

PROFESSUR FÜR  
WIRTSCHAFTSINFORMATIK  
DER FREIEN UNIVERSITÄT BERLIN



Abschlussbericht

Metaheuristiken  
Simulated Annealing  
Sommersemester 2017

Dennis Duysak, Luisa Karzel, Daniela Schmitt

## **Abstract**

Im Rahmen des Kurses Metaheuristiken im Sommersemester 2017 wurde ein Projekt in die Wege geleitet, um ein Nurse Rostering Problem mithilfe einer Heuristik initial zu lösen und zu optimieren. Das Projektteam, bestehend aus den Mitgliedern Dennis Duysak, Luisa Karzel und Daniela Schmitt, hat sich entschieden, dieses Optimierungsproblem mit der Simulated Annealing Heuristik zu lösen. Dabei sollten sowohl die Restriktionen des Nurse Rostering Problems und die Generierung der Initiallösung, als auch die Heuristik selbst in Java implementiert werden. Nach der Implementierung folgte ein Parameter-Testing der veränderbaren Parameter der Heuristik, um den Einfluss dieser auf die erzeugte finale Lösung zu analysieren. Durch diese Analyse soll es möglich sein, die optimalen Parameterwerte für das spezifische Nurse Rostering Problem zu finden. Dieser Report gibt eine Zusammenfassung über den Verlauf des Projektes und präsentiert die Ergebnisse der Parameter-Testing Analyse und ihre Implikationen. Die für das Projekt genutzten Nurse Rostering Daten wurden von der First International Rostering Competition bereitgestellt, welche 2010 gehalten wurde.

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	v
<b>Tabellenverzeichnis</b>	vii
<b>1 Einleitung</b>	1
<b>2 Gruppenmanagement</b>	3
2.1 Aufgabenaufteilung . . . . .	3
2.2 Zeitplanung . . . . .	3
2.3 Handhabung von Schwierigkeiten . . . . .	4
<b>3 Implementierung</b>	5
3.1 Einlesen . . . . .	5
3.2 Harte Restriktionen . . . . .	5
3.3 Initiallösung . . . . .	5
3.4 Metaheuristik . . . . .	7
3.5 Weiche Restriktionen . . . . .	8
3.6 Schrittweise Vorgehensweise . . . . .	8
3.7 Laufzeit . . . . .	9
<b>4 Simulated Annealing Parameter-Testing</b>	11
4.1 Erster Parameter Test . . . . .	11
4.2 Zweiter Parameter Test . . . . .	13
4.3 Diskussion der Ergebnisse . . . . .	16
<b>5 Ausblick</b>	19
<b>6 Anhang</b>	21
6.1 Getroffene Annahmen . . . . .	21
6.2 Erster Parameter Test . . . . .	22
6.3 Zweiter Parameter Test . . . . .	31

*Inhaltsverzeichnis*

# Abbildungsverzeichnis

2.1	Übersicht über Github Commits . . . . .	4
3.1	Schrittweise Vorgehensweise zur Implementierung . . . . .	8
4.1	Long01 mit fester Cooling Rate . . . . .	12
4.2	Long01 mit Starttemperatur . . . . .	13
4.3	Long01 mit fester Starttemperatur von 1000 . . . . .	15
4.4	Long01 mit fester Starttemperatur von 500 . . . . .	15
4.5	Long01 mit fester Cooling Rate von 1.0 . . . . .	16
4.6	Long01 mit fester Cooling Rate von 0.5 . . . . .	16
4.7	Long01 mit 5000 und 50000 Iterationen . . . . .	17
6.1	Long Hidden01 . . . . .	22
6.2	Long Hidden01 . . . . .	22
6.3	Long Hint01 . . . . .	22
6.4	Long Hint01 . . . . .	23
6.5	Long Late01 . . . . .	23
6.6	Long Late01 . . . . .	23
6.7	Long 01 . . . . .	24
6.8	Long 01 . . . . .	24
6.9	Medium Hidden01 . . . . .	24
6.10	Medium Hidden01 . . . . .	25
6.11	Medium Hint01 . . . . .	25
6.12	Medium Hint01 . . . . .	25
6.13	Medium Late01 . . . . .	26
6.14	Medium Late01 . . . . .	26
6.15	Medium01 . . . . .	26
6.16	Medium01 . . . . .	27
6.17	Sprint Hidden01 . . . . .	27
6.18	Sprint Hidden01 . . . . .	27

## Abbildungsverzeichnis

6.19 Sprint Hint01 . . . . .	28
6.20 Sprint Hint01 . . . . .	28
6.21 Sprint Late01 . . . . .	28
6.22 Sprint Late01 . . . . .	29
6.23 Sprint01 . . . . .	29
6.24 Sprint01 . . . . .	29
6.25 Toy1 . . . . .	30
6.26 Toy1 . . . . .	30
6.27 Long01 mit fester Cooling Rate von 10 . . . . .	31
6.28 Long01 mit fester Cooling Rate von 5 . . . . .	31
6.29 Long01 mit fester Cooling Rate von 1.0 . . . . .	31
6.30 Long01 mit fester Cooling Rate von 0.5 . . . . .	32
6.31 Long01 mit fester Cooling Rate von 0.1 . . . . .	32
6.32 Long01 mit fester Starttemperatur von 1000 . . . . .	32
6.33 Long01 mit fester Starttemperatur von 500 . . . . .	33
6.34 Long01 mit fester Starttemperatur von 100 . . . . .	33
6.35 Long01 mit fester Starttemperatur von 10 . . . . .	33

## **Tabellenverzeichnis**

3.1	Auswahl möglicher Initiallösungen . . . . .	6
3.3	Laufzeit Simulated Annealing Implementierung . . . . .	9
4.1	Auswahl möglicher Initiallösungen . . . . .	12
4.3	Testmodus 2 bei fester Cooling Rate . . . . .	13
4.4	Testmodus 2 bei fester Starttemperatur . . . . .	14



# 1 Einleitung

Planungen in Krankenhäusern erfordern einen erheblichen Aufwand. Das Erstellen eines optimalen Dienstplans für Krankenschwestern ist eine Kernaufgabe davon. Diese Dienstpläne regeln den Ablauf der verschiedenen Stationen in Krankenhäusern. Durch die Vielzahl an Qualifikationen und Vertragsarten der Schwestern wird die Planung jedoch erschwert. Um dieses Problem anzugehen wurde die Nurse Rostering Competition im Jahr 2010 geschaffen. Die Aufgabe besteht darin, einen optimalen Dienstplan zu entwickeln. Da die Einteilung der Krankenschwestern in Schichten sehr komplex gestaltet, ist es in der Regel nicht möglich einen optimalen Dienstplan zu entwickeln. Aus diesem Grund kommen für die Lösung der Competition verschiedene Heuristiken zum Einsatz, welche versuchen, eine annähernd optimale Lösung zu erzielen. Ein Beispiel für eine solche Lösung mit Hilfe einer Heuristik ist der Simulated Annealing Algorithmus. Dieser versucht durch iteratives Vorgehen den Dienstplan schrittweise zu verbessern. Mit einer anfänglichen Starttemperatur kühlt sich der Algorithmus bildlich gesprochen nach jeder Iteration um eine gegebene Abkühlungsrate ab. Diese Ausarbeitung beschäftigt sich mit der Implementierung des Simulated Annealing Algorithmus am Beispiel des Nurse Rostering Problems. Dabei wird im nächsten Abschnitt zunächst auf das Gruppenmanagement eingegangen. Die Aufgabenverteilung, die Zeitplanung sowie die Handhabung bei Schwierigkeiten werden in diesem Punkt erläutert. Im Abschnitt Implementierung wird konkret auf die Umsetzung des Vorhabens eingegangen. Dazu zählt die Einleseroutine, die Initiallösung, die Simulated Annealing Heuristik und auch die Restriktionen, die beachtet werden müssen. Anschließend folgen Vorgehensweisen der zu testenden Metaheuristik mit Hilfe von Parametertests. Die Ergebnisse der Tests werden anschließend diskutiert. Im letzten Abschnitt werden als Ausblick alternative Vorgehensweisen sowohl bei der Initiallösung als auch des Simulated Annealing Algorithmus betrachtet.

## *1 Einleitung*

## **2 Gruppenmanagement**

Dieses Kapitel dient dazu, einen kurzen Überblick über die Themen der Aufgabenaufteilung, Zeitplanung und Handhabung von Schwierigkeiten im Rahmen des Projektes zu geben.

### **2.1 Aufgabenaufteilung**

Das Projektteam weist eine gute Fähigkeitenverteilung auf und kann so die in Abschnitt “Vorgehen bei jedem dieser Schritte” genannten Schrittfolgen gut aufteilen. Die Aufgabe des Programmieren fällt dabei D. Duysak zu, da er schon viel Erfahrung mit der Java Programmierung hat. L. Karzel und D. Schmitt kommt die inhaltliche Aufbereitung der Teilprojekte und die Erstellung des Pseudocodes zu. Außerdem nehmen beide die Betrachtung der Ausnahmefälle vor und testen den Code, eine Fehlersuche und eventuelle Rücksprache bei Problemen findet meist mit dem gesamten Team statt. Zusammenfassend kann gesagt werden, dass das Team eine sehr gute Aufgabenverteilung hat und alle Mitglieder des Teams gleichermaßen zum Erfolg des Projektes beitragen.

### **2.2 Zeitplanung**

Die Zeitplanung hat sich so gestaltet, dass durch wöchentliche Treffen eine konstante Leistung erreicht wird. Dabei wird zuerst das ganze Projekt in kleine Arbeitspakete bestehend aus Einlesen, Harte Restriktionen, Initiallösung, Metaheuristik, Weiche Restriktionen und Parameter-Testing zerlegt. Jede Woche werden neue kleine Ziele innerhalb eines solchen Arbeitspaketes gesetzt und verfolgt. Der Verlauf lässt sich auch in Grafik 2.1 zu den Githubs Commits des Projektes nachvollziehen.

## 2 Gruppenmanagement

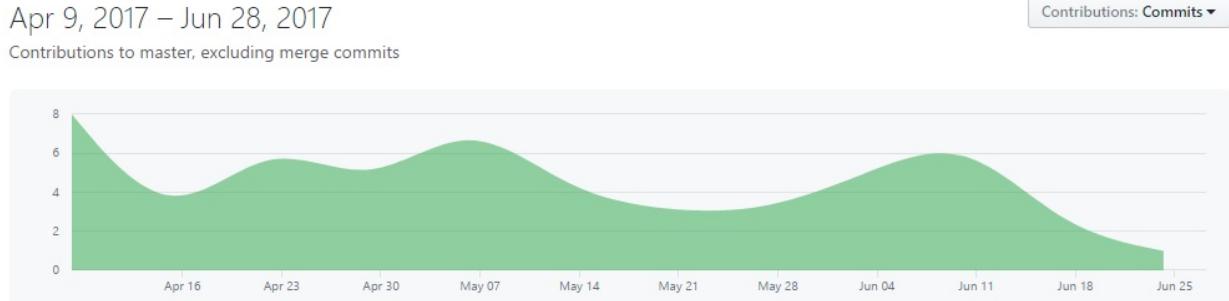


Abbildung 2.1: Übersicht über Github Commits

### 2.3 Handhabung von Schwierigkeiten

Insgesamt sind während des Projektes keine großen Schwierigkeiten aufgetreten. Zu aller erst stand die Vermutung im Raum, dass die Implementierung der Unwanted Patterns schwierig sein könnte, welche deshalb bis zum Schluss der Fertigstellung der harten Restriktionen aufgeschoben worden ist. Zum Schluss ist die Implementierung dieser dann allerdings reibungslos verlaufen und die erwarteten Probleme sind nicht zugetroffen. Das lässt sich darauf zurückführen, dass die Datenstruktur des Codes über die Zeit schon gut bekannt war und so die neue Restriktion leicht in das schon bestehende Gefüge eingebunden werden konnte. Im Nachhinein war es vielleicht sogar eine gute Idee, diese Restriktion solange aufzuschieben.

## **3 Implementierung**

Als Programmiersprache für die Implementierung wurde Java verwendet. Insgesamt gliedert sich die Vorgehensweise der Implementierung in folgende Schritte, welche sequentiell abgearbeitet werden: Einlesen, Harte Restriktionen, Initallösung, Metaheuristik und Weiche Restriktionen.

### **3.1 Einlesen**

Für das Auslesen der XML Datei wurde unter Java der DocumentBuilder verwendet. Dieser ermöglicht es einzelne Knoten aus der Datei zu lesen. Nach dem Definieren des Knoten können alle darunterliegenden Subknoten angesprochen werden. Mithilfe von lokalen Variablen wurde die Werte gespeichert und den Knotenklassen (z.B. Employee, Contract, etc.) zugewiesen. Anschließend wird ein Objekt der Klasse SchedulingPeriod erstellt, welches alle eingelesenen Werte enthält.

### **3.2 Harte Restriktionen**

Beim Nurse Rostering Problem lassen sich drei harte Restriktionen identifizieren. Eine Nurse darf nur eine Schicht an einem Tag erhalten, sie muss zusätzlich noch den passenden Skill für die Schicht aufweisen und der Bedarf an Schichten pro Tag muss genau gedeckt sein. Da ohne diese harten Restriktionen nicht überprüft werden kann, ob eine Lösung zulässig ist, ist den harten Restriktionen eine hohe Priorität zuzuordnen und sie werden als erstes implementiert.

### **3.3 Initallösung**

Zur Erzeugung einer Initallösung wurden unterschiedliche Herangehensweisen betrachtet, welche jeweils Lösungen mit unterschiedlicher Güte erzeugen. Insgesamt wurde dabei eine Auswahl von fünf Initialen Lösungen gesammelt, welche in nachfolgender Tabelle ausgewiesen sind.

### 3 Implementierung

Tabelle 3.1: Auswahl möglicher Initiallösungen

<b>Initiallösung</b>	<b>Beschreibung</b>	<b>Vorteil</b>	<b>Nachteil</b>
Abwandlung Nord-Süd-Regel	beginnend mit der knappsten Ressource (hier: Headnurses zuerst zuteilen), Aufteilung nach Tage	Programmieraufwand niedrig, Guter Verbesserungsbedarf	Relativ schlechte Initiallösung (Auswirkungen auf Laufzeit müssen erst durch Tests bestätigt werden)
Abwandlung Nord-Süd-Regel 2	beginnend mit der knappsten Ressource (hier: Headnurses zuerst zuteilen), Aufteilung nach Nurses	Programmieraufwand niedrig	
Abwandlung Nord-Süd-Regel 3	beginnend mit der knappsten Ressource (hier: Wochenenden, dann Headnurses), Aufteilung nach Nurses/ Tage/ Schichten	Initiallösung erzeugt tendenziell Lösungen mit höherer Güte	Programmieraufwand höher als Abwandlung Nord-Süd-Regel 1 + 2
Nurse Iteration	Über Nurses iterieren und jeweils die beste Dienstreihenfolge ermitteln	Bessere Initiallösung	Aufstellung weicher Restriktionen notwendig (präferierte Tage etc.), Programmieraufwand
Nurse Ziehen	X=0, Zuordnung losen von Nurses (Anzahl der Lose = max assignments)	Bessere Initiallösung/ Gleichverteilung der Arbeit Anzahl der Lose pro Nurse in Abhängigkeit zur Arbeitszeit gemäß Vertrag	Nach jeder Iteration muss nächstes Los bestimmt werden, Kann mit oder ohne zurücklegen implementiert werden, Programmieraufwand

Hierbei ist die Entscheidung der Umzusetzenden Initiallösung auf die Nord-Süd-Regel gefallen, welche für jeden Tag zuerst die Headnurses zuteilt und darauf folgend die normalen Schichten an die weiteren Nurses vergibt. Die Liste der Nurses wird dabei immer durchgegangen bis alle Schichten des Tages belegt sind. Für jeden neuen Tag setzt sie diese Liste fort. Dieses Verfahren wurde deshalb als Initiallösung ausgewählt, weil es einen guten Trade Off zwischen der Güte der Lösung und dem Programmieraufwand bietet. Die relativ schlechte Lösung bietet so eine gute Voraussetzung zu einer zukünftigen Verbesserung mit Hilfe der Heuristik. Während der Implementierung ist klar geworden, dass die Güte der Lösung nochmals schlechter ist, als anfangs angenommen. Der Grund dafür ist, dass in den zur Verfügung stehenden Dateien, die Headnurses jeweils am Anfang der Datei stehen. Da an jedem neuen Tag zuerst die Headnurse-Schichten zugeteilt werden, geht der Algorithmus alle restlichen normalen Nurses der Liste durch, bis wieder eine Headnurse gefunden wird, was jeweils erst wieder zu Beginn der Fall ist. So arbeiten die am Anfang der Liste stehenden Nurses jeden Tag und die am Ende der Liste stehenden nie, was eine sehr schlechte Initiallösung darstellt. Da eine schlechte Initiallösung für eine Heuristik jedoch von Vorteil sein kann, wurde dieses Verhalten des Algorithmus nicht geändert. Die zusätzlich vorgeschlagenen Initiallösungen stehen dabei für weitere zukünftige Varianten zur Verfügung, wurden so allerdings nicht mehr implementiert.

## 3.4 Metaheuristik

Die Implementierung der Initiallösung liefert die Basis für die Simulated Annealing Heuristik, welche als nächster Schritt implementiert wird. Da die Heuristik mit sogenannten Swaps arbeitet, ist entscheidend, wie ein solcher Swap definiert wird. Für dieses Projekt wurde ein Swap als Tausch eines ganzen Tages zwischen zwei Nurses angesehen. Es werden also zufällig ein Tag und danach zufällig zwei Nurses ausgewählt, welche dann ihre gesamten Schichten des Tages austauschen. Sollte die getauschte Lösung eine zulässige Lösung sein, kann mit dieser weitergearbeitet werden, sonst werden wieder zufällig neue Vertauschungen generiert. Alternativ können auch unterschiedliche Tage oder ganze Zeitperioden (mehrere Tage) zwischen zwei Nurses getauscht werden. Ein Ringtausch zwischen mehreren Nurses wäre eine weitere Variante. Zusätzlich wird beim Simulated Annealing die jemals beste erreichte Lösung gespeichert und am Ende ausgegeben, da die letzte Iteration der Heuristik nicht zwangsläufig die beste gefundene Lösung darstellt.

### 3.5 Weiche Restriktionen

Damit die Simulated Annealing Heuristik zwei unterschiedliche gültige Lösungen vergleichen kann, gilt es nun als letzten Schritt die weichen Restriktionen zu implementieren, um die Strafpunkte und somit die Güte der jeweiligen Lösung zu ermitteln. Dabei werden alle vorgegebenen weichen Restriktion der First International Nurse Rostering Competition eingebracht. Ausnahme hierzu bildet die “Alternative Skill Category”, da ihre Bedeutung unklar und sie in den Datensätzen immer auf null gesetzt ist. Die Strafpunkte jeder Restriktion werden dabei mit der Gewichtung in dem jeweiligen Vertrag multipliziert, um so die Wichtigkeit einzelner Restriktionen hervorzuheben. Unterschiede zu anderen Projektteams hinsichtlich ihrer Strafpunktverteilung können dadurch entstehen, dass bei manchen Restriktionen Interpretationsfreiheiten vorliegen, wie viele Strafpunkte ein Verstoß verursacht. So können beispielsweise bei der Restriktion “Complete Weekends” zwei Strafpunkte, wenn nur ein Tag des Wochenendes Fri/Sa/So gearbeitet wird, oder nur ein Strafpunkt vergeben werden, da die Restriktion insgesamt nicht erfüllt ist. Hier wird Variante eins gewählt, aber andere Projektteams können auch Variante zwei wählen, was am Ende zu einer unterschiedlichen Anzahl an Strafpunkten bei derselben Lösung führt. Die jeweiligen genau getroffenen Annahmen, welche im Verlauf des Projektes für jede Restriktion getroffen wurden, sind dabei im Anhang ersichtlich.

### 3.6 Schrittweise Vorgehensweise

Jedem dieser eben genannten Teilprojekte liegt dieselbe Vorgehensweise zugrunde, welche in Grafik 3.1 veranschaulicht wird.

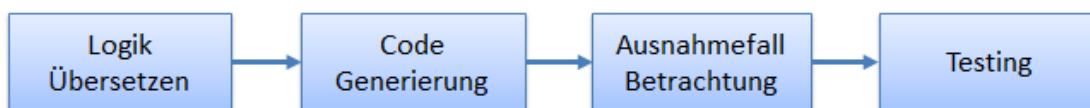


Abbildung 3.1: Schrittweise Vorgehensweise zur Implementierung

Als erstes wird die Logik und das Verständnis des Teilprojekts besprochen, außerdem werden die Annahmen festgelegt, dokumentiert und der Pseudocode erstellt. Anschließend findet die Implementierung in Java statt, wobei der Code zu diesem Zeitpunkt bereits ca. 90 Prozent der Fälle korrekt abdeckt. Die verbleibenden Fälle sind Ausnahmefälle und betreffen häufig die Handhabung des Codes am Anfang und Ende der Periode oder es gibt weitere Aspekte hinsichtlich der Head Nurse Schichten zu beachten. Der Code

wird an diese Ausnahmefälle angepasst und verbessert. Sollte aus logischer Sicht keine Einwände mehr vorliegen, wird der Code mit den vorhandenen Daten getestet, meist wird dabei die toy01 Datei und die sprint01 genutzt, zusätzlich kommt die long01 Datei zum Einsatz, wenn die Thematik der Head Nurses zu beachten ist. Wenn vorher nicht erkannte Probleme beim Testen auftreten, wird eine Fehlersuche gestartet, um die fehlerhaften Teile des Codes ausfindig zu machen und zu beheben. Dazu wurden beispielsweise bei den weichen Restriktionen die zu erwarteten Strafpunkte händisch ausgerechnet und mit tatsächlich vom Code generierten Strafpunkten verglichen. Zusätzlich werden auch die Ausnahmefälle betrachtet und überprüft. Erst wenn diese übereinstimmen und auch kleine Veränderungen in den Datensätzen zu keinen unterschiedlichen Ergebnissen führen, wird der Code als funktionierend klassifiziert. Die Schritte der Code Generierung und der Betrachtung von Ausnahmefällen sowie das Testing sind dabei jeweils von unterschiedlichen Personen durchgeführt worden. So wurde sichergestellt, dass Fehler eher gefunden wurden, da der Code jeweils von mehreren Projektmitgliedern analysiert wurde.

### 3.7 Laufzeit

Um exemplarisch einen Überblick über die Laufzeit des implementierten Simulated Annealing zu erhalten, wurde ein Testlauf mit der Long01 Datei durchgeführt. Die gemessenen Zeiten mit Parametereinstellungen von 1000 für die Starttemperatur und Cooling Rate von eins sind in untenstehender Tabelle ersichtlich.

Tabelle 3.3: Laufzeit Simulated Annealing Implementierung

	Laufzeit in ms
Einlesen	442
Initiallösung	344
Algorithmus	19555
Einmalige Strafpunktberechnung	13

Die Komplexität des Algorithmus hängt dabei maßgeblich von der gewählten Anzahl Iterationen ab. Diese ergibt sich aus einer Division von Starttemperatur durch Cooling Rate. Je mehr Iterationen durchgeführt werden, desto höher ist die resultierende Laufzeit und somit auch die Komplexität. Zusätzlich wird die Komplexität durch die Anzahl Nurses beeinflusst, wodurch mehr Rechenzeit in die sich wiederholende Strafpunktberechnung fließt. Durch eine kompaktere Aufteilung der Programmstruktur kann zusätzlich eine effizientere Laufzeit erreicht werden.

### *3 Implementierung*

## 4 Simulated Annealing Parameter-Testing

Die im weiteren Verlauf erläuterten Parameter Tests beschäftigen sich mit dem Einfluss unterschiedlicher Parametereinstellungen auf die Güte der von der Heuristik resultierenden Lösung. Hierbei stand die Frage im Raum, welche optimalen Einstellungen eine möglichst zeitsparende und gleichzeitig ausreichend hohe Lösungsgüte für das Nurse Rostering Problem liefert.

### 4.1 Erster Parameter Test

Insgesamt werden bei diesem ersten Parametertest eine Datei aus allen unterschiedlichen Dateitypen betrachtet. Dabei liegt die Vermutung zugrunde, dass unterschiedliche Dateitypen auch unterschiedliche optimale Parametereinstellungen besitzen.

Tabelle 4.1 gibt dabei Auskunft über die geplanten und durchgeführten Tests. Dabei wurden jeweils sechs Tests mit fester Cooling Rate und reduzierender Starttemperatur und vice versa durchgeführt. Für jeden Dateityp wurden somit zwölf Testdurchläufe absolviert. Zu jedem Testdurchlauf wurden dabei der Dateiname, die Cooling Rate, Starttemperatur, Iterationsanzahl (berechnet aus Starttemperatur geteilt durch Cooling Rate) und Score erfasst. Der Score wurde dabei als Mittel über fünf Durchläufe der Heuristik ermittelt.

Exemplarisch soll hier ein Testdurchlauf ab Beispiel des Testmodi 1 erläutert werden: Zuerst wird die Temperatur auf 10000 festgesetzt. Nun wird die Cooling Rate auf 10 gesetzt und pro Iteration um eins verringert. Nachdem die Cooling Rate nach zehn Iterationen null erreicht hat, wird die Temperatur um 1000 reduziert und die Cooling Rate auf zehn zurückgesetzt. Das Ganze wird wiederholt, bis die Temperatur 1000 erreicht.

Die Rechenzeit der Heuristik über alle Dateitypen und Testmodi belief sich dabei auf ca. 40 Stunden.

## 4 Simulated Annealing Parameter-Testing

Tabelle 4.1: Auswahl möglicher Initiallösungen

Testmodus	1	2	3	4	5	6
Oberes Ende für Cooling Rate	10	5	1	0.5	0.1	0.01
Veränderung der Cooling Rate	1	0.5	0.1	0.05	0.01	0.001
Oberes Ende für Temperatur	10000	5000	1000	500	100	10
Veränderung der Temperatur	1000	500	100	50	10	1

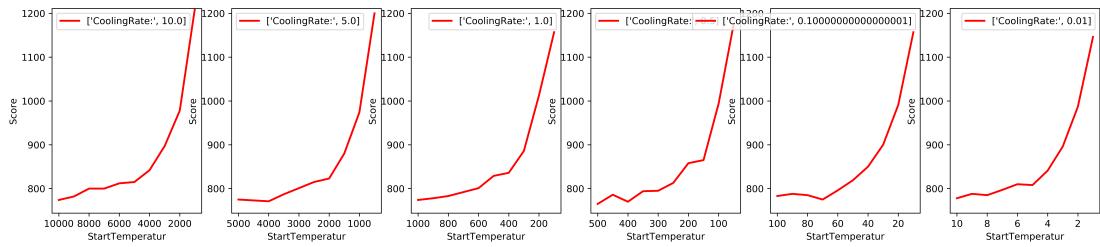


Abbildung 4.1: Long01 mit fester Cooling Rate

Nachfolgend haben sich dabei folgende Erkenntnisse herauskristallisiert: Innerhalb eines einzelnen Plots von Grafik 4.1 scheint eine sinkende Starttemperatur zu einem schlechteren Ergebnis zu führen bei gleichbleibender Cooling Rate. Bei Grafik 4.2 hingegen hat augenscheinlich eine veränderte Cooling Rate bei fester Starttemperatur innerhalb eines einzelnen Plots keine signifikanten Auswirkungen auf den Score. Dabei ist allerdings zu bedenken, dass auch die Temperatur für jeden Plot der sinkenden Cooling Rate sich verändert hat. Dasselbe gilt für eine gleichbleibende Cooling Rate und sinkender Temperatur. Zusätzlich konnte beim Vergleich mehrere Dateitypen jedoch nicht bestätigt werden, dass unterschiedliche Dateitypen unterschiedliche optimale Parametereinstellungen erfordern. So sind diese Ergebnisse nur bedingt brauchbar, um eine optimale Parametereinstellung zu bestimmen. Aus diesem Grund ist eine Wiederholung dieses Tests vorgesehen, bei welchem sich bei sinkender Temperatur die Cooling Rate bzw. umgekehrt bei sinkender Cooling Rate die Temperatur nicht ändert.

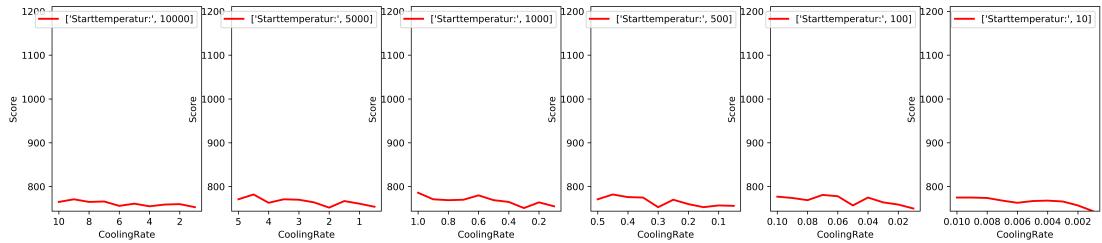


Abbildung 4.2: Long01 mit Starttemperatur

## 4.2 Zweiter Parameter Test

Im zweiten Parametertest soll nun beispielhaft für eine Datei die optimalen Parametereinstellungen ermittelt werden. Hierzu wurde die long01 Datei herangezogen, da diese auch Headnurses mit einbezieht und viele Restriktionen eingesetzt werden müssen. Die aus dem ersten Parametertest gesammelten Erkenntnisse führen dazu, dass in diesem zweiten Test eine sinkende Cooling Rate bei jeweils fester Temperatur sowie eine sinkende Temperatur bei fester Cooling Rate betrachtet wird. Tabelle 4.3 gibt Auskunft über die durchgeführten Tests bei fester Cooling Rate. Beispielsweise für eine Starttemperatur von 1000 und eine Cooling Rate von 10 werden nun 10 einzelne Durchläufe der Heuristik durchgeführt. Für jeden Durchlauf wird dabei der jeweilige Durchschnittsscore über 5 Iterationen mit diesen Parametern gespeichert. Nach jedem Durchlauf verringert sich die Starttemperatur bei gleichbleibender Cooling Rate um die inkrementelle Verringerung. Für den zweiten Durchlauf bedeutet das eine Starttemperatur von 900 und eine Cooling Rate von 10.

Tabelle 4.3: Testmodus 2 bei fester Cooling Rate

Start-temperatur	Inkrementelle Verringerung ST	Betrachtete Cooling Rates
1000	100	10, 5, 1, 0.5, 0.1
500	50	10, 5, 1, 0.5, 0.1
100	10	10, 5, 1, 0.5, 0.1
10	1	10, 5, 1, 0.5, 0.1

Dasselbe ergibt sich für eine feste Starttemperatur und variabler Cooling Rate.

#### 4 Simulated Annealing Parameter-Testing

Tabelle 4.4: Testmodus 2 bei fester Starttemperatur

Cooling Rate	Inkrementelle Verringerung CR	Betrachtete Starttemperaturen
10	1	1000, 500, 100, 10
5	0.5	1000, 500, 100, 10
1	0.1	1000, 500, 100, 10
0.5	0.05	1000, 500, 100, 10
0.1	0.01	1000, 500, 100, 10

Dabei erfolgt die Anpassung im Gegensatz zum ersten Parametertest nicht nach jedem Schritt. Der variable Parameter wird hierbei für jeden konstanten Parameter betrachtet. So soll es möglich sein, eine optimale Parametereinstellung zu ermitteln. Im weiteren wird eine Auswahl der Ergebnisse des Parametertests für jeweils eine konstante Cooling Rate und Starttemperatur gezeigt. Eine Visualisierung aller Test befindet sich im Anhang.

## 4.2 Zweiter Parameter Test

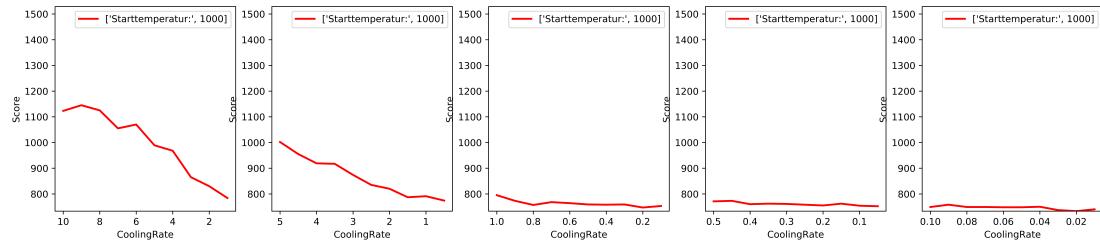


Abbildung 4.3: Long01 mit fester Starttemperatur von 1000

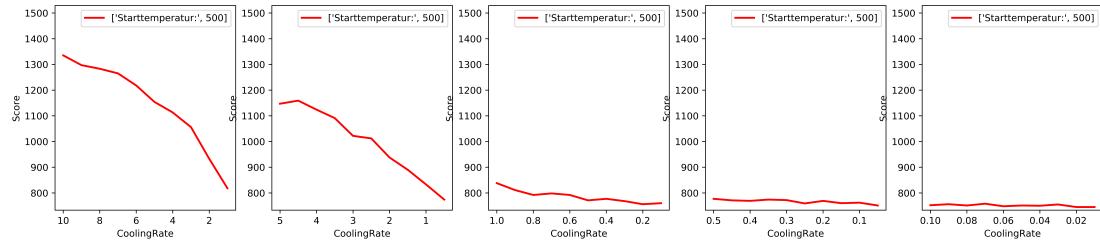


Abbildung 4.4: Long01 mit fester Starttemperatur von 500

Die Grafiken 4.3 und 4.4 zeigen nun ausschnittsweise die Ergebnisse der Tests für eine konstante Temperatur von 1000 bzw. 500 und variabler Cooling Rate zwischen 10 und 0.1. Hierbei ist gut zu erkennen, dass der erreichte Durchschnittsscore mit sinkender Cooling Rate bessere Werte annimmt. Für eine Starttemperatur von 500 werden dabei ab einer Cooling Rate von 0.5 nur noch marginale Verbesserungen erreicht.

In den Grafiken 6.32 und 6.30 sind jeweils die Starttemperaturen im Intervall von 1000 - 1 Einheiten bei konstanter Cooling Rate von 1.0 und 0.5 betrachtet worden. Hierbei zeigt sich, dass mit sinkender Temperatur bei gleichbleibender Cooling Rate eine schlechtere Lösung erzielt wird. Genauso ist auch zu erkennen, dass ab einer bestimmten hohen Temperatur keine nennenswerte Verbesserung der Lösung mehr erreicht wird. Das ist in 6.30 ab einer Temperatur von 600 Einheiten ersichtlich.

## 4 Simulated Annealing Parameter-Testing

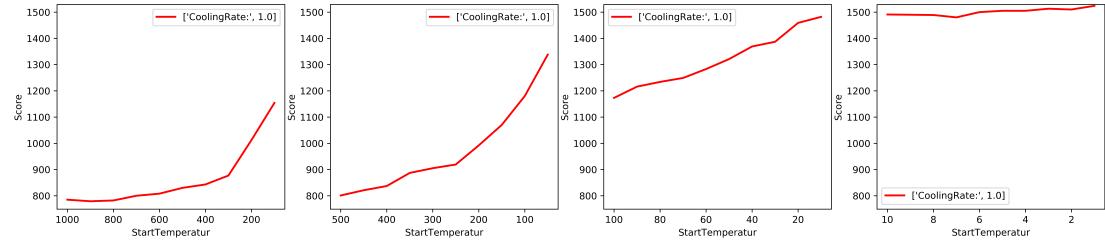


Abbildung 4.5: Long01 mit fester Cooling Rate von 1.0

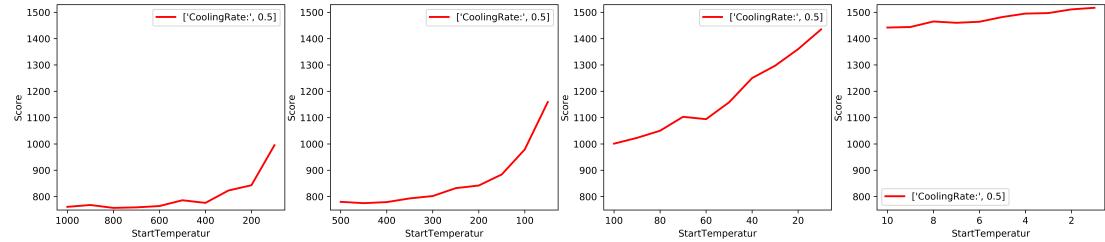


Abbildung 4.6: Long01 mit fester Cooling Rate von 0.5

Bei dem Vergleich der Grafiken mit fester Starttemperatur und Cooling Rate ist auch zu erkennen, dass insbesondere die Anzahl an Iterationen, welche sich aus Starttemperatur geteilt durch Cooling Rate ergibt, das ist, was zu einem guten Score führt. Die genaue Größe der Starttemperatur und der Cooling Rate scheint dabei nicht wichtig zu sein, solange die sich daraus ergebende Anzahl Iterationen hoch genug ist. Ab einem bestimmten Punkt wird allerdings auch durch eine höhere Iterationsanzahl nur noch eine geringe Verbesserung des Scores erreicht.

### 4.3 Diskussion der Ergebnisse

Zusammenfassend hat sich herausgestellt, dass das Verhältnis zwischen Starttemperatur und Cooling Rate entscheidend für die Güte der Lösung ist. Demnach sind die Fragestellungen von Exploration, entsprechend einer hohen Starttemperatur oder einer kleinen Cooling Rate, und Exploitation, entsprechend einer niedrigen Starttemperatur oder hoher Cooling Rate, aufgrund der Testergebnisse für die entwickelte Heuristik zur Lösung dieses Nurse Rostering Problem nicht signifikant relevant. Die Fragestellung der optimalen Parametereinstellung

lässt sich somit nicht eindeutig bestimmen. Allerdings kann gesagt werden, dass ab einer genügend hohen Iterationsanzahl keine starken Veränderungen in der Güte der Lösung erreicht werden.

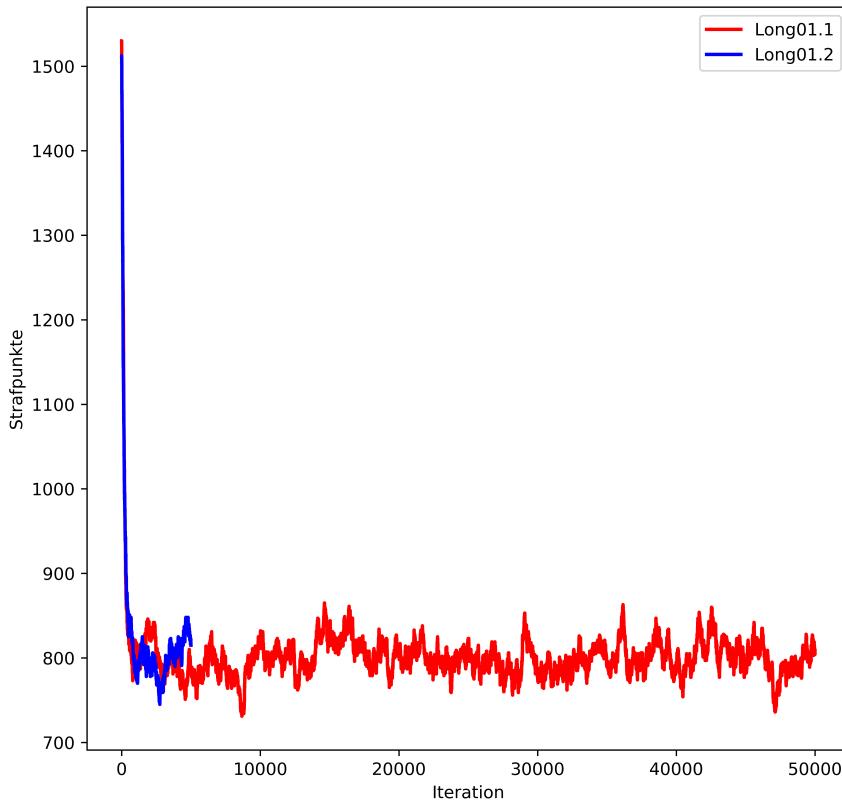


Abbildung 4.7: Long01 mit 5000 und 50000 Iterationen

Grafik 4.7 zeigt die schrittweise Verbesserung der Heuristik mit 5000 und 50000 Iterationen. Hierbei ist ersichtlich, dass die gefundenen Lösung bei kürzerer Iterationsanzahl nur geringfügig schlechter ausfällt. Zudem werden mit steigender Iterationsanzahl nun noch schwierere Lösungen erreicht. An dieser Stelle ist zu überlegen, ob diese Mehranzahl an Iterationen für nur eine geringfügige mögliche Verbesserung ökonomisch sinnvoll erscheint. Der Trade Off zwischen Rechenzeit und Lösungsgüte muss so individuell auf das Problem abgestimmt werden. Jedoch ist dabei immer zu bedenken, dass nicht festzustellen ist, ob es sich bei der gefundenen besten Lösung um ein globales oder lokales Optimum handelt.

#### *4 Simulated Annealing Parameter-Testing*

Zuletzt hat sich herausgestellt, dass die Wahl der eher schlechten Initiallösung einen positiven Einfluss auf die Ergebnisse der Heuristik hat. Dadurch war eine Verbesserung der Heuristik besser ersichtlich.

## 5 Ausblick

Während der Implementierung der Heuristik sind weitere Ideen zur Verbesserung des Simulated Annealing aufgekommen. Dabei könnte beispielsweise ein Reset nach einer vorher festgelegten Anzahl Iterationen stattfinden und die Heuristik von neu starten. Somit soll verhindert werden, dass die Heuristik in einem lokalen Optima feststecken bleibt. Alternativ könnte nach dieser Anzahl Iterationen ein Reset inklusive Rückkehr zu der bis dato best gefundenen Lösung stattfinden. Das ermöglicht eine zu Anfang große Exploration und nach dem Reset eine hohe Exploitation. Die bereits in der Implementierung vorgestellten Initiallösungen dienen außerdem als Grundlage für zusätzliche Tests. Weiterhin hat die Definition des Swaps große Auswirkungen auf das Verhalten des Simulated Annealing. Zukünftige Projekte oder Forschungsarbeiten könnten die genauen Einflüsse unterschiedlicher Swaps thematisieren um genauere Erkenntnisse über den Einfluss auf Exploration und Exploitation zu erlangen.

## *5 Ausblick*

## 6 Anhang

### 6.1 Getroffene Annahmen

MaxConsecutiveWorkingDays: für jeden einzelnen tag der über MaxConsecutiveWorkingDays liegt, wird ein Strafpunkt vergeben.

MinConsecutiveWorkingDays: für jeden einzelnen Tag der unter der MinConsecutiveWorkingDays liegt, wird ein Strafpunkt vergeben. Am Ende der Periode werden auch Strafpunkte verteilt, selbst wenn die nächste Periode noch unbekannt ist.

MaxConsecutiveFreeDays: für jeden einzelnen tag der über MaxConsecutiveFreeDays liegt, wird ein Strafpunkt vergeben.

MinConsecutiveFreeDays: für jeden einzelnen Tag der unter der MinConsecutiveFreeDays liegt, wird ein Strafpunkt vergeben. Am Ende der Periode werden auch Strafpunkte verteilt, selbst wenn die nächste Periode noch unbekannt ist.

MaxConsecutiveWorkingWeekends: (ab einem Tag Arbeit am Wochenende, wird dieses als gearbeitetes Wochenende definiert) Für jedes Wochenende über der Maximalanzahl wird jeweils ein Strafpunkt vergeben.

MinConsecutiveWorkingWeekends: (ab einem Tag Arbeit am Wochenende, wird dieses als gearbeitetes Wochenende definiert) Für jedes Wochenende unter der Minimalanzahl wird jeweils ein Strafpunkt vergeben.

MaxWorkingWeekends in FourWeeks: Für jedes Wochenende (ab einem Tag Arbeit am Wochenende, wird dieses als gearbeitetes Wochenende definiert) über der Maximalanzahl wird jeweils ein Strafpunkt vergeben.

CompleteWeekends: angebrochene Wochenenden am Anfang und Ende der Periode werden nicht mitgezählt, ansonsten gibt es jeweils einen Strafpunkt pro Tag des Wochenendes der nicht gearbeitet wurde (es sei denn, es wurde gar nicht am Wochenende gearbeitet).

NoNightShiftBeforeFreeWeekend: Es ist ein freies Wochenende, wenn sie am ersten Tag des Wochenendes (je nach Wochenenddefinition) nicht arbeitet.

IdenticalShiftTypesDuringWeekend: Der erste Tag des wochenendes definiert, wie der Rest des Wochenendes auszusehen hat.

CheckNumAssignments: Für jeden weiteren unter-/überschrittenen Tag gibt es jeweils einen Strafpunkt.

## 6 Anhang

UnwantedPatterns: Erst wenn das gesamte Pattern erfüllt ist, gibt es genau einen Strafpunkt.

### 6.2 Erster Parameter Test

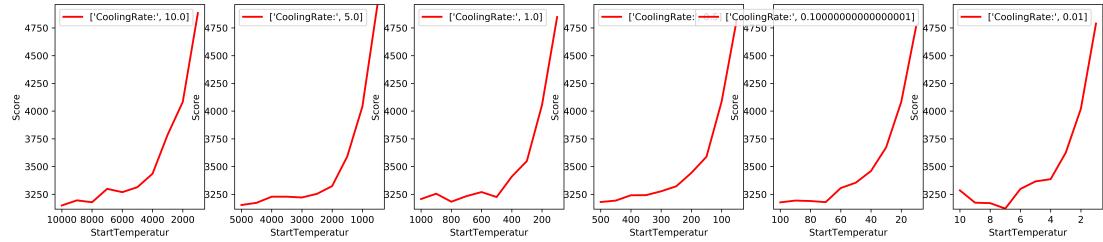


Abbildung 6.1: Long Hidden01

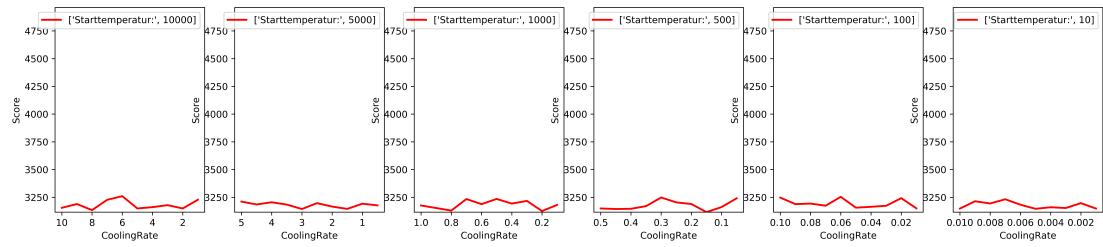


Abbildung 6.2: Long Hidden01

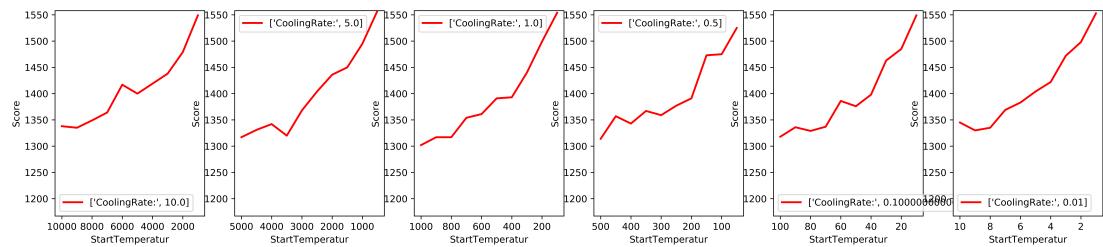


Abbildung 6.3: Long Hint01

## 6.2 Erster Parameter Test

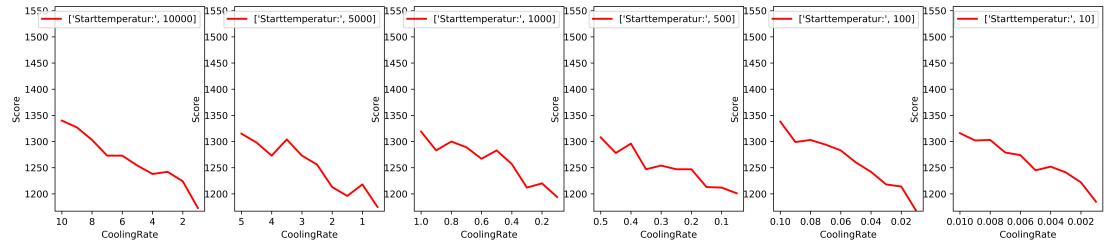


Abbildung 6.4: Long Hint01

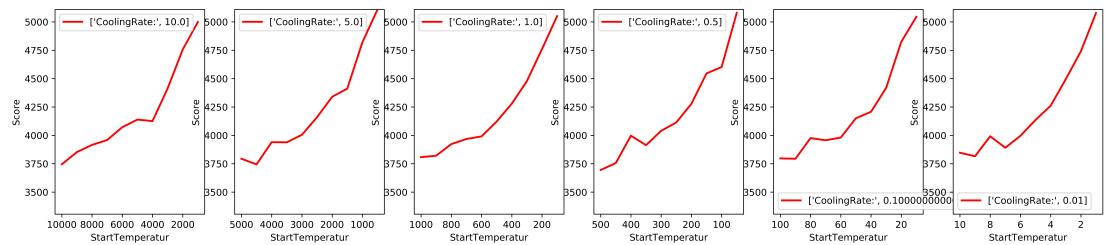


Abbildung 6.5: Long Late01

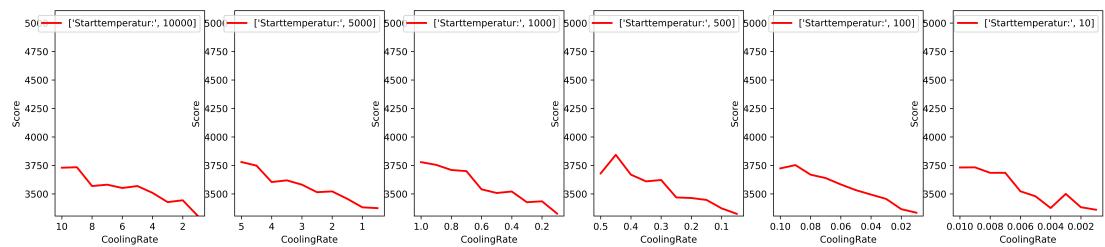


Abbildung 6.6: Long Late01

## 6 Anhang

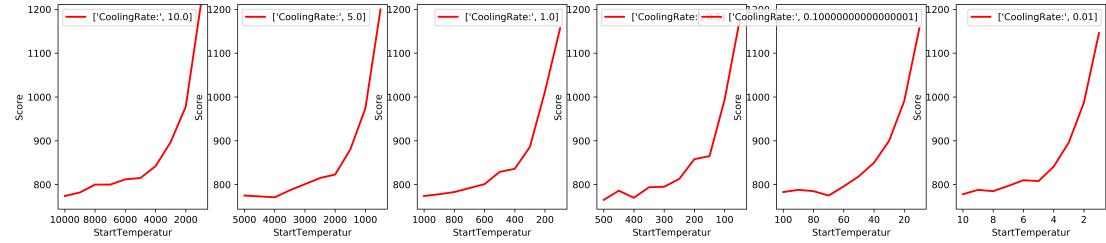


Abbildung 6.7: Long 01

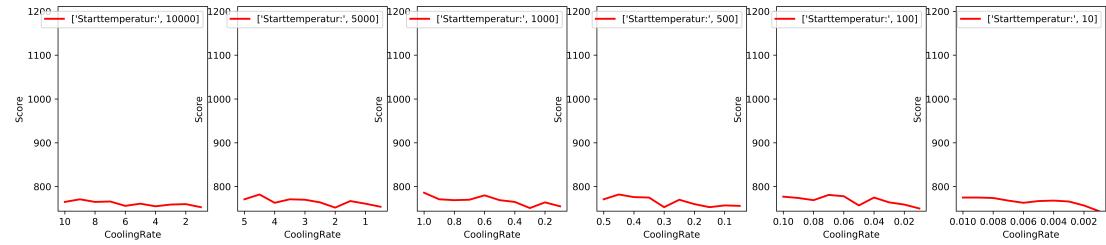


Abbildung 6.8: Long 01

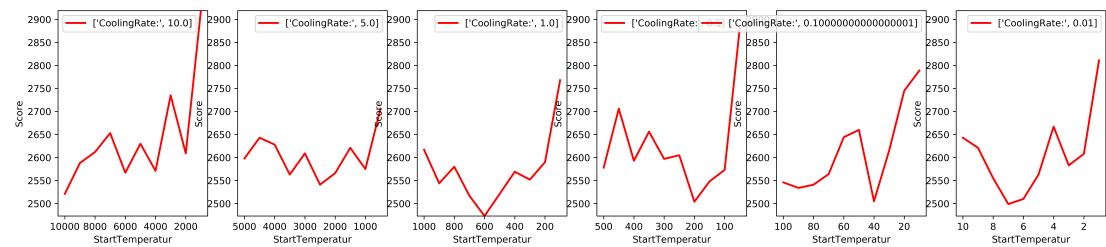


Abbildung 6.9: Medium Hidden01

## 6.2 Erster Parameter Test

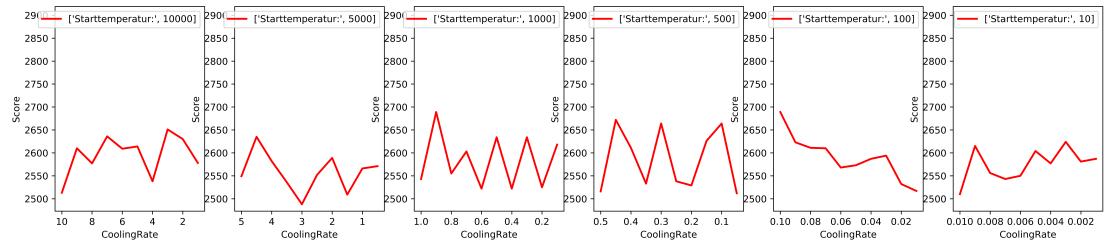


Abbildung 6.10: Medium Hidden01

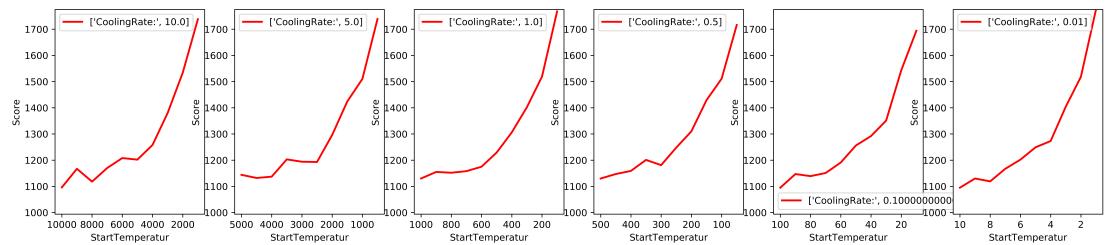


Abbildung 6.11: Medium Hint01

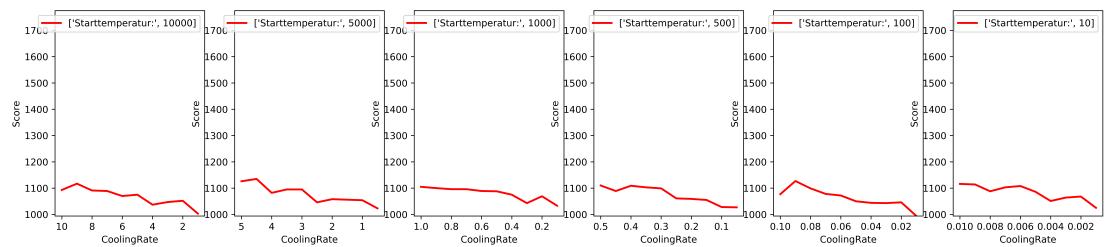


Abbildung 6.12: Medium Hint01

## 6 Anhang

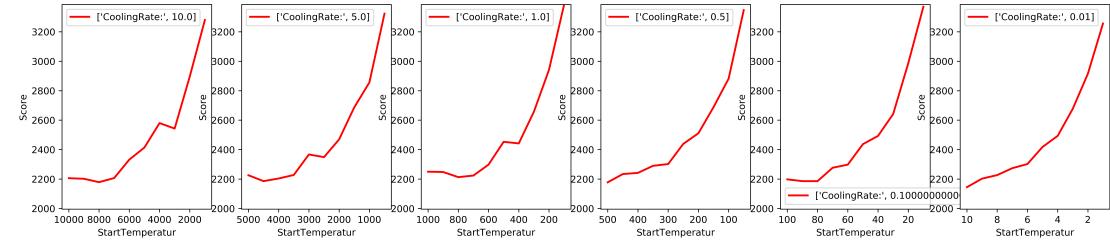


Abbildung 6.13: Medium Late01

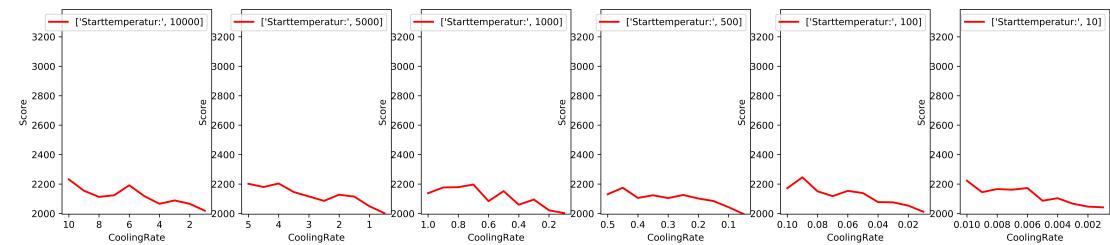


Abbildung 6.14: Medium Late01

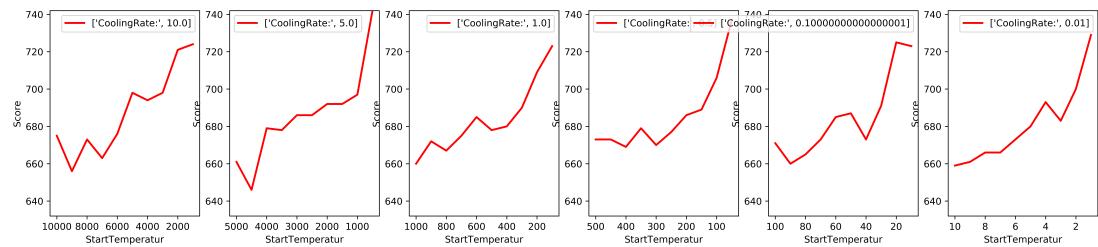


Abbildung 6.15: Medium01

## 6.2 Erster Parameter Test

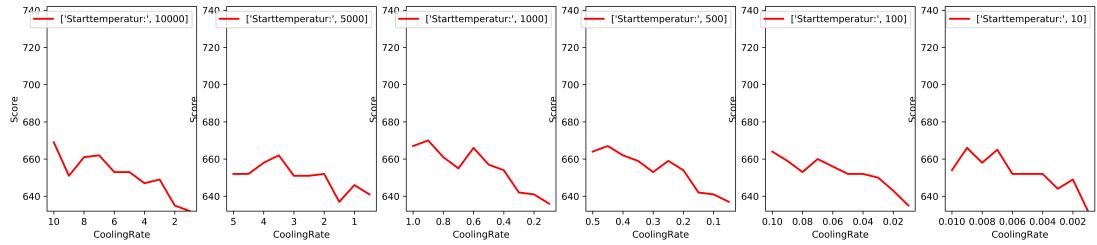


Abbildung 6.16: Medium01

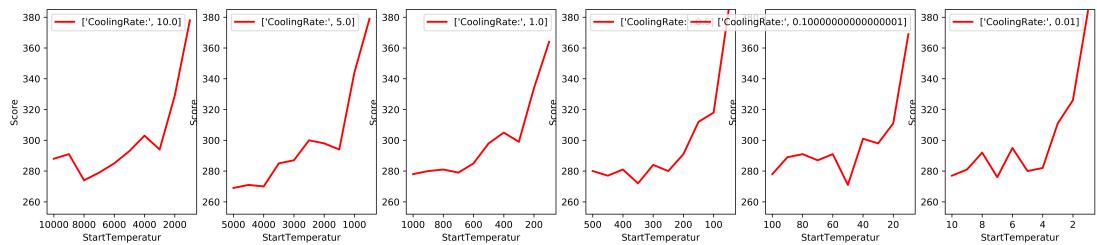


Abbildung 6.17: Sprint Hidden01

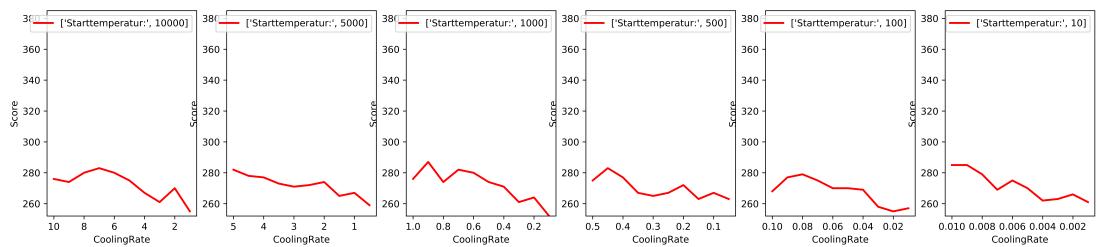


Abbildung 6.18: Sprint Hidden01

## 6 Anhang

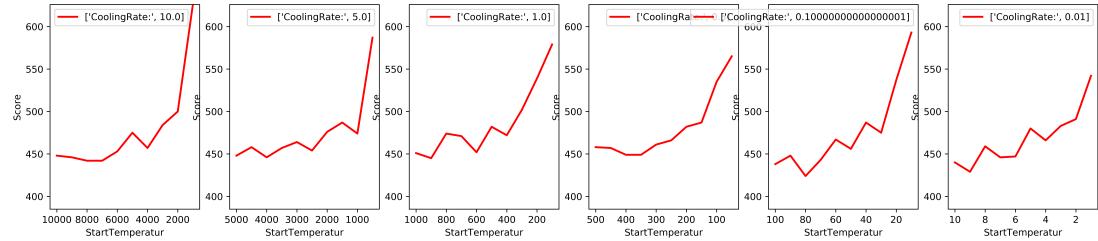


Abbildung 6.19: Sprint Hint01

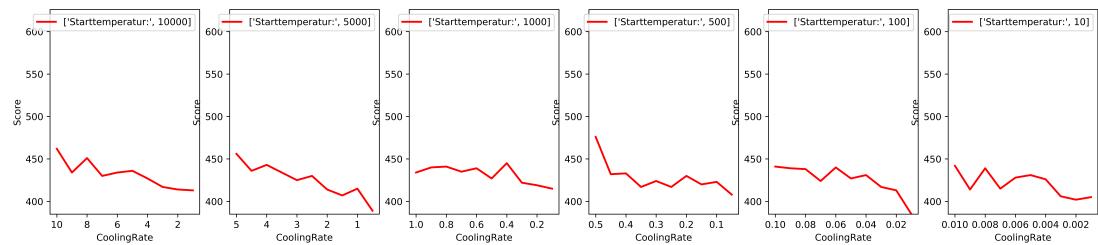


Abbildung 6.20: Sprint Hint01

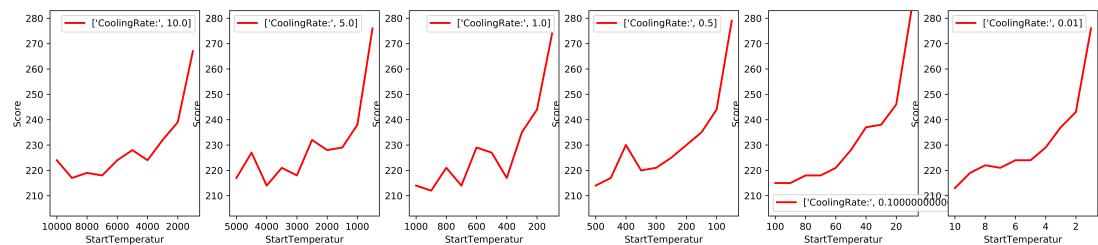


Abbildung 6.21: Sprint Late01

## 6.2 Erster Parameter Test

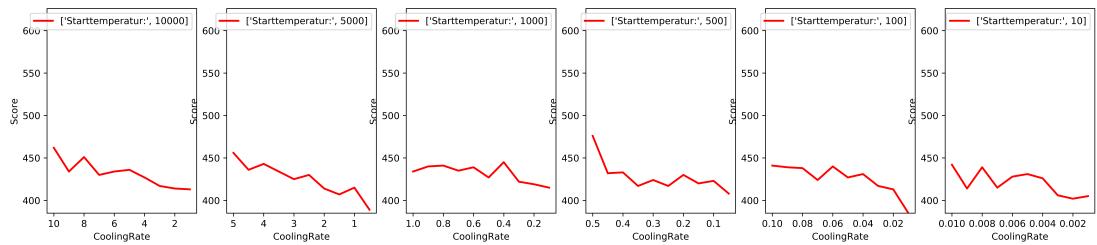


Abbildung 6.22: Sprint Late01

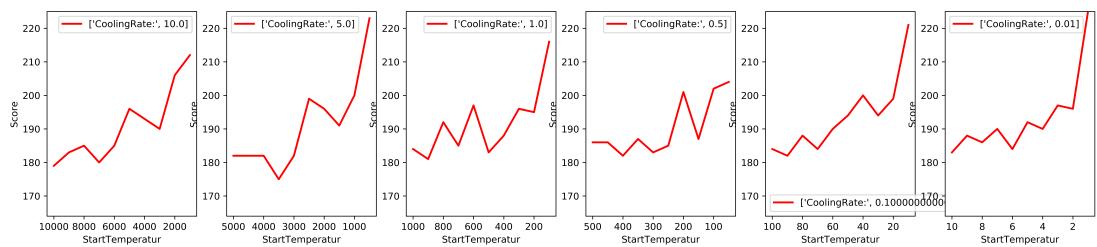


Abbildung 6.23: Sprint01

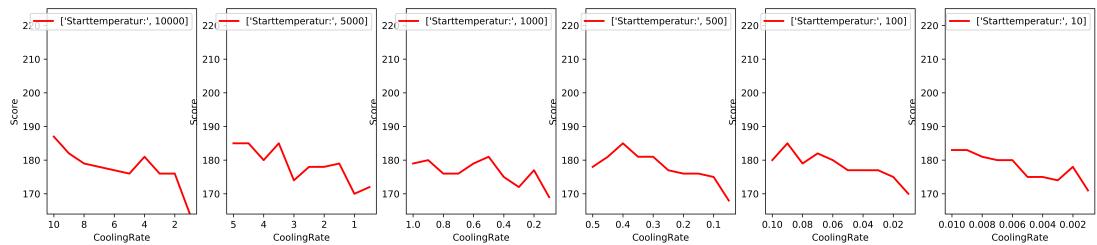


Abbildung 6.24: Sprint01

## 6 Anhang

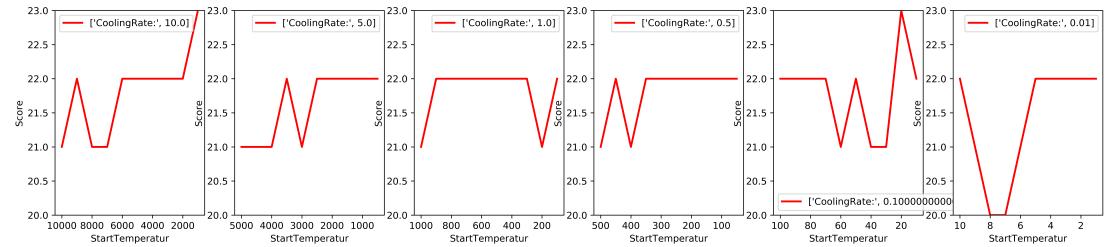


Abbildung 6.25: Toy1

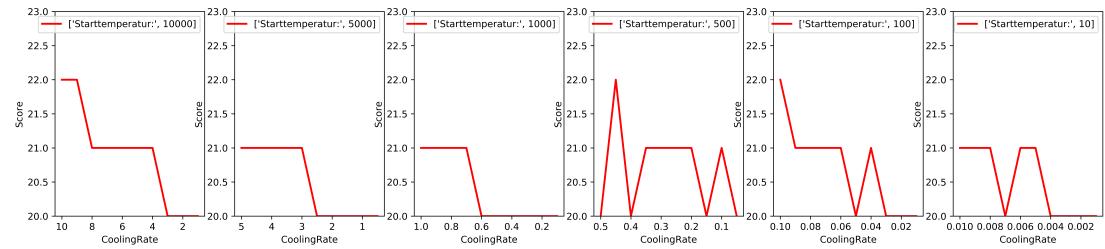


Abbildung 6.26: Toy1

### 6.3 Zweiter Parameter Test

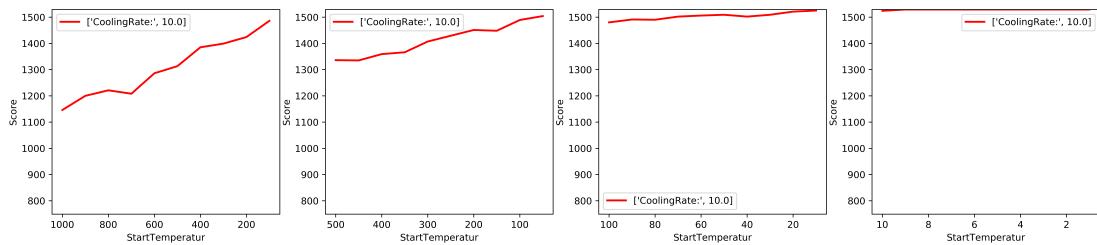


Abbildung 6.27: Long01 mit fester Cooling Rate von 10

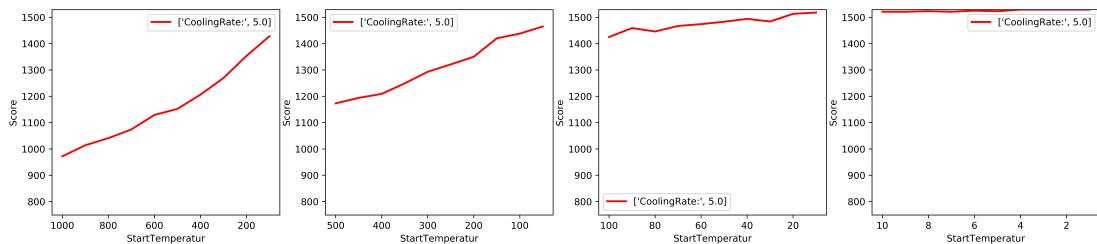


Abbildung 6.28: Long01 mit fester Cooling Rate von 5

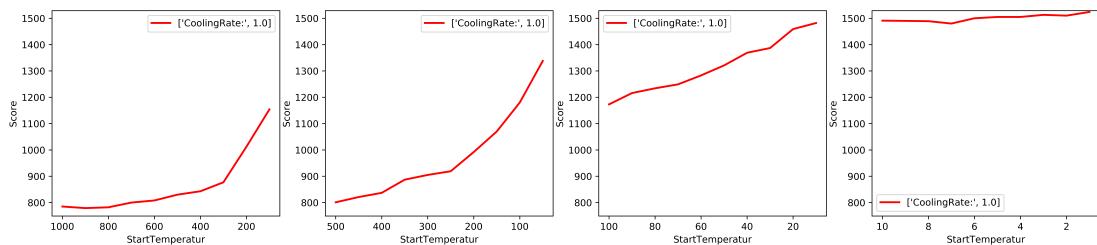


Abbildung 6.29: Long01 mit fester Cooling Rate von 1.0

## 6 Anhang

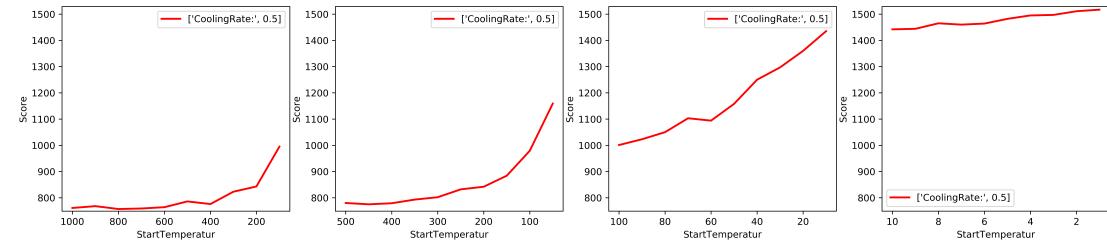


Abbildung 6.30: Long01 mit fester Cooling Rate von 0.5

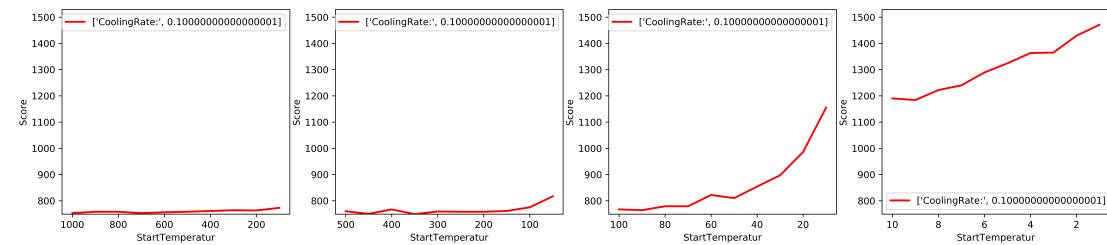


Abbildung 6.31: Long01 mit fester Cooling Rate von 0.1

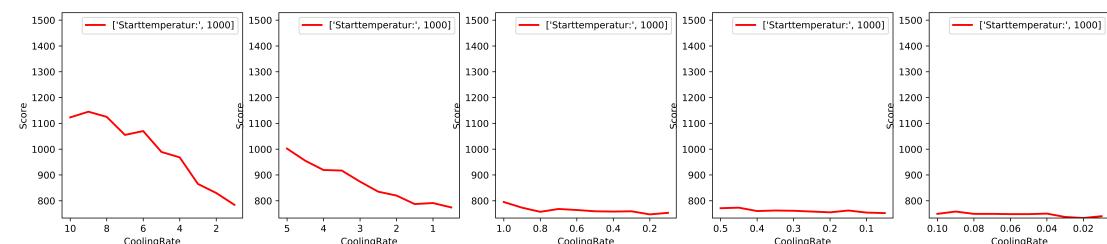


Abbildung 6.32: Long01 mit fester Starttemperatur von 1000

### 6.3 Zweiter Parameter Test

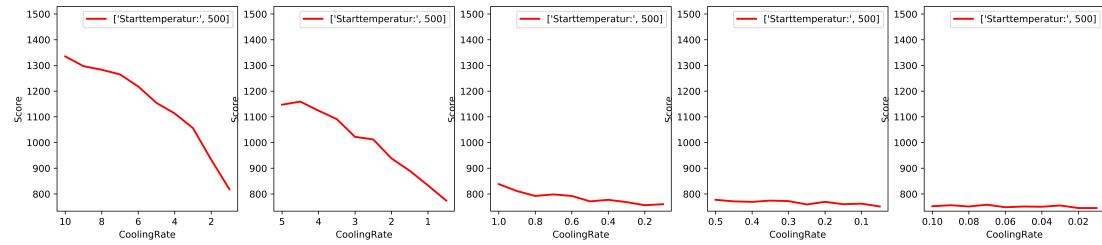


Abbildung 6.33: Long01 mit fester Starttemperatur von 500

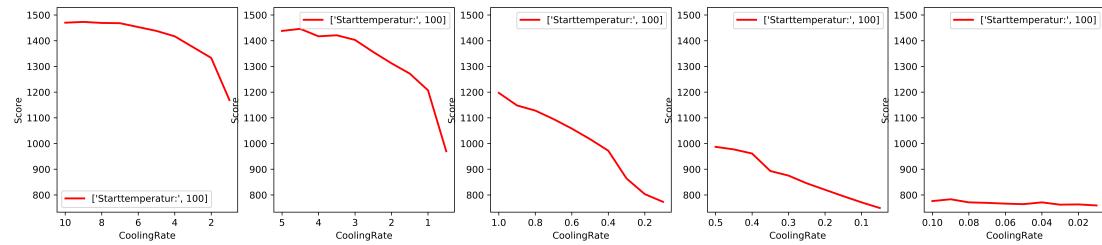


Abbildung 6.34: Long01 mit fester Starttemperatur von 100

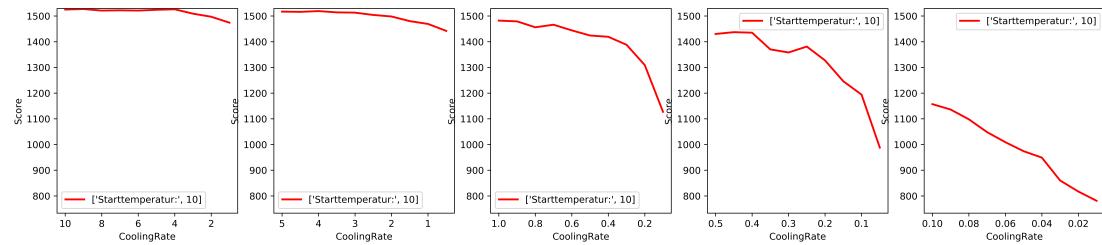


Abbildung 6.35: Long01 mit fester Starttemperatur von 10