mathbb{R}$ ein stetiges Vektorfeld ist. $\mathcal{U} = L^\infty(\mathbb{R}, U)$ mit

$$L^\infty(\mathbb{R}, U) = \left\{ u : \mathbb{R} \to U \;\middle|\; \begin{array}{c} u \text{ ist messbar und außerhalb einer} \\ \text{Lebesgue-Nullmenge beschränkt} \end{array} \right\}$$

ist die Menge der zulässigen Kontrollfunktionen und für $u \in \mathcal{U}$ gilt $u(t) \in U$ für alle t.

$x(t, t_0, x_0, u(.))$ bezeichnet eine Lösung für einen gegebenen Anfangswert x_0 zur Anfangszeit t_0 und Kontrollfunktion $u(\cdot)$ zum Zeitpunkt t.

Für Kontrollsysteme dieser Art gilt folgender Satz, welcher die Wahl der Klasse an Kontrollfunktionen rechtfertigt und die Existenz und Eindeutigkeit einer Lösung unter den relativ schwachen Voraussetzungen garantiert.

Satz 2: (Satz von Carathéodory, vgl. [21])
Gegeben sei ein zeitinvariantes Kontrollsystem wie oben mit den Eigenschaften:

(i) Raum der Kontrollfunktionen ist gegeben durch $\mathcal{U} = L^\infty(\mathbb{R}, U)$

(ii) Vektorfeld $f : \mathbb{R}^n \times U \to \mathbb{R}^n$ ist stetig.

(iii) Für alle $R > 0$ existiert $L_R > 0$, so dass die Abschätzung

$$\|f(x_1, u) - f(x_2, u)\| \leq L_R \|x_1 - x_2\|$$

für alle $x_1, x_2 \in \mathbb{R}^n$ und alle $u \in U$ mit $\|x_1\|$, $\|x_2\|$, $\|u\| \leq R$ gilt.

Dann gibt es für jeden Punkt $x_0 \in \mathbb{R}^n$ und jede Kontrollfunktion $u \in \mathcal{U}$ ein maximales offenes Intervall I mit $0 \in I$ und genau eine absolut stetige Funktion $x(t)$, die die Integralgleichung

$$x(t) = x_0 + \int_0^t f(x(\tau), u(\tau)) d\tau$$

für alle $t \in I$ erfüllt.

Gesucht ist in dieser Arbeit nicht eine Lösung, sondern die Menge aller Zustände, welche zu einem gegebenen Endzeitpunkt T von allen zulässigen Trajektorien erreicht werden.

Definition 3: (Erreichbarkeitsmenge)
Gegeben sei ein Kontrollsystem gemäß obiger Definition, dann ist die Erreichbarkeitsmenge $R(T) := R(T, t_0, X_0)$ zum Endzeitpunkt T mit Anfangszeit t_0 und Anfangszustandsmenge X_0 gegeben durch

$$\begin{aligned} R(T) &:= R(T, t_0, X_0) \\ &= \left\{ x(T, t_0, x_0, u(.)) \;\middle|\; \begin{array}{l} u(.) : [t_0, T] \to U \text{ zul. Kontrollfunktion} \\ \quad \text{also } u \in L^\infty([t_0, T], U) \\ t_0 \text{ Anfangszeit} \\ x_0 \in X_0 \text{ Anfangswert} \end{array} \right\} \end{aligned}$$

Falls $X_0 = \{x_0\}$ gilt, so schreibt man auch $R(T) = R(T, t_0, x_0)$.

Zur numerischen Berechnung der Erreichbarkeitsmenge muss man das System zunächst zeitlich diskretisieren. Dazu löst man das zugrunde liegende Kontrollsystem mit einem passendes Einschrittverfahren, zum Beispiel ein explizites Runge-Kutta-Verfahren. Betrachte nun also ein auf $[t_0, T]$ äquidistantes Zeitgitter $\{t_0, t_1, \ldots, t_{N_t}\}$ mit $t_i = t_0 + ih$ und Schrittweite $h = (T - t_0)/N_t$.

Anstelle des Funktionenraums $L^\infty(\mathbb{R}, U)$ für u sei nun die Kontrolle gegeben als eine stückweise konstante Funktion $u_h(.) \in \mathcal{U}_h$, also:

$$u_h(t) = u_i \in U \text{ mit } t \in \begin{cases} [t_i, t_{i+1}) \text{ für } i = 0, \ldots, N_t - 2 \\ [t_i, t_{i+1}] \text{ für } i = N_t - 1 \end{cases}$$

Das kontinuierliche System lässt sich nun an den Stützstellen t_i mithilfe eines gegenen Einschrittverfahrens mit Inkrementfunktion Φ durch die Rekursion $x_h(t_{i+1}) = x_h(t_i) + h\Phi(t_i, x_h(t_i), u_h, h)$ auswerten.

Definition 4: (zeitdiskretisiertes Kontrollsystem)
Sei ein nichtlineares zeitinvariantes Kontrollsystem in kontinuierlicher Zeit wie oben gegeben. Ferner sei ein Einschrittverfahren mit Inkrementfunktion Φ und Schrittweite $h = (T - t_0)/N_t$ gegeben und sei $\mathcal{U}_h$ die Menge der auf Intervallen der Länge h stückweise konstanten Kontrollfunktionen so dass für $u_h \in \mathcal{U}_h$ gilt:

$$u_h(t) = u_i \in U \text{ für } t \in \begin{cases} [t_i, t_{i+1}) \text{ für } i = 0, \ldots, N_t - 2 \\ [t_i, t_{i+1}] \text{ für } i = N_t - 1 \end{cases}$$

Dann ist ein nichtlineares Kontrollproblem in diskretisierter Zeit $t \in \{t_0, t_1, \ldots, t_{N_t} = T\}$ mit Zustand $x \in \mathbb{R}^n$, $n \in \mathbb{N}$, Anfangszeit t_0, Anfangszustand $x_0 \in X_0 \subset \mathbb{R}^n$ und Menge an zulässigen Kontrollwerten $U \subset \mathbb{R}^m$ gegeben durch die Iteration

$$\begin{aligned} x_h(t_{i+1}) &= x_h(t_i) + h\Phi(t_i, x_h(t_i), u_h, h) \\ x(t_0) &= x_0 \\ u_h &\in \mathcal{U}_h \end{aligned}$$

$x_h(t, t_0, x_0, u_h(.))$ bezeichnet dabei eine Lösung für einen gegebenen Anfangswert x_0 zur Anfangszeit t_0 und Kontrollfunktion $u_h(\cdot)$ zum Zeitpunkt $t \in \{t_0, t_1, \ldots, t_{N_t} = T\}$.

Wie oben lässt sich auch für dieses zeitdiskrete System die Erreichbarkeitsmenge völlig analog definieren.

Definition 5: (zeitdiskrete Erreichbarkeitsmenge)
Gegeben sei ein diskretisiertes Kontrollsystem gemäß obiger Definition, dann ist die Erreichbarkeitsmenge $R_h(T) := R_h(T, t_0, X_0)$ zum Endzeitpunkt T mit Anfangszeit t_0, Anfangszustandsmenge X_0 und Menge zulässiger stückweise konstanter Kontrollfunktionen $\mathcal{U}_h$ gegeben durch

$$\begin{aligned} R_h(T) &:= R_h(T, t_0, X_0) \\ &= \left\{ x_h(T, t_0, x_0, u_h(\cdot)) \;\middle|\; \begin{array}{l} u_h \in \mathcal{U}_h \text{ stückweise konstante} \\ \text{ zulässige Kontrollfunktion} \\ t_0 \text{ Anfangszeit} \\ x_0 \in X_0 \text{ Anfangswert} \end{array} \right\} \end{aligned}$$

Im einfachsten Fall handelt sich bei dem Einschrittverfahren um ein explizites Eulerverfahren, wobei $\Phi(t, x, u, h) = f(t, x, u)$ ist. Folglich ergibt sich für dieses Verfahren die Rekursion $x_h(t_{i+1}) = x_h(t_i) + hf(x_h(t_i), u_h(t_i))$. Für dieses gilt unter obiger Diskretisierung der folgende Satz (siehe [21]):

Satz 6: Betrachte ein zeitinvariantes Kontrollsystem gemäß Definition 1, für dass die Voraussetzung (i)-(iii) des Satzes von Carathéodory gelten. Dann gelten für die mit dem expliziten Eulerverfahren bestimmte Lösung des zeitlich disretisierten Kontrollsystems und jede Konstante $R > 0$ die folgende Aussagen:

(i) Es existiert eine von R abhängige Konstante $K > 0$, so dass für jede Kontrollfunktion $u \in \mathcal{U}$ mit $\|u\|_\infty \leq R$ und jeden Anfangswert $x_0 \in \overline{B}_R(0)$ eine diskrete Kontrollfunktion $u_h \in \mathcal{U}_h$ existiert, so dass die Abschätzung

$$\|x_h(t, t_0, x_0, u_h) - x(t, t_0, x_0, u)\| \leq K\sqrt{h}e^{Lt}$$

gilt.

(ii) Umgekerht existiert eine von R abhängige Konstante $K > 0$, so dass für jedes $x_0 \in \overline{B}_R(0)$, jede diskrete Kontrollfunktion $u_h \in \mathcal{U}_h$ mit $\|u_h\|_\infty \leq R$ und die durch $u(\tau) := u_h(t)$ für $\tau \in [t, t + h)\,,\, t \in h\mathbb{N}_0$ definierte stückweise konstante (also messbare) Kontrollfunktion die Abschätzung

$$\|x_h(t, t_0, x_0, u_h) - x(t, t_0, x_0, u)\| \leq Kh(e^{Lt} - 1)$$

gilt für alle $t \in h\mathbb{N}_0$, für die die Lösungen in $\overline{B}_R(0)$.

Der Beweis für diesen Satz ist sehr technisch und kann unter [22] nachgeschlagen werden. Für den Sonderfall, dass es sich bei (i) um ein konvexes Kontrollsystem handelt, also wenn die Menge $f(x, U_R) = \{f(x, u) | u \in U_R\} \subset \mathbb{R}^n\}$ mit $U_R = \{u \in U | \|u\| \leq R\}$ für alle $x \in \mathbb{R}^n$ und hinreichend große $R > 0$ konvex ist, so gilt eine schärfere Abschätzung der Ordnung $O(h)$. Diese und den dazugehörigen Beweis kann man unter [21] finden.

Das Eulerverfahren besitzt also für solche zeitdiskreten Kontrollsystemen die Konvergenzordnung $O(\sqrt{h})$. Da es für jedes $u \in \mathcal{U}$ ein $u_h \in \mathcal{U}_h$ gibt, existiert für das zeitkontinuierliche System eine approximierte Lösung, die höchsten mit Ordnung $O(\sqrt{h})$ abweicht. Mit steigendem N_t findet man also eine zulässige Kontrollfolge wodurch die exakte Lösung immer besser approximiert wird.

Da $R(T)$ die Vereinigung aller Lösungen des kontinuierlichen Systems zum Zeitpunkt T ist und diese durch Lösungen des zeitdiskreten Systems aus $R_h(T)$ approximiert werden, andererseits diese Lösungen bis auf eine gewissen Abweichung der Ordnung $O(h)$ ebenfalls zulässige Lösungen des zeitkontinuierlichen Systems sind, rechtfertigt dieser Satz die

Approximation von $R(T)$ durch $R_h(T)$. Unter den im Satz 6 gegebenen Voraussetzungen gilt $R_h(T) \to R(T)$ für $h \to 0$.

Tatsächlich gilt für erreichbare Mengen sogar, dass die des zeitdiskreten System mit Ordnung $O(h)$ zu jenen des kontinuierlichen Systems konvergieren. Zunächst ist dazu aber wie in [6] der folgende Satz wichtig, welcher besagt, dass unter nicht zu starken Voraussetzungen die Erreichbarkeitsmengen des kontinuierlichen System nicht zu weit von jener des zugehörigen konvexifizierten Systems entfernt liegt. Hierbei wird das Problem als Differentialinklussion aufgefasst und statt des kontinuierlichen Systems, das äquivalente System

$$\begin{aligned} \tfrac{d}{dt}x(t) &\in F(t,x) = \bigcup_{u\in\mathcal{U}} \{f(t,x,u)\} \\ x(t_0) &= x_0 \end{aligned} \tag{DI}$$

betrachtet. Das dazu gehörige konvexifizierte System ist dagegen durch

$$\begin{aligned} \tfrac{d}{dt}x(t) &\in \overline{\text{co}}\, F(t,x) \\ x(t_0) &= x_0 \end{aligned} \tag{CDI}$$

gegeben.

Satz 7: (vgl. [6] und [8]) Sei die mengenwertige Abbildung $F : [t_0, T] \times \mathbb{R}^n \to \mathbb{R}^n$ gegeben welche die folgenden Bedingungen erfüllt:

(i) Für alle $R > 0$ existiert ein $L_R \in L^1_+([t_0, T], \mathbb{R}_+)$, so dass für fast alle $t \in [t_0, T]$ gilt, $F(t, \cdot)$ ist $L_R(t)$-Lipschitz auf $B_R(0)$, also lokal Lipschitzstetig in x.

(ii) F ist von linearem Wachstum, dies bedeutet, es existiert $\lambda \in L^1([t_0, T], \mathbb{R}_+)$ so dass $\|F(t, x)\| \leq \lambda(t)(1 + \|x\|)$ gilt.

Dann ist der Abschluss der Erreichbarkeitsmenge von (DI) kompakt und gleich der Erreichbarkeitsmenge des konvexifizierten Systems (CDI).

Dieser Satz (Beweis siehe [8]) besagt also, dass die Erreichbarkeitsmenge der Differentialinklussion (DI) dicht in der der konvexifizierten Differentialinklussion (CDI) liegt.

Satz 8: (vgl. [6] und [9]) Sei $F : [t_0, T] \times \mathbb{R}^n \to \mathbb{R}^n$ Lipschitzstetig in beiden Argumenten (t und x) und besitze kompakte, konvexe und nichtleere Bilder. Dann gilt für alle $N \in \mathbb{N}$ mit $h = \frac{T-t_0}{N}$ die Abschätzung

$$d_H(R(T), R_h(T)) \leq Ch$$

wobei d_H der Hausdorff-Abstand ist, welcher für zwei kompakte, nichtleere Mengen $S, \tilde{S} \in \mathbb{R}^n$ und einen Punkt $x \in \mathbb{R}^n$ definiert ist durch

$$\begin{aligned} \text{dist}(x, S) &= \inf_{s\in S} \|x - s\| \\ d(S, \tilde{S}) &= \sup_{s\in S} \text{dist}(s, \tilde{S}) \\ d_H(S, \tilde{S}) &= \max\{d(S, \tilde{S}), d(\tilde{S}, S)\end{aligned}$$

Der Beweis dieses Satzes ist unter [9] zu finden und lässt zusammen mit dem Dichtheitsargument des vorherigen Satzes die Aussage zu, dass unter schwachen Voraussetzungen $R_h(T) \to R(T)$ für $h \to 0$ mit Ordnung $O(h)$ gilt, selbst wenn es sich nicht um Differentialinklussionen mit konvexen Bildern handelt.

2.2. Algorithmus

2.2.1. Grundgedanke

Betrachte nun ein beschränktes Gebiet $G \subset \mathbb{R}^n$. Angenommen es gilt $R(T, t_0, x_0) \subset G$, so kann man die Erreichbarkeitsmenge vollständig beschreiben, wenn man für jeden Punkt $g \in G$ den Punkt der Menge sucht, der den Abstand minimiert. Dieses Problem lässt sich als nichtlineares Optimierungsproblem der Form

$$\begin{array}{lrcl} \min & \frac{1}{2}\|x(T) - g\|_2^2 & & \\ \text{s.d.} & \dot{x}(t) & = & f(x(t), u(t)) \text{ für } t \in [t_0, T] \\ & u & \in & \mathcal{U} \\ & x(0) & = & x_0 \end{array} \tag{OCP(g)}$$

definieren, wobei man $x^*(\cdot, g)$ als optimale Lösung bezeichnet, für die der Abstand zum Zeitpunkt T zu einem Gebietspunkt g minimal ist. $u^*(\cdot, g)$ ist hier die zugehörige Kontrollfunktion die zum Erreichen dieser Lösung nötig ist.

OCP steht dabei für Optimal Control Problem oder eben Optimalsteuerungsproblem. Diese sind in der Regel etwas komplexer definiert, nicht selten will man bei solchen ein Kostenfunktional minimieren, welches von der Steuerung und der Lösungstrajektorie abhängt. Das hier aufgelistete Optimalsteuerungsproblem ist also eine schon sehr spezielle Variante.

Prinzipiell genügt es, Punkte zu suchen, bei denen der Abstand zur Erreichbarkeitsmenge 0 ist, denn für jeden Gitterpunkt $g \in G$ mit zugehörigem $x_r = \underset{x \in R(T)}{\operatorname{argmin}} \|x - g\|_2$ existiert ein Punkt $g' \in G$ mit $\|x_r - g\|_2 = 0$. Dadurch gilt

$$\begin{aligned} R(T) &= \bigcup_{g \in G \cup R(T)} g \\ &= \bigcup_{g \in G} \{x_R \in R(T) | \|x_R - g\|_2 = 0\} \\ &= \bigcup_{g \in G} \{x_R \in R(T) | \|x_R - g\|_2 \text{ ist minimal}\} \end{aligned}$$

und somit lässt sich die Erreichbarkeitsmenge mithilfe des Optimierungsproblems auch durch

$$R(T, t_0, x_0) = \bigcup_{g \in G} \left\{ x^*(T, g) \;\middle|\; \begin{array}{l} x^*(\cdot, g) \text{ ist Lösung des} \\ \text{Minimierungsproblems (OCP}(g)) \end{array} \right\}$$

beschreiben.

Falls $R(T, t_0, x_0) \not\subset G$, also $R(T)$ nicht vollständig in G enthalten ist, so lässt sich auf diese weise immerhin die zu G relative Erreichbarkeitsmenge

$$R(T, t_0, x_0)|_G = R(T, t_0, x_0) \cap G$$

mit dieser Methode vollständig beschreiben. Es können mithilfe des Minimierungsproblems (OCP(g)) zwar unter Umständen einzelen Randpunkte von $R(T)$ berechnet werden, welche außerhalb des betrachteten Gebiets G liegen, allerdings in den seltensten Fällen die komplette Menge.

Analog zum kontinuierlichen Fall lässt sich auch die diskretisierte Variante für ein gegebenes Einschrittverfahren mithilfe eines Optimierungsproblems beschreiben.

$$\begin{array}{lrcl} \min & \|x_h(T) - g\|_2^2 & & \\ \text{s.d.} & x_h(t_{i+1}) & = & x_h(t_i) + h\Phi(t_i, x_h(t_i), u_h, h) \\ & & & \text{für } i = 0, \ldots, N_t - 1 \\ & x(0) & = & x_0 \\ & u(\cdot) & \in & \mathcal{U}_h \end{array} \tag{DOCP(g)}$$

$x_h^*(\cdot, g)$ bezeichnet hier die optimale Lösung. DOCP steht für Discret Optimal Control Problem oder Zeitdiskretes Optimalsteuerungsproblem.

Dadurch lässt sich die Erreichbarkeitsmenge des diskretisierten System auch durch

$$R_h(T, t_0, x_0) = \bigcup_{g \in G} \left\{ x_h^*(T, g) \;\middle|\; \begin{array}{l} x_h^*(\cdot, g) \text{ ist Lösung des} \\ \text{Minimierungsproblems (DOCP}(g)) \end{array} \right\}$$

charakterisieren.

Zu beachten ist hierbei, dass die zeitlich diskretisierte Erreichbarkeitsmenge nicht gleich jener des kontinuierlichen Systems sein muss. Aus diesem Grund ist es nötig, das betrachtete Gebiet unter Umständen der Approximation anzupassen.

Für eine gegebene Zahl an Stützstellen N_t und ein gegebenes Einschrittverfahren lässt sich dieses Optimierungsproblem umformulieren, so dass es lediglich von einem Punkt des Gebiets G, dem Zeitintervall $[t_0, T]$, dem Anfangszustand x_0 und einer Kontrollfolge $u_0, \ldots, u_{N_t-1} \in U$ abhängt. Wenn man hier das Eulerverfahren verwendet, so lässt sich das Problem vereinfacht schreiben als:

$$\begin{array}{lrcl} \min\limits_{u_0,\ldots,u_{N_t-1},x_0} & \|x_{N_t} - g\|_2 & & \\ \text{s.d.} & x_{i+1} & = & x_i + hf(x_i, u_i) \\ & & & \text{für } i = 0, \ldots, N_t - 1 \\ & u_i & \in & U \text{ für } i = 0, \ldots, N_t - 1 \\ & x_0 & \in & X_0 \end{array} \qquad (\text{DOCP}_{\text{Euler}}(g))$$

2.2.2. Räumliche Diskretisierung

Nun lässt sich numerisch natürlich nicht jeder Punkt des Gebiets G betrachten, weshalb man dieses Problem noch räumlich diskretisieren muss. Dabei ist der Grundgedanke, dass man ein in jede Richtung äquidistantes Gitter über das Gebiet legt und nur die Gitterpunkte betrachtet. Man geht nun zunächst von einem Gebiet der Form

$$G = \{x \in \mathbb{R}^n \,|\, l_i \leq x_i \leq u_i \,,\, l, u \in \mathbb{R}^n \,,\, l_i < u_i\}$$

aus, was keine größere Einschränkung ist, da für jedes beschränktes Gebiet $G \in \mathbb{R}^n$ ein Gebiet G' der obigen Form existiert, mit $G \in G'$ und die Beschreibung durch die Optimalsteuerungsprobleme dadurch nicht schlechter wird.

Sei nun $N_x > 0$ die Anzahl der Gitterpunkte in jede Richtung, dann ist die Menge der Gitterpunkte gegeben durch:

$$G_{N_x} = \left\{ g_{j_1,\ldots,j_n} \in G \;\middle|\; \begin{array}{l} g_{j_1,\ldots,j_n} = \begin{pmatrix} l_1 + j_1 \frac{u_1 - l_1}{N_x - 1} \\ \vdots \\ l_n + j_n \frac{u_n - l_n}{N_x - 1} \end{pmatrix} \\ \text{mit } j_i \in \{0, \ldots, N_x - 1\} \text{ für } i = 1, \ldots, n \end{array} \right\}$$

Klarerweise gilt sowohl $G_{N_x} \subset G$ als auch

$$\forall g \in G \, \exists \, \tilde{g} \in G_{N_x} : \|g - \tilde{g}\|_2 \leq \delta \text{ mit } \delta = \frac{\|u - l\|_2}{N_x - 1}$$

Wenn man sich nun also

$$R_{h,N_x}(T, t_0, x_0) = \bigcup_{\tilde{g} \in G_{N_x}} \left\{ x_h^*(T, \tilde{g}) \;\middle|\; \begin{array}{l} x_h^*(\cdot, \tilde{g}) \text{ ist Lösung des} \\ \text{Minimierungsproblems (DOCP}(\tilde{g})) \end{array} \right\}$$

betrachtet, so findet man für jeden Punkt $x \in R_h(T, t_0, x_0)$ einen anderen Punkt $\tilde{x} \in R_{h,N_x}(T, t_0, x_0)$, so dass $\|x - \tilde{x}\|_2 \leq \delta$ gilt.

Bemerkung 9: Auch hier spielen eigentlich nur die Gitterpunkte eine Rolle, welche innerhalb der Erreichbarkeitsmenge liegen. Gitterpunkte außerhalb der Erreichbarkeitsmenge bilden den Rand unter Umständen besser ab, allerdings verringert sich der Fehler im Inneren dadurch nicht.

Je größer man also nun die Zahl der Gitterpunkte in jede Richtung wählt, desto besser deckt man die Erreichbarkeitsmenge des zeitlich diskretisierten Systems ab, wie man in den nachfolgenden Grafiken erkennen kann. Diese Grafiken stellen Lösungen des Kenderov-Problems dar, welches unter 5.1.2 näher beschrieben ist.

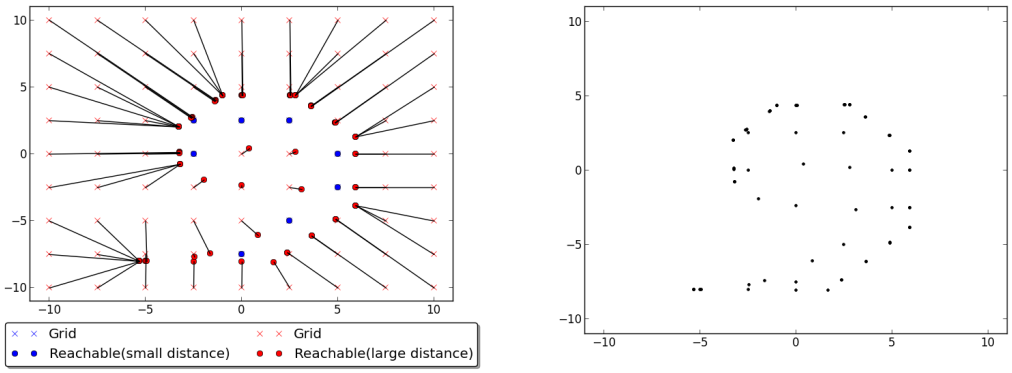


Abbildung 2.1.: Kenderov $N_t = 16$, $N_x = 17$

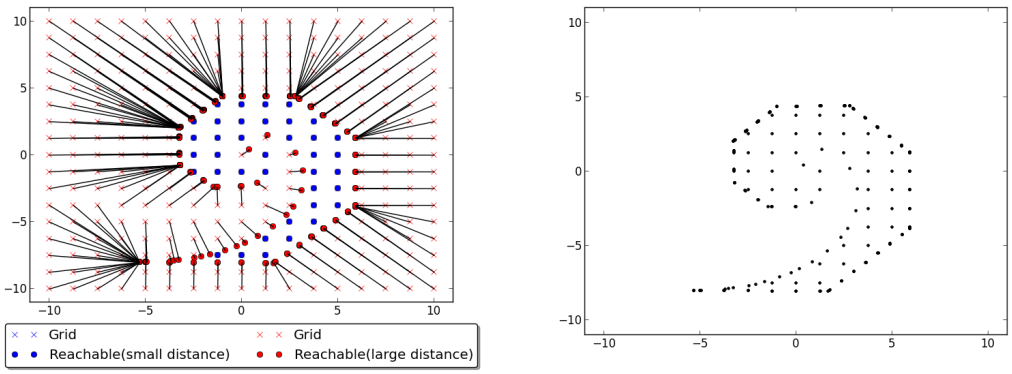


Abbildung 2.2.: Kenderov $N_t = 16$, $N_x = 17$

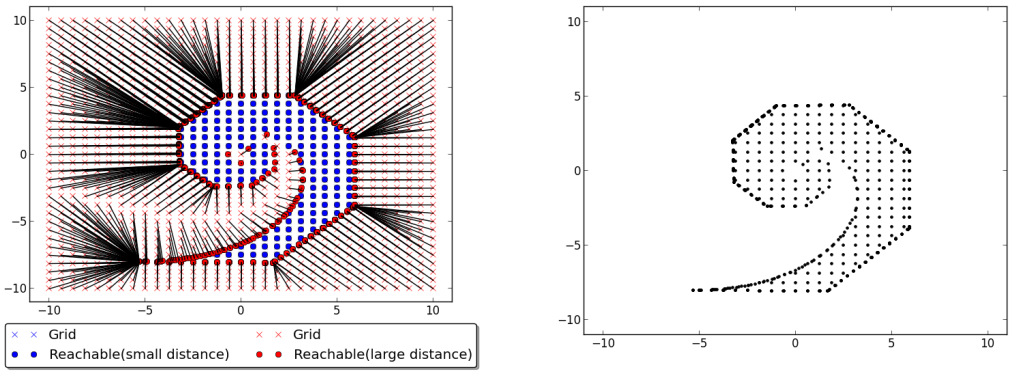


Abbildung 2.3.: Kenderov $N_t = 16$, $N_x = 33$

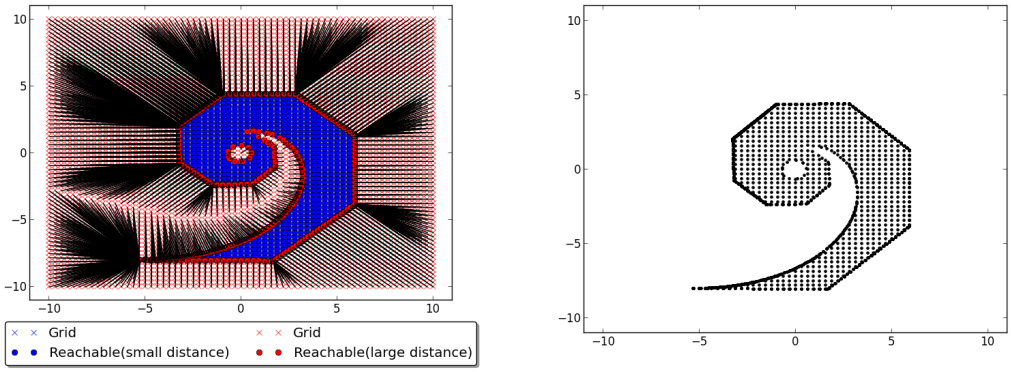


Abbildung 2.4.: Kenderov $N_t = 16$, $N_x = 65$

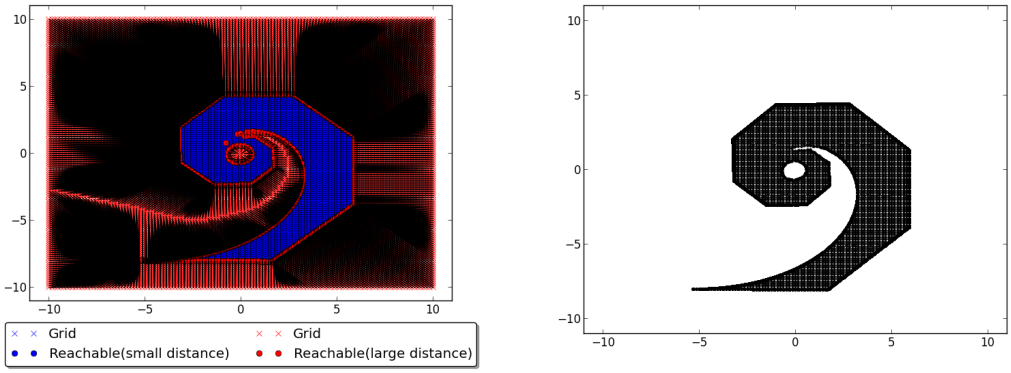


Abbildung 2.5.: Kenderov $N_t = 16$, $N_x = 129$

2.3. Adaptive Variante

Wie man sieht, genügt es eigentlich Bereiche des Gebiets zu untersuchen, die in der Nähe der approximierten Erreichbarkeitsmenge liegen. Bereiche, in denen der Abstand zur Erreichbaren Menge groß sind führen dazu, dass man viele Berechnungen macht, in denen man lediglich Randpunkte erreicht. Allerdings erreicht man solche Punkte mit ausreichender Genauigkeit bereits durch Gitterpunkte, die näher an der zu bestimmenden Menge liegen.

Um die Idee hinter der adaptiven Variante näher zu beschreiben, muss man zunächst von einer reinen Gitterpunktrepräsentation des betrachteten Gebiets hin zu einer Partitionierung in Boxen, welche durch gegenüberliegende Eckpunkte gegeben sind, übergehen. Eine Box ist gegeben durch zwei Punkte $l, u \in \mathbb{R}^n$ mit $l_i < u_i$ für $i = 1, \ldots, n$ und im folgenden wird die Bezeichnung

$$[l, u] = \{x \in \mathbb{R}^n | l_i \leq x_i \leq u_i \,,\, i = 1, \ldots, n\}$$

verwendet.

Wenn man nun das Gebiet $G = [l, u]$ betrachtet, so lässt sich für ein gegebenes N_x das räumlich diskretisierte Gebiet G_{N_x} wie oben durch

$$G_{N_x} = \left\{ g_{j_1,\ldots,j_n} \in G \;\middle|\; \begin{array}{l} g_{j_1,\ldots,j_n} = \begin{pmatrix} l_1 + j_1 \frac{u_1 - l_1}{N_x - 1} \\ \vdots \\ l_n + j_n \frac{u_n - l_n}{N_x - 1} \end{pmatrix} \\ \text{mit } j_i \in \{0, \ldots, N_x - 1\} \text{ für } i = 1, \ldots, n \end{array} \right\}$$

beschreiben. Mithilfe dieser Beschreibung lässt sich das Gebiet in Subboxen

$$[g_{j_1,\ldots,j_n}, g_{j_1+1,\ldots,j_n+1}] \text{ mit } j_i \in \{0, N_x - 1\} \,,\, i = 1, \ldots, n$$

unterteilen. Anfänglich wird mit einem groben Gitter gearbeitet, falls man aber einen Punkt in einer solchen Subbox findet, so betrachtet man das durch diese Subbox definierte Gebiet und legt über dieses ein bezüglich der einzelnen Koordinatenrichtungen äquidistantes Gitter.

Die Idee der adaptiven Variante lässt sich durch das folgende rekursive Schema beschreiben.

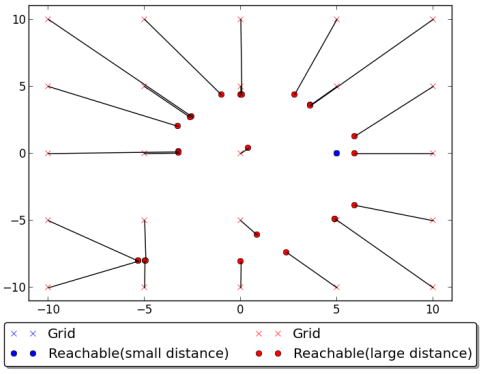
Algorithm 1 adaptive Variante

1. Initialisiere Rekursionstiefe R, Startgitterfeinheit $N_{x,0}$, Verfeinerungsgrad bei Rekursion $N_{x,R}$, Startgebeit $[l, u]$
2. Beginne mit einer groben Partitionierung auf $[l, u]$, also kleinem $N_{x,0}$
3. Berechne für jeden Gitterpunkt der aktuellen Box den Punkt der Erreichbarkeitsmenge der den Abstand minimiert
4. Teste für jede Subbox $[g_{j_1,\ldots,j_n}, g_{j_1+1,\ldots,j_n+1}]$ von $[l, u]$ ob ein durch den letzten Schritt gefundener Zielpunkt in dieser liegt. Falls ja, betrachte die Box $[l, u] = [g_{j_1,\ldots,j_n}, g_{j_1+1,\ldots,j_n+1}]$ mit Verfeinerung $N_{x,R}$ und verbleibender Rekursionstiefe $R-1$ und springe zu 3.
5. Beende den Algorithmus, wenn die gegebene Rekursionstiefe R für alle Boxen die Punkte der erreichbaren Menge enthalten erreicht ist.

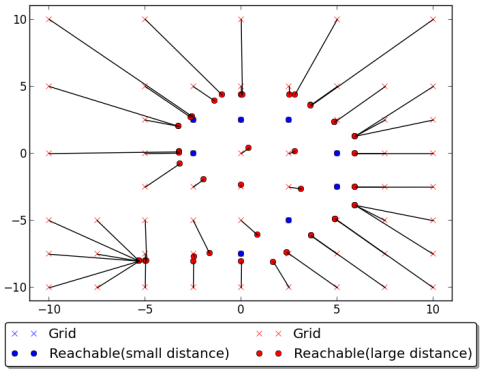
Im Folgenden ist $N_{x,R} = 3$, im zweidimensionalen wird also eine Box in 4 Subboxen unterteilt. Der Wert hier lässt sich variieren, die Feinheit des Gitters wird in dem Verfahren allerdings über die Rekursionstiefe bestimmt, so dass das Fixieren dieses Parameters sinnvoll ist. Der Parameter $N_{x,0}$ gibt die Startgitterfeinheit an, welche im Folgenden als N_x bezeichnet wird. Eine Variation dieses kann abhängig vom Problem vorteilhaft sein.

Mit dieser Notation lässt sich die nichtadaptive Variante auch über die adaptive realisieren, indem man $N_x = N_{x,0}$ groß wählt, und die Rekursionstiefe $R = 0$ setzt.

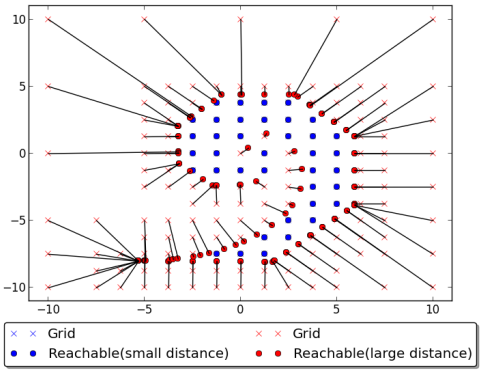
Für steigende Rekursionstiefe ergibt sich so folgender grafischer Verlauf.



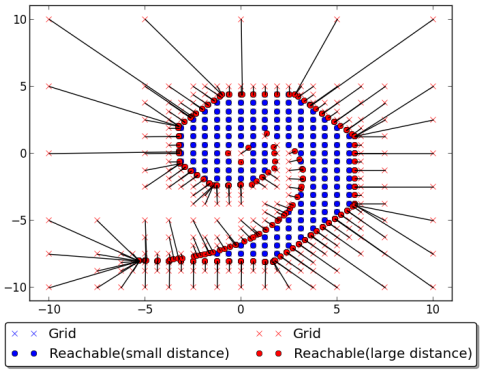
(a) $R = 0$



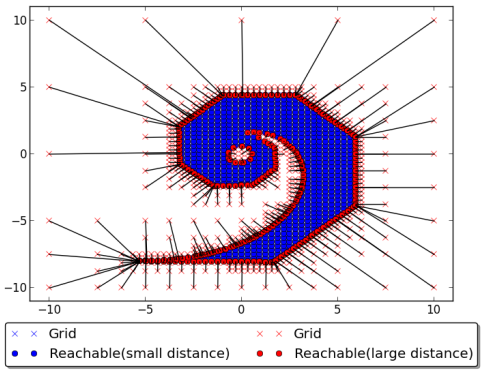
(b) $R = 1$



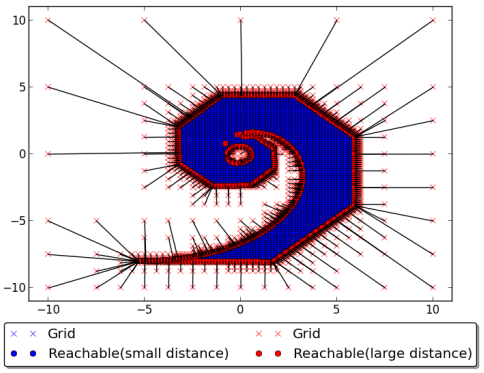
(c) $R = 2$



(d) $R = 3$



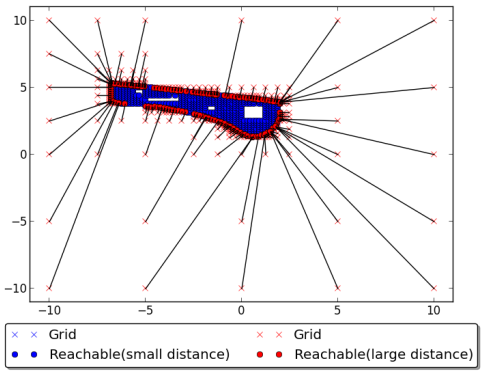
(e) $R = 4$



(f) $R = 5$

Abbildung 2.6.: Kenderov $N_t = 16$, $N_x = 5$

Abgesehen von dem Finden von lokalen statt globalen Minima kann man bei einer direkten Implementierung das Phänomen erkennen, dass Löcher innerhalb der Erreichbarkeitsmenge entstehen oder Teile dieser komplett fehlen. Wenn ein Gitterpunkt in der erreichbaren Menge liegt, müssen alle angrenzenden Boxen, welche eben diesen Punkt enthalten, verfeinert werden. Durch numerische Ungenauigkeiten, wird der Optimalwert des Optimierungsproblems DOCP(g) aber nie exakt 0. Die Lösung des zugrunde liegenden Kontrollproblems zum Zeitpunkt T entspricht also nicht notwendigerweise exakt dem Gitterpunkt, weshalb eben ein Zielpunkt nicht in allen angrenzenden Boxen liegt.



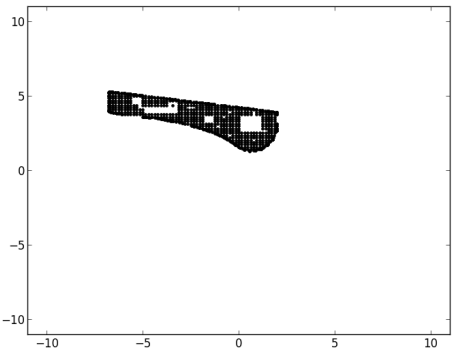


Abbildung 2.7.: Rayleigh, $N_t = 16$, $N_x = 5$, $R = 5$, adaptive Variante bei direkter Implementierung

Dieses Problem lässt sich dadurch lösen, dass man für einen Punkt feststellt ob er in der näheren Umgebung einer Box liegt. Statt in Punkt 4 des rekursiven Schemas zu testen, ob ein Punkt in einer Box liegt, überprüft man für ein gegebenes ϵ, ob ein Zielpunkt in der größeren Box $[g_{j_1,\dots,j_n}, g_{j_1+1,\dots,j_n+1}]_\epsilon$ liegt, welche durch

$$[l,u]_\epsilon := \{x \in \mathbb{R}^n \,|\, l_i - \epsilon(u_i - l_i) \leq x_i \leq u_i + \epsilon(u_i - l_i) \text{ für } i = 1,\dots,n\}$$

definiert ist. Mithilfe dieser Herangehensweise lassen sich die Blöcke im Inneren für sinnvolle Wahl von ϵ schließen.

Ein solcher Ansatz ist zumindest dann sinnvoll, sofern die Boxen in allen Koordinatenrichtungen ähnliche Ausmaße haben, also wenn $|u_i - l_i| \approx |u_j - l_j|$ für alle $i \neq j$ gilt, andernfalls muss die größere Box unter Umständen anders gewählt werden.

In der Praxis wird die Laufzeit mit diesem Verfahren massiv reduziert sofern die Menge $G \setminus R_h(T)$ groß ist, weshalb es auch in dieser Arbeit Anwendung findet.

Mehr Informationen zu dieser adaptiven Herangehensweise, weitere Probleme und Tests bezüglich der Wahl von Epsilon ist dem entsprechenden Artikel von Wolfgang Riedel [1] zu entnehmen.

3. Globale Optimierung

3.1. Grundlagen

In diesem Kapitel betrachtet man ein allgemeines Optimierungsproblem der Dimension n in folgender Form:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & F(x) \\ \text{sd.} \quad & l \leq x \leq u \quad \text{für } l, u \in \mathbb{R}^n \text{ mit } l \leq u \end{aligned} \tag{GOP}$$

Dabei bezeichnet $F : \mathbb{R}^n \to \mathbb{R}$ die Zielfunktion. Bei dieem Problem werden lediglich sogenannte Box-Constraints verwendet, also Nebenbedingungen der Form $l_i \leq x_i \leq u_i$. Dies ist für allgemeine Optimierungsprobleme generell eine große Einschränkung, allerdings für in dieser Arbeit betrachtete Probleme nur bedingt, da Kontrollprobleme die Steuerung und auch die Anfangswerte meist nur durch feste Werte begrenzen. Beispielsweise bei einem Wagen, der nur einen bestimmten Beschleunigungsbereich oder Einschlagwinkel der Räder zulässt, lassen sich die Beschränkungen so beschreiben.

Das Optimierungsproblem DOCP(g) lässt sich leicht in diese Form bringen. Dazu verwendet man als Optimierungsvariable $x = (u_0, \ldots, u_{N_t-1}, x_0)$ mit passenden Vektoren l und u, so dass $x_0 \in X_0$ und $u_i \in U$ für $i = 0, \ldots, N_t - 1$ gilt. Falls $X_0 = \{x_0\}$, so lässt sich das Optimierungsproblem reduzieren, indem man $x = (u_0, \ldots, u_{N_t-1})$ wählt.

Für die Klasse von Problemen existieren viele mögliche Lösungsverfahren, wobei die meisten allerdings lediglich lokale Lösungen liefern, was bedeutet, das ein gefundenes Optimum nur in einer beschränkten Umgebung optimal sein muss. Ein globales Optimum zu finden ist um ein vielfaches schwieriger.

Im Verlauf dieser Arbeit wurden dabei zwei Lösungsverfahren betrachtet, welche beide Teil der NAG-Bibliothek sind. Beim einen handelt es sich um einen Löser, der das Problem mit Hilfe einer Multi-Level Koordinaten Suche löst, wohingegen der andere ein Partikel Schwarm Optimierer ist.

Im Folgenden sollen diese Optimierer näher hinsichtlich ihrer allgemeinen Funktionsweise beschrieben werden ohne zu sehr ins Detail zu gehen.

3.2. Multilevel Coordinate Search (MCS)

Dieser Optimierer ist in der Lage, ein globales Optimum (Minimum oder Maximum) einer beliebigen Funktion zu finden und ist in der NAG-Dokumentation unter der Bezeichnung E05JBF geführt. Konvergenz wird allerdings nur garantiert, wenn die Zielfunktion in der nähe des globalen Optimums stetig ist.

Zunächst sei erwähnt, dass für diesen Abschnitt sowohl die Online-Dokumentation des Verfahrens [16] und der zugehörige Artikel [15] als Quelle verwendet wurde. Ferner wurde für einige Detailfragen der Matlabsourecode [18] des Verfahren zu Rate gezogen, da die oben genannte Dokumentation bei einigen Feinheiten lückenhaft ist. Allerdings ist dieser durch effiziente Speicherung der Informationen stellenweise auch hier keine große Hilfe, so dass in dieser Arbeit einige Details des Algorithmus unter Umständen nicht exakt beschrieben werden können. Für ein grundlegendes Verständnis sind diese allerdings auch nicht vonnöten.

Die Idee hinter dem Verfahren ist die, dass man den Suchraum in kleinere Räume unterteilt. Diese Partitionierung ist dabei nicht-uniform, sondern unterteilt Bereiche in denen kleine Funktionswerte erwartet werden weiter auf.

Von der durch die Grenzen des Optimierungsproblems definierten Startbox welche den kompletten Suchraum absteckt und sich durch zwei gegenüberliegende Randpunkte vollständig beschreiben lässt, werden Gebiete die bessere Funktionswerte enthalten weiter verfeinert, aber auch noch nicht ausreichend untersuchte Bereiche unterteilt. Jeder Teilbox wird ein Wert s zugeordnet, mithilfe dessen eine Abschätzung über die Anzahl der benötigen Splits möglich ist, welche zur Erzeugung dieser nötig waren. Es kann davon ausgegangen werden, dass Boxen mit kleinem s relativ groß sind, wohingegen für großes s ein Bereich bereits sehr fein abgedeckt wird. $s_{\max}$ begrenzt die Splittiefe und Boxen mit diesem Wert werden nicht mehr weiter unterteilt. Boxen werden immer in einer nach gewissen Regeln bestimmten Koordinatenrichtung aufgespalten, diese hängen sowohl vom aktuellen Level der Box als auch von bisherigen Aufteilungen die nötig waren um zu einer Box zu kommen ab.

Jede Box wird durch einen Basispunkt x, welcher sich meist auf einer Ecke befindet und dem am weitest davon entferntesten Eckpunkt y durch $B[x, y]$ identifiziert.

Am Beispiel einer Six-Hump-Funktion welche von der Form

$$f(x) = (4 - 2.1x_1^2 + \frac{1}{3}x_1^4)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2$$

ist und auf dem betrachteten Bereich $x_1 \in [-3, 3]$, $x_2 \in [-2, 2]$ sechs lokale Minima besitzt, lässt sich demonstrieren, wie eine mögliche Unterteilung des Suchraums durch das MCS-Verfahren aussehen kann. Diese Funktion besitzt zwei globale Minima bei $x^\star = (0.0898, -0.7126)$ und $x^{\star\star} = (-0.0898, 0.7126)$ mit $f(x^\star) = f(x^{\star\star}) = -1.0316$ und ist eine bekannte Funktion zum Testen globaler Optimierer (siehe [23]).

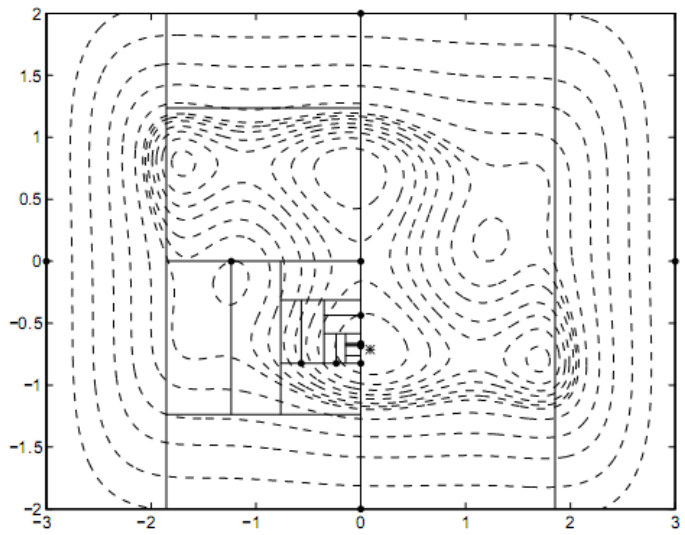


Abbildung 3.1.: Six-Hump-Funktion mit MCS minimiert (vgl. [15])

Die Punkte symbolisieren dabei die Basispunkte x, $\star$ ist hingegen ein globales Minimum.

3.2.1. Initialisierung und Sweeps

Falls eine Box das erste mal in der i-ten Koordinatenrichtung gesplittet wird, wird auf vom Nutzer bereitgestellte Werte und einige adaptiv bestimmte Zwischenwerte zurückgegriffen. Diese Werte werden über Initialisierungslisten der Form $l_i \leq x_i^1 < \cdots < x_i^{L_i} \leq u_i$ mit $L_i \geq 3$ eingegeben. Ferner benötigt man noch einen Startpunkt x^0, dessen Koordinaten ebenfalls Teil dieser Listen sein müssen, also $x_i^0 = x_i^j$ für ein $j \in \{1, \ldots, L_i\}$. Mit diesem Startpunkt lassen sich auch gute Startschätzungen für das globale Optimum angeben.

Zu Beginn des Algorithmus wird $x^\star = x_0$ gesetzt, von dieser ersten Schätzung für das globale Optimum aus wird folgende Prozedur durchlaufen um eine Menge an Initialisierungspunkten mit zugehörigen Zielfunktionswerten zu erzeugen.

Algorithm 2 Initialisierungsprozedur

for $i = 1, \ldots, n$ **do**
  for $j = 1, \ldots, L_i$ **do**
    Berechne $(\hat{x}^j, f_i^j)$ mit $\hat{x}_k^j = \begin{cases} x_k^\star & \text{falls } k \neq i \\ x_k^j & \text{sonst} \end{cases}$ und $f_i^j = F(\hat{x}^j)$
  end for
  Suche $j \in \{1, \ldots, L_i\}$, so dass f_i^j minimal ist.
  Setze $x^\star = \hat{x}^j$
end for

$x^\star$ wird später immer dann upgedated, wenn ein Basispunkt mit besserem Funktionswert als das bisherige $x^\star$ gefunden wird.

Mithilfe der errechneten Initialisierungspunkte und zugehörigen Funktionswerte lässt sich eine Startmenge an Boxen erstellen, die den anfänglichen Suchraum grob unterteilen. Dabei iteriert man über jede Koordinate $i = 1, \ldots, n$ und splittet die aktuelle Box abhängig davon, ob die Randpunkte der Initialisierungslisten auf den Grenzen liegen in $2L_i - 2$, $2L_i - 1$ oder $2L_i - 2$ Teilboxen. Es wird sowohl an den durch die Initialisierung gegebenen Punkten, als auch an jeweils einem Zwischenpunkt z_i^j geteilt, welcher gegeben ist durch $z_i^j = \hat{x}_i^{j-1} + q^m \left(\hat{x}_i^j - \hat{x}_i^{j-1} \right)$ mit $q = \frac{\sqrt{5}-1}{2}$, also dem Goldenen Schnitt, und $m \in \{1, 2\}$. m wird hier so gewählt, dass der Bereich welcher den kleineren Funktionswert an $\hat{x}_i^j$ und $\hat{x}_i^{j+1}$ besitzt den größeren Anteil erhält. Wenn die aufzuspaltende Box das Level s besitzt, so erhält der größere der beiden entstehenden Teilboxen das Level $s+1$ und der kleineren wird $\min\{s+2, s_{\max}\}$ zugeordnet. Die Box welche den gesamten Suchraum repräsentiert erhält zu Beginn das Level $s = 1$.

Beim Schritt $i \to i+1$, also bei der Betrachtung der nächsten Koordinatenrichtung, wird jene Subbox aus Schritt i zum Unterteilen ausgewählt, welche den besten Funktionswert am zugehörigen Basispunkt besitzt. Dies ist aufgrund der Regelung mit dem Goldenen Schnitt automatisch eine Box, welche das Level $s+1$ erhalten hat.

Für zwei der vom Optimierer gelieferten Standardeinstellungen kann die Partitionierung am Anfang beispielsweise wie folgt aussehen:

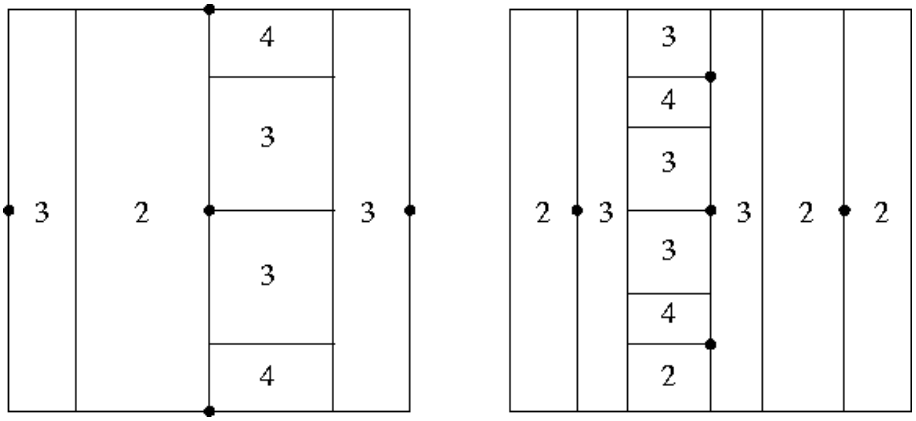


Abbildung 3.2.: Initialisierung der Boxen bei MCS (vgl. [15]), links MODE 0, rechts MODE 1 (siehe 5.5)

Hier lässt sich der Zusammenhang zwischen dem goldenen Schnitt und den entsprechenden Levels gut erkennen.

Anschließend werden die Koordinatenrichtungen abhängig von einer Abschätzung für die Variabilität der Zielfunktion bewertet und in einen Vektor ϕ abgelegt, so dass die Koordinate i die ϕ_i-höchste Variabilität besitzt. Für diese Abschätzung werden für jede Koordinate i drei beliebige aufeinanderfolgende $\hat{x}_i^j$, $\hat{x}_i^{j+1}$ und $\hat{x}_i^{j+2}$ betrachtet, über diese drei Punkte quadratisch interpoliert und jeweils das Minimum und Maximum dieser Interpolation auf dem Intervall $[\hat{x}_i^j, \hat{x}_i^{j+2}]$ gesucht. Dies macht man für alle drei aufeinanderfolgenden Punkte der Initialisierungliste und schätzt die Variabilität der Zielfunktion in i-ter Koordinate über die Differenz zwischen dem größten Maximum und dem kleinsten Minimum all dieser Interpolationen ab.

Zu Beginn erhält die Basisbox das Level $s = 1$, wann immer eine Box gesplittet wird, werden wie bereits erwähnt den entstehenden Subboxen je nachdem wie und wo unterteilt wird, das Level $s + 1$ oder $s + 2$ zugeordnet. Bei der Aufspaltung wird aber nicht jede Box auf einem Level s direkt gesplittet, sondern dies läuft in sogenannten Sweeps ab. Ein Sweep besteht dabei aus drei Schritten:

Algorithm 3 Sweep in MCS

1. Betrachte alle noch nicht unterteilten Boxen und erstelle eine Recordliste b_s, wobei b_s auf die Box zeigt, welche den niedrigsten Funktionswert am Basispunkt aller Boxen auf dem gleichen Level besitzt. Falls auf einem Level s keine Box die noch nicht gesplittet ist, setzte $b_s = 0$ und setze s auf den kleinsten Wert, so dass $b_s \neq 0$.

2. Die Box b_s ist ein Kandidat zum Aufspalten. Falls sie nicht gesplittet wird, setze ihren Level auf $s + 1$ und aktualisiere falls nötig die Recordliste b_s.

3. Setze $s = s + 1$. Falls $s = s_{\max}$, starte einen neuen Sweep (also gehe zu 1.). Wenn $b_s = 0$ für das neue s ist, wiederhole den dritten Schritt, bis entweder $b_s \neq 0$ (also ein Kandidat zur näheren Betrachtung gefunden wurde) oder $s = s_{\max}$.

Beim Aufteilen einer Box in eine unbeschränkte Richtung oder einer Koordinate deren zugehörige Ausmaße der Box sehr groß sind, möchte man dass die neuen Splitbereiche nicht zu groß sind. Wenn man also in eine Koordinatenrichtung unterteilen will deren

Ausmaße $[x_i, y_i]$ sind, so möchte man das die Positionen an denen eine Box geteilt wird zumindest in $[\xi^1, \xi^2] \subset [x_i, y_i]$ liegt, wobei

$$\begin{aligned}
&\xi^1 = \text{subint}(x_i, y_i)\\
&\xi^2 = x_i + \tfrac{1}{10}(\xi^1 - x_i)\\
&\text{subint}(x, y) = \begin{cases} \text{sgn}(y) & \text{falls } 1000|x| < 1 \text{ und } |y| > 1000 \\ 10\text{sgn}(y)|x| & \text{falls } 1000|x| \geq 1 \text{ und } |y| > 1000|x| \\ y & \text{sonst} \end{cases}
\end{aligned}$$

gilt.

3.2.2. Unterschiedliche Splitarten

Splitten nach Rangbedingung

Abhängig vom Level einer Box wird entschieden, wie und ob sie aufgespaltet wird. Falls $s > 2n(\min_{j=1,\ldots,n}(n_j) + 1)$, wobei n_j die Anzahl der bisherigen Unterteilungen der j-ten Koordinate ist welche nötig war um zur aktuellen Teilbox zu gelangen, so ist die Box schon näher betrachtet worden, wurde aber in einer oder mehrere Koordinatenrichtungen noch nicht häufig genug unterteilt. In diesem Fall wird immer in j-ter Richtung gesplittet, so dass n_j minimal ist. Falls mehrere $n_i = n_j$ für $i \neq j$ existieren, so wird jene Koordinate gewählt, welche die größere Variabilität besitzt, also jene welche das kleinere ϕ_i aus der Initialisierung besitzt.

Falls in diese Richtung noch nicht gesplittet wurde, wird auf die durch die Initialisierungsliste gegebenen Positionen zurückgegriffen und zusätzlich noch an durch den goldenen Schnitt bestimmten Zwischenpunkten unterteilt, andernfalls wird an der Stelle $z_i = x_i + \frac{2}{3}(\text{subint}(x_i, y_i) - x_i)$ und zusätzlich an der durch den goldenen Schnitt bestimmten Punkt. Hier wird wie bei der Initialisierung der Teilbox, welche den kleineren Anteil des goldenen Schnitts erhält das Level $\max\{s + 2, s_{\max}\}$ zugeordnet und den zwei anderen das Level $s + 1$.

Leider liefert die Dokumentation auch hier keine genaue Beschreibung hinsichtlich der exakten Positionen, weshalb ein Blick in den Sourcecode hilfreich ist. Die Dateien `split.m` und `split1.m` von [18] bestätigen die Annahme, dass das Intervall $[x_i, z_i]$ betrachtet wird und dieses gemäß dem goldenen Schnitt geteilt wird. Hier wird wieder abhängig von den Zielfunktionswerten $f(x_i)$ und $f(z_i)$ entschieden, welcher Subbox den größeren Anteil enthält.

Splitten nach erwarteter Verbesserung

Sollte hingegen $s \leq 2n(\min_{j=1,\ldots,n}(n_j) + 1)$ so sind alle Koordinaten in vorherigen Schritten ähnlich gut unterteilt worden. An dieser Stelle wird die Koordinate gewählt, welche die beste erwartete Verbesserung des bisher gefundenen Zielfunktionswertes bietet. In diesem Fall wird für jede Koordinate die beste Splitvariable und zugehöriger Funktionswert errechnet. Sollte dabei in einer Koordinatenrichtung noch nicht gesplittet worden sein, so wird die erwartete Verbesserung basierend auf den Funktionswerten den Punkte entsprechend der Initialisierungsliste errechnet. Andernfalls wird über vorherige Basispunkte und deren Funktionswerte quadratisch interpoliert und das Minimum z_i der darüber erhaltenen Funktion auf $[\xi^1, \xi^2]$ als beste erwartete Verbesserung verwendet.

Wenn der am besten zu erwartende Zielfunktionswert welcher über diese Methode gefunden wurde nicht besser, als das bisher beste global gefundene Optimum ist, so wird die Box nicht unterteilt und lediglich der Level um eins erhöht. Dies kann natürlich mehrmals

hintereinander passieren, so dass früher oder später der Level so groß ist, dass die Box gemäß der Rangregelung unterteilt wird.

An dieser Stelle wird klar, warum der Algorithmus für $s_{\max} \to \infty$ garantiert das globale Optimum findet, denn je größer $s_{\max}$ wird, desto öfter tritt die Regelung zum Splitten nach Rangbedingung in Kraft, so dass jeder Punkt bis auf beliebige Genauigkeit über Basispunkte angenähert wird.

Falls hingegen der erwartete Zielfunktionswert besser wird, so wird an der gefundenen Stelle z_i gesplittet und zusätzlich wieder wie bei dem Splitten nach Rangbedingung am durch den goldenen Schnitt bestimmten Punkt in $[x_i, z_i]$. Abhängig davon, wo die beste Abschätzung liegt, entstehen so zwei oder drei Subboxen, falls nämlich der gefundene Punkt an einer Grenze der Box liegt, so wird lediglich gemäß des goldenen Schnittes unterteilt. Je nach relativer Größe zueinander wird den Subboxen entweder das Level $s+1$ oder $\max\{s+2, s_{\max}\}$ zugeordnet. Die durch den goldenen Schnitt erhaltenen Subboxen erhalten wie vorher das Level $s+1$ für die größere Teilbox und $\max\{s+2, s_{\max}\}$ für die kleinere. Die dritte Teilbox bekommt wenn vorhanden das Level $s+1$ falls sie größer als die kleinere Teilbox des goldenen Schnittes ist, andernfalls wird auch ihr der Wert $\max\{s+2, s_{\max}\}$ zugeordnet.

3.2.3. Local Search

Mithilfe einer Local Search Option lässt sich die Konvergenz des Algorithmus beschleunigen. Zu diesem Zweck wird ein sogenannter Shopping Basket mit nützlichen Punkten angelegt. Dazu wird am Ende eines Sweeps für die Boxen $B[x, y]$, welche das Level $s = s_{\max}$ erreicht haben ausgehend von deren Basispunkt x lokal optimiert und falls ein noch nicht in der Liste enthaltener Punkt gefunden wird, dieser und dessen Funktionswert im Basket abgelegt. Genauer gesagt wird für einen Kandidaten x für den Shopping Basket zuerst getestet, ob er in Einzugsbereich eines lokalen Minimums welches bereits im Basket liegt ist. Falls dies nicht der Fall ist, wird ausgehend von dem Basispunkt mithilfe von Triple Search, einem Verfahren welches ein quadratisches Modell basierend auf drei Punkten x^l, x^m und x^u aufstellt, lokal optimiert. Die Punkte werden hier durch sukzessive Line Search-Verfahren bestimmt. Daraufhin wird nocheinmal getestet, ob das erhaltene Minimum nicht doch im Einzugsbereich eines bereits im Basket liegenden Punktes liegt und nur dann zu diesem hinzugefügt, wenn dies nicht der Fall ist.

Der Test ob ein Punkt im Einzugsgebiet eines lokalen Minimums liegt ist natürlich nur eine Abschätzung. Man testet hier für jeden Punkt w im Shopping Basket für einen Basispunkt x ob die Zielfunktion von x Richtung w monoton fallend ist. Hier wird an zwei Zwischenpunkten $x' = x + \frac{1}{3}(w-x)$ und $x'' = x + \frac{2}{3}(w-x)$ der zugehörige Zielfunktionswert $f(x')$ und $f(x'')$ berechnet.

Durch diese Methode ist eine kleinere maximale Splitanzahl nötig um gute Ergebnisse zu erzielen, da es genügt, wenn ein Basispunkt einer Box im Einzugsgebiet eines lokalen Minimums liegt. Ferner muss man so weniger häufig wegen der Regelung zum Splitten nach der erwarteten Verbesserung im Zielfunktionswert unterteilen, da diese Regel vom bisher besten Zielfunktionswert abhängt. Dieser wird durch die lokale Optimierung schneller verbessert als durch einfaches Splitten und anschließendes Betrachten der Funktionswerte an den Basispunkten.

Darüberhinaus sorgt man mit den lokalen Minima im Shopping Basket und dem Test ob ein Basispunkt bereits im Einzugsbereich eines schon bekannten Minimums ist dafür, die Anzahl der lokalen Optimierungen mittels Triple-Search niedrig zu halten. Beim anschließenden Test ob das gefundene lokale Optimum im Einzugsbereich eines bereits bekannten Minimums liegt versucht man die Zahl der Elemente im Shopping Basket zu beschränken.

Die Stärken dieser Methode lassen sich besonders bei Funktionen mit vielen lokalen Minima beobachten, ein typisches Beispiel hierfür ist die Rastrigin-Funktion, welche von der Form

$$f(x) = An \sum_{i=1}^{n} [x_i^2 - A\cos(2\pi x_i)]$$

ist.

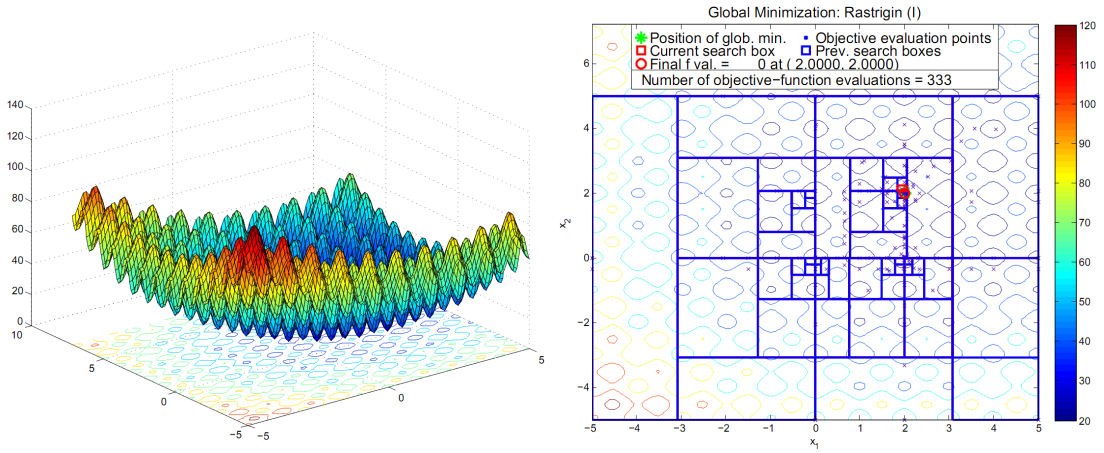


Abbildung 3.3.: Modifizierte Rastrigrin-Funktion [17]

Hier wurde eine leicht abgewandelte Version der Rastrigrin-Funktion verwendet.

$$f(x) = 10n \sum_{i=1}^{n} [(x_i - 2)^2 - A\cos(2\pi(x_i - 2))]$$

Diese Variante hat ihr globales Minimum bei $x^\star = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$ mit $f(x^\star) = 0$.

Bei Funktionen dieser Art müssen Bereiche häufig nicht näher untersucht werden, gleichzeitig allerdings auch aufgrund des Shopping Basket nicht zu oft lokal optimiert werden. Die Grafik zeigt auch, das hier nicht zu viele Zielfunktionsauswertungen nötig sind um gute Ergebnisse zu liefern. Dies ist besonders bei komplexeren Zielfunktionen, wie sie bei der in dieser Arbeit betrachteten Methode zur Berechnung erreichbarer Mengen auftreten, wichtig.

3.2.4. Pseudocode des Verfahrens

Um ein Gesamtbild des Algorithmus zu erhalten, ist es hilfreich das Verfahren als Ganzes in Form des folgenden Pseudocodes zu betrachten.

Algorithm 4 Multi-Level Coordinate Search - Algorithmus

Initialisierung
Berechne Initialisierungspunkte, deren Funktionswerte, die Variabilität ϕ, eine Startschätzung x_{best} mit f_{best} als bisher bestes globales Optimum und erzeuge eine Startmenge an Boxen gemäß Initialisierungsliste.
Erstelle Recordliste b_s.
while $s < s_{\max}$ **do**
  Finde die beste Box auf Level s über Recordliste b_s
  if $s > 2n(\min(n_j) + 1)$ **then**
    Wähle Koordinate nach Rang-Bedingung und markiere die Box zum Splitten
  else
    Suche Koordinate, welche die beste erwartete Verbesserung liefert.
    if keine Koordinatenrichtung gefunden, welche Optimum verbessert **then**
      Erhöhe den Level s der aktuellen Box um 1
    else
      Markiere aktuelle Box zum Splitten in gefundener Koordinatenrichtung
    end if
  end if
  if Aktuelle Box zum Splitten markiert **then**
    if In Koordinatenrichtung wird das erste mal gesplittet **then**
      Splitte an Punkten gegeben durch Initialisierungsliste und zusätzlich dazu an Zwischenpunkten die die Intervalle gemäß des Goldenen Schnitts teilen.
    else
      Splitte an Punkten, die durch die Splitregelung nach Rang oder erwarteter Verbesserung im Zielfunktionswert bestimmt sind.
    end if
    Update x_{best} und f_{best}
  end if
  if $s = s_{\max}$ **then**
    Wende falls gewünscht Local-Search-Methode zur Beschleunigung des Verfahrens an.
    Update x_{best} und f_{best}
    Beginne einen neuen Sweep, also bestimme neue Recordliste
    if $b_s = 0$ für alle Level s **then**
      return x_{best}, f_{best}
    end if
  end if
end while

3.2.5. Konvergenz des Verfahrens

Die folgenden Konvergenzaussagen sind in [15] zu finden.

Satz 10: (Konvergenz MCS)
Angenommen das nichtlineare Optimierungsproblem mit Boxed Constraints der Form

$$\begin{aligned} \min \quad & f(x) \\ \text{sd.} \quad & l \leq x \leq u \end{aligned}$$

besitzt eine Lösung in $\hat{x} \in [l, u]$ wobei $f : [l, u] \to \mathbb{R}$ in einer Umgebung von $\hat{x}$ stetig ist.

Sei $\epsilon > 0$, dann existiert ein s_0 so dass für alle $s_{\max} \geq s_0$ ein ein Basispunkt x gefunden wird, welcher $f(x) < f(\hat{x}) + \epsilon$ erfüllt.

Im schlechtesten Fall wird ein solcher Basispunkt gefunden, wenn alle Boxen das Level $s_{\max}$ besitzen, also die Menge aller Boxen mit Level $s < s_{\max}$ leer ist.

Die Anzahl an Sweeps die dafür nötig ist, ist höchstens $(p^{s_0-1} - 1)/(p - 1)$, wobei p die maximale Anzahl an Boxen ist, welche bei einer Unterteilung entstehen. Da innerhalb des Algorithmus pro Split maximal 3 Boxen entstehen, hängt p ausschließlich von den gewählten Initialisierungslisten ab.

Angesichts dessen, dass das globale Optimierungsproblem NP-schwer ist, ist diese exponentielle Entwicklung zu erwarten.

Für MCS mit dem weiter oben vorgestellten Local Search Ansatz lässt sich eine stärkere Konvergenzaussage treffen.

Satz 11: (Konvergenz MCS mit Local Search)
Zusätzlich zu den Voraussetzungen der obigen Konvergenzaussage nimmt man an, dass ein $\epsilon > 0$ existiert, so dass $f(y) > f(\hat{x}) + \epsilon$ für alle nichtglobalen lokalen Minimierer y der Zielfunktion und für alle $y \in [u, v]$ mit ausreichend großer Norm.

Dann existiert $L, S \in \mathbb{N}$, so dass für alle $s_{\max} > L$ der MCS-Algorithmus mit Local Search nach höchstens S Sweeps einen globalen Minimierer findet.

Diese Aussage kommt aufgrund dessen zustande, dass es genügt in den Attraktionsbereich des globalen Optimums zu kommen und man nicht notwendigerweise alle Boxen bis zum Level $s_{\max}$ unterteilen muss um das globale Optimum über Basispunkte anzunähern.

weitere Informationen

Die exakten Konvergenzsätze des Verfahrens sowohl mit als auch ohne Local Search sind unter [15] zu finden. Näheres zur Implementierung und weitere Details bezüglich der Local Search, eine Beschreibung des lokalen Triple Search Verfahrens und genauere Informationen zur Berechnung des erwarteten Zielfunktionswerts bei der Regelung zum Splitten nach erwarteter Verbesserung ist der Online Dokumentation der NAG-Library [16], dem zum Verfahren gehörigen Artikel [15] und dem ausreichend dokumentierten Matlabsourcecode [18] zu entnehmen.

3.3. Particle Swarm Optimization (PSO)

Anders als MCS ist der Ansatz bei Partikel Schwarm Optimierern (in NAG unter E05SAF zu finden) nicht den Suchraum zu unterteilen, sondern viele Partikel mit unterschiedlicher Initialisierung loszuschicken und in jedem Schritt sowohl hinsichtlich eigener Werte also auch Funktionswerte aller Partikel zu optimieren. Jeder Partikel x_j wird rekursiv über $x_j^{i+1} = x_j^i + v_j^{i+1}$ fortgeführt. v_j gibt dabei die Richtung an, in die sich der betrachtete Partikel bewegt und hängt neben dem Optimum des aktuellen Partikels auch von dem aller Partikel und jeweils zugehöriger Gewichte ab.

Das Verfahren endet, wenn ein passendes Abbruchkriterium erreicht ist, beispielsweise wenn ausreichend viele Partikel gegen ein Optimum konvergiert sind, ein bestimmter Zieloptimalwert erreicht ist, eine Anzahl an Funktionsauswertungen überschritten wird oder sich der Zielfunktionswert nach einer gewissen Anzahl an Iterationen nicht mehr ändert.

Der Algorithmus hängt dabei stark von zufällig erzeugten Variablen ab und ist in hohem Grade heuristisch. Sofern der Optimalwert nicht bekannt ist, lässt sich die Qualität der

Lösung nicht abschätzen, mehrere Durchläufe können unterschiedlichste Lösungen generieren.

Die durch die NAG-Bibliothek bereitgestellte Variante, dessen Dokumentaion unter [19] zu finden ist und für diesen Abschnitt als Quelle verwendet wurde, macht von folgender Notation Gebrauch:

n_{par}	Anzahl der erzeugten Partikel
x_j	j-ter Partikel
f_j	Funktionswert des Partikels x_j, also $f(x_j)$
v_j	Bewegungsrichtung oder Geschwindigkeit des j-ten Partikels
$\hat{x}_j$	bisher bester Kandidat für Optimum bezüglich Partikel x_j
$\hat{f}_j$	zugehöriger Optimalwert, also $\hat{f}_j = f(\hat{x}_j)$
$\tilde{x}$	bisher bester Kandidat für Optimum bezüglich aller Partikel
$\tilde{f}$	zugehöriger Optimalwert, also $\tilde{f} = f(\tilde{x})$
$R \in U(l, u)$	zufällig gewählter Vektor passender Dimension mit R_i auf $[l_i, u_i]$ gleichverteilten Komponenten

Jeder Partikel und dessen beste Position wird zu Beginn auf gleichmäßig verteilte Zufallswerte des Suchraums gesetzt, also $x_j = R \in [l, u]$ und $\hat{x}_j = R \in [l, u]$. Gleiches gilt für die Startrichtung $v_j = R \in U(-V_{\max}, V_{\max})$ allerdings bezüglich $V_{\max}$, der maximalen Bewegungsrichtung welche abhängig von dem Suchraum $[l, u]$ bestimmt wird.

In jedem Schritt wird die Bewegungsrichtung gemäß der Formel

$$v_j = w_j v_j + \underbrace{C_s D_1(\hat{x}_j - x_j)}_{\text{kognitiver/sozialer Anteil}} + \underbrace{C_g D_2(\tilde{x} - x_j)}_{\text{globaler Anteil}}$$

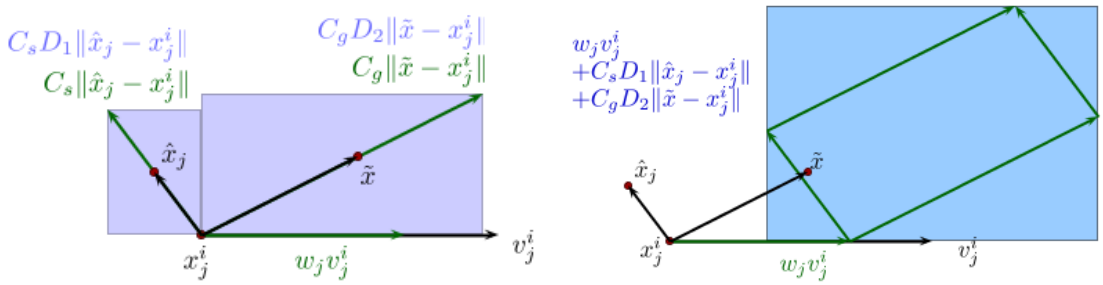
aktualisiert und anschließend die neue Position mit $x_j = x_j + v_j$ berechnet. Der kognitive Anteil sorgt dabei dafür, dass der Partikel zu seiner bisher besten Position strebt, wohingegen der globale Anteil den Partikel in Richtung der bisher besten globalen Position, also dem besten Wert aller Partikel, drängt.

D_1 und D_2 sind Diagonalmatrizen, deren Diagonalelemente in jedem Schritt zufällig aus dem Bereich $[0, 1]$ gleichverteilt gewählt werden, also $D_i = \text{diag}(r_1, \ldots, r_n)$ mit $r_i \in U(0, 1)$.

C_s und C_g sind dagegen Konstanten, welche beeinflussen wie stark ein Partikel gegen die jeweiligen Positionen gesteuert werden. Hohe Werte haben zur Folge, dass Partikel über die jeweiligen Positionen hinausgeschleudert werden. Wenn man C_s zu groß wählt, kann es sein, dass die Partikel nicht mehr gegen das bisher beste globale Optimum konvergieren, sonder sich zerstreuen. Auch wenn man C_g zu groß wählt, wird unter Umständen zu weit über $\tilde{x}$ hinausgeschleudert und eine Konvergenz verhindert. Aus diesem Grund ist es ratsam, C_s und C_g so zu wählen, dass $0 \leq C_s, C_g \leq 2$ gilt, wobei aber mindestens $C_s \neq 0$ oder $C_g \neq 0$ gelten muss.

w_j ist ein Parameter, welche die vorherige Bewegungsrichtung gewichtet und bewegt sich in einem Bereich von $[w_{\min}, w_{\max}]$ mit $0 \leq w_{\min} \leq w_{\max} \leq 1$. Diese Gewichte werden falls gewünscht nach bestimmten Regeln initialisiert und in jedem Schritt aktualisiert, allerdings gilt $w_j^{i+1} \leq w_j^i$. Zur Initialisierung kann man man w_j auf $w_{\max}$ setzen, es mit einem gegebenen Wert w_{init} initialisieren, oder man wählt einen zufälligen Wert $w_j = R \in U(w_{\min}, w_{\max})$. Die Gewichtung ist wie bereits erwähnt monoton fallend bezüglich der Iterationsschritte, es steht also dem Benutzer frei w_j konstant auf dem initialisierten Wert zu lassen, ihn bis zu einer maximalen Iterationsstufe linear fallen zu lassen oder in jedem Schritt um einen bestimmten Prozentsatz zu reduzieren.

In jeder Iteration wird also für zufällig gewählte D_1 und D_2 die neue Bewegungsrichtung bestimmt. Dadurch lässt sich zwar nicht die exakte Bewegungsrichtung in jedem Schritt angeben, allerdings lässt sich so bestimmen, in welchem Bereich sich ein Partikel nach der Iterationsvorschrift x_j^{i+1} aufhalten kann.



(a) Mögliche Kandidaten für Attraktionsvektoren (b) Bereich in dem sich der Partikel im nächsten Schritt aufhält

Abbildung 3.4.: PSO-Iterationsschritt, Illustration

Für Diagonalmatrizen der Form $D_i = r\ \mathrm{diag}(1, \ldots, 1)$ mit $r \in U(0, 1)$, also einer etwas vereinfachten Form des Partikelschwarmverfahrens bei dem nur die Länge der Vektoren durch einen skalaren Wert verändert wird, kann ein Iterationsschritt so aussehen:

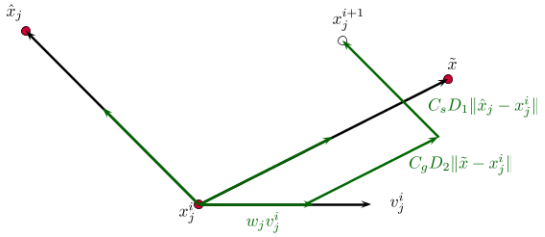


Abbildung 3.5.: PSO-Iterationsschritt für vereinfachtere Variante

Falls der Partikel x_j bei $\tilde{x}$ landet, so wird dieser zusammen mit seiner Startbewegungsrichtung v_j, Gewichtung w_j und $\hat{x}_j$ neu initialisiert.

Nun kann v_j so gewählt sein, dass das resultierende x_j den Suchraum verlässt. An dieser Stelle gibt es unterschiedliche Herangehensweisen, wie man weiter fortfährt. Die durch NAG bereitgestellten Strategien sind durch die hier aufgelisteten Optionen gegeben:

- IGNORE - Ignoriere die Boxen und fahre einfach fort.
- RESET - Reinitialisiere den Partikel, ohne dabei aber die bisher beste Position zu vergessen.
- FLOATING - Belasse den Partikel außerhalb des Suchraums, beziehe aber die resultierenden Funktionswerte nicht mit in Berechnungen ein. Der Partikel wird möglicherweise durch die kognitive und globale Attraktion wieder in den Suchraum gesteuert.
- HYPERSPHERICAL - Der Suchraum wird als n-dimensionale Hypersphere aufgefasst, falls also ein Partikel den Suchraum auf einer Grenze verlässt, kommt er an der anderen Seite wieder hinein.

- FIXED - Der Partikel wird auf die bei den übertretenen Grenzen zurück auf die Grenze gesetzt und sicherzustellen, dass der Suchraum nicht verlassen wird.

Nach der Anwendung dieser Regel, muss man $\hat{x}_j$, $\hat{f}_j$, $\tilde{x}$ und $\tilde{f}$ updaten. Falls sich das globale Optimum $\tilde{f}$ in einem Schritt verbessert hat, so kann man einen lokalen Minimierer mit dem Startwert $(\tilde{x}, \tilde{f})$ auf das Problem anwenden. Der Algorithmus stellt dabei verschiedene lokale Optimierer bereit, für diese Arbeit werden allerdings nur jene verwendet, welche lediglich den Zielfunktionswert und die Grenzen benötigen ohne auf den Gradienten der Zielfunktion zurückzugreifen.

Zusammengefasst funktioniert der Algorithmus also gemäß des folgenden einfachem Schemas:

Algorithm 5 Partikel Schwarm Algorithmus

Initialisiere: $x_j = R \in U(l, u)$
$\hat{x}_j = R \in U(l, u)$
$v_j = R \in U(-V_{\max}, V_{\max})$
$\hat{f}_j = f(\hat{x}_j)$
$w_j = \begin{cases} w_{\max} \\ w_{\text{init}} \\ R \in U(w_{\min}, w_{\max}) \end{cases}$ je nach gewünschter Initialisierung

while Abbruchkriterium noch nicht erreicht **do**
  for all Partikel x_j **do**
    wende BOUNDARY-Regel auf x_j an
    update $\hat{x}_j$, $\hat{f}_j$, $\tilde{x}$ und $\tilde{f}$
  end for
  if $\tilde{f}$ hat sich verbessert **then**
    versuche $\tilde{f}$ mit lokalem Optimierer mit Startwert $\tilde{x}$ weiter zu verbessern
  end if
  for all Partikel x_j **do**
    Berechne v_j neu
    Wende Richtungsvektor v_j auf x_j an
    if x_j nahe $\tilde{x}$ **then**
      initialisiere x_j, $\hat{x}_j$, w_j und v_j neu
    else
      wende falls gewünscht Abstiegsregel auf w_j an
    end if
  end for
end while
versuche den besten Kandidaten $\tilde{x}$ mit einem lokalen Optimierer zu verbessern
return $\tilde{x}$, $\tilde{f}$

Konvergenzaussagen ließen sich bisher für das Partikelschwarmverfahren nicht treffen, was sich auch in den Abbruchkriterien widerspiegelt. Zwar lässt sich die Bewegung eines einzelnen Partikels näher analysieren, indem man ein diskretes Dynamisches System betrachtet und zumindest Konvergenz der einzelnen Partikel unter bestimmten Bedingungen an die Variablen w_j, C_s und C_g nachweisen, allerdings eben keine Konvergenz zu einem globalen Optimum.

Dieses diskrete dynamische System ist für den j-ten Partikel gegeben durch

$$\begin{aligned} v_j^{i+1} &= w_j^i v_j^i + C_s D_1(\hat{x}_j^i - x_j^i) + C_g D_2(\tilde{x} - x_j^i) \\ x_j^{i+1} &= x_j^i + v_j^{i+1} \end{aligned}$$

wofür sich unter bestimmten Voraussetzungen das Langzeitverhalten abschätzen lässt.

Eine Konvergenz der einzelnen Partikel macht wenn man sich das System betrachtet auch durchaus Sinn. Unter der Annahme, dass w_j streng monoton fallend ist, spielt $w_j^i v_j^i$ bei der Berechnung der neuen Bewegungsrichtung nach vielen Iterationen kaum noch eine Rolle. $C_s D_1(\hat{x}_j - x_j) + C_g D_2(\tilde{x} - x_j)$ ist also dann der dominante Term. Nimmt man weiter an, dass D_1 und D_2 im Mittel $\mathrm{diag}(\frac{1}{2}, \ldots, \frac{1}{2})$ sind und sich im laufe der Iterationen weder $\hat{x}_j$ noch $\tilde{x}$ ändern, wandert der Partikel auf lange Sicht abhängig von C_s und C_g auf eine Position zwischen $\hat{x}_j$ und $\tilde{x}$. Dieses Verhalten kann man auch gut in Abbildung 3.5 sehen.

Unter diesen Voraussetzung lässt sich das Dynamische System vereinfachen und eine Parameterwahl bestimmen, so dass Konvergenz vorliegt.

Dabei wird wie in [20] zunächst das System mit diesen Einschränkungen etwas vereinfacht. Statt Diagonalmatrizen D_1 und D_2 wird hier der Faktor $\frac{1}{2}$ verwendet, ferner nimmt man dort $w_j^i = a$ konstant an. Wenn man nun das eingeschränkte System mit der Substitution

$$\begin{aligned} b &= \tfrac{C_s + C_g}{2} \\ p &= \tfrac{C_s}{C_s + C_g}\hat{x}_j + \tfrac{C_g}{C_s + C_g}\tilde{x} \end{aligned}$$

betrachtet, so vereinfacht sich dieses zu

$$\begin{aligned} v_j^{i+1} &= a v_j^i + b(p - x_j^i) \\ x_j^{i+1} &= x_j^i + v_j^{i+1} \end{aligned}$$

wobei b der Durchschnitt der Attraktionskoeffizienten C_s und C_g und p der gewichtete Mittelpunkt zwischen $\hat{x}_j$ und $\tilde{x}$ ist.

In Matrixform lässt sich dieses System auch schreiben als:

$$\begin{aligned} &y_{k+1} = A y_k + B p \\ &y_k = \begin{pmatrix} x_k \\ v_k \end{pmatrix},\ A = \begin{pmatrix} 1-b & a \\ -b & a \end{pmatrix},\ B = \begin{pmatrix} b \\ b \end{pmatrix} \end{aligned}$$

Diese System besitzt das Equilibrium $y^\star = \begin{pmatrix} p \\ 0 \end{pmatrix}$, also $x^\star = p$ und $v^\star = 0$. Nun lassen sich Parameter bestimmen, so dass dieses Equlibrium ein Attraktor ist, also so dass $\lim_{k\to\infty} x_k = p$ gilt.

Dies ist der Fall, wenn die Eigenwerte der Matrix A betragsmäßig kleiner 1 sind, was genau dann erfüllt ist, wenn die Ungleichungen

$$a < 1 \quad , \quad b > 0 \quad , \quad 2a - b + 2 > 0$$

erfüllt sind.

Wenn sich $\hat{x}_j$ oder $\tilde{x}$ ändern, ändert sich auch der Punkt zu dem sich der Partikel bewegt. Wenn man hingegen die zufällig besetzten Diagonalmatrizen nicht mittelt, so bewegt sich der Partikel in einem Bereich um diesen Punkt, so dass Konvergenzaussagen lediglich auf Bereiche möglich sein dürften.

Näheres zu diesem Dynamischen System und eine Parameterabschätzung für w, C_s und C_g unter den angemerkten Einschränkungen ist [20] zu entnehmen.

4. Implementierung

Bei der Implementierung wurde als Basis die adaptive Variante von Wolfgang Riedl aus [1] verwendet, welche neben passenden Datenstrukturen zum Speichern der Gitterpunkte und der zugehörigen Zielpunkte in den Erreichbarkeitsmenge auch das rekursive Schema aus Algorithmus 1 enthält.

Der lokale Löser wurde entfernt und ein Backend entworfen, welches es erlaubt unterschiedliche Modelle und Optimierer einzusetzen. Dazu wurde das in Fortran90 enthaltene Feature der Module genutzt, welche es erlauben einen etwas objektorientierteren Ansatz zu verfolgen.

Hierbei wurden Module für das eigentliche Kontrollproblem und für die globalen Optimierer geschrieben. Erst beim Kompilieren wird entschieden, welchen Optimierer und welches Modell man verwendet.

4.1. Modul für Kontrollsystem

Ein Modell-Modul muss dabei unterschiedliche kontrollproblemspezifische Variablen und Subroutinen bereitstellen. Neben der Dimensionen für den Zustandsraum und den Kontrollraum ist hier auch die Systemdynamik und die Grenzen für die einzelnen Kontrollwerte relevant. Ferner muss eine Menge zulässiger Anfangswerte mittels Boxconstraints oder ein einzelner Anfangswert gegeben sein.

Neben diesen kontrollsystemspezifischen Informationen werden hier auch Werte für den Adaptiven Algorithmus gesetzt, dies beinhaltet einen booleschen Wert `ADAPTIVE`, welcher festsetzt, ob man die adaptive oder die reguläre Variante des Verfahrens verwenden möchte. Zusätzlich werden hier modellspezifisch auch N_t und N_x, beziehungsweise N_t, $N_{x,0}$ und R falls man adaptiv vorgehen möchte, gesetzt und natürlich auch das betrachtete Gebiet $G = [A, B]$ über das man anschließend ein Gitter legt wird hier definiert.

Darüber hinaus gibt es neben Initialisierungs- und Cleanup-Subroutinen, welche der Speicherverwaltung und Initialisierung von Arrays dienen, noch eine Subroutine, welche es ermöglicht, eine Startsteuerung basierend auf einem gegebenen Gitterpunkt zurückzugeben um einen Optimierer falls gewünscht passend zu initialisieren.

Zusammengefasst ist das Modellmodul also wie folgt aufgebaut:

```
MODULE MODEL
  ! dimensions of state space and controll space
  INTEGER              :: DIM_X
  INTEGER              :: DIM_U

  ! start and finish time of system
  DOUBLEPRECISION      :: T0
  DOUBLEPRECISION      :: TF

  ! timesteps for euler
  INTEGER              :: NT
```

```
  ! boolean to set, if x_0 is part of a set X_0 or a single
  !  initial value is used
  LOGICAL                                  :: X0_SET

  ! initial value or boxed constraints for the set X_0
  DOUBLEPRECISION, DIMENSION(:), POINTER   :: X0
  DOUBLEPRECISION, DIMENSION(:), POINTER   :: LB_X0, UB_X0

  ! boxed constraints for the controls
  DOUBLEPRECISION, DIMENSION(:), POINTER   :: LB_U, UB_U

  ! area [Ai,Bi] for grid
  DOUBLEPRECISION, DIMENSION(:), POINTER   :: A, B

  ! use adaptive version of algorithm and
  !  number of recursion
  LOGICAL                 :: ADAPTIVE
  INTEGER                 :: NX
  INTEGER                 :: R
CONTAINS
  ! memory allocation and setting of bounds
  SUBROUTINE MODEL_INIT()

  ! system dynamic
  SUBROUTINE DAE(T, X, U, F, IUSER, USER)

  ! deallocation of memory
  SUBROUTINE MODEL_CLEANUP()

  ! returns custom initial guess for u_0 and x_0
  !  depending on the current gridpoint.
  SUBROUTINE MODEL_OPTIMIZATION_INIT(X_GRID,
                                     X0_INIT, U_INIT)
END MODULE MODEL
```

4.2. Modul für globale Optimierer

Die geschriebenen NAG-Optimierungsmodule beinhalten hingegen für die jeweiligen Optimierer spezifische Optionen und von den Routinen verlangte Variablen. Sinnvolle Initialisierungen wurden dabei über verschiedene Modi realisiert, bei MCS verschiedene Arten zur Initialisierung, bei PSO hingegen verschiedene lokale Optimierer. Obendrein wurden jeweils verschiedene Reskalierungsoptionen für die Zielfunktion implementiert.

Beide Optimierer verlangen eine Zielfunktion, in welcher das Kontrollsystem für über die Optimierungsvariable übergebene Steuerungen und falls nötig Anfangswert mittels einem dort direkt implementieren Eulerverfahren gelöst wird. Anschließend wird die reskallierte Distanz dieser Lösung zum Endzeitpunkt T zu dem über ein Benutzerarray übergebenen Gitterpunkt als Zielfunktionswert dem Optimierer zurückgegeben.

Ferner benötigen beide Optimierer eine Monitoring-Subroutine, welche es erlaubt zur Laufzeit nützliche Informationen auszugeben. Diese wurden allerdings als leere Dummy-Routinen implementiert um Ausgaben auf das nötigste zu beschränken.

Der Aufbau eines solchen Moduls ist also wie folgt:

```
MODULE NAG_OPTIMIZATION
  ! dimension of the optimization problem
  INTEGER         :: N

  ! different variabls/arrays needed from the routines
  ! (...)

  ! initialisation presets
  INTEGER         :: MODE

  ! objective-function rescale
  INTEGER         :: RESCALE
CONTAINS
  ! memory allocation, option and parameter
  ! initialization
  SUBROUTINE OPTIMIZATION_INIT()

  ! memory cleanup
  SUBROUTINE OPTIMIZATION_CLEANUP()

  ! objective-function, controllsystem is solved here
  ! for a given optimizationvaraible and a rescaled
  ! objectivevalue is returned
  SUBROUTINE OBJFUN(...)

  ! dummy monitoring-routine, nothing is done here
  SUBROUTINE MONIT(...)
END MODULE NAG_OPTIMIZATION

! the global optimization routine is called here for a
! given gridpoint and options set by the module
! NAG_OPTIMIZATION target-point with the correctly scaled
! distance is returned
SUBROUTINE CALC_REACHABLESET(A1, A2, B1, B2, NT, GRIDPOINT,
                             TARGETPOINT, DISTANCE)
```

An den nötigen Stellen wird direkt auf Daten aus dem Modell-Modul zugegriffen. In der Zielfunktion wird beispielsweise die rechte Seite der Differentialgleichung bei der Berechnung einer Lösung mittels Euler aufgerufen oder bei der Speicherallokation in `OPTIMIZATION_INIT` werden direkt die Dimensionen des Kontroll- und Zustandsraums verwendet um zusammen mit `N_T` die Dimension des Optimierungsproblems zu bestimmen.

Durch diese Implementierung wurde die Übergabe von Parameter an Funktionen auf ein Minimum reduziert.

4.3. Hauptroutine

Die Main-Routine `REACHABLESET` ruft zunächst die Initialisierungssubroutinen der einzelnen Module auf und liest relevante Daten wie das betrachtete Gebiet oder die Rekursionstiefe aus dem Modell-Modul aus. Anschließen wird abhängig von der im Modell gesetzten Variable `ADAPTIVE` entweder nichtadaptiv über alle Gitterpunkte iteriert und der lokale Optimierer aufgerufen. Andernfalls wird die rekursiv deklarierte Routine `RECURSIVE`

`SUBROUTINE REACHABLE_SET_ADAPTIVE(A1, A2, B1, B2, NT, NX, ROOT, R)` aufgerufen, welche sich dann jeweils selbst falls nötig mit aktueller Subbox und kleinerer verbleibender Rekursionstiefe neu aufruft.

Die Funktion `TARGET_IN_BBOX(A1, A2, B1, B2, EPS, DIM, TARGET)` wird innerhalb der adaptiven Routine aufgerufen um zu prüfen, ob eine Liste an Targetpunkten in einer Umgebung der gegebenen Subbox $[A_i, B_i]_\epsilon$ gemäß dem im Kapitel 2.3 vorgestellten Verfahren liegt.

Näheres zur genaueren Implementierung und insbesondere der einzelnen Modelle ist dem dieser Arbeit beigelegten Programmcode zu entnehmen.

5. Numerische Resultate

5.1. Testsysteme

Im Laufe dieser Arbeit wurden hauptsächlich zwei System betrachtet, Rayleigh und Kenderov. Bei beiden handelt es sich um zweidimensionale Systeme mit eindimensionaler Kontrolle, welche bei der Variante mit lokalem Optimierer unterschiedliche Probleme verursachen und sowohl in [1] also auch in [5] näher untersucht wurden.

5.1.1. Rayleigh

$$
\begin{aligned}
\dot{x}_1(t) &= x_2(t) \\
\dot{x}_2(t) &= -x_1(t) + x_2(t)(1.4 - 0.14x_2(t)^2) + 4u(t) \\
u(t) &\in [-1, 1] \forall t \in [t_0, T] \\
t_0 &= 0 \\
T &= 2.5 \\
X_0 &= \left\{ x_0 = \begin{pmatrix} -5 \\ -5 \end{pmatrix} \right\}
\end{aligned}
\qquad \text{(Rayleith)}
$$

Das Rayleighproblem hat seinen Ursprung in elektrischen Schaltungen und erzeugt nichtkonvexe Erreichbarkeitsmengen der nachfolgenden Form:

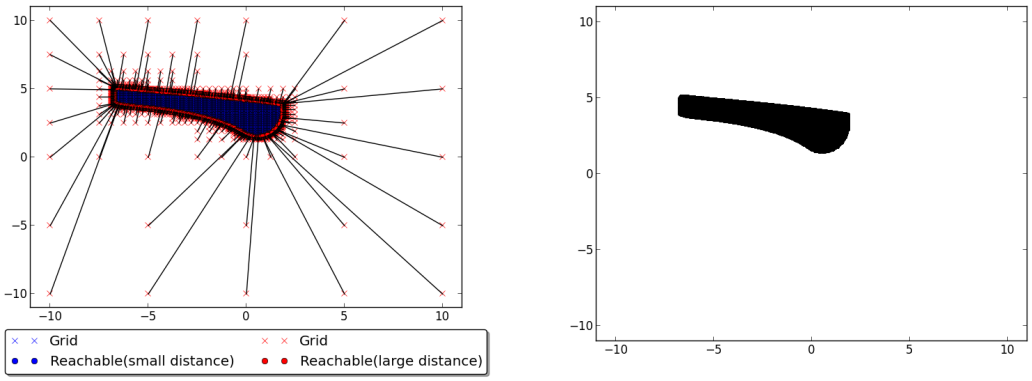
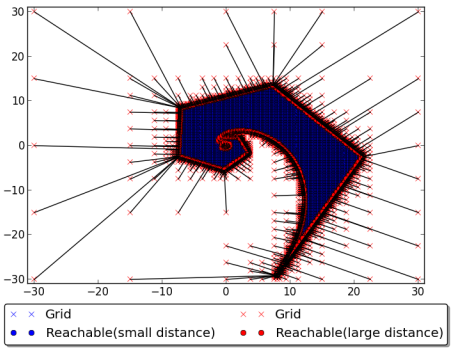


Abbildung 5.1.: $R_h(T)$ für Rayleigh, $N_t = 20$

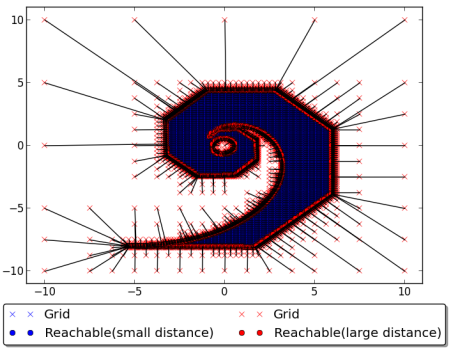
5.1.2. Kenderov

$$
\begin{aligned}
\sigma &= 0.9 \\
\alpha_1 &= \sigma^2 - 1 \\
\alpha_2 &= \sigma\sqrt{1-\sigma^2} \\
\dot{x}_1(t) &= 8\left(\alpha_1 x_1(t) + \alpha_2 x_2(t)(1-2u(t))\right) \\
\dot{x}_2(t) &= 8\left(\alpha_1 x_2(t) - \alpha_2 x_1(t)(1-2u(t))\right) \\
u(t) &\in [-1.0, 1.0] \forall t \in [t_0, T] \\
t_0 &= 0 \\
T &= 1 \\
X_0 &= \left\{ x_0 = \begin{pmatrix} 2 \\ 2 \end{pmatrix} \right\}
\end{aligned}
\tag{Kenderov}
$$

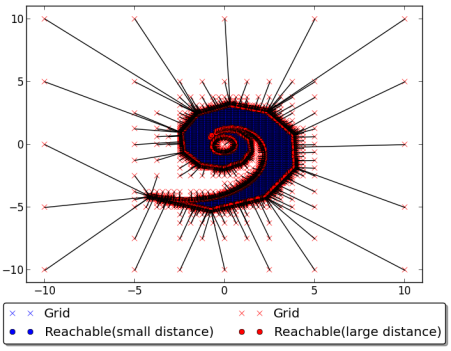
Das von Petar Kenderov konstruierte Problem ist so gestaltet, dass die Erreichbarkeitsmenge des zeitkontinuierlichen Systems eine Nullmenge ist. Die Erreichbarkeitsmenge des zeitlich diskretisierten Systems hängt allerdings stärker von N_t als bei Rayleigh ab und so ergeben sich für unterschiedliche Feinheiten verschiedene Grafiken.



(a) $N_t = 10$



(b) $N_t = 16$



(c) $N_t = 20$

Abbildung 5.2.: $R_h(T)$ für Kenderov-Problem

5.2. Probleme mit lokalem Optimierer

Betrachtet man zunächst die Lösung des Rayleigh-Problem mithilfe eines lokalen Optimierers, so stellt man unterschiedliche Phänomene fest.

- Für grobe zeitliche Diskretisierung werden viele lokale Minima gefunden und die erzeugten Grafiken zeigen, dass ein großer Teil der Erreichbarkeitsmenge nur schlecht abgedeckt wird oder gar ganz fehlen kann.

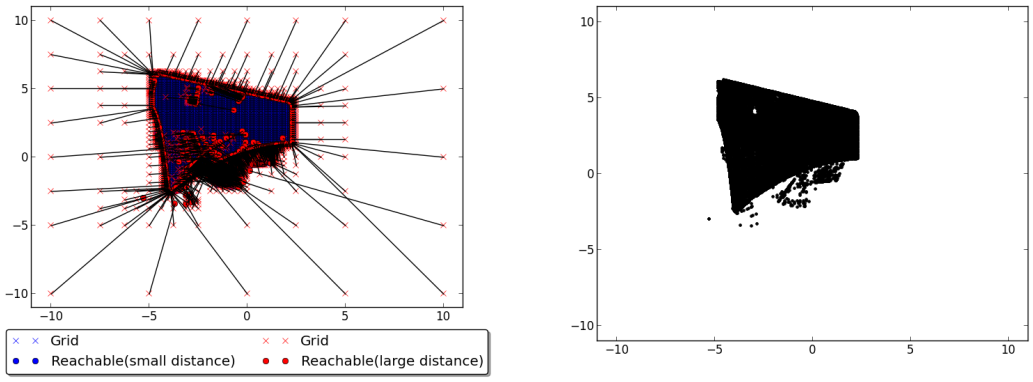


Abbildung 5.3.: Rayleigh $N_t = 8$, $N_x = 5$, $R = 6$, lokale Variante

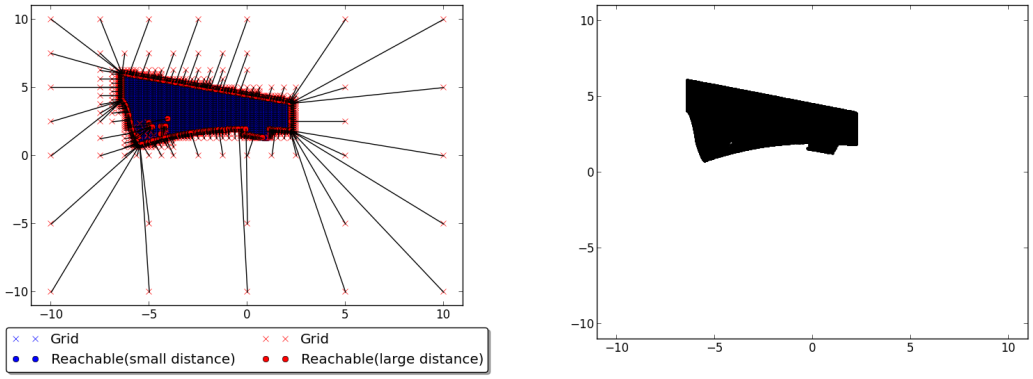


Abbildung 5.4.: Rayleigh $N_t = 10$, $N_x = 5$, $R = 6$, lokale Variante

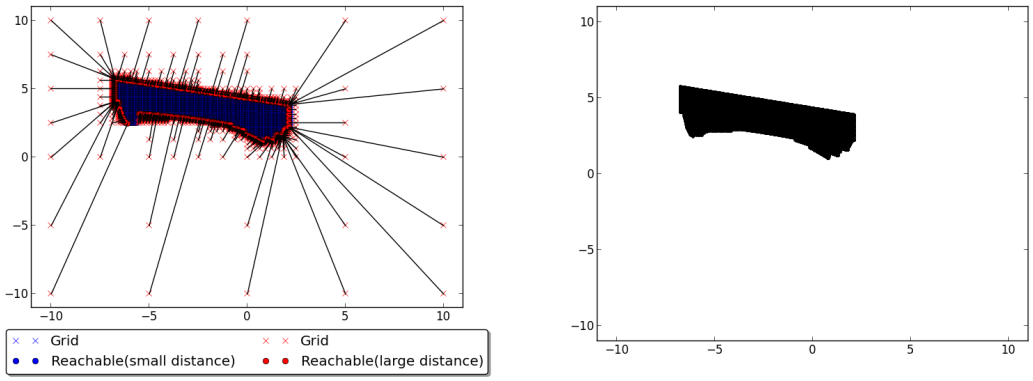


Abbildung 5.5.: Rayleigh $N_t = 12$, $N_x = 5$, $R = 6$, lokale Variante

Bei allen Grafiken sieht man deutlich, das Punkte, welche als Randpunkte gelten weil deren Abstand zu einem Gitterpunkt zu groß ist, im Inneren der eigentlichen Menge liegen. Besonders für $N_t = 12$ fällt auf, das Punkte im Inneren der Erreichbarkeitsmenge gefunden werden, ohne von Randpunkten umschlossen zu sein.

- Falls N_t klein ist, ist die Laufzeit relativ groß.

- Für feinere Diskretisierung erkennt man im rechten unteren Bereich der approximierten Erreichbarkeitsmenge, dass gezackte Kanten entstehen.

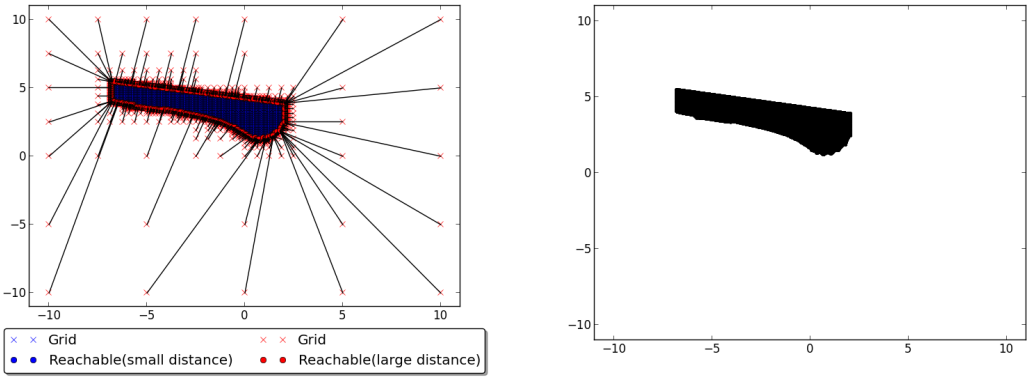


Abbildung 5.6.: Rayleigh $N_t = 14$, $N_x = 5$, $R = 6$, lokale Variante

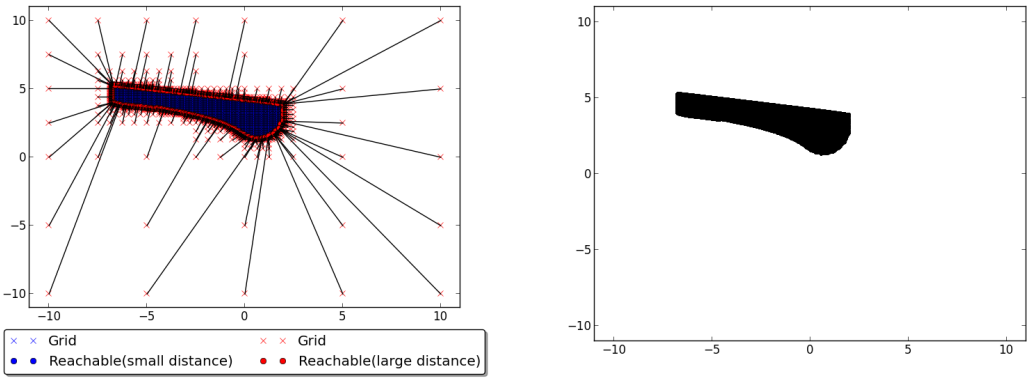


Abbildung 5.7.: Rayleigh $N_t = 16$, $N_x = 5$, $R = 6$, lokale Variante

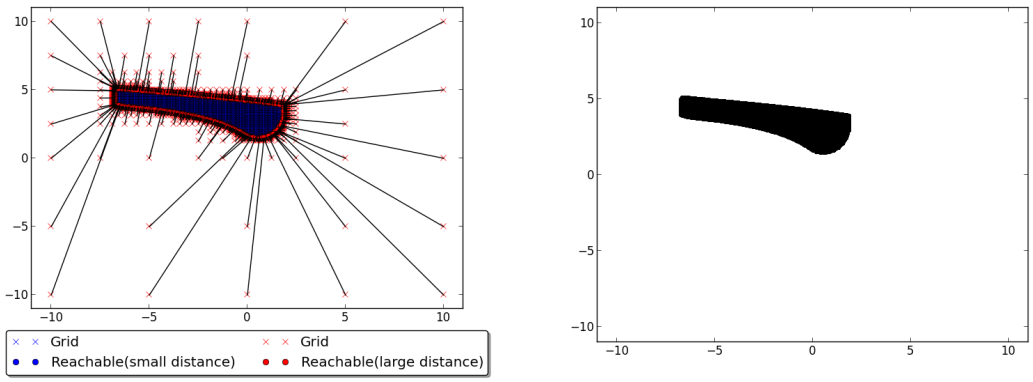


Abbildung 5.8.: Rayleigh $N_t = 20$, $N_x = 5$, $R = 6$

Diese Kanten werden mit steigender Stützstellenanzahl geglättet und verschwinden zunehmend. Hier ist von Interesse, ob dies bedingt ist durch die Systemdynamik des Kontrollproblems oder ob diese Kanten aufgrund von lokalen Optimallösungen entstehen und bestimmte Punkte einfach nicht gefunden werden.

Beim Kenderov-Problem kann man hingegen eine Reihe anderer Probleme feststellen.

- Das Kenderov-Problem ist wesentlich empfindlicher, was schlechte Startwerte für die Kontrolle $u_i = u_0$ für $i = 0, \ldots, N_t - 1$ angeht. Große Teile der Erreichbarkeitsmenge werden nur schlecht oder gar nicht erst berechnet.

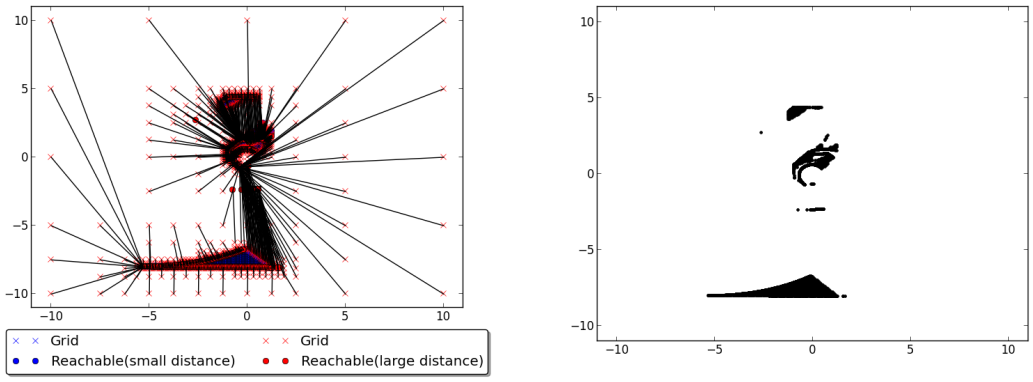


Abbildung 5.9.: Kenderov $N_t = 16$, $N_x = 5$, $R = 6$, $u_0 = -1.0$, lokale Variante

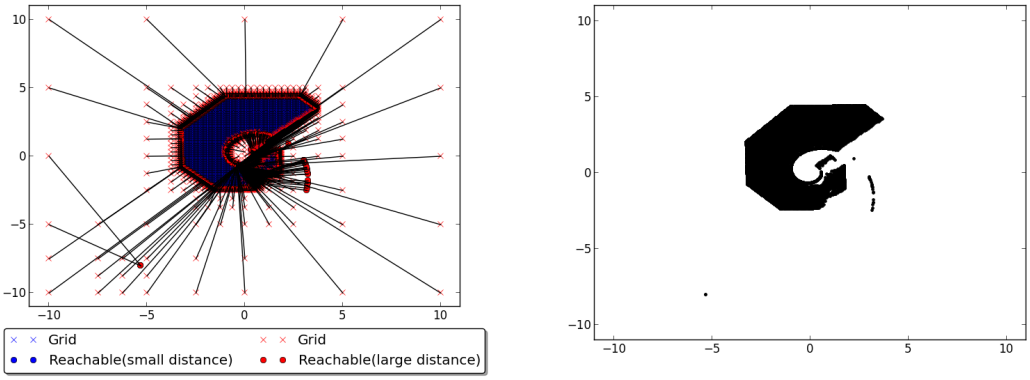


Abbildung 5.10.: Kenderov $N_t = 16$, $N_x = 5$, $R = 6$, $u_0 = 0.0$, lokale Variante

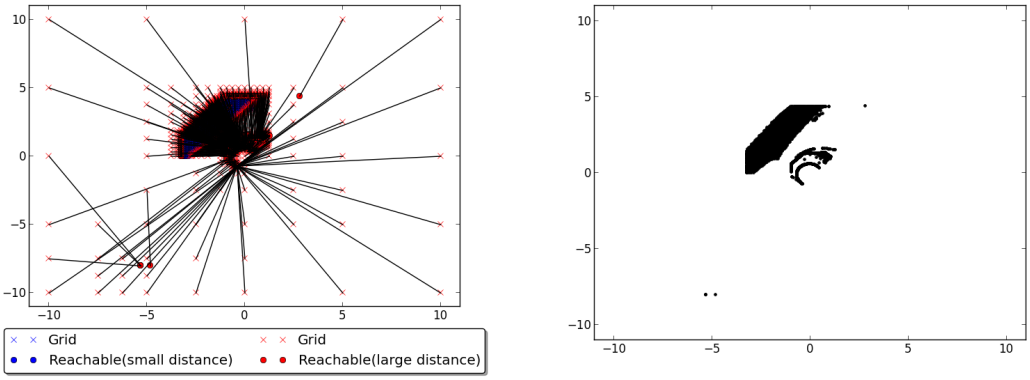


Abbildung 5.11.: Kenderov $N_t = 16$, $N_x = 5$, $R = 6$, $u_0 = 1.0$, lokale Variante

- Selbst wenn man über verschiedene Startwerte für u_0 iteriert (wobei $u_i = u_0$ gewählt wird) und den besten der so erhaltenen Punkte als Lösung wählt, also einen teilweise globalen Ansatz verfolgt, entstehen Lücken im Inneren.

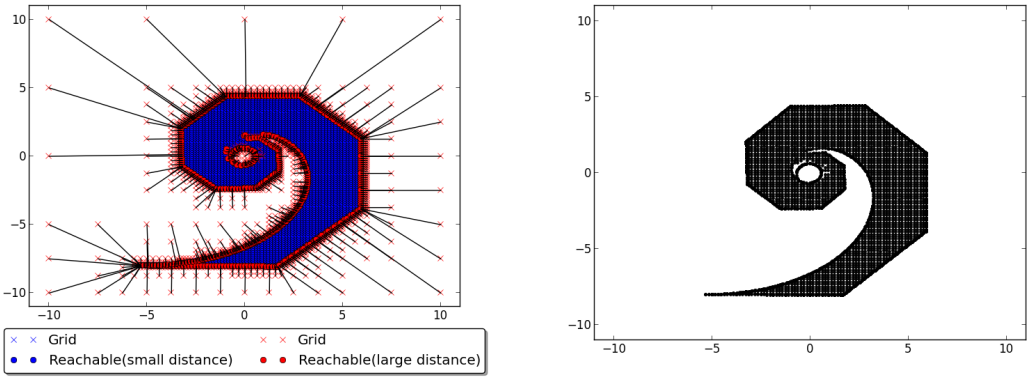


Abbildung 5.12.: Kenderov $N_t = 16$, $N_x = 5$, $R = 6$, Schleife über u_0, lokale Variante

Diese Lücken lassen sich nur schließen, wenn man zusätzlich für jeden Startwert über verschiedene Skalierungen der Zielfunktion iteriert. Statt also das Optimie-

rungsproblem $\min F(x)$ mit $F(x) = \|x - g\|_2$ zu betrachten, löst man das Problem für $F(x) = c\|x - g\|_2^2$ für $c \in \{0.01, 0.1, 1.0, 10.0, 100.0\}$ und verwendet die beste Lösung.

5.3. Allgemeine Einstellungen

Grundsätzlich wurde für diese Arbeit das adaptive Verfahren verwendet, da es mit den hier betrachteten Problemen schon bei der lokalen Version gut funktioniert und die Laufzeiten massiv reduziert. Für die Überprüfung ob ein Punkt in der Box $[l, u]_\epsilon$ ist, wurde aufgrund vorheriger Versuche mit der lokalen Variante $\epsilon = 0.1$ gesetzt.

5.4. Reskalierung der Zielfunktion

Nach ersten Tests wurde schnell klar, dass es nicht besonders sinnvoll ist, das Optimierungsproblem

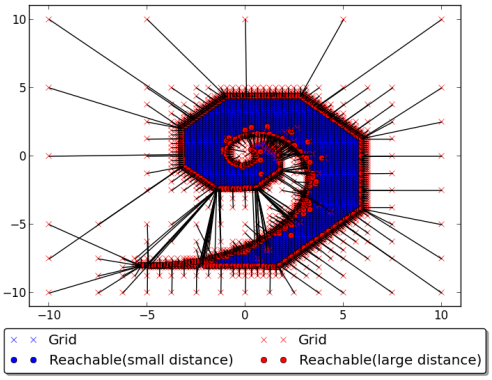
$$\begin{array}{ll} \min & \|x_h(T) - g\|_2 \\ \text{sd.} & x_h(T) \text{ ist Lösung des Kontrollproblems} \end{array}$$

exakt so zu lösen. Gerade MCS bricht in diesem Fall häufig die Berechnung vorzeitig ab und verweist darauf, dass die Zielfunktion schlecht skaliert sei. Die Dokumentation zum MCS-Verfahren rät hier dazu, eine andere Initialisierungsmethode zu wählen, die Grenzen des Problems zu lockern oder eben eine anders skalierte Zielfunktion zu verwenden.

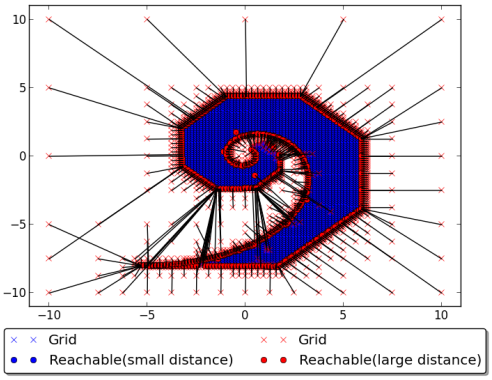
Da diese Meldungen auch für andere Initialisierungen auftreten und eine Lockerung der Boxed Constraints systembedingt nicht in Frage kommt, wurden an dieser Stelle einige Versuche unternommen die Zielfunktion anders zu skalieren. Statt also $\|x_h(T) - g\|_2$ zu betrachten, untersuchte man verschiedene Skalierungen der Zielfunktion

- RESCALE 0 - $\|x_h(T) - g\|_2$
- RESCALE 1 - $\|x_h(T) - g\|_2^2$
- RESCALE 2 - $\|x_h(T) - g\|_2^4$
- RESCALE 3 - $n\|x_h(T) - g\|_2^2$ (n ist Dimension des Zustandsraums)
- RESCALE 4 - $100\|x_h(T) - g\|_2^2$
- RESCALE 5 - $c\|x_h(T) - g\|_2^2$ für $c \in \{0.01, 0.1, 1.0, 10.0, 100.0\}$

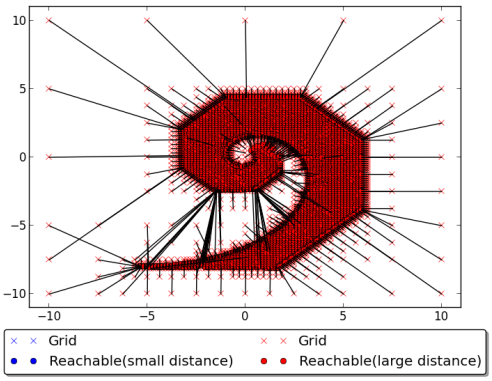
welche allesamt die gleichen globalen Minima besitzen wie das Ausgangsproblem.



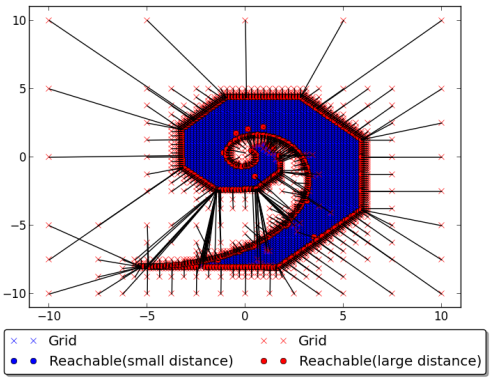
(a) RESCALE 0



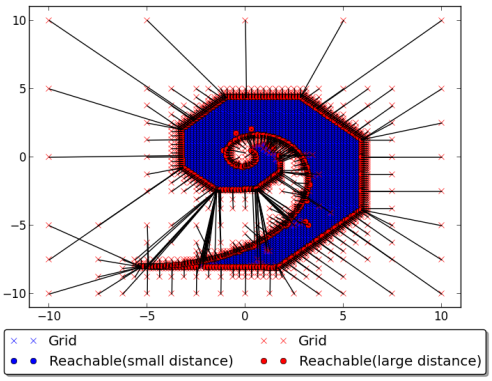
(b) RESCALE 1



(c) RESCALE 2



(d) RESCALE 3



(e) RESCALE 4

Abbildung 5.13.: Kenderov, MCS, $N_t = 16$, $N_x = 3$, $R = 6$, unterschiedliche Zielfunktionsskalierung

Hierbei wird deutlich, dass man mit quadrierter Norm in Option 1 sehr gute Resultate erzielt. Wenn man hingegen Option 2 verwendet sieht man, dass kaum noch Punkte unter einer gewissen Toleranz gefunden werden. Dies liegt daran, dass für sehr kleine Abstände der Wert von $\|x_h(T) - g\|_2^4$ noch viel kleiner wird und sich Änderungen an der eigentlichen Distanz in der Zielfunktion nur wenig bemerkbar machen. Ein kleiner Unterschied bei

dieser Skalierung wird zu einem größeren Fehler bei der Betrachtung der durch die euklidische Norm berechneten Distanz. Der Optimierer endet an dieser Stelle vorzeitig, da das Optimierungsproblem mit ausreichender Genauigkeit gelöst wurde, bei der Rückrechnung auf die euklidische Distanz ist der Fehler allerdings größer.

Andererseits ändern Vorfaktoren am Ergebnis nur sehr wenig, qualitativ lässt sich hier für MCS kein nennenswerter Unterschied beobachten.

Aus diesem Grund wurde für alle weiteren Experimente die Option RESCALE 1 verwendet. Auf die Variante RESCALE 5 wird später näher eingegangen, nachdem grundlegende Resultate der eigentlichen Optimierer vorliegen.

5.5. Berechnungen mit MCS

Zunächst ist es unabhängig von der Initialisierung sinnvoll, die Anzahl an zulässigen Funktionsauswertungen zu erhöhen, da sonst bei zu vielen Gitterpunkten der Optimierer mit Fehlern aussteigt und keine Lösung findet, die durch ein anderes Abbruchkriterium gefunden wird. Die Anzahl wurde von $100n^2$ auf $500n^2$ gesetzt, was für unveränderte maximale Splitanzahl $s_{\max}$ und aktiver Local Search Option ausreichend ist.

Die Local Search-Option wurde hier wie in der Standardkonfiguration aktiviert, da diese den Algorithmus massiv beschleunigt und die Ergebnisse deutlich besser sind.

Was verschiedene Initialisierungslisten angeht, wurden zunächst alle unterschiedlichen Voreinstellungen getestet. Diese umfassen neben zufälligen Initialisierungslisten auch zwei Standardunterteilungen und eine Möglichkeit Initialisierungslisten mittels einem Line-Search-Verfahren zu generieren. Ferner besteht auch die Möglichkeit eigene Initialisierungslisten einzugeben. Diese Variante wurde dazu genutzt um eigenständige Initialisierungen zu implementieren.

- MODE 0 Setzt Initialisierungslisten auf $x_i^1 = l_i \leq x_i^2 = \frac{u_i+l_i}{2} \leq x_i^3 = u_i$ mit $x_0 = \left(x_1^1, \ldots, x_n^1\right)$, also Punkte auf den Grenzen und in der Mitte der einzelnen Koordinatengrenzen. Dies ist die vom Verfahren genutzte Standardvariante.

- MODE 1 Setzt Initialisierungslisten auf $x_i^1 = \frac{5l_i+u_i}{6} \leq x_i^2 = \frac{u_i+l_i}{2} \leq x_i^3 = \frac{l_i+5u_i}{6}$ und ebenfalls $x_0 = \left(x_1^1, \ldots, x_n^1\right)$, also Punkte etwas von den Begrenzungen entfernt und wieder in der Mitte. Auch diese Variante wird bereits durch den Algorithmus geliefert.

- MODE 2 Berechnet eine Initialisierungsliste mithilfe von Line-Search in jeder Koordinatenrichtung, initialisiert die Listen also entsprechend dem Optimierungsproblem.

- MODE 3 Setzt in jeder Koordinatenrichtung eine höhere Anzahl an gleichmäßig verteilten Punkten, wobei x_0 frei gewählt auf einen der Punkte gesetzt werden kann. Für die in dieser Arbeit getesteten Problem wurde x_0 wie bei MODE 0 und MODE 1 in die Mitte gesetzt. Bei der Implementierung wurde hier auf die Möglichkeit der benutzerdefinierten Initialisierung zurückgegriffen.

- MODE 4 Besetzt die Initialisierungsliste mit zufälligen Werten.

- MODE 5 Erlaubt eine vom aktuellen Gitterpunkt abhängige Initialisierung des Optimierers.

- MODE 6 Das Optimierungsproblem wird mit verschiedenen Startinitialisierungen gelöst. Dabei setzt man $x_i^1 = l_i$, $x_i^3 = u_i$ und wählt für x_i^2 Zwischenwerte, so dass

das zugrundeliegende Einschrittverfahren mit einem auf allen Zeitschritten gleichen u_0 initialisiert wird.

Dies ist so implementiert, dass im zulässigen Kontrollraum ein äquidistantes Gitter mit k Punkten in jede Koordinatenrichtung betrachtet wird, so das kein Gitterpunkt auf dem Rand liegt. Es werden also für ein gegebenes k alle Punkte aus der Menge

$$\left\{ u = \begin{pmatrix} u_1 \\ \vdots \\ u_m \end{pmatrix} \;\middle|\; \begin{array}{l} u_i = l_i + (u_i - l_i)\frac{0.5+j}{k} \\ \text{für } i = 1, \ldots, m \text{ und } j = 1, \ldots, k-1 \end{array} \right\}$$

als Startkontrolle betrachtet. Hier sind l und u die Boxed Constraints des Kontrollraums und nicht die des Optimierungsproblems.

Falls eine Lösung mit Optimalwert 0 gefunden wird , also eine Lösung des zeitdiskreten Kontrollproblems gefunden wurde, welche zum Zeitpunkt T gleich dem betrachteten Gitterpunkt g des Zustandsraum ist, so wird die Iteration über die verschiedenen Startwerte beendet, da kein besserer Optimalwert für andere Startwerte gefunden werden kann.

Zusätzlich dazu wurden noch Experimente bezüglich der maximalen Splitanzahl s_{max} unternommen, da mit steigender Splitanzahl Konvergenz garantiert wird.

5.5.1. Rayleigh

Betrachtet man zunächst eine grobe zeitliche Diskretisierung und testet verschiedene Initialisierungsmodi, so liefert MCS folgende Ergebnisse.

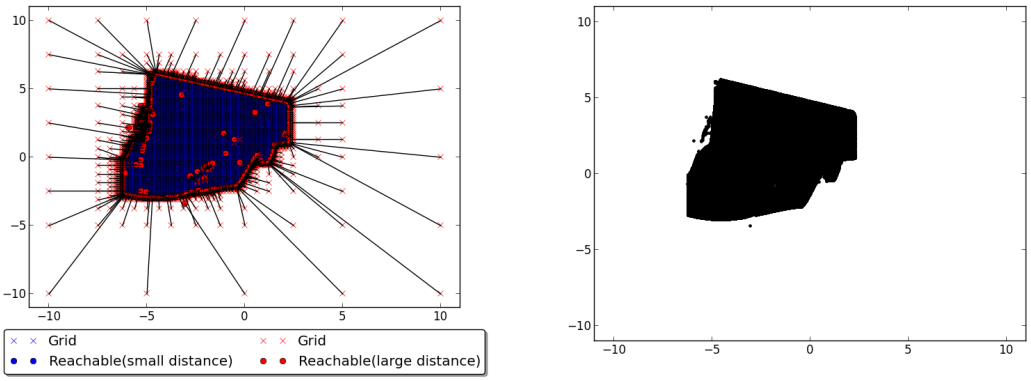


Abbildung 5.14.: Rayleigh, $N_t = 8$, $N_x = 5$, $R = 6$, MODE 0

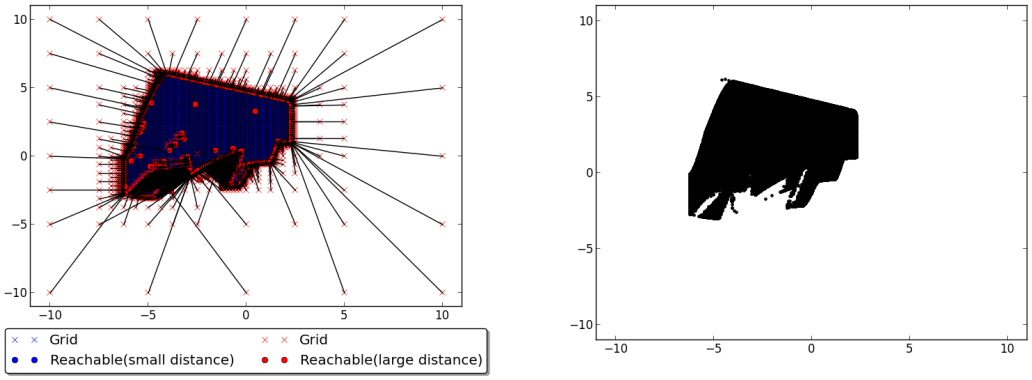


Abbildung 5.15.: Rayleigh, $N_t = 8$, $N_x = 5$, $R = 6$, MODE 1

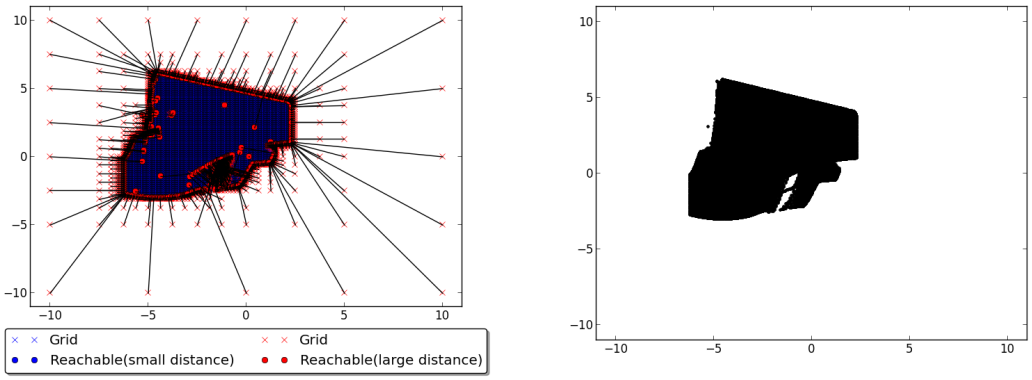


Abbildung 5.16.: Rayleigh, $N_t = 8$, $N_x = 5$, $R = 6$, MODE 2

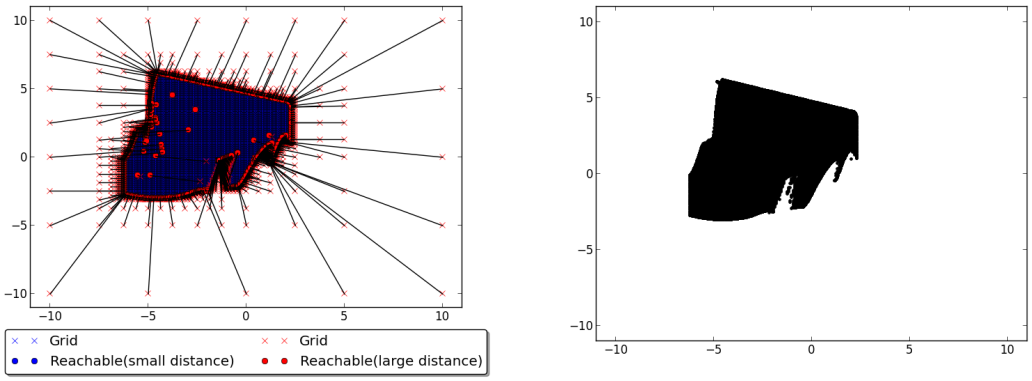


Abbildung 5.17.: Rayleigh, $N_t = 8$, $N_x = 5$, $R = 6$, MODE 3, 15 Initialisierungspunkte pro Koordinate, Startschätzung Mitte

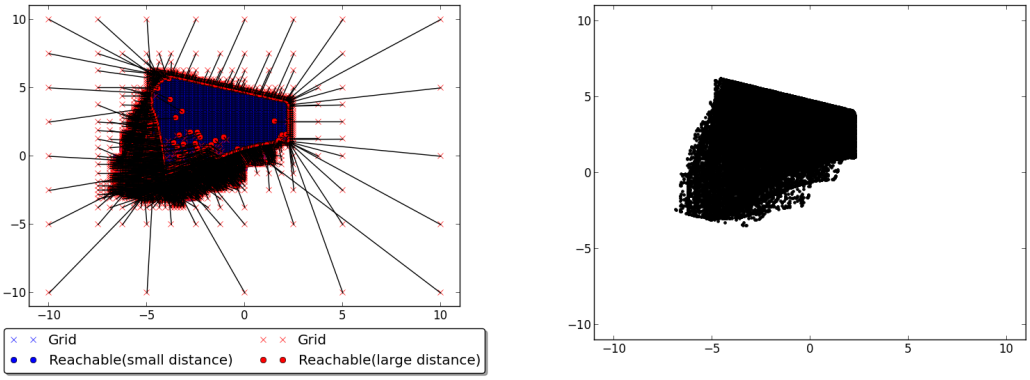


Abbildung 5.18.: Rayleigh, $N_t = 8$, $N_x = 5$, $R = 6$, MODE 4

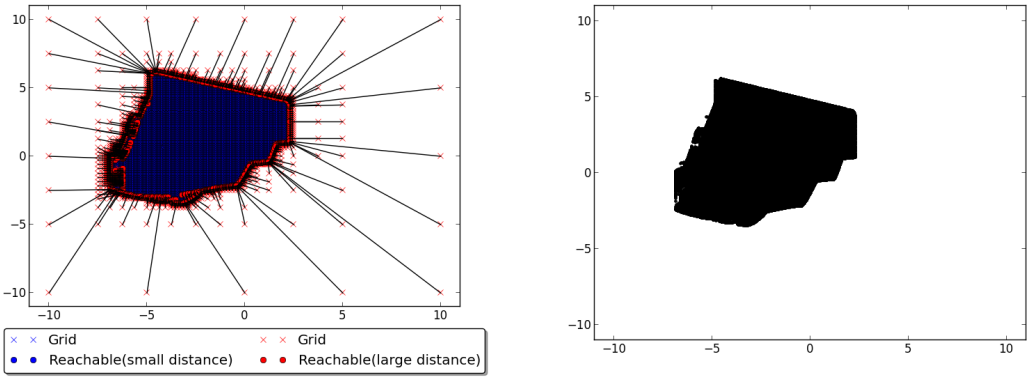


Abbildung 5.19.: Rayleigh, $N_t = 8$, $N_x = 5$, $R = 6$, MODE 6

Bereits hier wird deutlich, dass MCS in hohem Maße von der Art der Initialisierung abhängt und stark schwankende Resultate liefert. Lässt man MODE 6 außen vor, so wird aber bei Rayleigh klar, dass die Lösungen des lokalen Optimierers qualitativ doch deutlich übertroffen werden. Im unteren Bereich werden Teile der Erreichbarkeitsmenge gefunden, deren Existenz das lokale Verfahren nur vermuten lies. Gleiches gilt für das linke Gebiet, bei dem der lokale Löser eine glatte Kante erzeugt welche kaum einen Verdacht erweckte, dass außerhalb davon noch Punkte in der gesuchten Menge $R_h(T)$ liegen.

Allerdings sieht man auch deutlich, das viele lokale Minima gefunden werden, und auch hier Bereiche nicht vollständig entdeckt werden. Betrachtet man MODE 4, so mag das Gesamtbild zwar schlechter sein, als bei den anderen Initialisierungen, allerdings werden zufallsbedingt auch Punkte im linken oberen Gebiet gefunden welche durch andere Varianten verborgen bleiben.

MODE 6 erzielt erwartungsgemäß die besten Resultate und liefert so zumindest für das Areal rechts unten einen Benchmark. Da man aber hier über verschiedene Startschätzungen iteriert um ein globales Optimum zu finden, sprechen diese Ergebnisse nicht unbedingt für die Qualität des verwendeten MCS-Verfahrens.

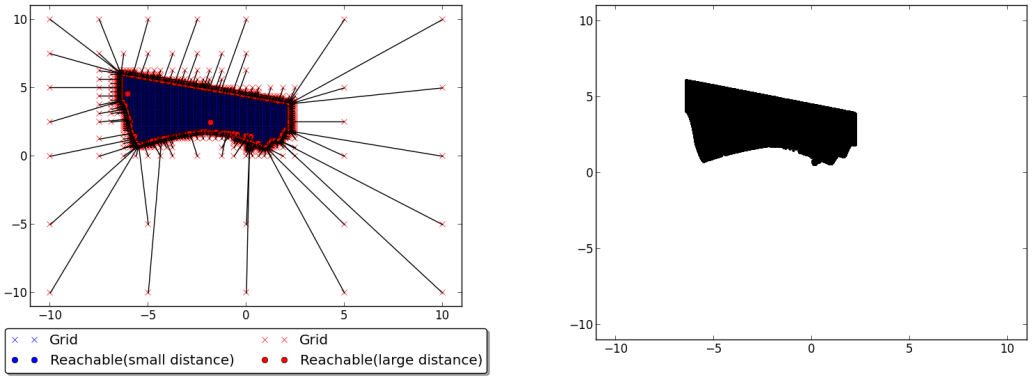


Abbildung 5.20.: Rayleigh, $N_t = 10$, $N_x = 5$, $R = 6$, MODE 0

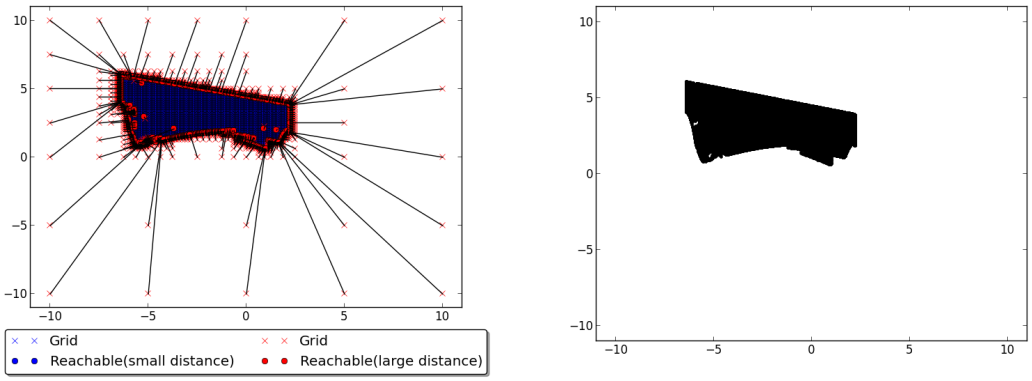


Abbildung 5.21.: Rayleigh, $N_t = 10$, $N_x = 5$, $R = 6$, MODE 1

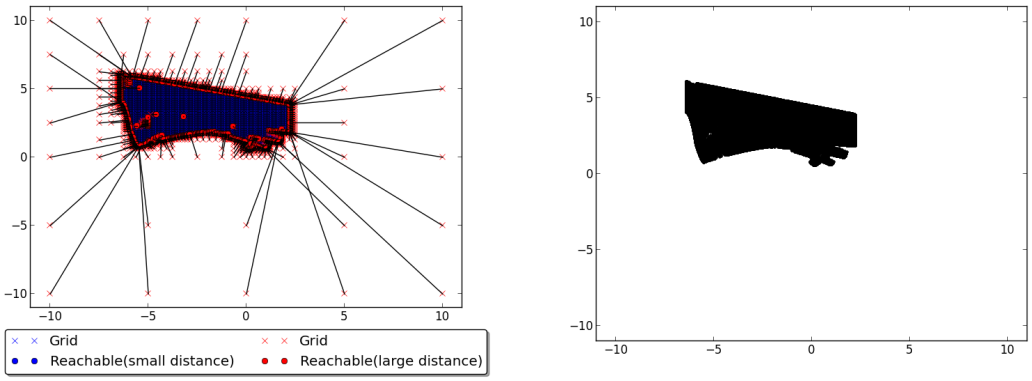


Abbildung 5.22.: Rayleigh, $N_t = 10$, $N_x = 5$, $R = 6$, MODE 2

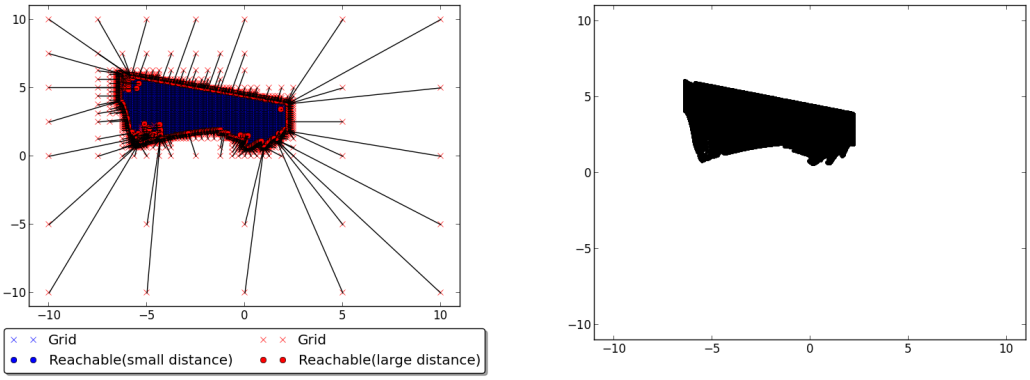


Abbildung 5.23.: Rayleigh, $N_t = 10$, $N_x = 5$, $R = 6$, MODE 3, 15 Initialisierungspunkte pro Koordinate, Startschätzung Mitte

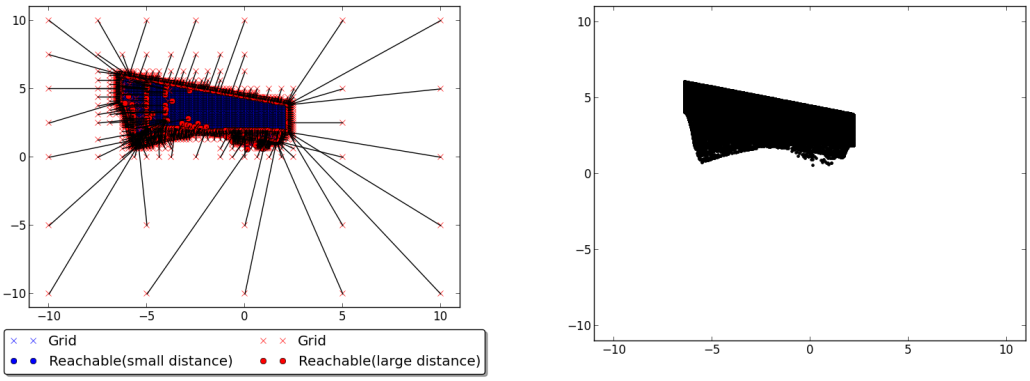


Abbildung 5.24.: Rayleigh, $N_t = 10$, $N_x = 5$, $R = 6$, MODE 4

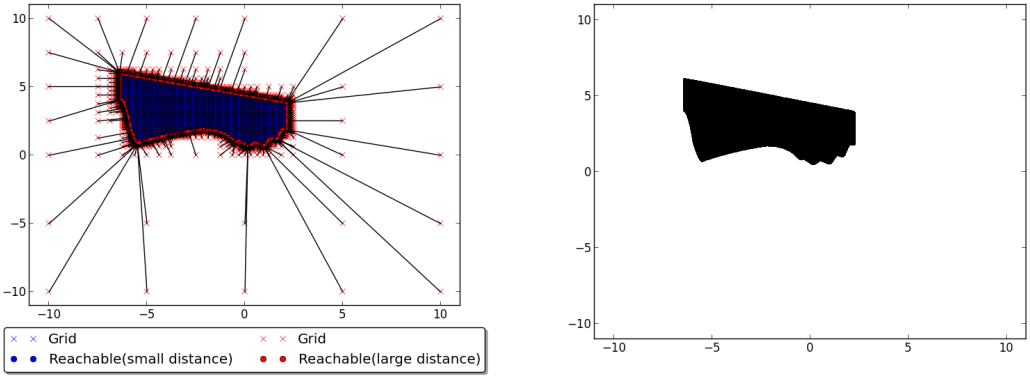


Abbildung 5.25.: Rayleigh, $N_t = 10$, $N_x = 5$, $R = 6$, MODE 6

Auch für $N_t = 10$ ergibt sich ein wesentlich besseres Gesamtbild als bei der lokalen Variante. Hier liefern die voreingestellten Initialisierungen (abgesehen von MODE 4 mit zufallsgenerierten Initialisierungslisten) qualitativ ähnlich gute Resultate.

Was allerdings hier bereits deutlich wird, sind die Kanten im rechten Bereich, welche

beim lokalen Verfahren erst ab $N_t = 12$ auftreten. Dies lässt die Vermutung zu, dass diese durch die Dynamik des zugrundeliegenden diskretisierten Kontrollsystems entstehen und nicht dem Optimierungsverfahren zuzuschreiben sind.

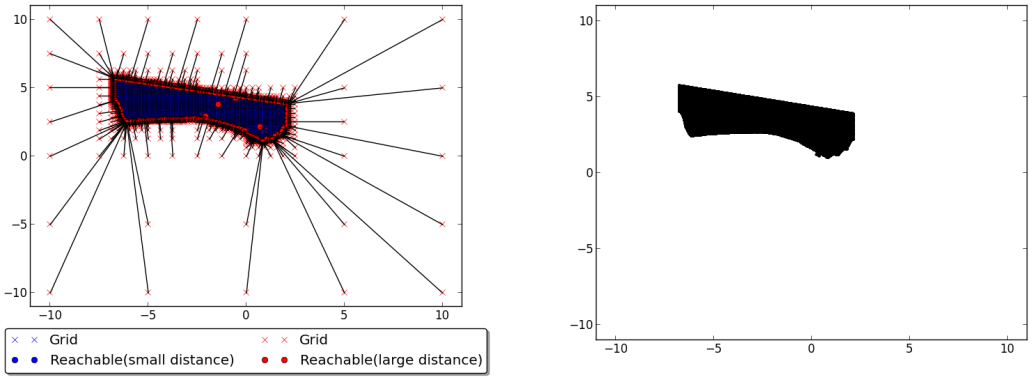


Abbildung 5.26.: Rayleigh, $N_t = 12$, $N_x = 5$, $R = 6$, MODE 0

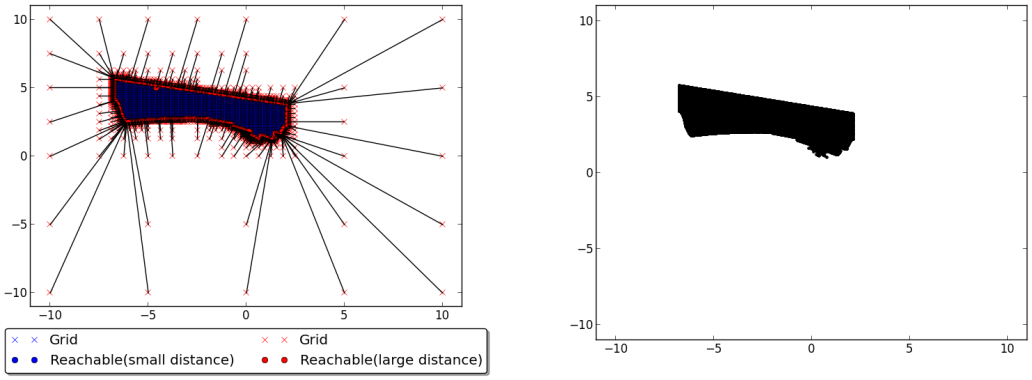


Abbildung 5.27.: Rayleigh, $N_t = 12$, $N_x = 5$, $R = 6$, MODE 1

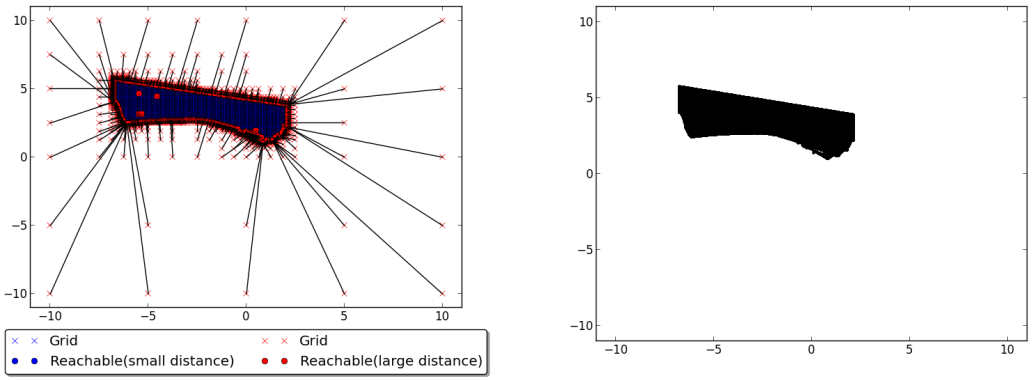


Abbildung 5.28.: Rayleigh, $N_t = 12$, $N_x = 5$, $R = 6$, MODE 2

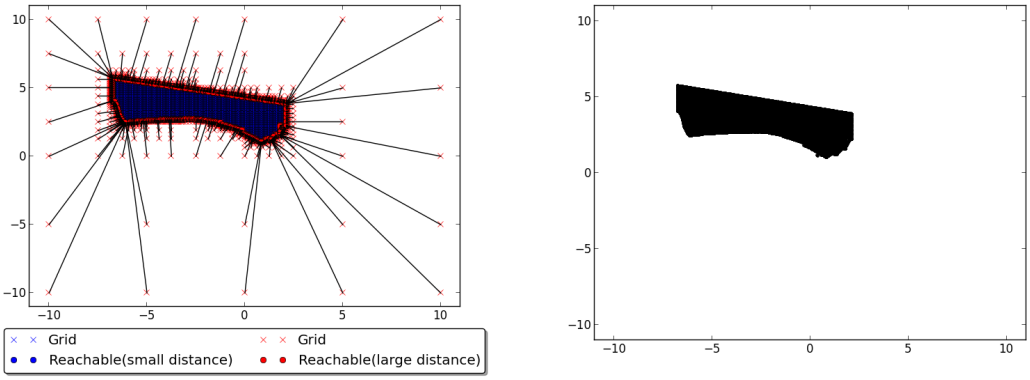


Abbildung 5.29.: Rayleigh, $N_t = 12$, $N_x = 5$, $R = 6$, MODE 3, 15 Initialisierungspunkte pro Koordinate, Startschätzung Mitte

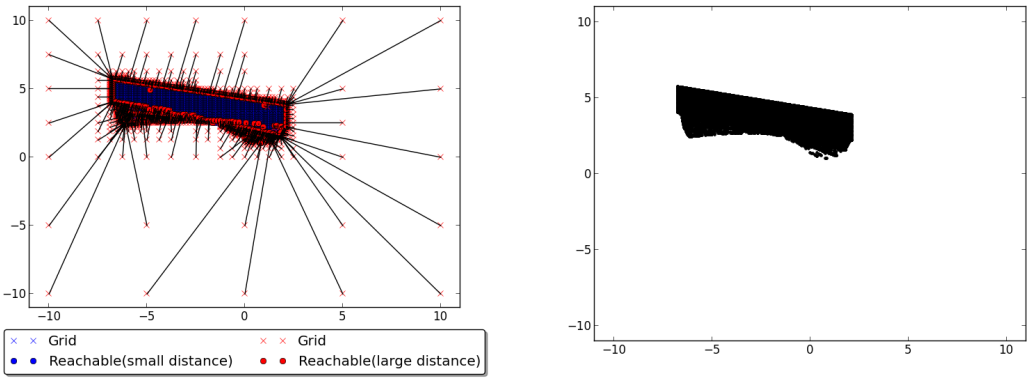


Abbildung 5.30.: Rayleigh, $N_t = 12$, $N_x = 5$, $R = 6$, MODE 4

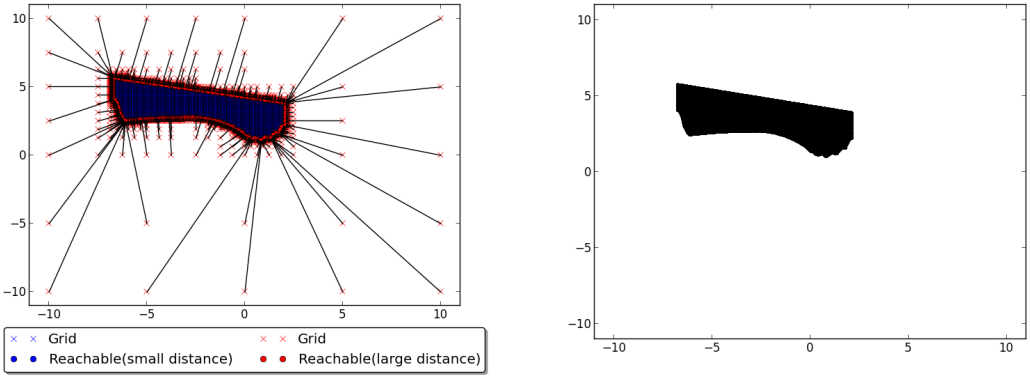


Abbildung 5.31.: Rayleigh, $N_t = 12$, $N_x = 5$, $R = 6$, MODE 6

Wie beim lokalen Verfahren werden die Ergebnisse bei Rayleigh mit steigendem N_t immer besser und das abgesehen von den zufallsgenerierten Listen mit MODE 4 unabhängig von der gewählten Art der Initialisierung. Auch hier sieht man noch deutlich die Kanten im rechten Bereich der Erreichbarkeitsmenge.

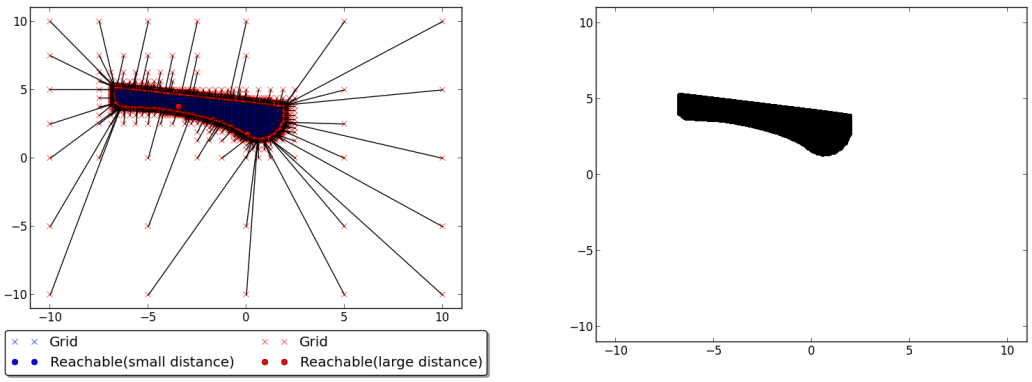


Abbildung 5.32.: Rayleigh, $N_t = 16$, $N_x = 5$, $R = 6$, MODE 0

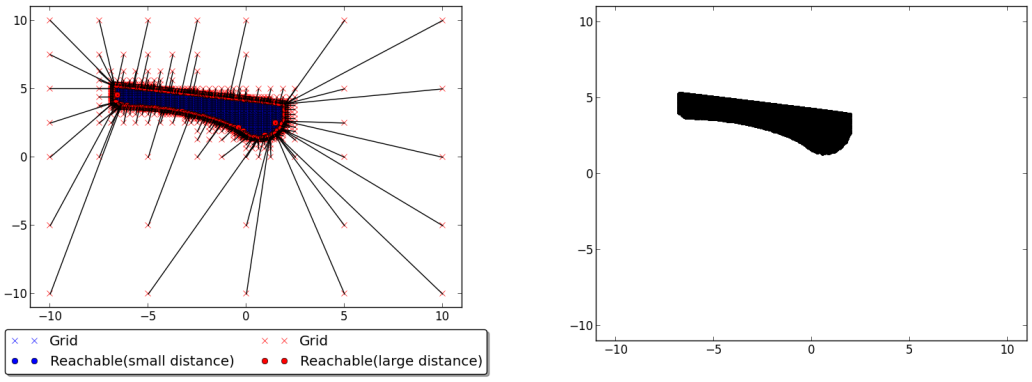


Abbildung 5.33.: Rayleigh, $N_t = 16$, $N_x = 5$, $R = 6$, MODE 1

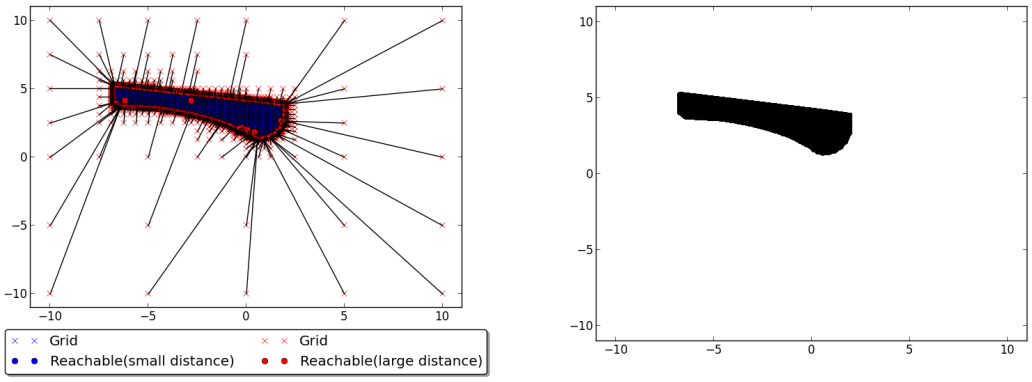


Abbildung 5.34.: Rayleigh, $N_t = 16$, $N_x = 5$, $R = 6$, MODE 2

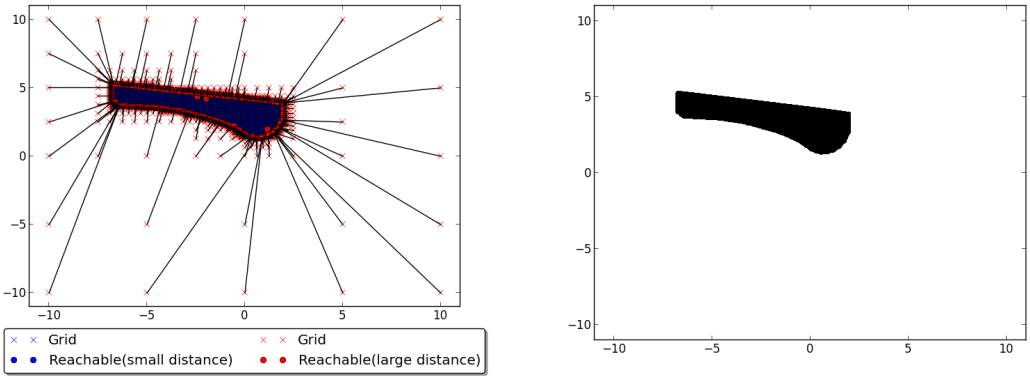


Abbildung 5.35.: Rayleigh, $N_t = 16$, $N_x = 5$, $R = 6$, MODE 3, 15 Initialisierungspunkte pro Koordinate, Startschätzung Mitte

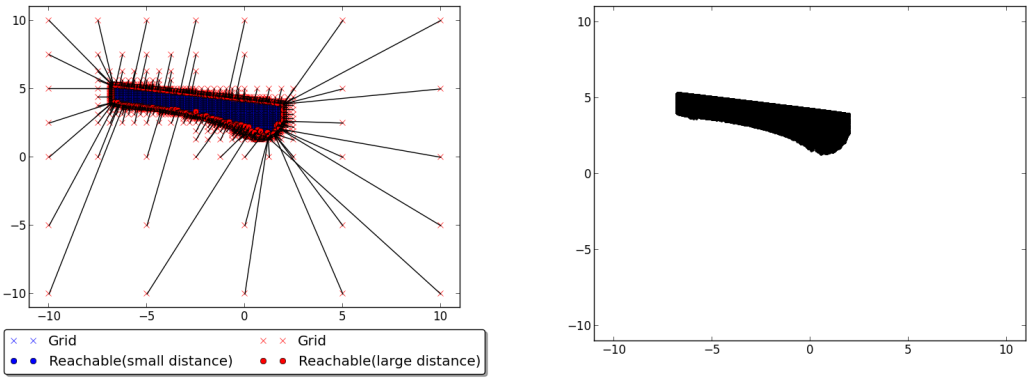


Abbildung 5.36.: Rayleigh, $N_t = 16$, $N_x = 5$, $R = 6$, MODE 4

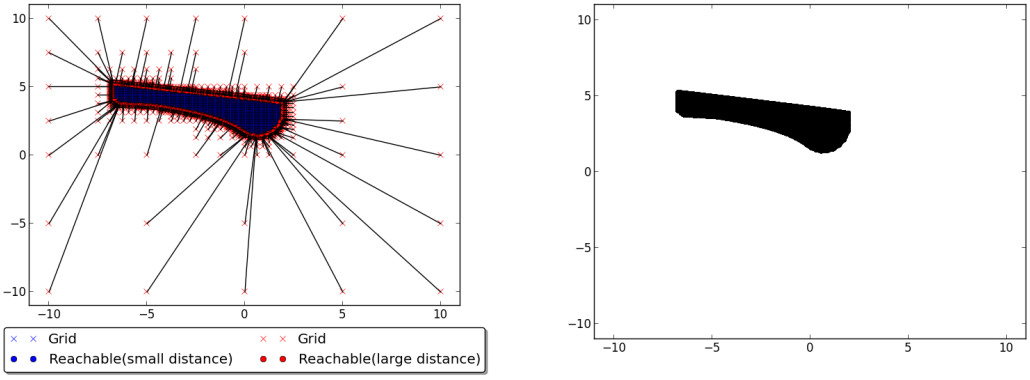


Abbildung 5.37.: Rayleigh, $N_t = 16$, $N_x = 5$, $R = 6$, MODE 6

Zwischen dem lokalen und dem globalen Verfahren gibt es ab $N_t = 16$ kaum noch Unterschiede. Eines der nach wie vor auftretenden Probleme ist das Beenden des Optimierers Fehlern für einzelne Gitterpunkte und dem Verweis auf schlechte Skalierung des Problems. Dies ist dann klar ersichtlich, wenn im Inneren der Erreichbarkeitsmenge ein einzelner Tar-

getpunkt liegt, welcher zu weit von dem zugehörigen Gitterpunkt entfernt liegt, wie zum Beispiel bei MODE 0 und MODE 2 in der Mitte der Erreichbarkeitsmenge.

An dieser Stelle ist auch bereits die Frage geklärt, ob die Kanten im rechten unteren Bereich durch schlechte Lösungen des Optimierungsproblems entstehen oder doch aufgrund der Systemdynamik auftretten. Das lokale Verfahren arbeitet an dieser Stelle nicht schlecht sondern die erreichbare Menge hat für diese groben zeitlichen Diskretisierungen tatsächlich diese Form.

5.5.2. Kenderov

Da bereits bei Rayleigh klar wird, das die Startinitialisierung großen Einfluss auf die Ergebnisse zu haben scheint, ist es sinnvoll, das Kenderov-Problem für einzelne Initialisierungen zu betrachten, da die lokale Variante ebenfalls sehr unterschiedliche Ergebnisse für verschiedene Startwerte liefert.

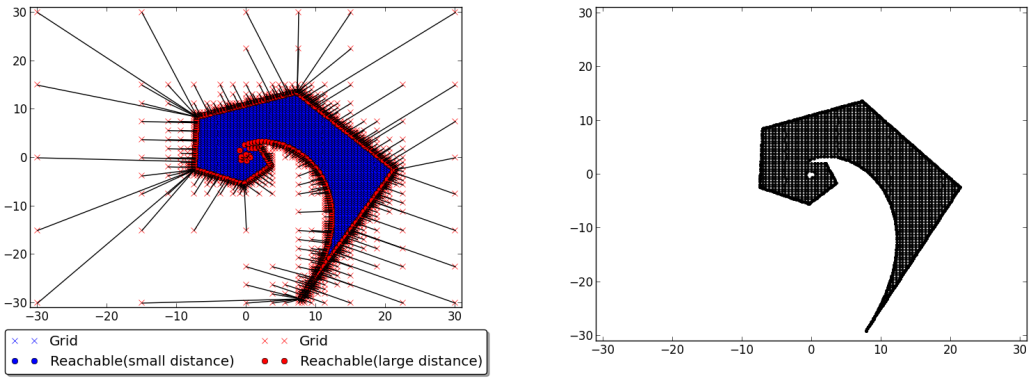


Abbildung 5.38.: Kenderov, $N_t = 10$, $N_x = 5$, $R = 5$, MODE 6

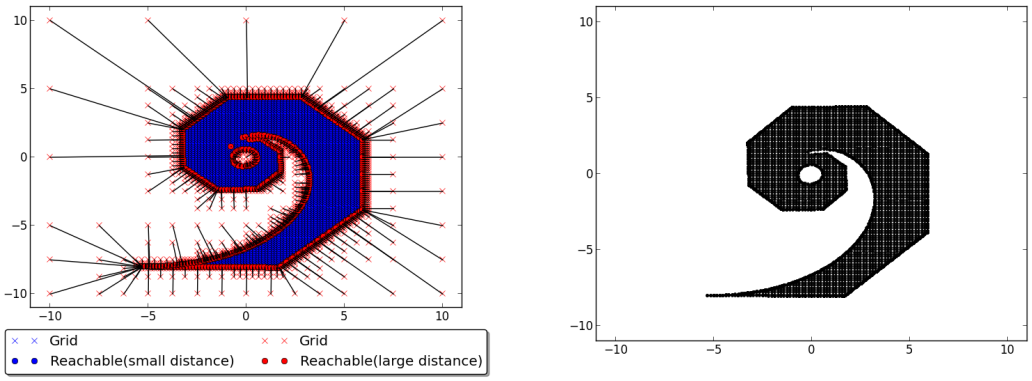


Abbildung 5.39.: Kenderov, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 6

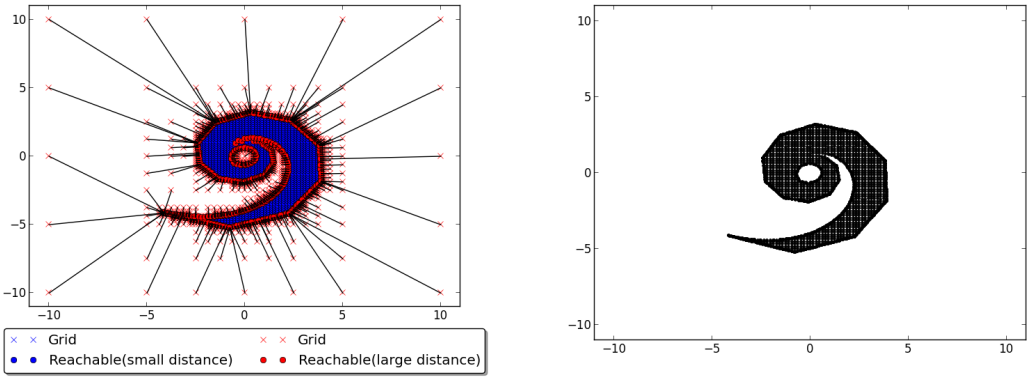


Abbildung 5.40.: Kenderov, $N_t = 20$, $N_x = 5$, $R = 5$, MODE 6

Wie bei Rayleigh dienen die Ergebnisse, bei welchen man über verschiedene Startwerte iteriert und anschließend den besten verwendet lediglich als Benchmark. Anders als beim lokalen Verfahren ist es bei MCS nicht nötig zusätzlich noch über verschiedene Reskalierungen zu iterieren. Die Menge wird hier bereits sehr sauber und ohne Löcher im Inneren dargestellt.

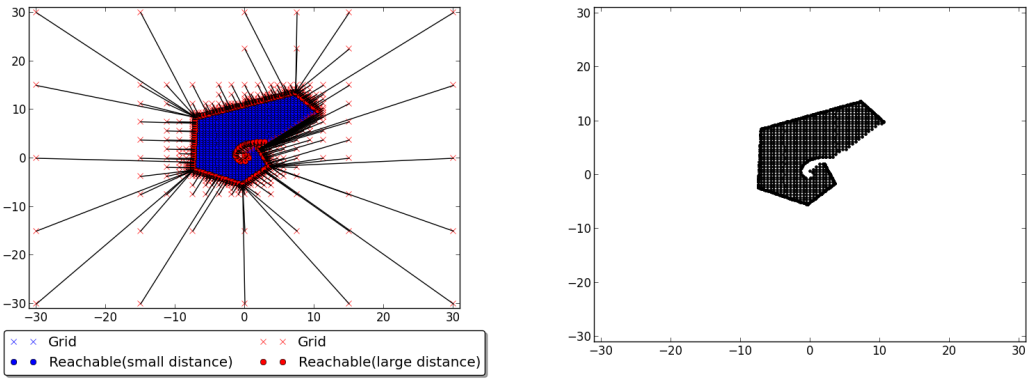


Abbildung 5.41.: Kenderov, $N_t = 10$, $N_x = 5$, $R = 5$, MODE 0

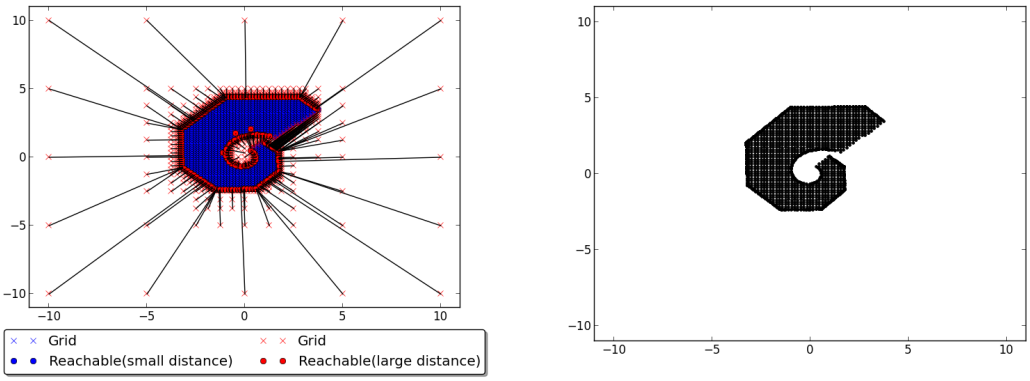


Abbildung 5.42.: Kenderov, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 0

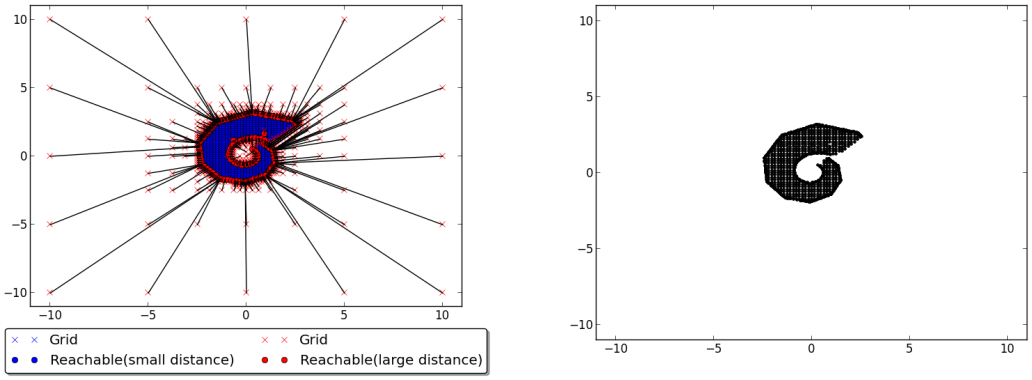


Abbildung 5.43.: Kenderov, $N_t = 20$, $N_x = 5$, $R = 5$, MODE 0

An dieser Stelle wird deutlich, wie stark der Algorithmus von guten Startschätzungen abhängt, denn die erzeugten Grafiken ähneln doch sehr jenen der lokalen Variante mit Startschätzung $u_i = 0.0$ für $i = 1, \ldots, N_t$. Dies ist auch die Startschätzung die bei MODE 0 und MODE 1 verwendet wird, denn tatsächlich zeichnet sich für MODE 1 das gleiche Bild ab. Gleiches gilt auch für MODE 3 mit 15 Punkten pro Koordinatenrichtung in der Initialisierungsliste und Startwert in der Mitte.

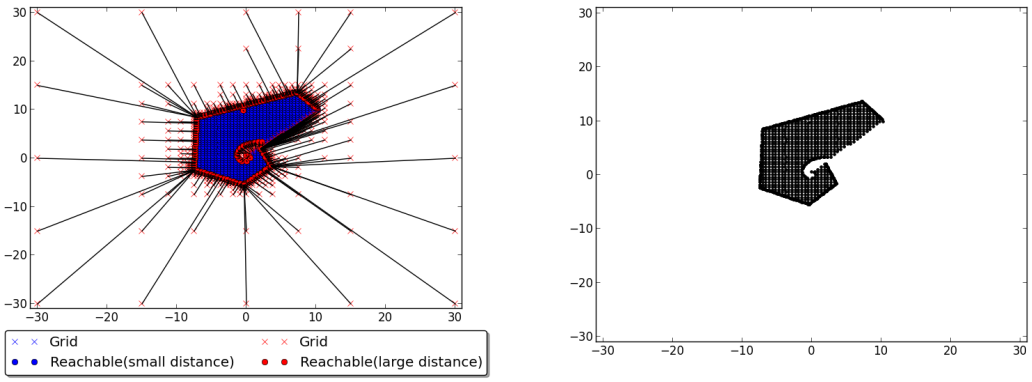


Abbildung 5.44.: Kenderov, $N_t = 10$, $N_x = 5$, $R = 5$, MODE 1

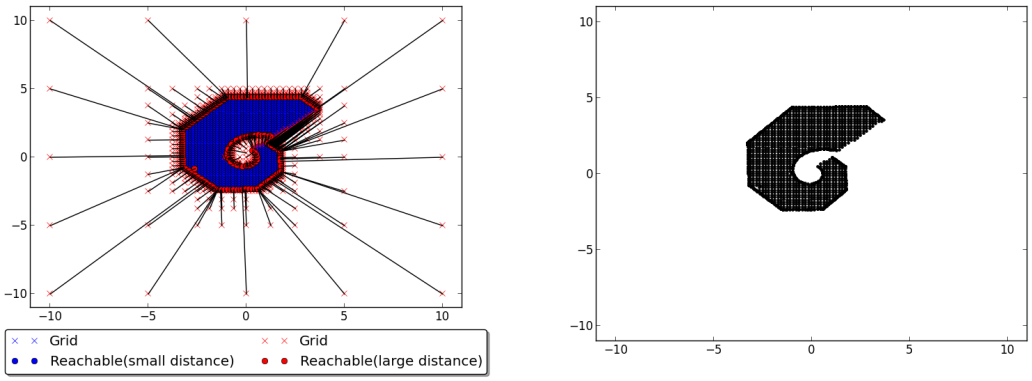


Abbildung 5.45.: Kenderov, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 1

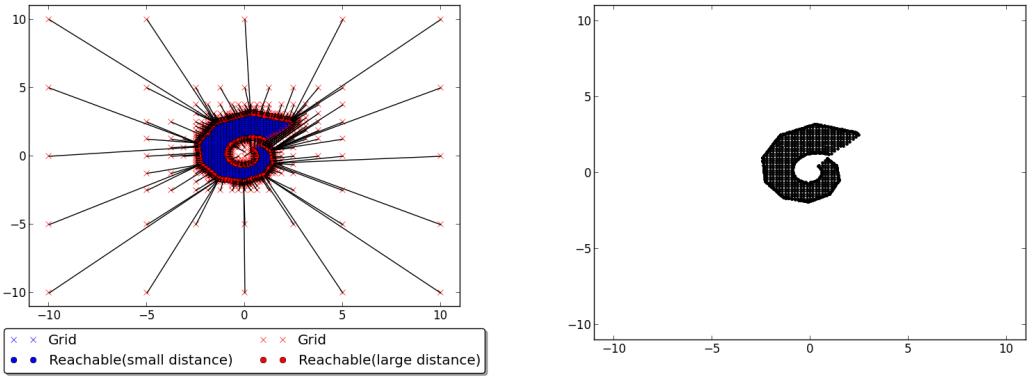


Abbildung 5.46.: Kenderov, $N_t = 20$, $N_x = 5$, $R = 5$, MODE 1

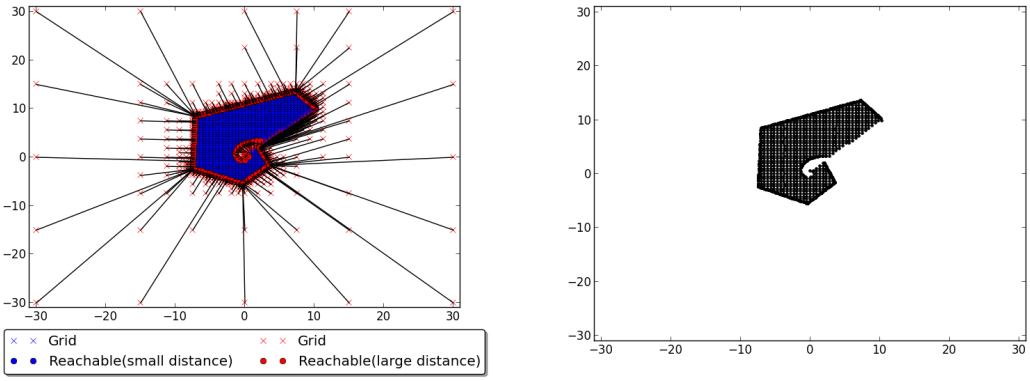


Abbildung 5.47.: Kenderov, $N_t = 10$, $N_x = 5$, $R = 5$, MODE 3

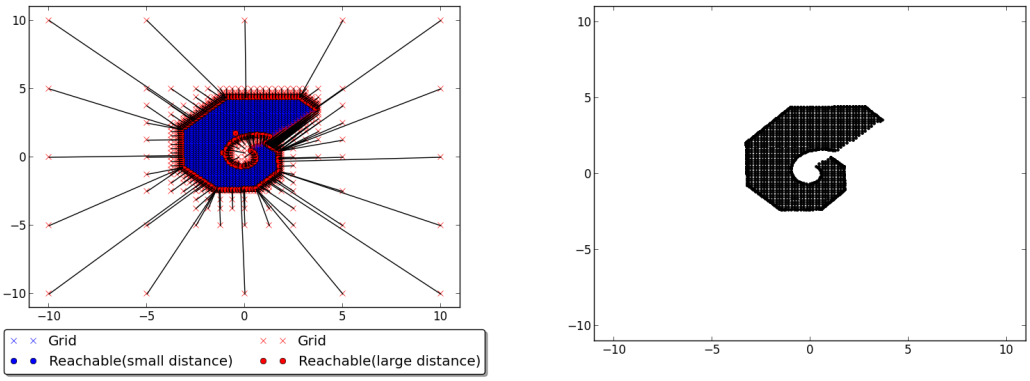


Abbildung 5.48.: Kenderov, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 3

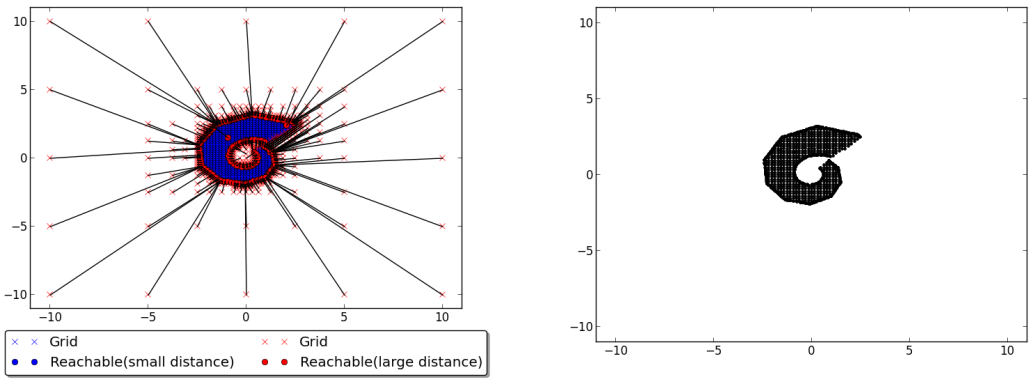


Abbildung 5.49.: Kenderov, $N_t = 20$, $N_x = 5$, $R = 5$, MODE 3

Dieses Ergebnis ist angesichts der Tatsache, dass es sich um einen globalen Optimierer handelt, doch etwas ernüchternd. Bei beiden Modi fehlt ein großer Teil der Erreichbarkeitsmenge, sowohl außen, als auch im inneren Bereich, wo der Kreis geschlossen sein müsste.

Insgesamt sind die Ergebnisse aber dennoch wesentlich besser, als mit dem lokalen Optimierer mit Startschätzung $u_i = 0.0$. Nicht nur wird ein größerer Bereich gefunden, dieser wird auch glatter abgegrenzt. Allerdings werden die einzelnen Punkte, die die lokale Version im rechten Bereich für $N_t = 16$ nicht abgebildet.

Verwendet man nun MODE 2, also die Initialisierung, bei der die Listen und Startschätzungen mithilfe eines Line-Search-Verfahrens ermittelt werden, so wandelt sich das Bild allerdings etwas.

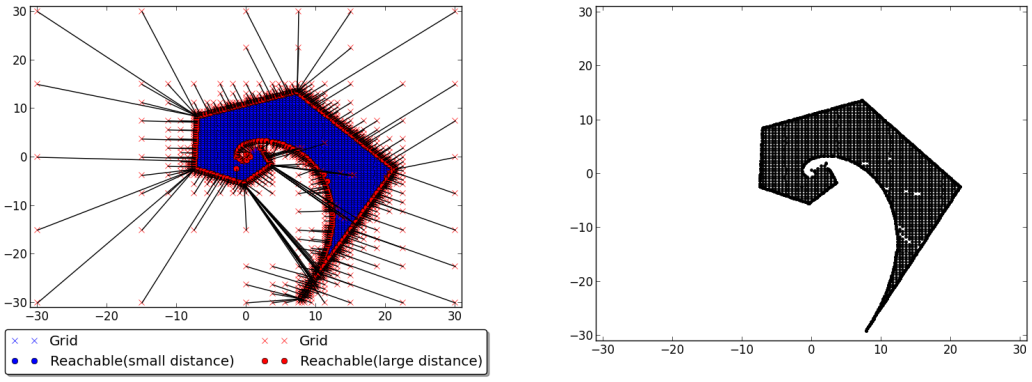


Abbildung 5.50.: Kenderov, $N_t = 10$, $N_x = 5$, $R = 5$, MODE 2

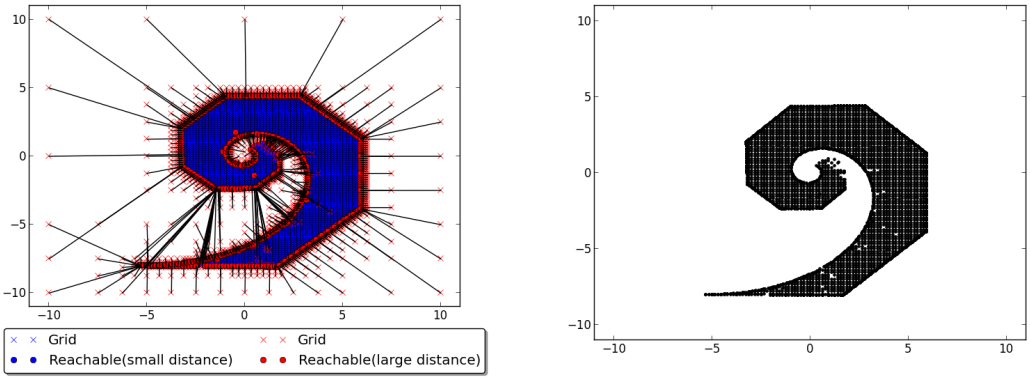


Abbildung 5.51.: Kenderov, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 2

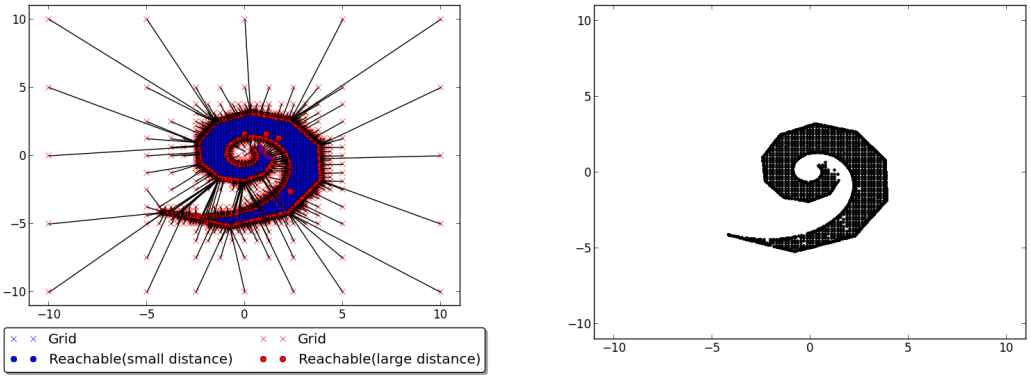


Abbildung 5.52.: Kenderov, $N_t = 20$, $N_x = 5$, $R = 5$, MODE 2

Zwar erkennt man auch hier, dass der innere Bereich nicht geschlossen ist, und ein Teil der Erreichbarkeitsmenge nicht berechnet wird, allerdings wird doch ein Großteil der Menge dargestellt. Bedauerlicherweise erkennt man aber auch hier, das der Optimierer nicht selten lokale Minima findet. Ferner ist auch hier das Problem erkennbar, dass der Optimierer trotz RESCALE 1 mit Fehlern vorzeitig endet.

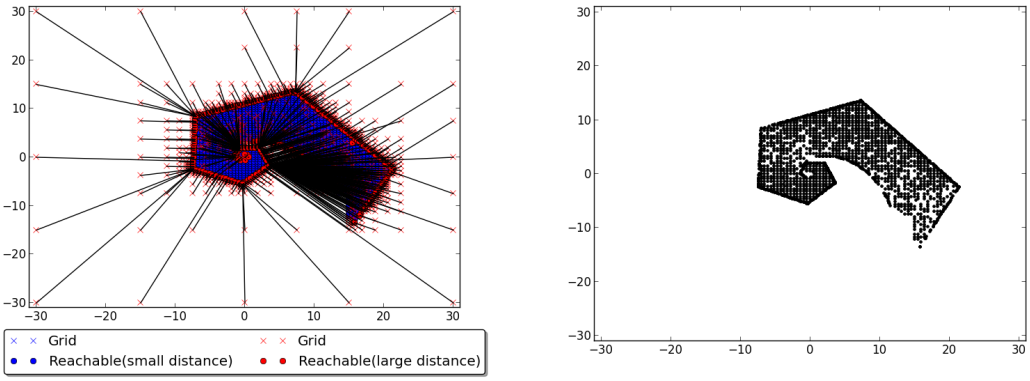


Abbildung 5.53.: Kenderov, $N_t = 10$, $N_x = 5$, $R = 5$, MODE 4

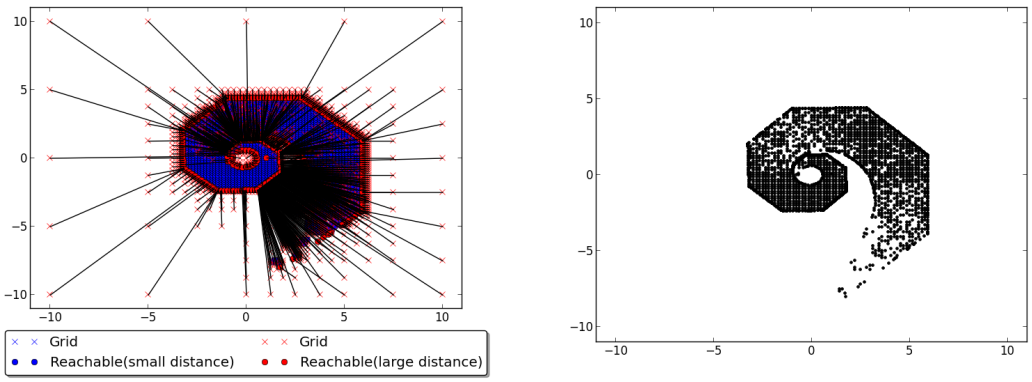


Abbildung 5.54.: Kenderov, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 4

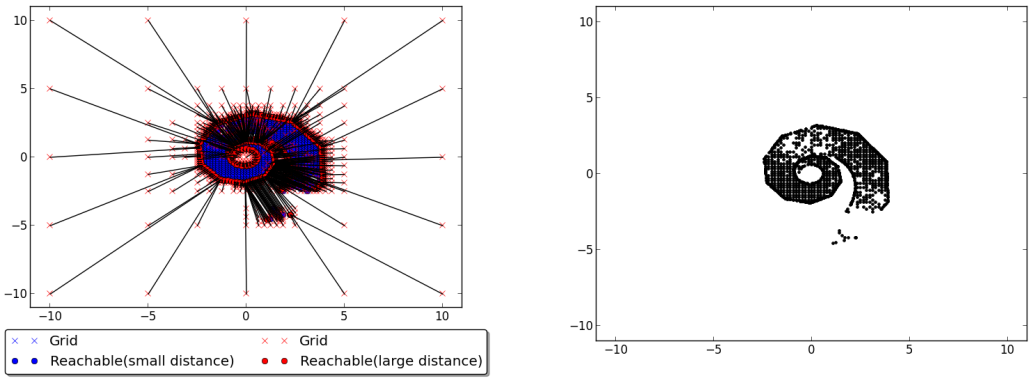
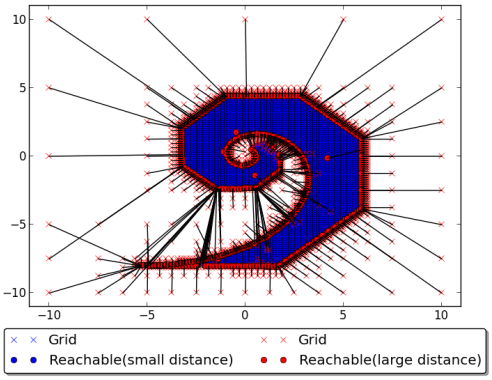


Abbildung 5.55.: Kenderov, $N_t = 20$, $N_x = 5$, $R = 5$, MODE 4

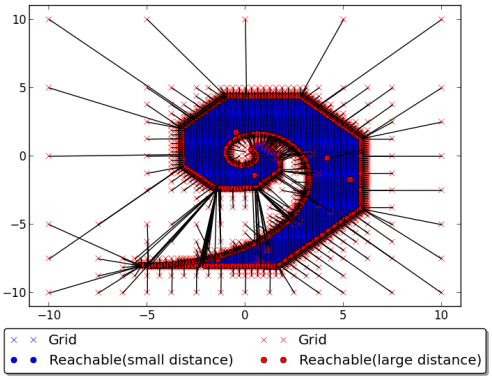
Für zufallsgenerierte Initialisierungslisten ergibt sich ähnlich wie bei Rayleigh ein durchwachsenes Bild. Die Qualität der erzeugten Grafiken schwankt dabei mit jedem Durchlauf, da das eine mal der äußere Bereich gut abgedeckt wird, ein anderes mal hingegen fehlen kann. Hier wird aber nie die Qualität der mit MODE 2 erzeugten Grafiken erreicht. Allerdings wird deutlich, dass das innere Areal, bei dem der Kreis geschlossen sein müsste, bei dieser Initialisierung besser dargestellt wird, als bei anderen Varianten.

5.5.3. Variation der maximalen Splitanzahl $s_{\max}$

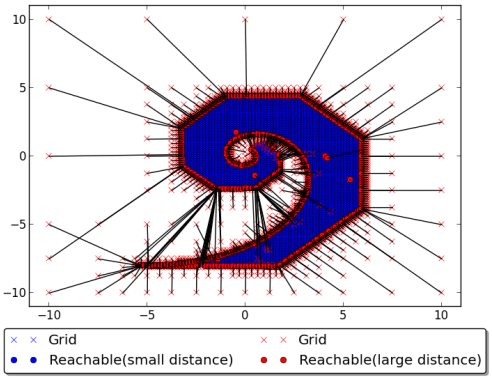
Nun stellt sich natürlich die Frage, in wie weit eine Variation von $s_{\max}$ das Gesamtbild ändert, denn zumindest theoretisch liefert der Algorithmus für $s_{\max} \to \infty$ das globale Minimum. Der Standardwert hier ist $s_{max} = \lfloor \frac{d}{3}(n+2) \rfloor$ mit $d = 15$, also $s_{max} = 5n + 10$.



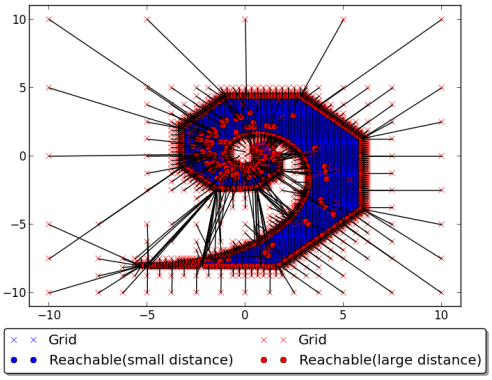
(a) $s_{\max} = 10n$



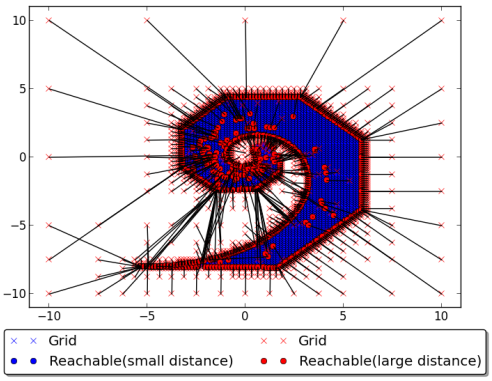
(b) $s_{\max} = 20n$



(c) $s_{\max} = 50n$



(d) $s_{\max} = 100n$



(e) $s_{\max} = 100n$

Abbildung 5.56.: Kenderov, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 2

Wie man sieht, ändern sich die Ergebnisse nicht zum Besseren, sie werden stattdessen sogar noch schlechter, was auch für andere Initialisierungen gilt. Für eine zu groß gewählte maximale Splitanzahl bricht für viele Gitterpunkte der Optimierer mit Verweis auf eine schlecht skalierte Zielfunktion ab. Betrachtet man zusätzlich die Laufzeiten in Bezug auf Änderungen an $s_{\max}$ so lohnt sich eine Variation diese Wertes nicht.

MODE / $s_{\max}$	MODE 0	MODE 1	MODE 2
5n + 10	0m45.775s	0m31.858s	1m27.201s
10n	1m7.932s	0m58.816s	2m20.989s
20n	2m32.722s	2m23.393s	6m28.204s
50n	9m28.476s	8m38.292s	26m6.926s
100n	35m7.712s	36m26.609s	93m10.193s
200n	189m58.516s	185m50.121s	419m42.310s

Da die Laufzeiten mit steigender Splitanzahl trotz Local Search explodieren, kann nicht abgeschätzt werden, wie groß $s_{\max}$ für diese Probleme sein muss, damit alle globalen Optima erreicht werden.

Wenn man Local Search deaktiviert und sich lediglich auf Funktionswerte der Basispunkte einzelner Punkte verlässt muss die Splitanzahl um ein vielfaches höher sein um passable Ergebnisse zu erzielen.

Für MODE 0 mit deaktivierte Local Search Option und $s_{\max} = (n+2)^{20}$ ergibt sie beispielsweise ein solches Bild:

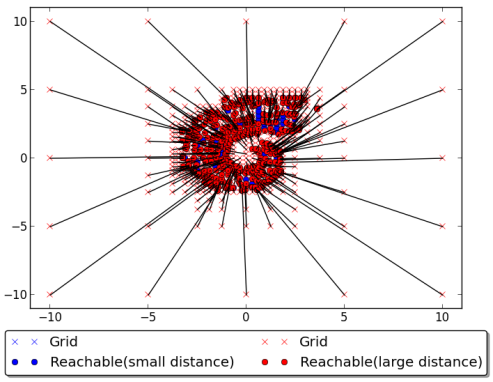


Abbildung 5.57.: Beispiel mit deaktivierter Local Search

Dies ist auch der Grund, warum Local Search für alle anderen Versuche aktiviert ist

5.5.4. Schleife über verschiedene Reskalierungen der Zielfunktion

Nun stellt sich die Frage wie eine Schleife über verschiedene Reskalierungen die Ergebnisse verbessert. Bei der lokalen Variante war diese Iteration nötig, um die Löcher oberhalb der Kreise um den Nullpunkt zu schließen, welche bei einer Schleife über verschiedene Startschätzungen entstehen.

Exemplarisch werden hier die Ergebnisse mit MODE 2 betrachtet, da dieser die besten Resultate auch ohne eine solche Iteration lieferte.

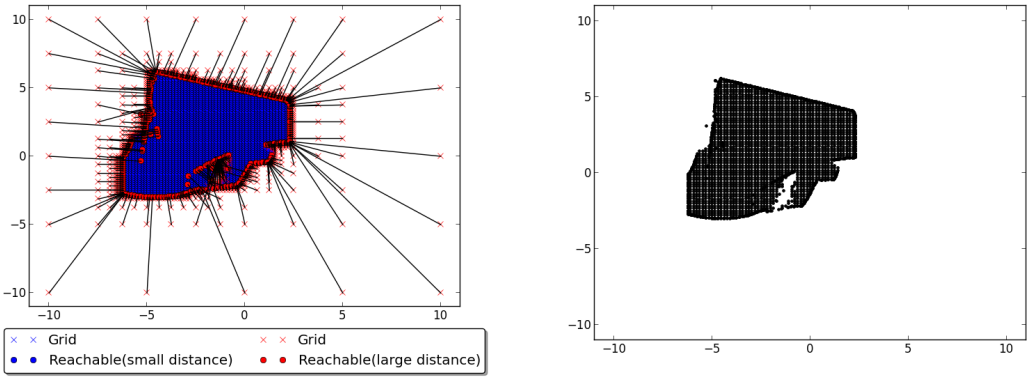


Abbildung 5.58.: Rayleigh, $N_t = 8$, $N_x = 5$, $R = 5$, MODE 2, RESCALE 5

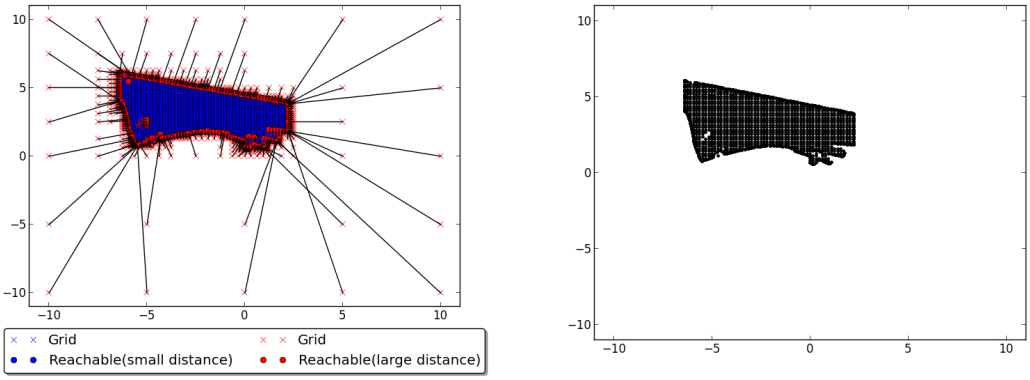


Abbildung 5.59.: Rayleigh, $N_t = 10$, $N_x = 5$, $R = 5$, MODE 2, RESCALE 5

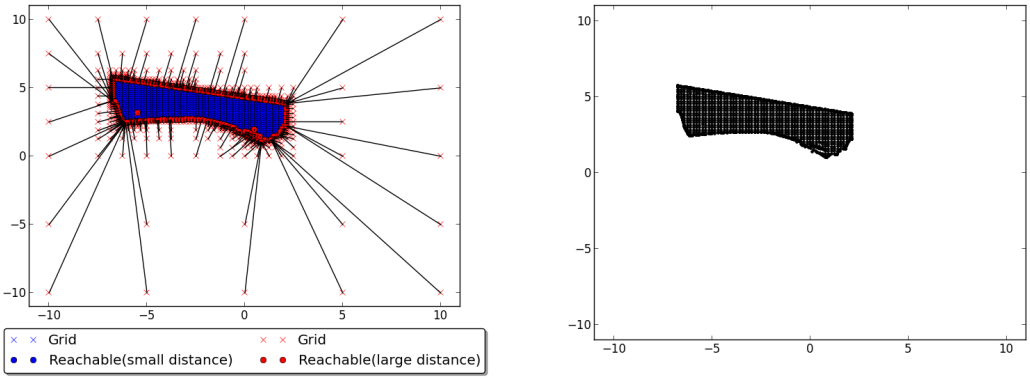


Abbildung 5.60.: Rayleigh, $N_t = 12$, $N_x = 5$, $R = 5$, MODE 2, RESCALE 5

Bei Rayleigh sind die Ergebnisse für kleine N_t praktisch identisch mit denen von RESCALE 1.

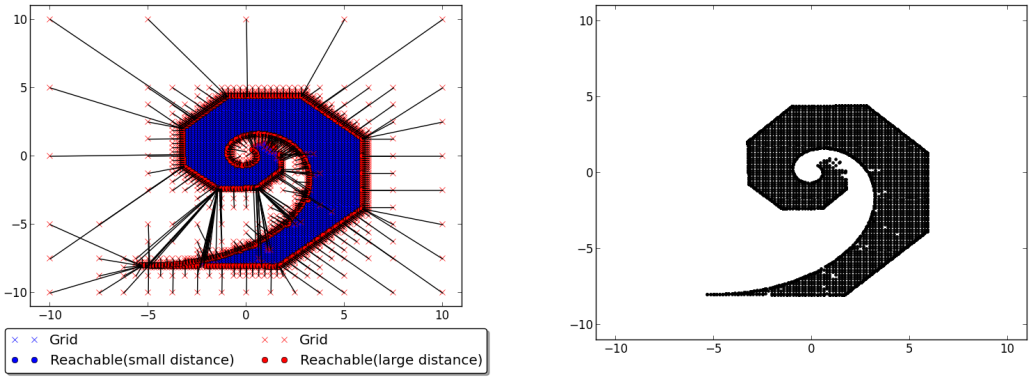


Abbildung 5.61.: Kenderov, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 2, RESCALE 5

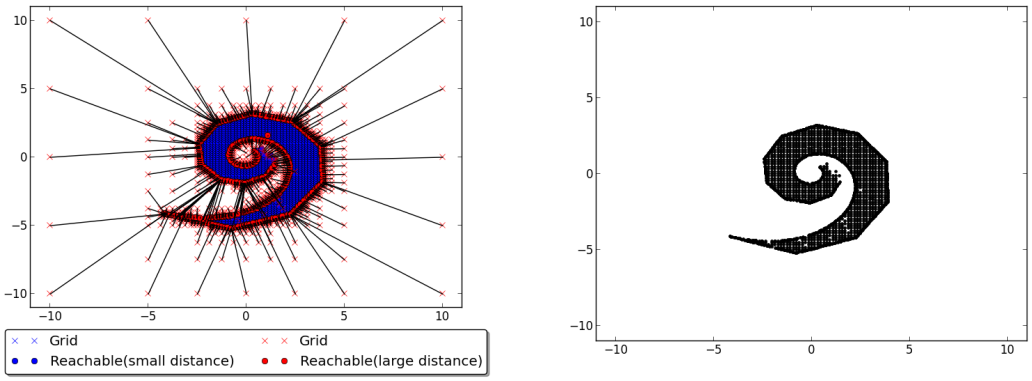


Abbildung 5.62.: Kenderov, $N_t = 20$, $N_x = 5$, $R = 5$, MODE 2, RESCALE 5

Auch bei Kenderov sind die Ergebnisse etwas ernüchternd. Der Kreis in der Mitte wird bei dieser Iteration nicht geschlossen. Allerdings stellt man fest, das im Inneren keine vereinzelten Punkte mit Distanz zum Gitterpunkt größer 0 gefunden werden. Verwendet man RESCALE 1 so kommt es immer wieder zu solchen vereinzelnen Punkten welche durch Abbruch des Optimieres mit Fehlermeldungen entstehen.

Falls bei RESCALE 5 der Optimierer eine Fehlermeldung liefert, wird einfach der Vorgang für eine anders skalierte Zielfunktion neu gestartet bis eine richtige Lösung gefunden wurde. Dieses Ergebnis kommt also wenig überraschend.

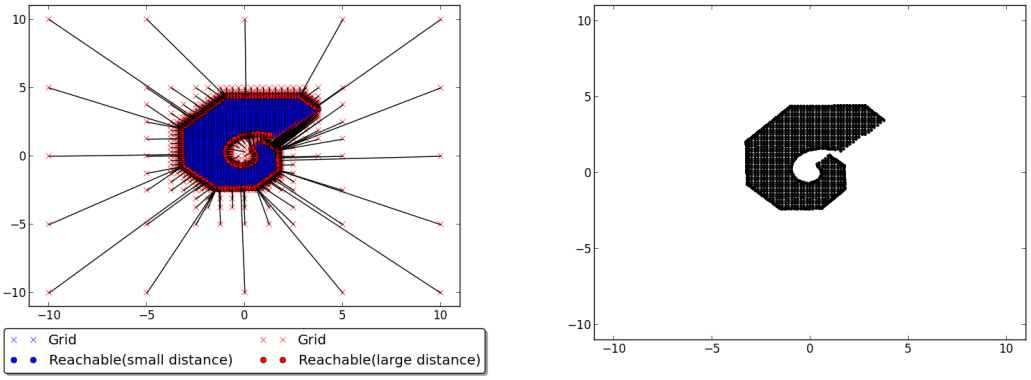


Abbildung 5.63.: Kenderov, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 0, RESCALE 5

Für MODE 0 und MODE 1 verbessern sich die Ergebnisse ebenfalls nur in diesem Rahmen und die Laufzeiten sind bei allen Modi erwartungsgemäß deutlich höher.

	MODE 0		MODE 1		MODE 2	
R	RECALE 1	RESCALE 5	RECALE 1	RESCALE 5	RECALE 1	RESCALE 5
3	0m4.651s	0m16.028s	0m3.478s	0m12.044s	0m9.612s	0m36.045s
4	0m13.876s	0m39.045s	0m9.888s	0m28.922s	0m27.269s	1m24.587s
5	0m46.220s	1m50.379s	0m32.013s	1m17.671s	1m27.167s	3m42.213s
6	2m40.345s	5m33.124s	1m50.682s	3m47.101s	4m59.182s	10m38.483s

Tabelle 5.1.: Laufzeitvergleich Schleife über Reskalierungen, MCS, Kenderov, $N_t = 16$, $N_x = 5$

Anders als bei der lokalen Variante muss man bei MCS leider das Fazit ziehen, dass sich eine solche Schleife nicht lohnt. Zwar werden die Lösungen minimal besser, allerdings werden nach wie vor Teile der Erreichbarkeitsmenge nicht gefunden. Für den Grad der Verbesserung lohnt sich der Aufwand den man betreibt nicht.

5.6. Berechnungen mit PSO

Der Partikelschwarmoptimierer lässt sich nur bis zu einem gewissen Grad anpassen. Zwar ist es möglich das generelle Verhalten der Partikel zu beeinflussen, eine gute Startschätzung lässt sich allerdings nicht angeben.

Primär wurde versucht die Ergebnisse mit verschiedenen lokalen Optimierern zu verbessern.

- MODE 0 - Dies ist die Standardeinstellung, welche keinen lokalen Optimierer verwendet um ein neu gefundenes globales Optimum weiter zu verbessern.
- MODE 1 - Verwendet den durch NAG bereitgestellten lokalen Optimierer E04CBF, ein Downhill-Simplex-Verfahren oder auch Nelder-Mead-Simplex-Verfahren um durch Partikel gefundene Zielfunktsionswerte weiter zu verbessern.
- MODE 2 - Verwendet E04JYF, einen Quasi-Newton-Algorithmus zur Bestimmung eines lokalen Minimums.

Die beiden lokalen Optimierer benötigen lediglich die Zielfunktion und nicht deren Ableitung. Die NAG-Library bietet zwar noch eine Reihe weiterer Optimierer an, allerdings benötigen diese neben der Zielfunktion auch noch passende Gradienten.

Anschließend wurden noch Versuche unternommen, die Parameter C_s und C_g, also die soziale und globalen Konstanten in der Bewegungsrichtungsgleichung, welche das Schwarmverhalten beeinflussen, zu variieren.

Im Allgemeinen wurde für die Boundary-Regelung die Option FLOATING und $C_s = C_g = 2.0$ als Konstanten für den Schwarm verwendet. Die Gewichtung der vorherigen Bewegungsrichtung w_j wird in jedem Iterationsschritt um jeweils 1% verringert.

Die Anzahl der Partikel wurde auf der Standardeinstellung $n_{par} = 10n$ belassen. Mit steigender Dimension steigt also auch die Zahl der Partikel.

5.6.1. Rayleigh

Auch hier ist zunächst eine Untersuchung des Verfahrens hinsichtlich qualitativer Aspekte nötig und besonders ein Vergleich zum lokalen Verfahren für grobe zeitliche Diskretisierung sinnvoll.

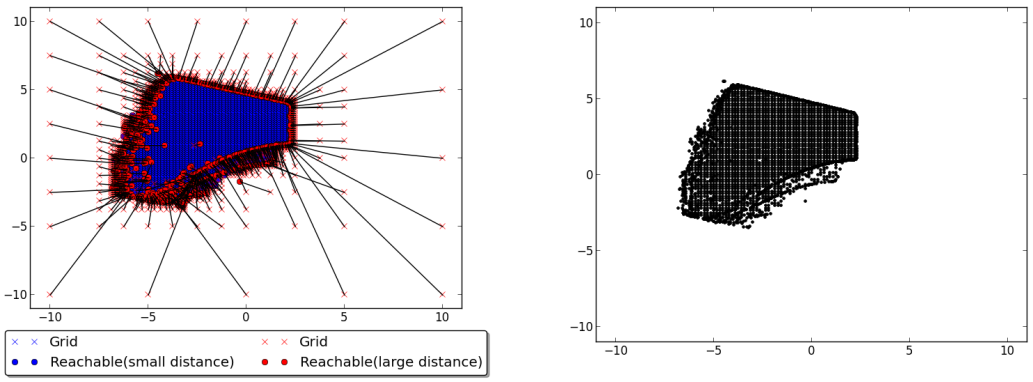


Abbildung 5.64.: Rayleigh, $N_t = 8$, $N_x = 5$, $R = 5$, MODE 0

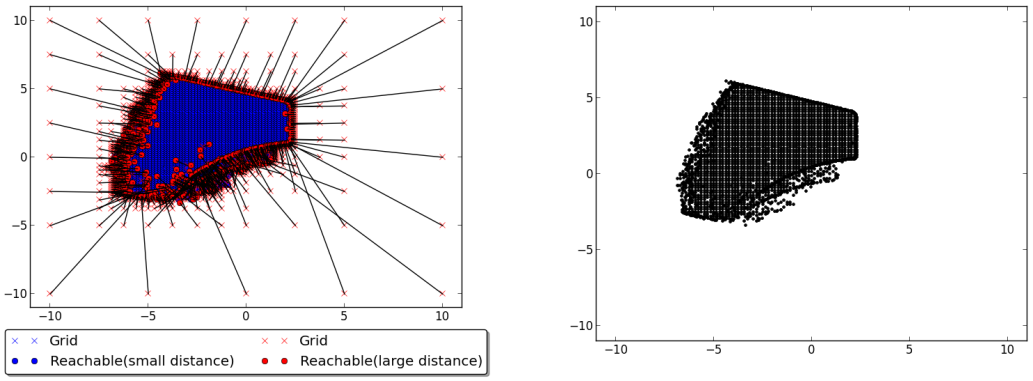


Abbildung 5.65.: Rayleigh, $N_t = 8$, $N_x = 5$, $R = 5$, MODE 1

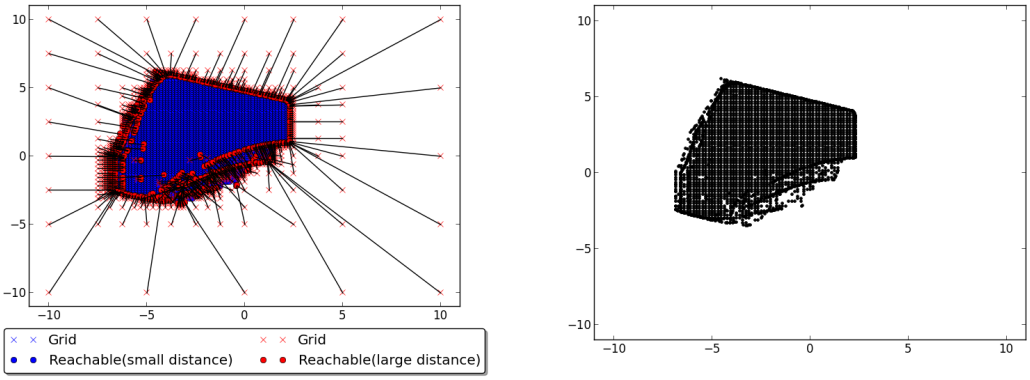


Abbildung 5.66.: Rayleigh, $N_t = 8$, $N_x = 5$, $R = 5$, MODE 2

Im Vergleich zu den Grafiken mit dem lokalen Optimierer fällt auf, dass zwar der untere Bereich besser dargestellt wird und gerade links unten eine weitaus größere Menge gefunden wird, dagegen aber im oberen linken Bereich bei allen Modi ein kleiner Teil fehlt.

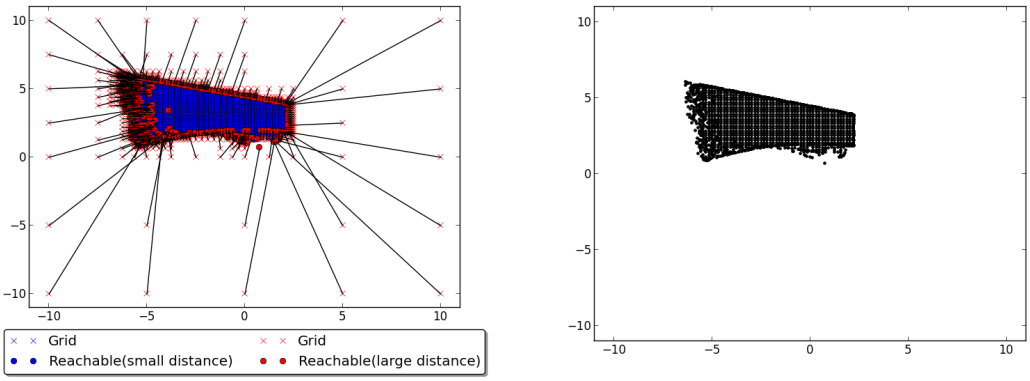


Abbildung 5.67.: Rayleigh, $N_t = 10$, $N_x = 5$, $R = 5$, MODE 0

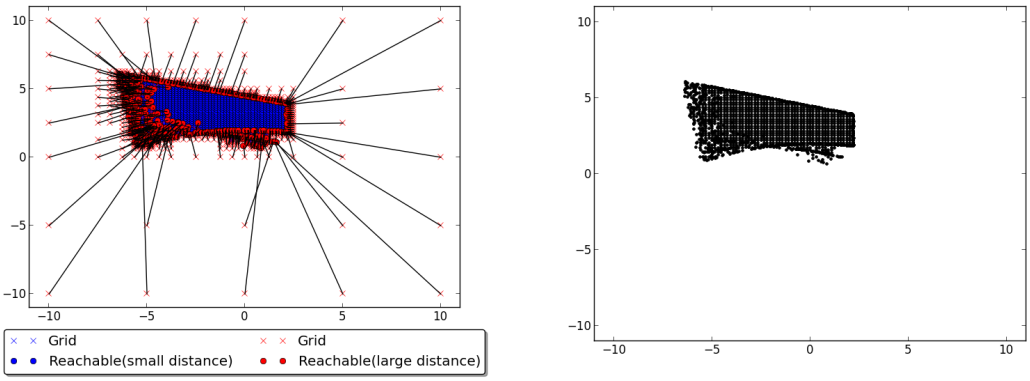


Abbildung 5.68.: Rayleigh, $N_t = 10$, $N_x = 5$, $R = 5$, MODE 1

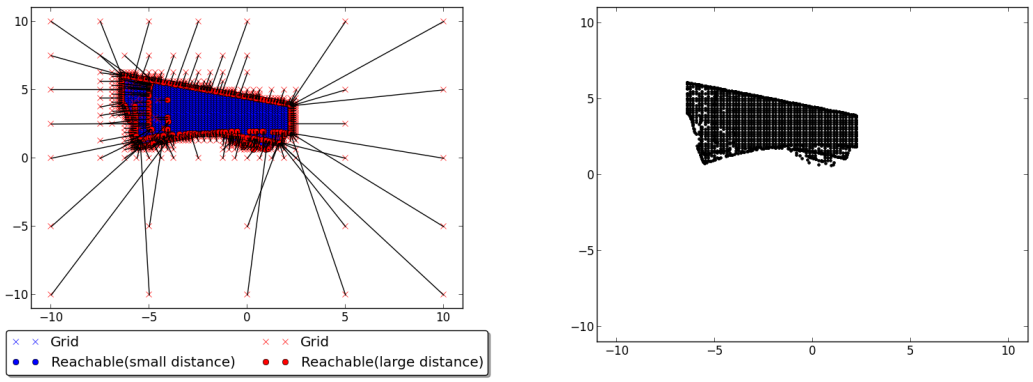


Abbildung 5.69.: Rayleigh, $N_t = 10$, $N_x = 5$, $R = 5$, MODE 2

Betrachtet man die Ergebnisse für $N_t = 10$ so wird deutlich, welchen Unterschied ein lokaler Optimierer im Partikelschwarmalgorithmus macht. Der linke Bereich bei dem die Kontrollen für dicht beieinanderliegenden Zielpunkten stark schwanken wird bei allen schlecht erreicht. Auch unten, wo bereits lokale Optimierer schlechte Ergebnisse liefern, variieren die erzeugten Mengen stark und die Wahl des lokalen Optimierers verändert die Qualität merklich.

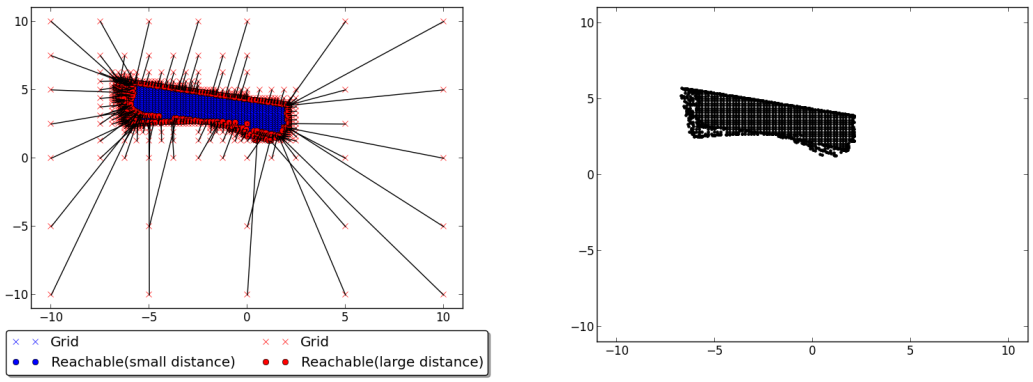


Abbildung 5.70.: Rayleigh, $N_t = 12$, $N_x = 5$, $R = 5$, MODE 0

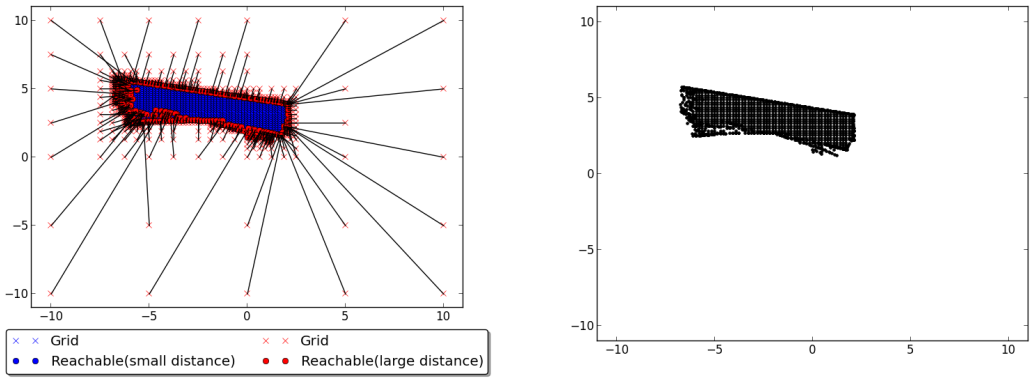


Abbildung 5.71.: Rayleigh, $N_t = 12$, $N_x = 5$, $R = 5$, MODE 1

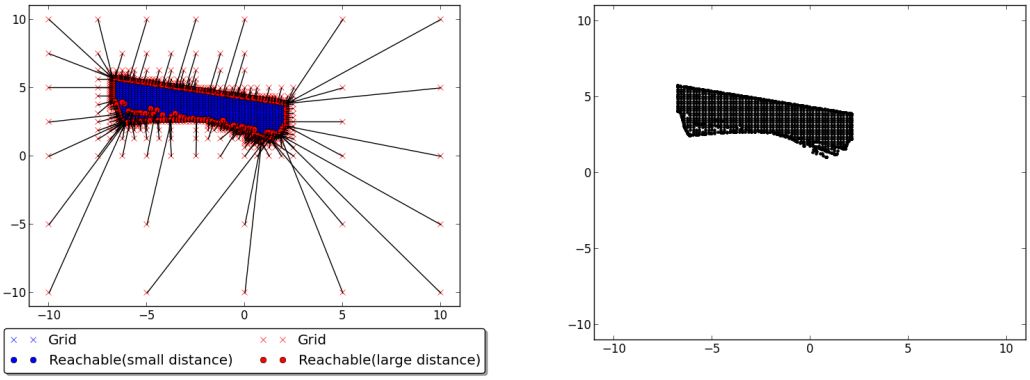


Abbildung 5.72.: Rayleigh, $N_t = 12$, $N_x = 5$, $R = 5$, MODE 2

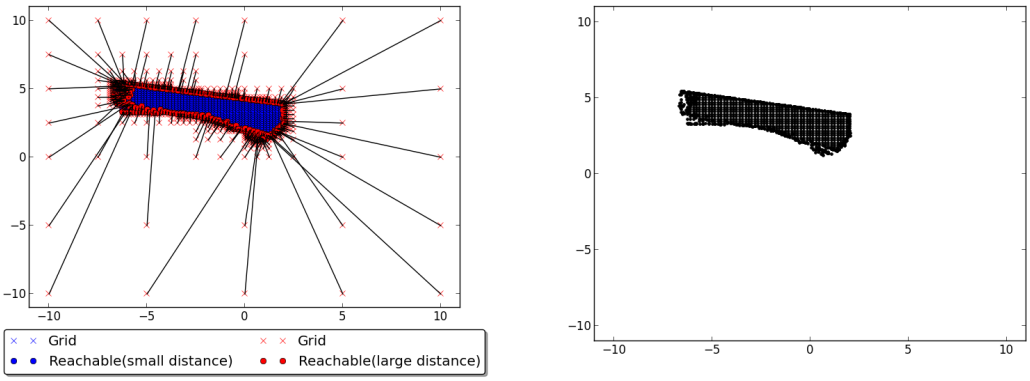


Abbildung 5.73.: Rayleigh, $N_t = 14$, $N_x = 5$, $R = 5$, MODE 0

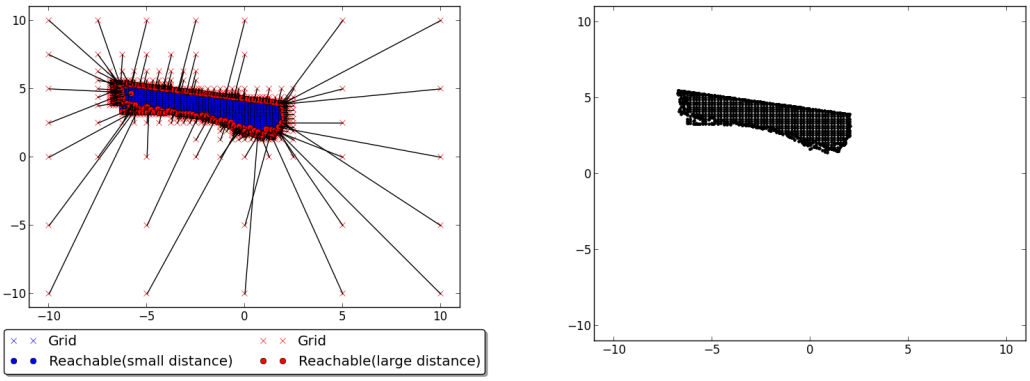


Abbildung 5.74.: Rayleigh, $N_t = 14$, $N_x = 5$, $R = 5$, MODE 1

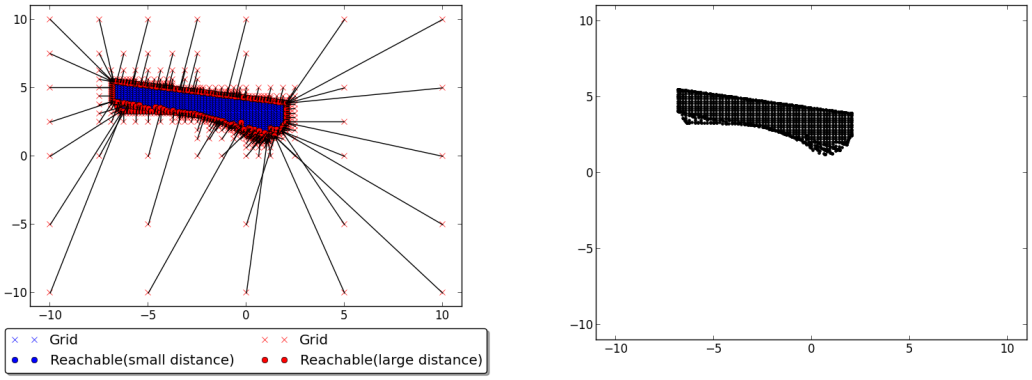


Abbildung 5.75.: Rayleigh, $N_t = 14$, $N_x = 5$, $R = 5$, MODE 2

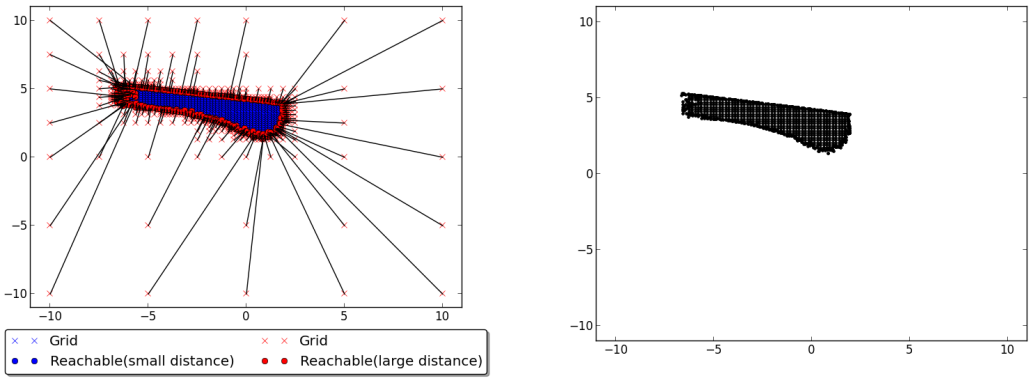


Abbildung 5.76.: Rayleigh, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 0

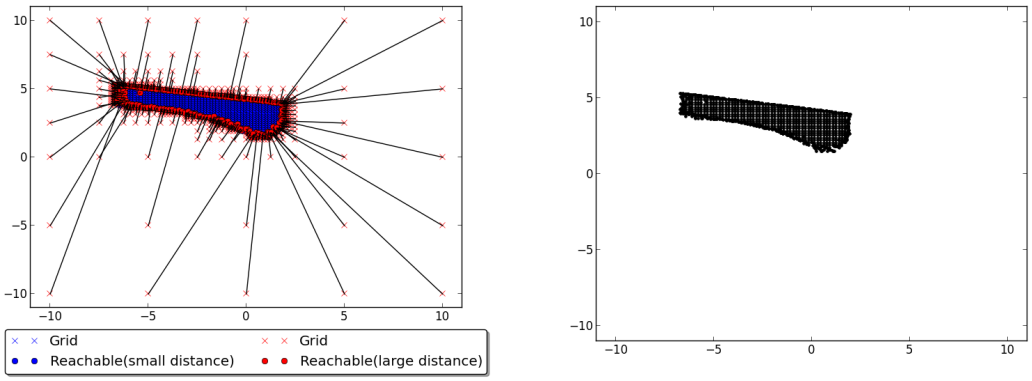


Abbildung 5.77.: Rayleigh, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 1

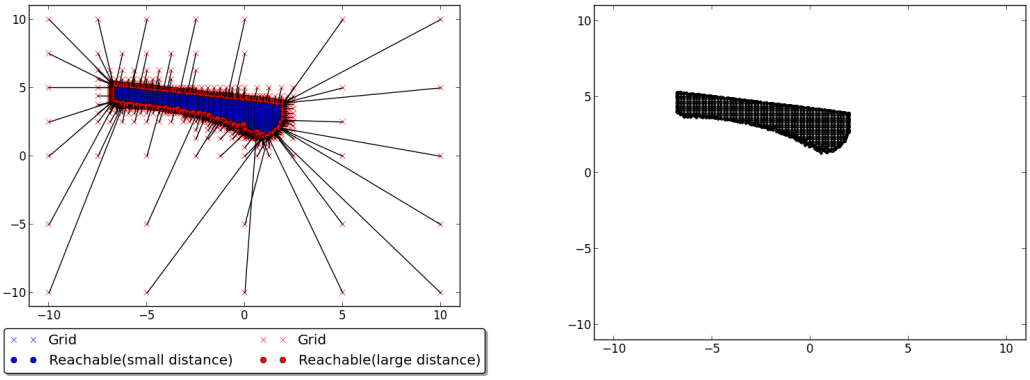


Abbildung 5.78.: Rayleigh, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 2

Mit steigendem N_t erkennt man, dass abgesehen von MODE 2 die erzeugten Grafiken teils deutlich schlechter sind, als die des lokalen Verfahrens. Gerade links wird der Rand nicht sauber dargestellt und es werden keine globalen Minima gefunden.

Betrachtet man die Ergebnisse mit MODE 2, so sieht man auch hier, dass die Ränder nicht so glatt sind wie sie sein müssten, aber zumindest links unten ein Bereich gefunden wird, welcher bisher unerschlossen war.

5.6.2. Kenderov

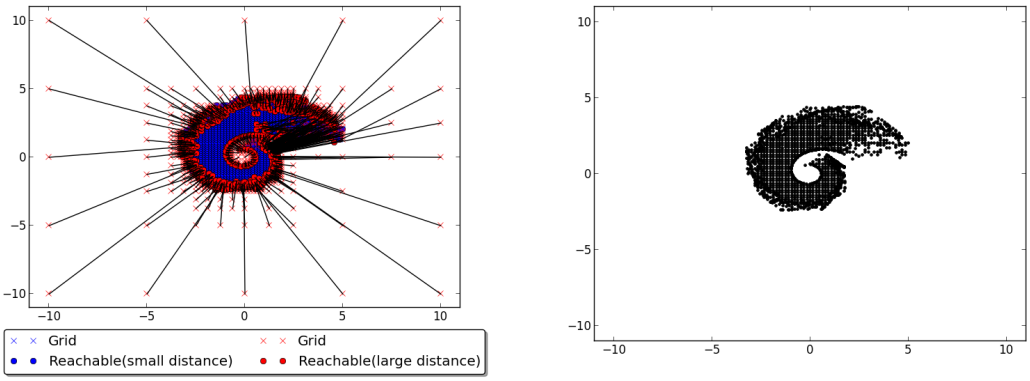


Abbildung 5.79.: Kenderov, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 0

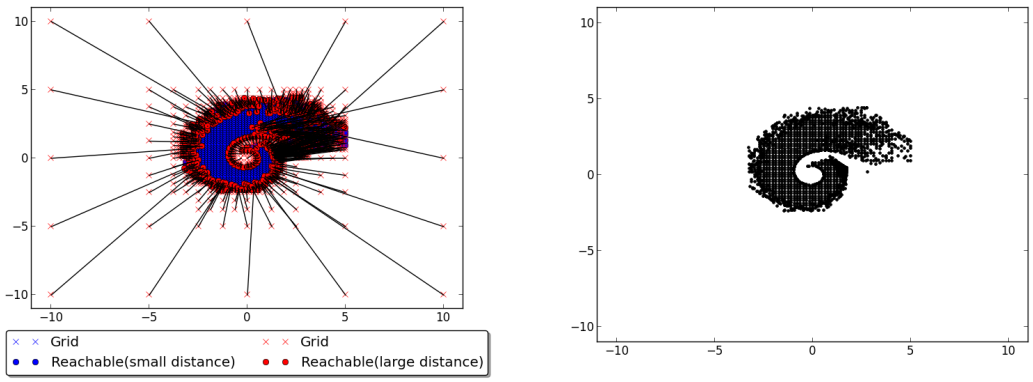


Abbildung 5.80.: Kenderov, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 1

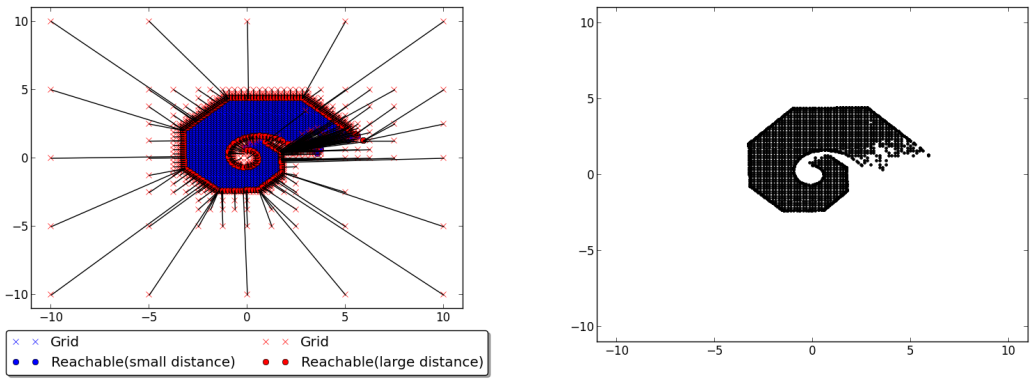


Abbildung 5.81.: Kenderov, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 2

Das Kenderov-Kontrollproblem offenbart die Schwächen des Partikelschwarmoptimierers noch deutlicher als das Rayleigh-Problem. Für MODE 0 und MODE 1 werden die äußeren Ränder überhaupt nicht korrekt berechnet und die Ergebnisse sind deutlich schlechter als beim lokalen Verfahren.

Mit MODE 2 wird immerhin der äußere Rand soweit sauber berechnet, dies ist allerdings dem verwendeten lokalen Optimierer im PSO-Verfahren zuzuschreiben.

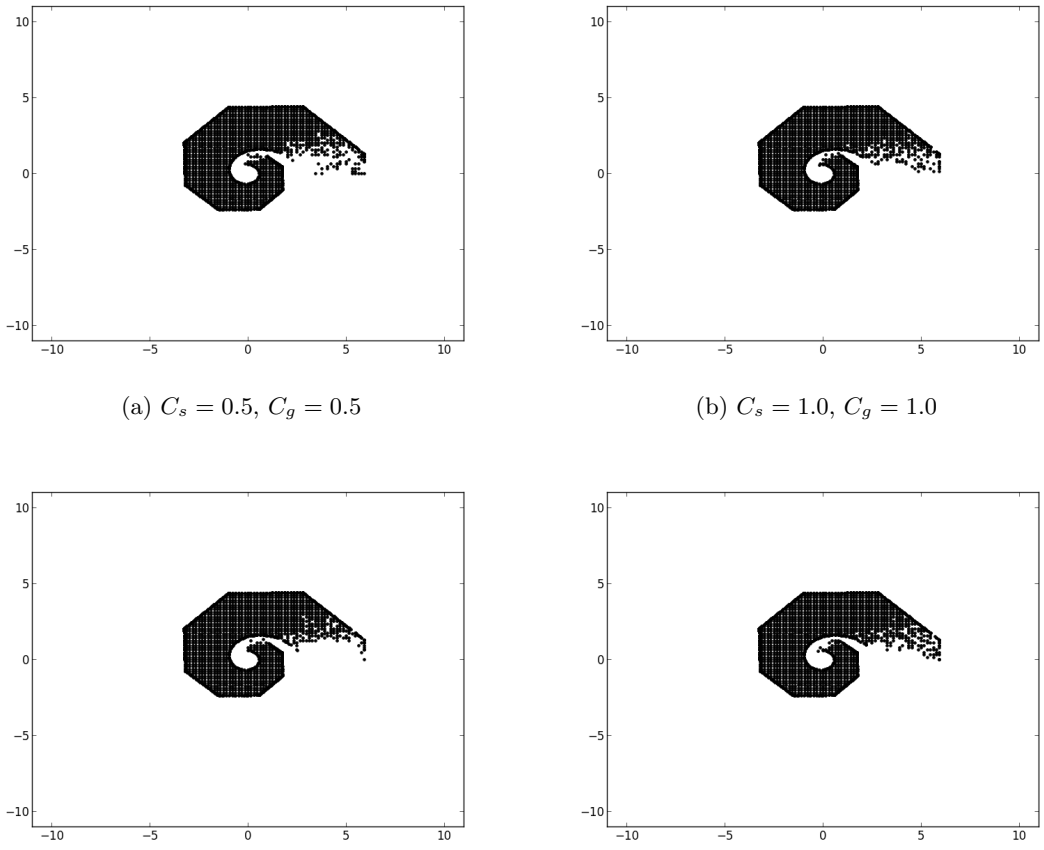
Bei allen Modi ist aber deutlich erkennbar, dass ein großer Teil der Erreichbarkeitsmenge komplett fehlt, genau der Teil, der auch bei der lokalen Variante nicht dargestellt wird, wenn man Startschätzung $u_i = 0.0 \, \forall i = 0, \ldots, N_t - 1$ verwendet. Der PSO-Algorithmus hat also ähnlich wie das MCS-Verfahren den Hang dazu, große Teile der Erreichbarkeitsmenge nicht zu finden und liefert häufig lokale Minima.

5.6.3. Optionsvariation des Optimierers

An diesem Punkt sind Experimente nötig, welche klären ob sich die Ergebnisse im Rahmen der durch das Verfahren bereitgestellten Optionen weiter verbessern lässt. Klarerweise haben die Konstanten C_s und C_g den größten Einfluss auf das Verhalten der einzelnen Partikel, sofern man die Gewichtung der Vorgängerbewegungsrichtung w_j in jedem Schritt prozentual fallen lässt. Mit steigender Iterationsanzahl spielt die Vorgängerrichtung v_j^i nur noch eine geringe Rolle und die Terme, welche Attraktion zum globalen und bisher besten Optimum steuern, sind dominant.

Dazu wurden für MODE 2, welcher bei Standardwerten von $C_s = C_g = 2.0$ die besten Ergebnisse lieferte, diese Konstanten im Bereich $[0.5, 2.0]$ variiert.

Wie die nachfolgenden Grafiken zeigen, ändert sich an den Ergebnissen leider wenig. Es werden zwar gerade bei Kenderov im rechten äußeren Bereich unterschiedliche Punkte gefunden, diese sind aber wohl ausschließlich dem Zufallsgenerator zu verdanken, der hin und wieder schöne Startwerte für einzelne Partikel wählt. Qualitativ ändert sich an den Ergebnissen also sowohl beim Rayleigh- also auch beim Kenderov-Problem nichts.



(a) $C_s = 0.5$, $C_g = 0.5$

(b) $C_s = 1.0$, $C_g = 1.0$

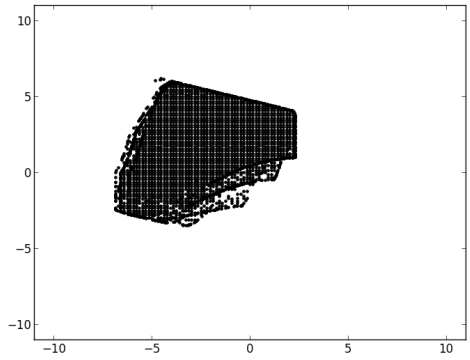
(c) $C_s = 2.0$, $C_g = 0.5$

(d) $C_s = 0.5$, $C_g = 2.0$

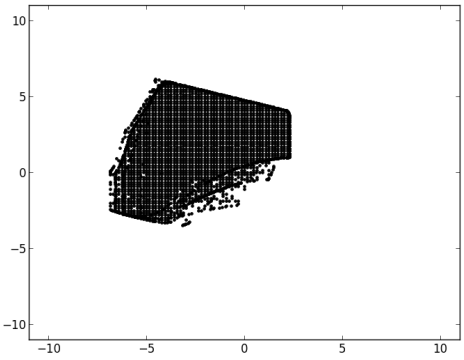
Abbildung 5.82.: Kenderov, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 2

c_s \ c_g	0.5	1.0	1.5	2.0
0.5	2m12.026s	1m28.951s	1m42.017s	1m55.322s
1.0	3m51.722s	2m24.466s	1m49.222s	2m14.343s
1.5	5m13.796s	3m28.144s	3m5.308s	3m42.074s
2.0	6m43.673s	4m51.916s	5m8.163s	12m29.000s

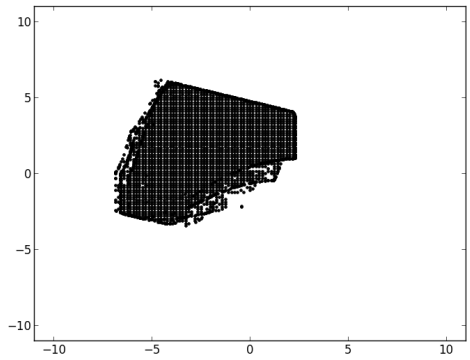
Tabelle 5.2.: Laufzeit PSO, Kenderov, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 2



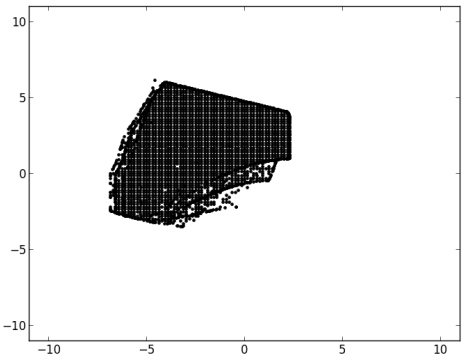
(a) $C_s = 0.5$, $C_g = 0.5$



(b) $C_s = 1.0$, $C_g = 1.0$



(c) $C_s = 2.0$, $C_g = 0.5$

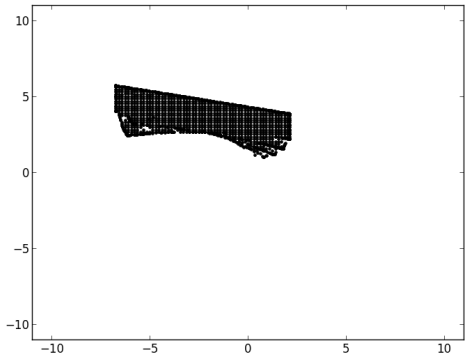


(d) $C_s = 0.5$, $C_g = 2.0$

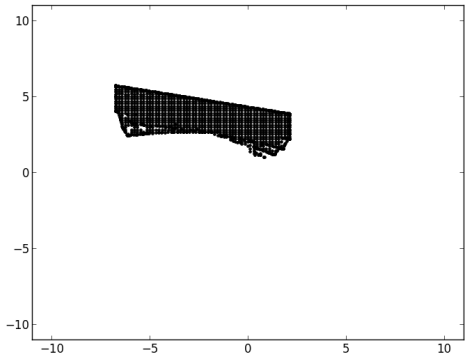
Abbildung 5.83.: Rayleigh, $N_t = 8$, $N_x = 5$, $R = 5$, MODE 2

c_s \ c_g	0.5	1.0	1.5	2.0
0.5	0m44.890s	0m29.157s	0m28.402s	0m30.746s
1.0	1m16.613s	0m41.497s	0m36.110s	0m36.092s
1.5	1m40.509s	1m1.167s	0m48.589s	0m46.773s
2.0	3m2.897s	1m24.085s	1m7.090s	1m15.371s

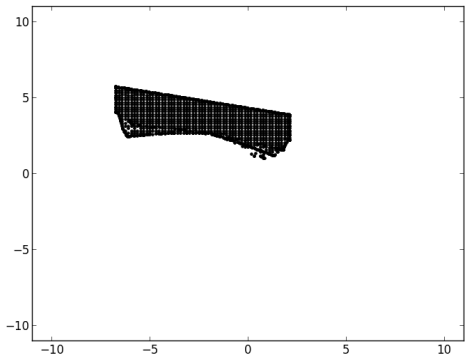
Tabelle 5.3.: Laufzeit PSO, Kenderov, $N_t = 8$, $N_x = 5$, $R = 5$, MODE 2



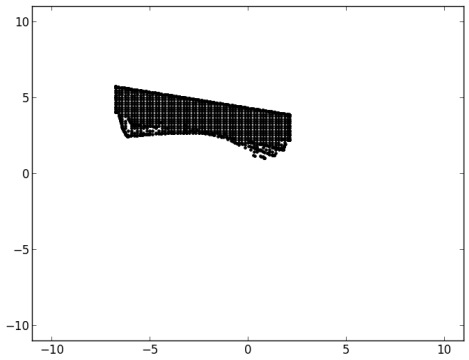
(a) $C_s = 0.5$, $C_g = 0.5$



(b) $C_s = 1.0$, $C_g = 1.0$



(c) $C_s = 2.0$, $C_g = 0.5$



(d) $C_s = 0.5$, $C_g = 2.0$

Abbildung 5.84.: Rayleigh, $N_t = 12$, $N_x = 5$, $R = 5$, MODE 2

c_s \ c_g	0.5	1.0	1.5	2.0
0.5	0m26.463s	0m26.715s	0m28.320s	0m31.498s
1.0	0m30.864s	0m29.787s	0m31.889s	0m34.831s
1.5	0m40.747s	0m35.434s	0m37.401s	0m42.086s
2.0	1m11.656s	0m44.081s	0m47.189s	0m58.888s

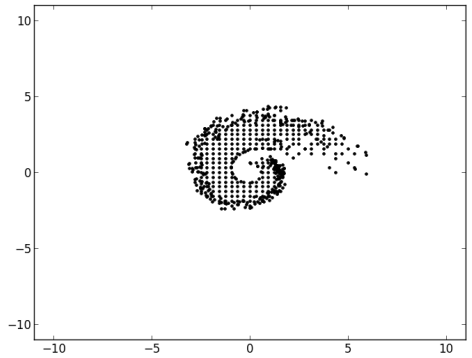
Tabelle 5.4.: Laufzeit PSO, Kenderov, $N_t = 12$, $N_x = 5$, $R = 5$, MODE 2

Untersucht man die Laufzeiten der unterschiedlichen Parameterkombinationen so wird deutlich, dass wenn die soziale Komponente C_s deutlich überwiegt, die Laufzeit höher wird. Dies liegt daran, dass je höher die Attraktion zum eigenen gefundenen Optimum stärker als die zum globalen aller Partikel ist, desto länger brauchen die Partikel um zum globalen Optimum zu kommen und zurückgesetzt zu werden. Da die Anzahl der Partikel, die zum aktuellen globalen Optimum gesteuert und zurückgesetzt wurden, eines der wichtigsten Abbruchkriterien ist, ist eine höhere Laufzeit hier verständlich. Wenn hingegen C_g die dominantere Komponente ist, konvergieren die einzelnen Partikel schneller zum bisher gefundenen besten Optimum. Dieses Verhalten wird zumindest bei Rayleigh deutlich, ist im groben aber auch bei Kenderov erkennbar.

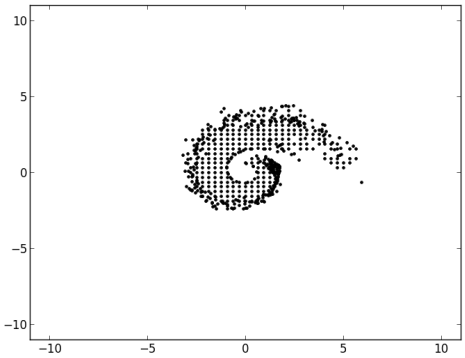
Bei Kenderov auf der anderen Seite schwanken die Zeiten teilweise anders, dies liegt allerdings daran, dass doch vereinzelt Punkte gefunden, welche im rechten äußeren Bereich liegen, gefunden werden. Die sehr kleine Zeit für $C_s = 1.0$ und $C_g = 1.5$ folgt aus der Tatsache, das zufallsbedingt ein großer Block nicht mit berechnet wurde.

Insgesamt lässt sich sagen, dass eine Variation der Konstanten C_s und C_g zumindest mit dem verwendeten lokalen Optimierer in MODE 2 im Hinblick auf die Qualität der Ergebnisse kaum einen Nutzen bringt. Zwar lässt sich die Laufzeit vermindern, indem man die soziale Komponente C_s kleiner wählt, an der eigentlichen Qualität der Ergebnisse ändert dies jedoch nichts.

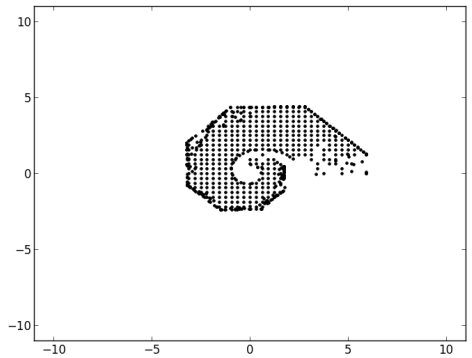
Betrachtet man allerdings das reine PSO-Verfahren mit MODE 0 ohne in diesem einen lokalen Optimierer zu verwenden, so zeigt sich sich bei Kenderov jedoch, dass eine Variation der Parameter im Allgemeinen sinnvoll ist.



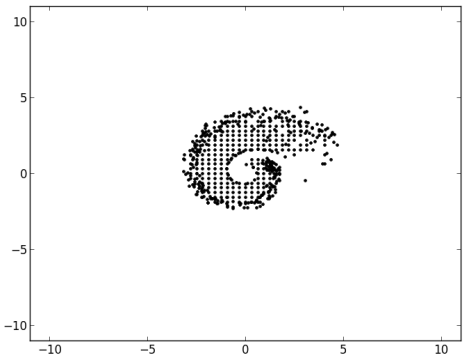
(a) $C_s = 0.5$, $C_g = 0.5$



(b) $C_s = 1.0$, $C_g = 1.0$



(c) $C_s = 2.0$, $C_g = 0.5$



(d) $C_s = 0.5$, $C_g = 2.0$

Abbildung 5.85.: Kenderov, $N_t = 16$, $N_x = 5$, $R = 4$, MODE 0

Hier wird deutlich, dass falls C_s größer als C_g ist, neigen die Partikel dazu, sich weiter voneinander zu verstreuen und finden so Punkte im Suchraum, die bei anderer Parameterwahl nicht erreicht werden. Die so gefundenen Punkte werden bei den anderen Modi allerdings als Basis für die lokale Optimierung gewählt, weshalb an dieser Stelle klar wird, warum bei MODE 2 keine besseren Ergebnisse erzielt werden. Prinzipiell arbeitet der lokale Optimierer hier außerordentlich gut, kann allerdings bei schlechter Startparameterwahl keine besseren Lösungen finden.

c_s \ c_g	0.5	1.0	1.5	2.0
0.5	6m47.315s	5m47.063s	6m32.830s	6m50.146s
1.0	11m16.592s	7m31.417s	8m3.651s	7m7.666s
1.5	18m12.301s	10m24.863s	8m25.980s	9m32.511s
2.0	33m37.170s	13m55.405s	11m55.927s	12m2.772s

Tabelle 5.5.: Laufzeit PSO, Kenderov, $N_t = 16$, $N_x = 5$, $R = 5$, MODE 2

Wie auch bei MODE 2 erkennt man, dass mit steigendem sozialem Anteil die Laufzeit höher wird. Ebenso ist das Verhalten des Schwarms mit zugehörigem Abbruchkriterium der Grund für diesen Zuwachs.

5.7. Laufzeitvergleiche

Neben einer qualitativen Betrachtung der Verfahren spielt natürlich auch die Laufzeit dieser mit unterschiedlichen Optionen vor allem im Hinblick auf die lokale Variante eine wichtige Rolle.

Zunächst ist aber erwähnenswert, dass sich die CPU-Zeiten und die Realzeiten beim lokalen Verfahren massiv voneinander unterscheiden. Es lassen sich hier nur Vermutungen über den Grund dafür anstellen. Es liegt im Bereich des möglichen, dass durch Systemausgaben im lokalen Optimierer viel Zeit verloren geht, da dieser eine Unmenge an Daten erzeugt, die für Optimalsteuerungsprobleme von Interesse sein können, für die Berechnung von Erreichbarkeitsmengen aber nur eine sehr untergeordnete Rolle spielen. Bei den globalen Optimierern wurden die Ausgaben aus diesem Grund auf das nötigste beschränkt.

Allerdings treten auch große Differenzen auf, wenn man die Ausgabe des lokalen Optimierers nach /dev/null umleitet und so unterdrückt. Es ist also auch vorstellbar, dass intern viel Speicher allokiert und deallokiert wird, ein eindeutiger Schuldiger lässt sich allerdings nicht ermitteln.

Insgesamt werden aufgrund dessen neben den realen Laufzeiten auch noch die CPU-Zeiten der Variante mit dem lokalen Optimierer angegeben.

Beim Globalen Optimierer halten sich die Differenzen in Grenzen, sowohl beim MCS- als auch beim PSO-Verfahren sind die Unterschiede bei den üblichen Standardeinstellung über alle Modi hinweg minimal.

	Real	CPU
MODE 0	2m41.865s	2m41.638s
MODE 1	1m51.193s	1m51.051s
MODE 2	5m1.309s	5m0.223s
MODE 3	2m8.906s	2m8.516s
MODE 4	2m31.747s	2m31.525s
MODE 6	19m4.085s	19m3.075s

(a) MCS

	Real	CPU
MODE 0	120m46.735s	120m45.537s
MODE 1	95m39.099s	95m39.055s
MODE 2	39m38.437s	39m38.125s

(b) PSO

Tabelle 5.6.: Real- und CPU-Zeitvergleiche, Kenderov, $N_t = 16$, $N_x = 5$, $R = 6$

Bei MCS konnte zwar ein größerer Unterschied der Zeiten bei deutlich erhöhter maximaler Splitanzahl und MODE 2 ausgemacht werden, aber selbst hier liegen sie nicht um ein

Vielfaches höher. Beispielsweise beträgt für $s_{\max} = 200n$ die Realzeit 459m22.948s, wohingegen die CPU-Zeit auf 419m42.310s schrumpft. Die Abweichung hier ist zwar groß, aber sind nicht ansatzweise in dem Bereich jener des lokalen Optimierers in Extremfällen und treten in diesem Maße auch nicht bei den in diesem Abschnitt betrachteten Zeitvergleichen auf.

Aus diesem Grund wird für die globalen Verfahren darauf verzichtet die CPU-Zeiten mit anzugeben.

Ferner sei erwähnt, dass alle Versuche bei keiner oder wenn dann minimaler Belastung des System durch andere Programme durchgeführt wurden. Der verwendete Rechner wies die hier aufgelisteten Spezifikationen auf.

CPU	AMD Phenom II x4 940 Quadcore mit 3000 MHz pro Kern
Arbeitsspeicher	4 Gigabyte
Betriebsystem	CrunchBang 11 „Waldorf“ x68 - 64bit based on Debian 7 „Wheezy“
Kernel	3.2.0-4-amd64
Compiler	GNU Fortran (Debian 4.6.3-14) 4.6.3
NAG-Library	NAG Fortran Mark 23 for GNU Fortran 64 bit (FLL6A23DFL)

5.7.1. Rayleigh

Das Rayleigh-Problem eignet sich besonders gut zum Vergleich der Laufzeiten, da die berechneten approximierten Erreichbarkeitsmengen für unterschiedliche N_t ähnliche Ausmaße haben und sich qualitativ nicht zu sehr voneinander unterscheiden. Ferner lässt sich für dieses Problem leicht ein großer Bereich an unterschiedlichen N_t betrachten und so die Optimierer und deren Optionen im Hinblick auf unterschiedlich dimensionale Probleme in Bezug auf Geschwindigkeit vergleichen.

Für das Rayleigh-Problem ergeben sich für unterschiedliche N_t die nachfolgend aufgelisteten Laufzeiten.

R	MODE 0	MODE 1	MODE 2	MODE 3	MODE 4	MODE 6
3	0m5.504s	0m4.022s	0m8.265s	0m4.580s	0m5.325s	1m17.732s
4	0m16.025s	0m11.765s	0m19.755s	0m13.494s	0m14.591s	2m36.950s
5	0m53.501s	0m40.060s	0m59.685s	0m47.225s	0m43.843s	6m11.485s
6	3m17.878s	2m25.082s	3m25.066s	2m53.087s	2m31.662s	16m12.803s

(a) MCS, Realzeit

R	MODE 0	MODE 1	MODE 2
3	0m45.536s	0m38.943s	0m7.578s
4	1m41.249s	1m24.869s	0m22.298s
5	4m47.005s	3m59.267s	1m13.532s
6	15m16.780s	13m53.956s	4m28.634s

(b) PSO, Realzeit

R	REAL	CPU
3	0m47.156s	0m15.433s
4	3m3.423s	0m59.828s
5	13m18.305s	4m6.643s
6	49m58.070s	14m38.151s

(c) Locale Version

Tabelle 5.7.: Laufzeitvergleich, Rayleigh, $N_t = 8$, $N_x = 5$

R	MODE 0	MODE 1	MODE 2	MODE 3	MODE 4	MODE 6
3	0m2.724s	0m2.401s	0m2.878s	0m2.303s	0m1.993s	0m29.066s
4	0m8.205s	0m6.320s	0m8.552s	0m6.529s	0m5.095s	0m55.395s
5	0m26.968s	0m19.345s	0m27.992s	0m20.156s	0m16.890s	2m2.668s
6	1m37.638s	1m8.097s	1m39.147s	1m8.855s	0m55.301s	4m50.390s

(a) MCS, Realzeit

R	MODE 0	MODE 1	MODE 2
3	0m45.686s	0m34.252s	0m5.772s
4	1m35.710s	1m21.818s	0m15.562s
5	4m13.372s	3m15.267s	0m48.119s
6	12m0.674s	9m25.912s	2m46.641s

(b) PSO, Realzeit

R	REAL	CPU
3	0m53.833s	0m17.941s
4	1m23.793s	0m27.046s
5	4m23.312s	1m21.525s
6	14m19.159s	4m28.165s

(c) Locale Version

Tabelle 5.8.: Laufzeitvergleich, Rayleigh, $N_t = 10$, $N_x = 5$

R	MODE 0	MODE 1	MODE 2	MODE 3	MODE 4	MODE 6
3	0m3.000s	0m2.763s	0m3.740s	0m2.531s	0m2.385s	0m33.871s
4	0m8.222s	0m6.754s	0m9.192s	0m6.240s	0m5.757s	1m1.540s
5	0m25.197s	0m19.428s	0m27.332s	0m19.165s	0m16.380s	2m3.142s
6	1m27.709s	1m6.259s	1m31.872s	1m3.134s	0m55.424s	4m34.942s

(a) MCS, Realzeit

R	MODE 0	MODE 1	MODE 2
3	1m21.386s	1m1.288s	0m9.096s
4	2m29.523s	2m0.500s	0m22.001s
5	5m44.397s	4m29.303s	0m59.007s
6	15m27.469s	12m8.695s	3m13.384s

(b) PSO, Realzeit

R	REAL	CPU
3	0m3.878s	0m1.352s
4	0m8.516s	0m3.016s
5	0m18.907s	0m6.664s
6	1m18.413s	0m25.814s

(c) Locale Version

Tabelle 5.9.: Laufzeitvergleich, Rayleigh, $N_t = 12$, $N_x = 5$

R	MODE 0	MODE 1	MODE 2	MODE 3	MODE 4	MODE 6
3	0m3.210s	0m2.697s	0m4.131s	0m2.888s	0m2.433s	0m36.574s
4	0m8.183s	0m6.878s	0m9.957s	0m6.955s	0m6.260s	1m10.511s
5	0m25.766s	0m20.866s	0m29.393s	0m20.596s	0m19.137s	2m29.585s
6	1m28.008s	1m11.973s	1m38.990s	1m9.911s	1m5.983s	5m47.552s

(a) MCS, Realzeit

R	MODE 0	MODE 1	MODE 2
3	1m56.166s	1m29.505s	0m13.889s
4	3m28.777s	3m4.404s	0m29.136s
5	8m14.349s	6m18.788s	1m18.447s
6	20m19.305s	17m29.950s	4m1.420s

(b) PSO, Realzeit

R	REAL	CPU
3	0m6.515s	0m2.204s
4	0m13.569s	0m4.592s
5	0m28.065s	0m8.705s
6	1m2.397s	0m20.029s

(c) Locale Version

Tabelle 5.10.: Laufzeitvergleich, Rayleigh, $N_t = 14$, $N_x = 5$

R	MODE 0	MODE 1	MODE 2	MODE 3	MODE 4	MODE 6
3	0m4.180s	0m4.016s	0m4.616s	0m3.730s	0m3.967s	0m51.773s
4	0m10.482s	0m10.445s	0m12.112s	0m9.313s	0m10.896s	1m44.983s
5	0m30.838s	0m28.385s	0m35.721s	0m26.593s	0m26.494s	3m31.773s
6	1m41.672s	1m31.292s	1m58.995s	1m26.975s	1m26.093s	7m47.824s

(a) MCS, Realzeit

R	MODE 0	MODE 1	MODE 2
3	2m52.820s	2m14.917s	0m22.324s
4	5m47.481s	4m27.171s	0m49.012s
5	11m8.273s	9m19.281s	1m49.899s
6	28m21.598s	22m55.324s	5m37.931s

(b) PSO, Realzeit

R	REAL	CPU
3	0m6.702s	0m2.084s
4	0m15.192s	0m4.892s
5	0m33.220s	0m10.701s
6	1m18.364s	0m26.538s

(c) Locale Version

Tabelle 5.11.: Laufzeitvergleich, Rayleigh, $N_t = 16$, $N_x = 5$

R	MODE 0	MODE 1	MODE 2	MODE 3	MODE 4	MODE 6
3	0m5.064s	0m5.671s	0m6.213s	0m5.464s	0m5.379s	1m14.724s
4	0m12.929s	0m13.894s	0m15.872s	0m12.994s	0m12.815s	2m34.553s
5	0m37.771s	0m36.738s	0m46.629s	0m35.564s	0m37.173s	5m15.337s
6	2m4.496s	1m58.534s	2m29.033s	1m51.618s	1m55.126s	11m15.253s

(a) MCS, Realzeit

R	MODE 0	MODE 1	MODE 2
3	4m5.921s	3m24.891s	0m48.582s
4	8m28.264s	5m46.021s	1m27.960s
5	17m37.854s	12m31.272s	3m10.798s
6	43m1.679s	31m17.804s	9m6.287s

(b) PSO, Realzeit

R	REAL	CPU
3	0m6.702s	0m2.084s
4	0m15.192s	0m4.892s
5	0m33.220s	0m10.701s
6	1m18.364s	0m26.538s

(c) Locale Version

Tabelle 5.12.: Laufzeitvergleich, Rayleigh, $N_t = 18$, $N_x = 5$

R	MODE 0	MODE 1	MODE 2	MODE 3	MODE 4	MODE 6
3	0m6.710s	0m8.145s	0m7.887s	0m7.803s	0m7.281s	1m31.515s
4	0m16.620s	0m19.749s	0m20.260s	0m18.238s	0m16.331s	3m14.963s
5	0m50.154s	0m57.229s	0m59.077s	0m50.937s	0m44.453s	6m41.549s
6	2m38.805s	2m55.178s	3m10.262s	2m40.673s	2m23.876s	14m34.776s

(a) MCS, Realzeit

R	MODE 0	MODE 1	MODE 2
3	7m23.052s	4m37.943s	1m26.533s
4	12m27.563s	10m43.125s	2m29.030s
5	24m37.387s	19m22.362s	6m46.202s
6	58m28.904s	45m28.034s	16m24.362s

(b) PSO, Realzeit

R	REAL	CPU
3	0m10.066s	0m3.248s
4	0m20.620s	0m6.824s
5	0m45.692s	0m13.517s
6	1m43.969s	0m31.062s

(c) Locale Version

Tabelle 5.13.: Laufzeitvergleich, Rayleigh, $N_t = 20$, $N_x = 5$

Wenn man sich die Laufzeittabellen näher betrachtet, so stellt man verschiedene Phänomene fest.

- Die Zeiten sind bei allen Varianten für kleine N_t im Bereich 8 bis 10 ähnlich. Dies liegt an dem Verhalten des diskretisierten Kontrollsystems, welches für kleine N_t größere Erreichbarkeitsmengen erzeugt und für größere N_t gegen die Erreichbarkeitsmenge des kontinuierlichen Systems welche unabhängig von N_t ist konvergiert.
  Da hier überall die adaptive Variante getestet wurde, ist es nicht verwunderlich, dass die Laufzeit mit flächenmäßig größerer Erreichbarkeitsmenge größer wird, immerhin muss in diesem Fall ein größerer Bereich durch das adaptive Verfahren verfeinert werden.
  Im Gegenzug sind aber die resultierenden Optimierungsprobleme für kleine N_t niedrigdimensional, nämlich von der Dimension N_t da die Dimension des Kontrollraums bei Rayleigh 1 ist.
  Daher sinken die Zeiten wenn man die zeitliche Diskretisierung von $N_t = 8$ auf $N_t = 10$ erhöht, steigen dann aber beim Schritt von $N_t = 10$ auf $N_t = 12$.

- Je größer man ab $N_t = 10$ die Anzahl an Zeitschritten wählt, desto länger werden die erreichten Laufzeiten aufgrund der höheren Komplexität der zu lösenden Optimierungsprobleme. Da die errechneten Mengen aber hinsichtlich der Größe auf einem ähnlichen Niveau bleiben, wird der zunehmende Aufwand nicht ausgeglichen.

- Die Laufzeiten mit PSO sind für MODE 0 und MODE 1 erstaunlich hoch, selbst wenn die Lösungen zumindest für größere N_t ähnlich gut sind, wie die des lokalen Optimierers oder dem MCS-Verfahren, ist die deutlich höhere Laufzeit von Nachteil. MODE 2 schafft es die Laufzeit gegenüber den anderen Modi massiv zu reduzieren und dabei ähnlich gute, wenn nicht sogar bessere, Resultate zu erzielen. Beispielsweise ist für $N_t = 20$ die Laufzeit von MODE 0 ungefähr viermal so hoch, als bei MODE 2.

- Vergleicht man die Laufzeiten des PSO-Verfahrens mit denen der lokalen Variante auch im Hinblick darauf, dass die Ergebnisse für kleine N_t qualitativ zwar besser sind weil ein größerer Bereich gefunden wird, so lohnt der zeitliche Aufwand hier nicht.
  Gerade mit steigendem N_t werden die Zeiten schnell sehr groß und können nicht im Ansatz die verwendete Rechenzeit in besseren Resultaten widerspiegeln.

- Mithilfe des MCS-Verfahrens lassen sich die Probleme augenscheinlich erstaunlich schnell lösen. Die Unterschiede zwischen MODE 0, MODE 1, MODE 3 (mit 15 gleichverteilten Punkten in den Initialisierungslisten) und MODE 4 sind hier vernachlässigbar. Die Schwankungen kommen wohl daher, das im einen Fall nach der Initialisierungsroutine schon bessere Punkte gefunden als bei anderen vordefinierten Initialisierungslisten.
  Betrachtet man aber die Zeiten von MODE 4, also der Version mit zufälligen Initialisierungslisten, so fällt auf, dass das Problem wohl relativ gnädig mit schlechten Startschätzungen umgeht. Dies soll heißen, dass auch wenn zufallsbedingt schlechte Startwerte gewählt werden, wirkt sich dies bei Rayleigh nicht zu sehr auf die Laufzeit aus. Gleiches gilt auch für eine qualitative Betrachtung der erzeugten Grafiken.
  MODE 2 hat etwas höhere Laufzeiten, was aufgrund der Suche nach sinnvollen Initialisierungslisten verständlich ist, da die erzeugten Grafiken allerdings insgesamt gerade für $N_t \geq 12$ am besten sind, ist der zeitliche Aufwand im Rahmen und absolut akzeptabel.

- Anders als bei PSO ist der MCS mit MODE 0, MODE 1, MODE 3 und MODE 4 für größere N_t ($\in [12, 20]$) nur etwas langsamer als der lokale Optimierer im Hinblick auf Realzeiten. Der zusätzliche Aufwand hält sich auch bei MODE 2 im Rahmen.

- Für $N_t = 8$ und $N_t = 10$ sind die Laufzeiten des MCS-Verfahrens abgesehen von dem Benchmark MODE 6 durchweg besser als der lokale Optimierer. Nicht nur die Reallaufzeit wird deutlich unterboten, selbst die CPU-Zeit wird geschlagen. Dies ist äußerst bemerkenswert, da die berechneten Mengen deutlich näher am Benchmark mit MODE 6 liegen, als die des lokalen Verfahrens.
  Dies ist auch deshalb erstaunlich, da die Erreichbarkeitsmengen mit dem MCS-Verfahren nicht nur sauberer berechnet werden, es werden auch Bereich gefunden, die vom lokalen Verfahren komplett unerschlossen bleiben. Es wird also ein größerer Bereich der Erreichbarkeitsmenge in kleinerer Zeit berechnet, was den Schluss zulässt, dass MCS für solch niedrigdimensionale Probleme, womit das lokale Verfahren Schwierigkeiten hat, ausgesprochen gut geeignet ist.

		Realzeit	CPU-Zeit
MCS	MODE 0	13m22.103s	13m21.794s
	MODE 1	18m11.249s	18m11.108s
	MODE 2	18m31.812s	15m59.816s
	MODE 3	14m24.265s	14m23.330s
	MODE 4	13m53.420s	13m53.284s
	MODE 6	64m20.446s	64m19.753s
Local		11m52.188s	2m38.834s

Tabelle 5.14.: Rayleigh, $N_t = 40$, $N_x = 5$, $R = 6$

Selbst bei höherdimensionalen Problemen schneidet das MCS-Verfahren nicht deutlich schlechter ab. Wenn man die Laufzeiten des Tests für $N_t = 40$ betrachtet, so sind diese zwar über alle Initialisierungen hinweg höher, dennoch ist die Geschwindigkeit für dieses System mit etwas gesteigerter Dimension überraschend hoch.

Für sehr komplexe Probleme mit weitaus höherer Dimension wird die Kluft aber mit Sicherheit deutlich größer, so dass ab einem Punkt der Griff zu lokalen Verfahren unvermeidlich wird.

5.7.2. Kenderov

Anders als das Rayleigh-Problem lassen sich Laufzeiten der unterschiedlichen Optimierer schlechter miteinander vergleichen, da wie man gesehen hat die qualitativen Unterschiede enorm sind. Für sich betrachtet mögen die im folgenden aufgelisteten Laufzeiten wenig aussagen, im Detail lassen sich aber besonders im Hinblick auf die berechneten Menge verschiedene Schlüsse ziehen.

R	MODE 0	MODE 1	MODE 2	MODE 3	MODE 4	MODE 6
3	0m2.343s	0m1.229s	0m4.487s	0m1.189s	0m1.776s	0m38.642s
4	0m6.599s	0m3.275s	0m12.120s	0m3.140s	0m5.072s	1m20.761s
5	0m20.516s	0m10.385s	0m36.393s	0m9.408s	0m14.806s	2m58.350s
6	1m12.509s	0m37.050s	2m0.145s	0m32.548s	0m54.325s	6m39.858s

(a) MCS, Realzeit

R	MODE 0	MODE 1	MODE 2
3	1m18.875s	1m17.173s	0m15.643s
4	3m10.919s	2m58.623s	0m37.409s
5	7m12.695s	8m32.556s	1m51.447s
6	23m44.536s	23m17.688s	6m51.917s

(b) PSO, Realzeit

	Schleife über u_0	
R	Real	CPU
3	2m20.008s	0m45.719s
4	6m35.359s	1m56.867s
5	24m36.736s	6m39.397s
6	103m57.421s	26m20.447s

(c) Locale Version

	$u_0 = -1.0$		$u_0 = 0.0$		$u_0 = 1.0$	
R	Real	CPU	Real	CPU	Real	CPU
3	0m1.445s	0m0.616s	1m11.914s	0m21.605s	0m30.164s	0m8.605s
4	0m2.829s	0m1.164s	5m28.512s	1m33.214s	0m30.702s	0m8.989s
5	0m22.489s	0m7.580s	15m6.371s	4m11.380s	1m11.222s	0m21.529s
6	0m31.254s	0m10.165s	64m51.683s	16m21.969s	4m39.153s	1m5.916s

(d) Locale Version

Tabelle 5.15.: Laufzeitvergleich, Kenderov, $N_t = 10$, $N_x = 5$

R	MODE 0	MODE 1	MODE 2	MODE 3	MODE 4	MODE 6
3	0m4.651s	0m3.478s	0m9.612s	0m4.188s	0m4.599s	1m35.087s
4	0m13.876s	0m9.888s	0m27.269s	0m11.654s	0m13.793s	3m28.551s
5	0m46.220s	0m32.013s	1m27.167s	0m37.609s	0m39.530s	7m57.010s
6	2m40.345s	1m50.682s	4m59.182s	2m9.258s	2m31.479s	19m12.156s

(a) MCS, Realzeit

R	MODE 0	MODE 1	MODE 2
3	5m28.146s	5m35.366s	1m41.028s
4	15m6.482s	16m3.722s	3m50.898s
5	33m34.044s	35m57.118s	9m45.518s
6	115m35.944s	97m14.730s	39m35.600s

(b) PSO, Realzeit

	Schleife über u_0	
R	Real	CPU
3	3m17.792s	0m48.871s
4	8m1.077s	2m13.316s
5	21m30.035s	5m41.569s
6	73m46.579s	16m36.070s

(c) Locale Version

	$u_0 = -1.0$		$u_0 = 0.0$		$u_0 = 1.0$	
R	Real	CPU	Real	CPU	Real	CPU
3	0m4.159s	0m1.400s	0m5.163s	0m1.692s	0m2.046s	0m0.748s
4	0m8.920s	0m3.268s	0m59.579s	0m19.105s	0m6.415s	0m2.308s
5	0m19.330s	0m7.204s	5m22.548s	1m31.450s	0m30.717s	0m10.905s
6	0m51.941s	0m18.193s	11m22.571s	3m25.693s	4m49.274s	1m26.561s

(d) Locale Version

Tabelle 5.16.: Laufzeitvergleich, Kenderov, $N_t = 16$, $N_x = 5$

R	MODE 0	MODE 1	MODE 2	MODE 3	MODE 4	MODE 6
3	0m5.437s	0m4.463s	0m9.779s	0m5.422s	0m5.124s	1m53.078s
4	0m13.183s	0m10.935s	0m25.136s	0m13.834s	0m11.981s	4m7.642s
5	0m37.881s	0m30.316s	1m13.101s	0m40.074s	0m37.417s	9m6.855s
6	2m2.340s	1m37.248s	3m51.940s	2m9.316s	1m57.810s	20m55.753s

(a) MCS, Realzeit

R	MODE 0	MODE 1	MODE 2
3	9m23.767s	7m59.395s	4m17.056s
4	21m7.144s	17m36.130s	7m26.234s
5	44m58.742s	44m34.747s	15m12.171s
6	127m34.330s	135m43.922s	50m31.817s

(b) PSO, Realzeit

	Schleife über u_0	
R	Real	CPU
3	1m54.352s	0m30.694s
4	4m58.692s	1m16.133s
5	12m14.839s	3m21.141s
6	34m8.489s	8m0.250s

(c) Locale Version

	$u_0 = -1.0$		$u_0 = 0.0$		$u_0 = 1.0$	
R	Real	CPU	Real	CPU	Real	CPU
3	0m3.228s	0m0.980s	0m4.484s	0m1.472s	0m1.814s	0m0.636s
4	0m8.122s	0m2.644s	0m11.934s	0m3.732s	0m3.826s	0m1.220s
5	0m16.756s	0m4.968s	0m31.552s	0m9.317s	0m7.035s	0m2.312s
6	0m37.584s	0m11.633s	2m15.868s	0m40.119s	0m13.081s	0m3.640s

(d) Locale Version

Tabelle 5.17.: Laufzeitvergleich, Kenderov, $N_t = 20$, $N_x = 5$

Zusammengefasst wird beim Blick auf die Tabellen folgendes klar:

- Vergleicht man die Laufzeiten von MCS mit MODE 0, MODE 1, MODE 3 und MODE 4 mit der lokalen Variante mit Startschätzung $u_i = 0.0$, so sind die realen Laufzeiten für $N_t = 20$ in etwas gleich auf. Je kleiner aber N_t wird, desto schneller wird der MCS-Algorithmus, wohingegen die lokale Version deutlich höhere Laufzeiten aufweist.
  Selbst die CPU-Zeiten werden von den Realzeiten des MCS-Algorithmus hier übertroffen. Wie bei Rayleigh ist diese Leistung für sich betrachtet schon erstaunlich, bedenkt man aber noch, dass die berechneten Mengen eine insgesamt bessere Qualität aufweisen, so ist die Geschwindigkeit sicher ein Qualitätsmerkmal des Algorithmus. Auch hier spielt er für Probleme kleinerer Dimension seine Stärken aus.

- MCS mit MODE 2, also der mit Line-Search generierten Initialisierungsliste hat zwar höhere Laufzeiten, allerdings werden die Erreichbarkeitsmengen abgesehen von dem mittleren Bereich oberhalb des Kreises vollständig berechnet.
  Diese Leistung ist angesichts der benötigten Laufzeit bemerkenswert und kommt ausgesprochen unerwartet.

- Auch die Laufzeiten von MCS mit MODE 4 sind bemerkenswert betrachtet man die erzielten Ergebnisse. Zwar sind die Grafiken schlechter als jene mit MODE 2, allerdings erlauben auch sie eine relativ gute Abschätzung darüber, wie die Erreichbarkeitsmenge aussieht, da größere Bereiche gefunden werden als beim lokalen Optimierer mit Startschätzung $u_i = 0.0$.

- MCS mit MODE 6, die Benchmarkvariante, arbeitet ebenfalls erstaunlich schnell und unterbietet die realen Laufzeiten des lokalen Optimierers mit ähnlicher Schleifenstrategie über u deutlich, man nähert sich sogar den CPU-Laufzeiten von diesem an, liefert aber deutlich bessere Ergebnisse. Hier war es nicht nötig über verschiedene Rescalierungen zu iterieren um den kritischen Bereich oberhalb des Zentrums zu schließen.

- Wie bei Rayleigh sind die Laufzeiten beim PSO-Verfahren sehr hoch. Bedauerlicherweise sind die Ergebnisse dabei für MODE 0 und MODE 1 auch noch wesentlich schlechter als beim bisherigen lokalen Verfahren.
  MODE 2 erzielt auch bei Kenderov die besten Laufzeiten und Ergebnisse, allerdings liegt dies wie bei der qualitativen Betrachtung der Resultate an dem verwendeten lokalen Optimierer.
  Die Laufzeiten mit MODE 2 sind angesichts der qualitativen Verbesserung der Grafiken gegenüber $u_i = 0.0$ beim lokalen Verfahren in einem akzeptablen Bereich.

6. Fazit und Ausblick

6.1. Fazit

Wie bei vielen Dingen im Leben ist auch bei globalen Optimierern nicht alles Gold was glänzt. Die qualitativen Unterschiede sind enorm und angesichts der in dieser Arbeit erzielten Resultate ist die Bezeichnung global in der Praxis selten angebracht.

Beim direkten Vergleich mit dem lokalen Verfahren treten hier häufig ähnliche Probleme auf. Teile der Erreichbarkeitsmenge werden nicht gefunden, es werden nur lokale Minima erreicht oder die Ergebnisse entsprechen so garnicht den Erwartungen wenn man die Resultate des Partikelschwarmoptimierers betrachtet.

Bei der Betrachtung der Optimierer für Probleme dieser Art geht das Multi-Level Coordinate Search-Verfahren als klarer Sieger hervor. Dieser Optimierer liefert mit relativ kleinem Aufwand bei der Implementierung erstaunliche Ergebnisse sowohl bei qualitativer Betrachtung als auch hinsichtlich der benötigen Rechenzeit.

Die erzielten Ergebnisse mit diesem Verfahren sind durchweg besser als beim lokalen Verfahren, die Laufzeit ähnlich und für kleine N_t teils deutlich besser. Mit dem Ändern der Initialisierungsart durch das setzen eines einzelnen Parameters beim Aufruf der Routine erzielt man grandiose Ergebnisse ohne viel Rechenzeit aufwenden zu müssen.

Dennoch bleibt er teilweise hinter den Erwartungen zurück. Es lassen sich durch eine Schleife über verschieden Startschätzungen zwar sehr gute Ergebnisse erzielen, dies sollte aber bei einem globalen Optimierer nicht nötig sein. Auch im Hinblick auf die garantierte Konvergenz ist das Ergebnis ernüchternd, da selbst bei großer Steigerung der maximal erlaubten Splittiefe $s_{\max}$ keine Verbesserung zu erkennen ist. Stattdessen wird wieder auf schlecht skalierte Zielfunktionen verwiesen und der Optimierer oft vorzeitig abgebrochen.

Insgesamt sprechen die erzielten Ergebnisse aber für den Optimierer. Besonders hinsichtlich der Laufzeit dürfte das Verfahren somit auch in anderen Einsatzgebieten Anwendung finden, sofern die Dimension des Problems nicht zu hoch ist. Bedauerlicherweise funktioniert dieser Algorithmus lediglich mit Boxed Constraints, was den Einsatz beispielsweise in einem nichtlinearen modellprädiktiven Regler unter Umständen verhindert. Für eine passende Problemklasse dürfte der Optimierer aber eine hervorragende Wahl sein.

Der Partikelschwarmoptimierer macht im direkten Vergleich dazu eine schlechte Figur, zwar erzielt man im Allgemeinen bessere Ergebnisse als bei der lokalen Variante, allerdings ist die Rechenzeit enorm hoch. Besonders der in MODE 2 verwendete lokale Optimierer verbessert das Ergebnis hinsichtlich Qualität und Laufzeit deutlich, was aber kein Gütesiegel des PSO-Verfahrens darstellt sondern vielmehr eines des in diesem Modus verwendeten lokalen Verfahrens ist.

Nun mag das Konzept der Optimierung mithilfe von Partikelschwärmen zwar häufig Anwendung finden und oft gute Resultate liefern, allerdings ist es für die in dieser Arbeit betrachteten Klasse von Problemen ungeeignet. Die Laufzeit ist zu hoch und die Ergebnisse für problematische Kontrollsystem wie Kenderov zu schlecht. Auch bei einfacheren System wie Rayleigh sind die Ergebnisse kaum besser als die bisheriger lokaler Verfahren, für steigende Dimension sogar schlechter.

Selbst eine Variation der für den Optimierer wichtigsten Variablen C_s und C_g ändert

an diesem Ergebnis nichts.

Da sich das Partikelschwarmverfahren relativ leicht parallelisieren lässt, ist eine Verbesserung der Laufzeit natürlich möglich, NAG bietet auch eine parallelisierte Variante an, an den erzielten Ergebnissen dürfte dies aber qualitativ wenig ändern.

Das Einsatzgebiet dieser Art von Optimierern ist also wohl eher nicht dort, wo viele Probleme möglichst schnell mit relativ guten Ergebnissen, sondern eher bei sehr großen Problemen bei denen es schwer ist den kompletten Suchraum zu untersuchen. Denkbar ist auch das man für ein hochdimensionales Problem ein bekanntest lokales Minimum unabhängig vom Aufwand weiter verbessern will.

Darüber hinaus ist es vorteilhaft, dass es für das PSO-Verfahen eine Variante gibt, welche auch mit nichtlinearen Grenzen des Suchraums klar kommt. Damit lasst sich eine weitaus größerer Klasse von Problemen untersuchen. Beispielweise lässt sich so ein System betrachten, bei dem die Kontollbereiche voneinander abhängen, also etwa bei einem Wagen, welcher für kleine Lenkbewegungen große Geschwindigkeit zulässt, für große Lenkbewegungen die Geschwindigkeit aber einschränkt.

6.2. Ausblick

In dem Zeitraum, in dem diese Arbeit angefertigt wurde, stand lediglich MARK 23 der NAG Fortran Library zur Verfügung. Die inzwischen erschienene Version MARK 24 bietet neben MCS und PSO auch noch einen Multi-Start-Optimierer (NAG-Bezeichnung E05UCF), welcher ein nichtlineares Optimierungsproblem mit nichtlinearen Grenzen löst, indem verschiedene Startwerte gewählt werden und das Problem dann für diese Startwerte mittels eines SQP-Verfahrens, also einem lokalen Optimierer, gelöst werden. Es wäre also auch interessant, wie sich die in dieser Arbeit betrachteten Probleme mit diesem Optimierer lösen lassen, da dieser Ansatz in etwa der Herangehensweise mit der Schleife über verschiedene Startwerte u_i entspricht um sowohl die Ergebnisse des lokalen Verfahrens als auch die des MCS-Algorithmus zu verbessern.

Dieser Multistart-Algorithmus erlaubt es entweder zufällige Startwerte aus dem Suchraum zu verwenden oder lässt dem Nutzer die Wahl diese innerhalb einer Subroutine zu setzen.

Die Ergebnisse dürften allerdings entweder stark vom Zufall abhängen, oder davon wie gut der Nutzer die Startwerte wählt. Um gute Startwerte angeben zu können ist aber eine genaue Kenntnis über das Verhalten des Kontrollsystems und der erreichbaren Menge notwendig. Da das Setzen einer Startschätzung abhängig vom Gitterpunkt bereits bei der lokalen Version mit Kenderov getestet wurde und ebenfalls einige fruchtlose Versuche mit MCS unternommen wurden, dürfte sich auch für diesen Multi-Start-Optimierer eine problemspezifische Wahl der Startwerte erübrigen.

Darüber hinaus wären Versuche mit den Optimierern auch in anderen Anwendungsgebieten sinnvoll, da häufig aus Zeitgründen lokale Optimierer eingesetzt werden, allerdings meist globale Optima gesucht werden.

A. Anhang

A.1. Inhalt auf CD

Die dieser Arbeit beigelegte CD enthält neben der lokalen Version des Algorithmus auch die globale Version, die vollständigen numerischen Resultate und diese Arbeit als PDF und reine Latex-Fassung.

Zusätzlich dazu befindet sich auf dieser auch die verwendete Literatur abgesehen vom Matlabsource-Code des MCS-Verfahrens als PDF.

Die Ordnerstruktur ist dabei die folgende:

```
CD/
  local_version/              (Basisverzeichniss lokale Version)
    sqpfiltertoolbox/
    OCPIDDAE1-V1.3/
      EXAMPLES/               (Sourcecode locale Version)
      plot_reachable_set.py
      (...)
  global_version/             (Sourcecode globale Version)
  nag_library/                (verwendete Mark 23 NAG-Library)
  literatur/                  (verwendete Literatur)
  results/                    (numerische Resultate
    global_version/            (der globalen Version)
    local_version/             (der lokalen Version)
    time_messurement/          (Zeitmessungen)
  tex/                        (Latex-version dieser Arbeit)
  diplom.pdf                  (PDF-Version dieser Arbeit)
```

Bei den numerischen Resultaten zum lokalen Optimierer ist die Verzeichnissstruktur

```
CD/
  results/
    local_version/
      kenderov_u0_-1.0/     (Kenderov, Startschätzung u_i = -1.0)
      kenderov_u0_u0_0.0/   (Kenderov, Startschätzung u_i = 0.0)
      kenderov_u0_1.0/      (Kenderov, Startschätzung u_i = 1.0)
      kenderov_u0_loop/     (Kenderov, Schleife über u_i)
      rayleigh/             (Rayleigh)
```

wohingegen bei dem globalen Verfahren

```
CD/
  results/
    global_version/
      rescale_1/              (Ergebnisse für Option RESCALE 1
        mcs/                  (mit MCS-Verfahren)
          kenderov/           (beim Kenderov-Problem)
            modeX/            (und MODE X, X = 0, 1, 2, 3, 4, 6)
              NT_NX_R.png     (Grafik mit Gitterpunkten)
              NT_NX_R_ng.png  (Grafik ohne gitterpunkte)
          rayleigh/(...)      (beim Rayleigh-Problem)
          kenderov_splittest  (Splittest Kenderov MCS)
        pso/                  (mit PSO-Verfahren
          kenderov/(...)      (Ergebnisse kenderov mit C_s und C_g Tests)
          rayleigh/(...)      (Ergebnisse kenderov mit C_s und C_g Tests)
        (...)                 (Gesonderte Ordner für Illustrationen)
      rescale_5               (Testergebnisse für RESCALE 5)
        kenderov_mcs/(...)
        rayleigh_mcs/(...)
      other_tests/(...)       Sonstige Tests.
```

verwendet wird.

Für die Bilddateien wird hier immer die Syntax `NT_NX_R.png` und `NT_NX_R_ng.png` als Dateinamen verwendet. Eine Datei mit der Bezeichnung `16_5_6.png` gehört also zu den Parametern $N_t = 16$, $N_x = 5$ und $R = 6$. Bei gesonderten Tests wurden weitere Parameter gemäß diesem Schema hinzugefügt.

A.2. Kompilieren lokale Variante

Zum Kompilieren der lokalen Variante sind zunächst folgende Schritte notwendig, um die für den Algorithmus nötigen Librarys zu erzeugen.

```
cd local_version/
cd sqpfiltertoolbox/
make libfgs
cd ..

cd OCPIDDAE1-V1.3/
make libocpiddae1
```

Um die einzelnen Beispiele, also Rayleigh oder Kenderov zu übersetzen muss man ausgehend vom Basisverzeichnis der lokalen Version

```
cd locale_version/OCPIDDAE1-V1.3/
make rayleigh (oder) make kenderov
```

verwenden. Zum ausführen ist

```
cd local_version/OCPIDDAE1-V1.3/
./rayleigh (oder) ./kenderov
./plot_reachable_set.py
```

zu verwenden um sowohl das Programm zu starten als auch anschließend die Grafiken zu erzeugen.

Der Programmcode für diese Variante ist unter

```
/CD/local_version/OCPIDDAE1-V1.3/EXAMPLES/
```

zu finden.

A.3. Kompilieren globale Version

Die globale Version setzt eine auf dem System installierte und lauffähige Version der NAG-Library MARK 23 voraus.

Dazu muss neben dem eigentlichen Softwarepaket auch die NAG-Lizenz mit passender Umgebungsvariable `$NAG_KUSARI_FILE` installiert sein. Verwendet wurde im Laufe dieser Arbeit die Version:

NAG Fortran Library, Mark 23
FLL6A23DFL - License Managed
Linux 64 (Intel&reg 64 / AMD64), GNU gfortran, Double Precision

Diese 64-bit Version setzt Version 4.6 des GNU-Fortran-Compilers voraus. Es wird ferner verlangt, dass die Umgebungsvariable `$NAG_LIBRARY_PATH` gesetzt wurde, welche auf das Verzeichnis des installierten NAG-Library zeigt. Falls dies nicht der Fall ist, so ist es nötig die `Makefile` entsprechend anzupassen.

Unter der Voraussetzung, dass eine diesen Anforderungen entsprechende Installation der NAG-Library und des Kompilers vorliegt, so lässt sich die globale Version übersetzten indem man folgende Schritte ausführt:

```
cd global_version/
make rayleigh_mcs      (für Rayleigh mit MCS)
make rayleigh_pso      (für Rayleigh mit PSO)
make kenderov_mcs      (für Kenderov mit MCS)
make kenderov_pso      (für Kenderov mit PSO)
```

Zum Aufruf und Erzeugen der Grafiken sind anschließend die Befehle

`./rayleigh_mcs` (oder) `./rayleigh_pso` (oder)
`./kenderov_mcs` (oder) `./kenderov_pso`
`./plot_reachable_set`

zu verwenden.

Aufgrund des modularen Ansatzens ist es unter Umständen nötig, vor dem kompilieren die alten Objekt- und Module-Dateien zu entfernen, was mit `make cleanup` realisiert wird.

Zur Zeitmessung wurde ferner ein Bash-Skript erstellt, welches die Routinen für verschiedene Werte aufruft, die Zeiten misst und diese anschließend in einer Datei `time_messure.txt` speichert. Falls diese Variante genutzt wird, so muss das die Hauptroutine in `adaptiv_main.f90` leicht abzuändern.

Je nachdem über welche Werte man iteriert ist es nötig die entsprechenden Variablen bei der Moduleinbindung `USE <MODULE>, ONLY: <VARIABLE>` anzuhängen und weiter unten per `READ(*,*) <VARIABLE>` einzulesen.

Falls dies nicht gewünscht wird, werden die im Sourcecode gesetzten Variablen aus den einzelnen Modulen verwendet.

Die globale Variante umfasst die hier nachfolgend aufgelisteten Dateien.

```
CD/
  global_version/
    Makefile                    (Makefile zum Kompilieren)
    adaptive_main.f90           (Mainfile mit adaptivem Algorithmus)
    tree.f90                    (Datenstrukturen für adaptive Version)
    model_kenderov.f90          (Kenderov Kontrollsystem)
    model_rayleigh.f90          (Rayleigh Kontrollsystem)
    nag_mcs_optimization.f90    (MCS-Modul)
    nag_pso_optimization.f90    (PSO-Modul)
    plot_reachable_set.py       (Skript zum erzeugen der Grafiken)
    test.sh                     (exemplarisches Bashskript zur)
                                (Iteration verschiedener Werte)
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Wolfgang Riedl. Adaptive algorithm to calculate reachable sets. *Preprint*, 2013.

[2] Robert Baier and Matthias Gerdts. Set-valued numerical analysis and optimal control. 2005.

[3] Walter Alt, Robert Baier, Matthias Gerdts, and Frank Lempio. Error bounds for euler approximation of linear-quadratic control problems with bang-bang solutions. *Preprint*, 2(3):547—570, 2013.

[4] Robert Baier, Ilyes Aïssa Chahma, and Frank Lempio. Stability and convergence of euler's method for state-constrained differential inclusions. *SIAM Journal on Optimization*, 18(3):1004–1026, 2007.

[5] Robert Baier and Matthias Gerdts. A computational method for non-convex reachable sets using optimal control. In *Proceedings of the European Control Conference (ECC)*, pages 23–26, 2009.

[6] Robert Baier, Matthias Gerdts, Ilaria Xausa, et al. Approximation of reachable sets using optimal control algorithms. *Numerical Algebra, Control and Optimization*, 3(3):519–548, 2013.

[7] Ilyses A. Chahma. Set-valued discrete approximation of state-constrained differential inclusions. *Bayreuther Mathematische Schriften*, 67:3–162, 2003.

[8] Hélène Frankowska and Franco Rampazzo. Filippov's and filippov–ważewski's theorems on closed domains. *Journal of Differential Equations*, 161(2):449–478, 2000.

[9] Asen L. Dontchev and EM. Farkhi. Error estimates for discretized differential inclusions. *Computing*, 41(4):349–358, 1989.

[10] Asen L. Dontchev and William W. Hager. Euler appoximation of the feasible set. *Numerical Functional Analysis and Optimization*, 15(3-4):245–261, 1994.

[11] Asen L. Dontchev, William W. Hager, and Vladimir M. Veliov. Second-order runge–kutta approximations in control constrained optimal control. *SIAM Journal on Numerical Analysis*, 38(1):202–226, 2000.

[12] Asen Dontchev and Frank Lempio. Difference methods for differential inclusions: a survey. *SIAM review*, 34(2):263–294, 1992.

[13] Ilaria Xausa, Robert Baier, Matthias Gerdts, Mark Gonter, Christian Wegwerth, et al. Avoidance trajectories for driver assistance systems via solvers for optimal control problems. In *20th International Symposium on Mathematical Theory of Networks and Systems*, 2012.

[14] Mattias Sandberg. Convergence of the forward euler method for nonconvex differential inclusions. *SIAM Journal on Numerical Analysis*, 47(1):308–320, 2008.

[15] Waltraud Huyer and Arnold Neumaier. Global optimization by multilevel coordinate search. *Journal of Global Optimization*, 14(4):331–355, 1999.

[16] NAG. Nag library routine document - e05jbf. Technical report. NAG Fortran Librarydocumentation, Mark 23.

[17] Global optimization by multilevel coordinate search. Rastrigrin Example with Mark 22 of the NAG Fortran Library.

[18] Arnold Neumaier. Matlabsourcecode, multilevel coordinate search. http://www.mat.univie.ac.at/∼neum/software/mcs/.

[19] NAG. Nag library routine document - e05saf. Technical report. NAG Fortran Librarydocumentation, Mark 23.

[20] Ioan Cristian Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters*, 85:317–325, 2003.

[21] Lars Grüne. Numerische dynamik von kontrollsystemen. *Vorlesungsskript Sommersemester*, 2004.

[22] Roberto L Gonzalez and Mabel M Tidball. On a discrete time approximation of the hamilton-jacobi equation of dynamic programming. *INRIA Rapports de Recherche*, 1375, 1991.

[23] Marcin Molga and Czesław Smutnicki. Test functions for optimization needs. 2005.

Erklärung

Hiermit erkläre ich, dass ich diese Diplomarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel erstellt habe.

Diese Arbeit lag in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vor.

Bayreuth, den 28. März 2014

Michael Bodenschatz