RWTH RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN

Fraunhofer ISE

Simulation und Auslegungsoptimierung solarthermischer Kraftwerke unter Einsatz evolutionärer Algorithmen und neuronaler Netze

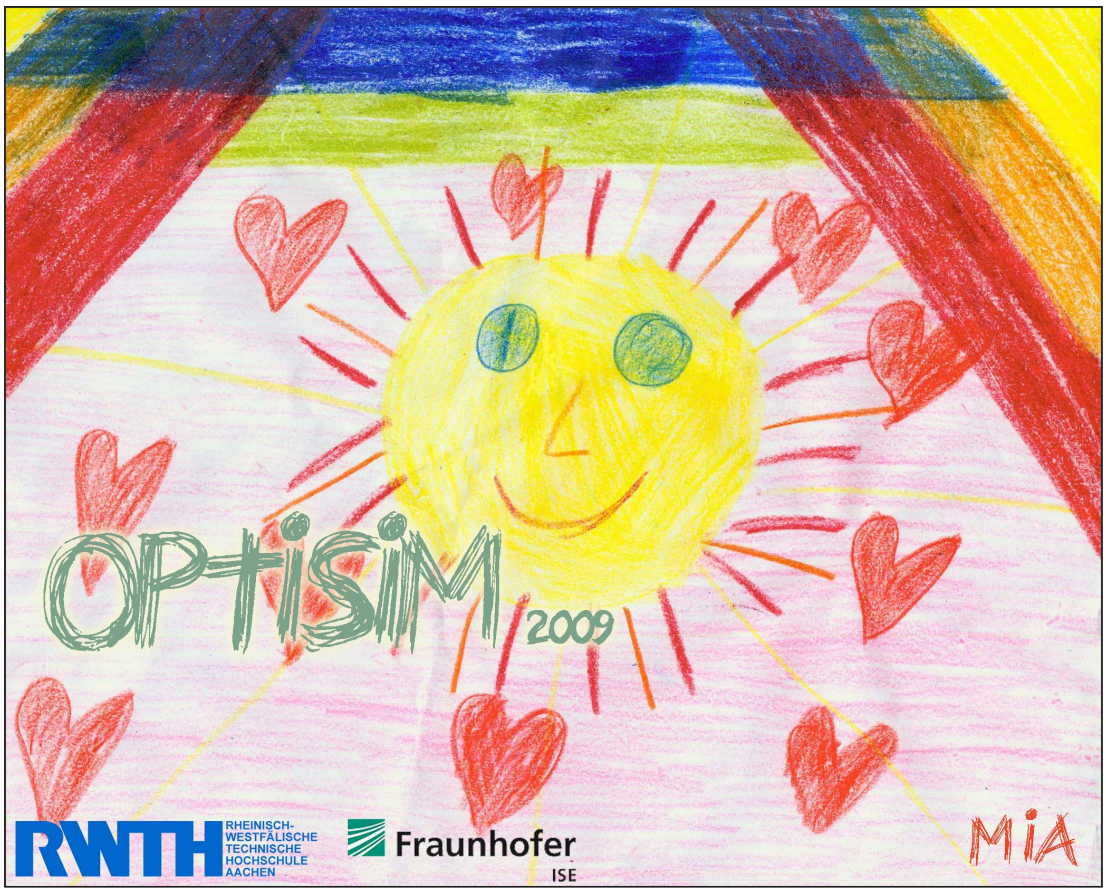
von
Pascal Richter

Diplomarbeit in Informatik

vorgelegt der
Mathematisch-Naturwissenschaftlichen Fakultät der
Rheinisch-Westfälischen Technischen Hochschule Aachen
im Dezember 2009

Angefertigt bei
Theorie der hybriden Systeme
Prof. Dr. Erika Ábrahám

Externe Betreuer
Prof. Dr. rer. nat. Volker Wittwer
Dipl.-Wi.-Ing. Gabriel Morin
Fraunhofer Institut für Solare Energiesysteme



Für meine kleine Familie.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Hintergrund

Durch die Verabschiedung von Einspeisegesetzen und den Aufbau eines Emissionshandelssystems in den EU-Mitgliedsstaaten haben die Entwicklung und der Einsatz erneuerbarer Energietechnik stark zugenommen. Das EU-Klimapaket von Dezember 2008[1] setzt das ehrgeizige Ziel der EU, den Anteil der regenerativen an der gesamten Stromerzeugung bis zum Jahre 2020 auf 20 Prozent zu steigern. Damit führt die regenerative Energieerzeugung längst kein Nischendasein mehr unter den Stromerzeugern, sondern stellt eine ernsthafte Alternative dar, vor allem zu Zeiten knapper werdender Ölreserven und des Klimawandels.

Eine Möglichkeit der regenerativen Energieerzeugung bieten solarthermische Kraftwerke. Das Prinzip erscheint recht simpel: Spiegel bündeln Sonnenstrahlen, um Wasser zu erhitzen, der entstehende Dampf treibt Turbinen an, wodurch Strom gewonnen wird. Solarthermische Kraftwerke bieten mehrere Vorteile: Sie bieten die Möglichkeit, im großen Stil Energie zu speichern. Somit kann der Strom zum Beispiel auch nachts zur Verfügung gestellt werden. Darüber hinaus funktionieren sie ähnlich wie konventionelle Dampfkraftwerke (z.B. Kohlekraftwerke). Damit können die hier langjährig gesammelten Erfahrungen ebenfalls für solarthermische Kraftwerke genutzt werden.

Solarthermische Kraftwerke sind seit Jahrzehnten erprobt. Bereits in den achtziger Jahren wurden in Kalifornien die ersten Parabolspiegel-Kraftwerke gebaut. Aber noch ist Solarthermie nicht so kosteneffizient wie Kohle oder Kernkraft.

[1] EU-Klimapaket von 2008: Richtlinien über erneuerbare Energien, die dritte Phase des Emissionshandelssystems, etc. Quelle: http://www.europarl.europa.eu/

Solarthermische Kraftwerke zur Stromerzeugung lohnen sich laut Ehrenberg [5] nicht im sonnenarmen Mitteleuropa. Jedoch in sonnenreichen Regionen wie Südspanien, Nordafrika oder dem Nahen Osten hat die Technik großes Potential (siehe Abbildung 1.1). Beispielsweise entwickelte eine Vereinigung von Wissenschaftlern das Desertec-Konzept, wonach riesige Spiegelkraftwerke in der Sahara theoretisch genug Strom für den Bedarf der gesamten Welt decken könnten. Spätestens seitdem sich Mitte 2009 ein Konsortium von mehreren Großkonzernen wie die Deutsche Bank, Siemens oder RWE zur Desertec Industrial[2] Initiative zusammentat, erscheint eine Umsetzung des Konzept immer wahrscheinlicher. Das Ziel der Initiative ist die Deckung von 15 Prozent des europäischen Bedarfs mit Strom aus der Wüste bis zum Jahr 2050.

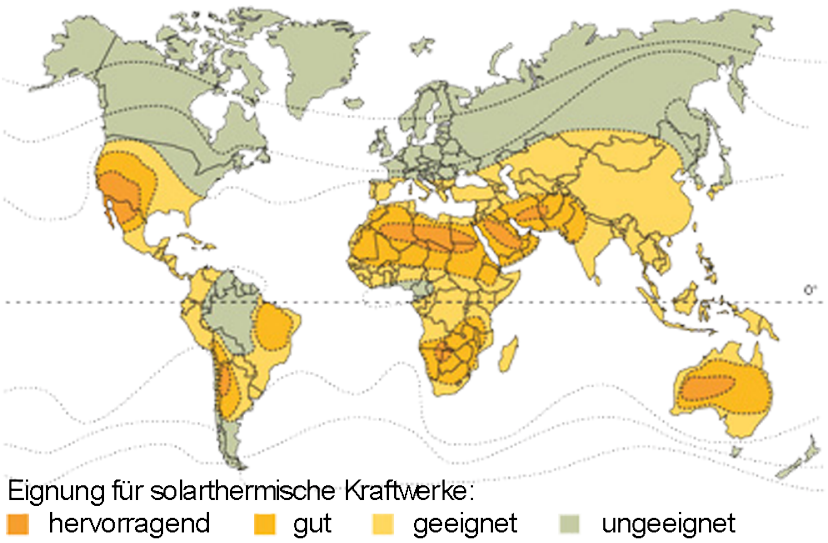


Abbildung 1.1: Sonnengürtel der Erde. Quelle: Solar Millennium

1.2 Ziele der Arbeit

Ein solarthermisches Kraftwerk (siehe Abbildung 1.2) besteht im Wesentlichen aus zwei Subsystemen: dem Solarkollektor, der die Sonnenenergie aufnimmt und dem Kraftwerksblock, der diese in Strom umwandelt. Die Wirtschaftlichkeit eines solarthermischen Kraftwerks wird durch die Effizienz (Wirkungsgrade) und die Kosten seiner Subsysteme bestimmt und lässt sich durch die Stromgestehungskosten (Kosten je kWh Strom) ausdrücken.

[2]Desertec Industrial: http://www.desertec.org/

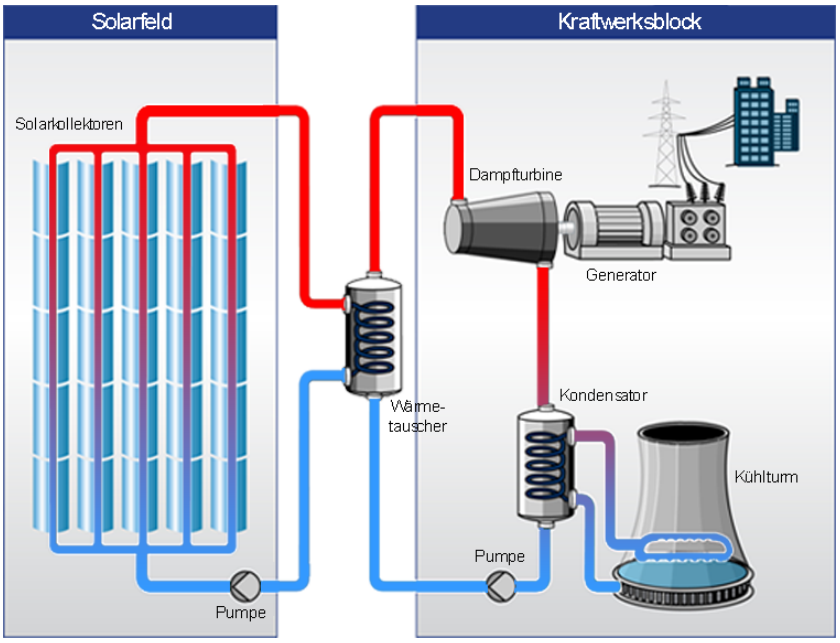


Abbildung 1.2: Schematischer Aufbau eines Solarkraftwerks. Spiegel bündeln Sonnenstrahlen, um ein Wärmemedium zu erhitzen, der entstehende Dampf treibt Turbinen an, wodurch Strom gewonnen wird.
Quelle: Solar Millennium

Die Subsysteme beeinflussen sich dabei gegenseitig, so dass die Wirkungsgradoptimierung des einen Subsytems auch zur Wirkunsgradreduzierung des anderen Subsystems führen kann: Eine Anhebung der Betriebstemperatur des Solarkollektors führt zu einem Wirkungsgradanstieg des Kraftwerkblocks. Im Kollektor hingegen bewirkt dies eine Verstärkung der Wärmeverluste, wodurch sich der Wirkungsgrad des Kollektors reduziert. Folglich ist es für die Optimierung eines solarthermischen Kreislaufes nicht sinnvoll die Subsysteme getrennt zu betrachten, sondern sogar notwendig ihre Abhängigkeiten in einem Modell zu berücksichtigen.

Da ein solches integriertes Modell bislang noch nicht existiert, besteht diesbezüglich noch Forschungsbedarf. Ziel dieser Diplomarbeit ist die Entwicklung einer Simulationssoftware, welche ein solarthermisches Kraftwerk mit allen Subsystemen abbildet. Ein solches Modell bietet die Grundlage, um die Stromgestehungskosten zu optimieren. Somit stellt die Auslegung von solarthermischen Kraftwerksanlagen im mathematischen Sinne ein Optimierungsproblem dar. Eine Vielzahl an Auslegungsparametern müssen so gewählt werden, dass eine Zielfunktion optimiert wird, die die Stromgestehungskos-

ten beschreibt.

Für die Simulation von solarthermischen Kraftwerken verfügt das Fraunhofer-Institut für Solare Energiesysteme über die Programmmodule Thermoflex und ColSim, die nach wirtschaftlichen Gesichtspunkten eine optimale Auslegung des Kraftwerksblocks und des Solarkollektors zulassen.

Im Rahmen dieser Arbeit wurde eine Simulationssoftware entwickelt, die diese beiden Programmmodule verbindet und an eine Optimierungseinheit anschließt. Aufgrund der großen Anzahl der zu optimierenden Variablen stellt es eine besondere Herausforderung dar, ein Optimierungsverfahren zu implementieren, welches sowohl zeiteffizient ist, als auch hinreichend genaue Ergebnisse liefert. In der Diplomarbeit von Gutjahr [9] wurde zur Optimierung des Kraftwerksblocks erfolgreich ein genetischer Algorithmus gewählt. Aufbauend auf diesen Erfahrungen wird im Folgenden der genetische Algorithmus verwendet, um ein solarthermisches Kraftwerk zu optimieren.

Zur Verbesserung der Zeiteffizienz, wurde die implementierte Software um ein weiteres Werkzeug der künstlichen Intelligenz erweitert. Es wurde untersucht, in wie weit sich neuronale Netze für die Anwendung in einer solarthermischen Kraftwerkssimulation eignen.

1.3 Gliederung der Arbeit

Die vorliegende Arbeit ist folgendermaßen gegliedert. In Kapitel 2 werden die energietechnischen Grundlagen für Dampfkreisläufe beschrieben und auf unterschiedliche Arten solarthermischer Kraftwerke eingegangen. In Kapitel 3 wird das Konzept und die Implementierung der Simulationssoftware vorgestellt. Dabei wird die Funktionsweise jeder Programmeinheit erläutert und die Datenübergabe zwischen diesen beschrieben. In Kapitel 4 wird eine Erweiterung der Simulationssoftware vorgestellt, welche basierend auf neuronalen Netzen die Simulationsgeschwindigkeit stark erhöht. Im Zentrum der Untersuchung steht die Güte des Konzepts, die aus unterschiedlichen Blickwinkeln betrachtet und bewertet wird. Im abschließenden Kapitel 5 werden die Ergebnisse zusammengetragen und Ausblicke bezüglich weiterer Untersuchungen gegeben.

Kapitel 2

Grundlagen der solaren Energietechnik

In diesem Kapitel werden zunächst die energietechnischen Grundlagen von Dampfkraftwerken beschrieben und anschließend auf die unterschiedlichen Arten solarthermischer Energiequellen eingegangen.

2.1 Dampfkraftwerke

Wasser-Dampf-Kreisläufe dienen nicht nur der konventionellen Stromerzeugung wie in Kohle- und Atomkraftwerken, sondern auch der regenerativen Energieerzeugung in Solarkraftwerken.

Zur Stromerzeugung wird die Dampfturbine in einem Wasser-Dampf-Kreislauf wie folgt benutzt (siehe Abbildung 2.1): Der zum Betrieb der Dampfturbine notwendige Wasserdampf wird durch eine Wärmequelle erzeugt. Dabei nehmen die Temperatur und das spezifische Volumen des Dampfes zu. Der Dampf strömt in die Dampfturbine, wo er einen Teil seiner zuvor aufgenommenen Energie als Bewegungsenergie an die Turbine abgibt. An die Turbine ist ein Generator angekoppelt, der die mechanische Leistung in elektrische Leistung umwandelt. Danach strömt der entspannte und abgekühlte Dampf in den Kondensator, wo er wieder flüssig wird. Durch die Verflüssigung des Dampfes wird ein großer Teil der zuvor aufgenommenen Wärmeenergie irreversibel an die Umgebung abgegeben. Anschließend kann das Wasser zu der Wärmequelle gepumpt und erneut erhitzt werden.

Die Aufgabe von Dampfkraftwerken ist also die Umwandlung von Wärmeenergie in

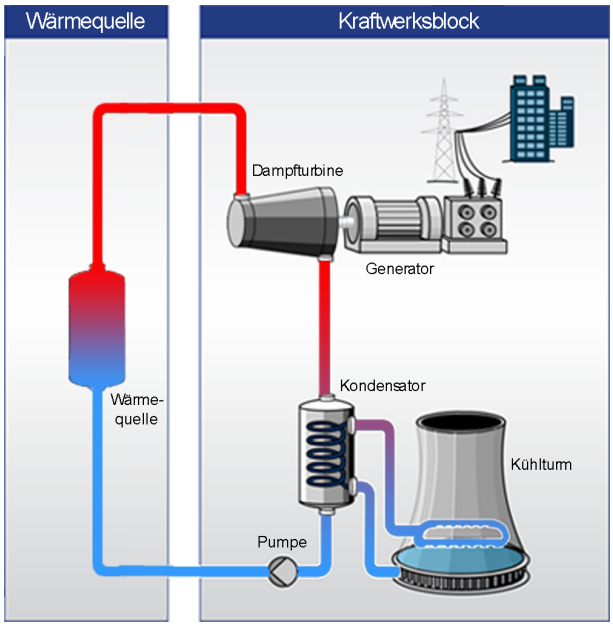


Abbildung 2.1: Schematischer Aufbau eines Dampfkraftwerks.

mechanische Energie und anschließend in elektrischen Strom. Ziel ist, dass möglichst viel der Wärmeenergie in elektrische Energie umgewandelt wird. Der Anteil der erzeugten elektrischen Energie an der zugeführten Wärmeenergie wird *Wirkungsgrad* genannt.

Wasser-Dampf-Kreisläufe sind thermodynamische Kreisprozesse. Zahoransky [29] beschreibt diese als Prozesse bei denen Wasser thermodynamische Zustandsänderungen durchläuft und wieder auf seinen ursprünglichen Zustand zurückgeführt wird. In den folgenden Abschnitten werden Dampfkraft-Prozesse und ihre wichtigsten Einflussgrößen erläutert.

2.1.1 Clausius-Rankine Prozess

Wittig [27] bezeichnet den Clausius-Rankine-Prozess als den idealisierten Basisprozess der allen realen Dampfkraftwerken zugrunde liegt. Alle Zustandsänderungen des Wassers vollziehen sich bei konstantem Druck (isobare Zustandsveränderung) oder bei konstanter Entropie[1] (isentrope Zustandsveränderung). Im Wesentlichen besteht der

[1] R. Clausius führte den Begriff Entropie (Einheit J/K) 1865 zur Beschreibung von Kreisprozessen ein. Es handelt sich um eine thermodynamische Größe, mit der Wärmeübertragungen und irreversible Vorgänge in thermodynamischen Prozessen rechnerisch erfasst und anschaulich dargestellt werden können, siehe [3].

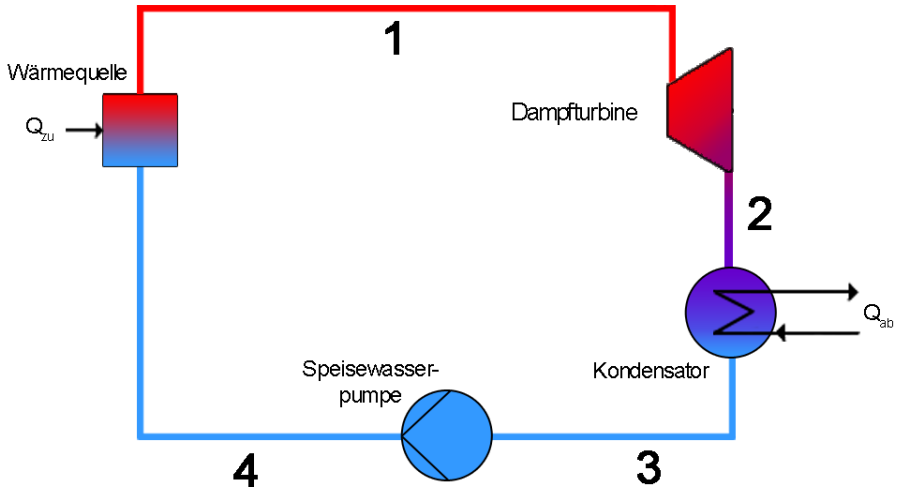


Abbildung 2.2: Anlagenschema des Clausius-Rankine Prozesses. In der Wärmequelle wird durch Energiezufuhr Wasserdampf erzeugt (Prozessschritt 4 → 1). Der Dampf strömt durch die Dampfturbine wodurch diese angetrieben wird (Prozessschritt 1 → 2). Anschließend verflüssigt sich der Dampf im Kondensator zu Wasser (Prozessschritt 2 → 3). Das Wasser wird zu der Wärmequelle gepumpt und erneut erhitzt (Prozessschritt 3 → 4).

Clausius-Rankine-Kreisprozess (siehe Abbildung 2.2) aus den Schritten:

- Isentrope Druckerhöhung des Wassers durch die Speisewasserpumpe, Prozessschritt 3 → 4
- Isobare Wärmezufuhr im Dampferzeuger, das Wasser wird in einer Wärmequelle erhitzt, verdampft und überhitzt[2], Prozessschritt 4 → 1
- Isentrope Dampfexpansion in der Turbine, Prozessschritt 1 → 2
- Isobare Wärmeabfuhr im Kondensator, Prozessschritt 2 → 3

Für den Clausius-Rankine Prozess kann der thermische Wirkungsgrad η_{th} bestimmt werden, der sich laut Cerbe [3] aus dem Verhältnis der nutzbaren Arbeit (Nutzarbeit W_{nutz}) zum zugeführten Wärmestrom Q_{zu} ergibt. Gemäß der in Abbildung 2.2 verwendeten Indizierungen lässt sich die Nutzarbeit als Differenz der zu- und abgeführten Wärme darstellen

$$W_{nutz} = Q_{zu} - |Q_{ab}|$$

[2]Beim Überhitzen wird Wasserdampf über seine Verdampfungstemperatur hinaus weiter erhitzt.

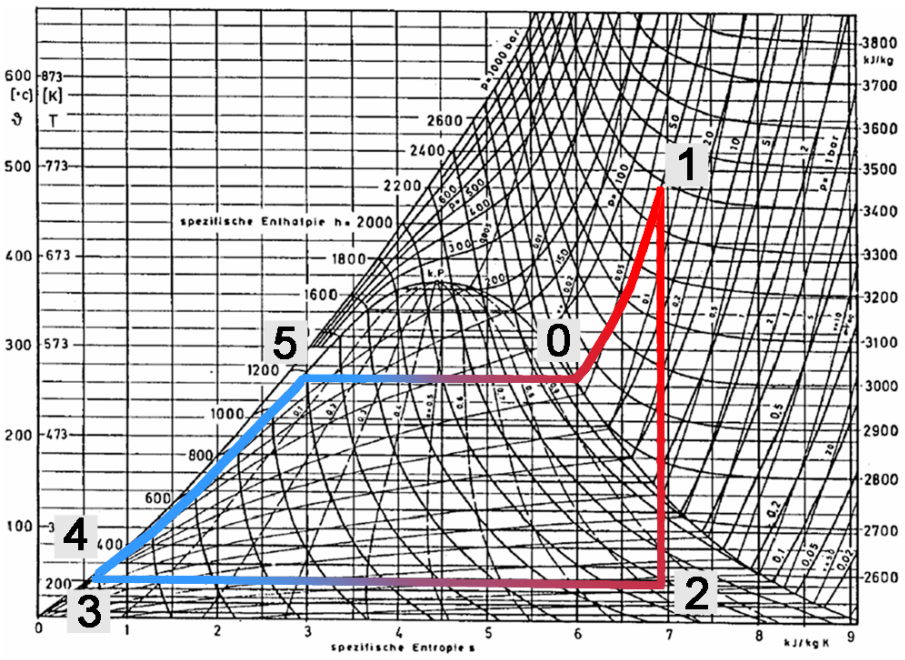


Abbildung 2.3: Clausius-Rankine Prozess im T-s Diagramm von Wasser. In der Wärmequelle wird durch Energiezufuhr Wasser erhitzt (Prozessschritt $4 \rightarrow 5$), verdampft (Prozessschritt $5 \rightarrow 0$) und überhitzt (Prozessschritt $0 \rightarrow 1$).Der Dampf strömt durch die Dampfturbine wodurch diese angetrieben wird (Prozessschritt $1 \rightarrow 2$). Anschließend verflüssigt sich der Dampf im Kondensator zu Wasser (Prozessschritt $2 \rightarrow 3$). Das Wasser wird zu der Wärmequelle gepumpt und erneut erhitzt (Prozessschritt $3 \rightarrow 4$). Die Fläche, die durch die Prozessschritte $4 \rightarrow 5 \rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ markiert wird, entspricht der erzeugten Nutzarbeit.

Somit kann der thermische Nettowirkungsgrad η_{th} angegeben werden als

$$\eta_{th} = \frac{W_{nutz}}{Q_{zu}} = 1 - \frac{|Q_{ab}|}{Q_{zu}} \tag{2.1}$$

Der Wirkungsgrad lässt sich auch im so genannten T-s Diagramm ablesen. In dem Zustandsdiagramm wird die spezifische Entropie s gegen die absolute Temperatur T aufgetragen. In Abbildung 2.3 ist das T-s Diagramm für den Clausius-Rankine Prozess dargestellt.

Die Fläche unterhalb der durch die Prozessschritte $4 \rightarrow 5 \rightarrow 0 \rightarrow 1$ markierten Verlaufslinie entspricht laut Cerbe [3] der zugeführten Wärme. Die vom Kondensator abgeführte Wärme entspricht der Fläche unterhalb der Punkte $2 \rightarrow 3$. Wird die abgeführte Wärme von der zugeführten Wärme abgezogen, ergibt sich die Nutzarbeit, die

der Fläche $3 \rightarrow 4 \rightarrow 5 \rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow 3$ entspricht.

Im Folgenden werden einige energietechnische Möglichkeiten zur Steigerung des thermischen Wirkungsgrades erläutert. In der konventionellen Kraftwerkstechnik führt dies dazu, dass Brennstoff gespart wird und somit zu einer Kostenreduzierung des erzeugten, elektrischen Stroms führt. Bei solaren Kraftwerken bewirkt dies eine Reduzierung der nötigen Kollektorfläche, was ebenfalls zur Senkung der Kosten beiträgt.

2.1.2 Wirkungsgradsteigernde Maßnahmen

Zu den häufig verwendeten wirkungsgradsteigernden Maßnahmen gehören die regenerative Speisewasservorwärmung und die Zwischenüberhitzung.

Regenerative Speisewasservorwärmung

Laut Morin [19] wird bei der regenerativen Speisewasservorwärmung in Anzapfungen längs der Turbine Dampf abgezweigt, um das Speisewasser vorzuwärmen. Nach Waas [25] werden in der Praxis zwischen einer und zehn Anzapfungen vorgesehen. Insgesamt wird etwa 25% bis 35% der Frischdampfmenge dazu verwendet. Durch die Vorwärmung steigt die Speisewassertemperatur und damit die mittlere Temperatur der Wärmezufuhr, was eine wirkungsgradsteigernde Funktion hat, vgl. [19].

In den Abbildungen 2.4 und 2.5 sind beispielhaft ein Anlagenschema mit Vorwärmer und das dazugehörige T-s-Diagramm dargestellt.

Zwischenüberhitzung

Eine weitere wirkungsgradsteigernde Maßnahme ist die Zwischenüberhitzung. Die Turbinen der meisten Kraftwerke bestehen aus einem Hochdruck- und einem Niederdruckteil. Bei der Zwischenüberhitzung wird der im Hochdruckteil entspannte Dampf wieder aufgeheizt, bevor er im Niederdruckteil weitere Arbeit verrichtet (siehe Abbildung 2.6). Laut Morin [19] steigert auch diese Maßnahme die mittlere Temperatur der Wärmezufuhr und damit den thermischen Wirkungsgrad des Dampfkraftprozesses. Die Abbildungen 2.6 und 2.7 zeigen beispielhaft ein Anlagenschema mit Zwischenüberhitzung und die dazugehörigen Prozessschritte im T-s-Diagramm.

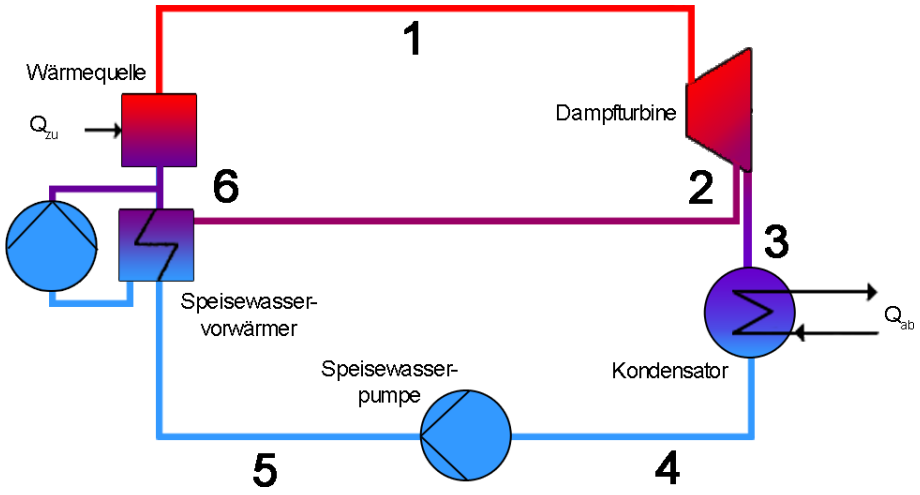


Abbildung 2.4: Anlagenschema eines Clausius-Rankine Prozesses mit einem Vorwärmer. Mit Anzapfdampf aus der Dampfturbine wird das Speisewasser vorgewärmt (Prozessschritt 2 → 6).

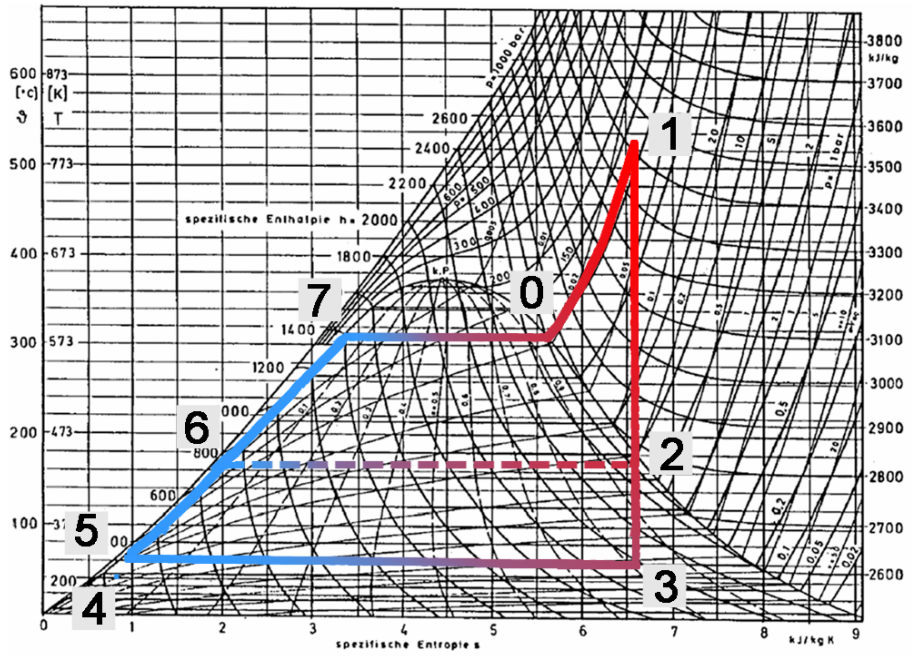


Abbildung 2.5: Clausius-Rankine Prozess mit Vorwärmer im T-s Diagramm. Mit Anzapfdampf aus der Dampfturbine wird das Speisewasser vorgewärmt (Prozessschritt 2 → 6) und somit die Fläche, die der Nutzarbeit entspricht, im Vergleich zum Clausius -Rankine Prozess vergrößert.

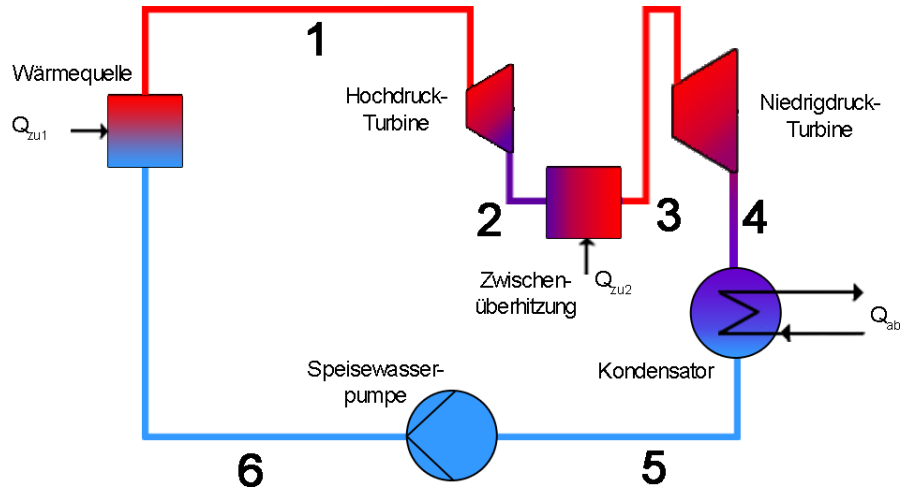


Abbildung 2.6: Anlagenschema eines Clausius-Rankine Prozesses mit Zwischenüberhitzung. Der Dampf strömt durch den Hochdruckteil der Turbine (Prozessschritt 1 → 2). Anschließend wird er im Zwischenüberhitzer erneut erhitzt (Prozessschritt 2 → 3) bevor er durch den Niederdruckteil der Turbine strömt (Prozessschritt 3 → 4).

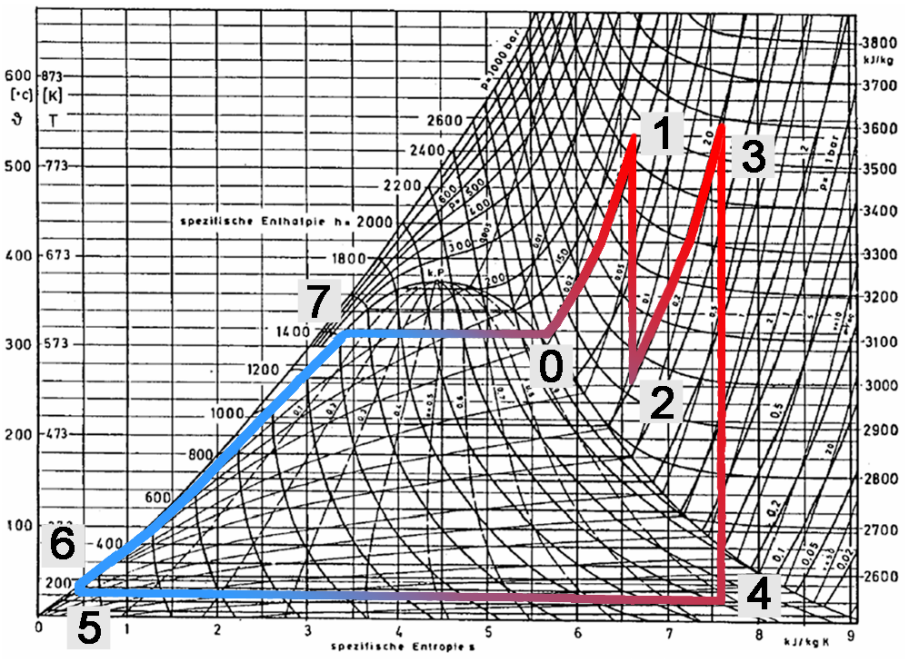


Abbildung 2.7: Clausius-Rankine Prozess mit einfacher Zwischenüberhitzung im T-s Diagramm. Der Dampf strömt durch den Hochdruckteil der Turbine (Prozessschritt 1 → 2). Anschließend wird er im Zwischenüberhitzer erneut erhitzt (Prozessschritt 2 → 3) bevor er durch den Niederdruckteil der Turbine strömt (Prozessschritt 3 → 4). Die Fläche, die der Nutzarbeit entspricht hat sich im Vergleich zum Clausius Rankine Prozess vergrößert.

2.2 Solarthermische Dampfkraftwerke

Die in Abschnitt 2.1 vorgestellten Wasser-Dampf-Kreisläufe werden wie bereits erwähnt bei vielen Kraftwerkstypen verwendet. Sie unterscheiden sich im Wesentlichen nur durch ihre Wärmequelle. Im Folgenden werden Dampfkraftwerke mit verschiedenen solarthermischen Wärmequellen vorgestellt.

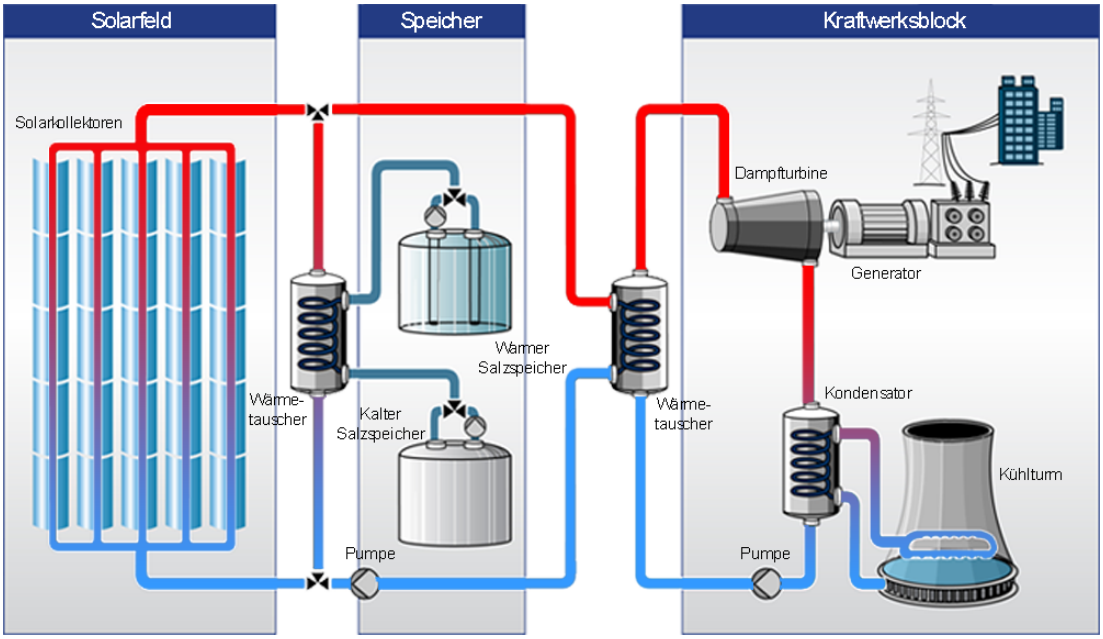


Abbildung 2.8: Schematischer Aufbau des 50 MW-Solarkraftwerkes „Andasol-I“ in Adalusien (Spanien). Das solar erhitzte Wärmemedium wird über Wärmetauscher an den konventionellen Dampfkreislauf abgegeben. Der anfallende Dampf wird in der Turbine entspannt und das Thermoöl wird wieder dem Prozess zu geführt. Die Wärme kann in dem Salzspeicher zwischengespeichert werden.
Quelle: Solar Millennium

Solarthermische Dampfkraftwerke (siehe Abbildung 2.8) basieren auf der Idee, die direkte Sonnenstrahlung gebündelt als Wärmequelle zu nutzen. Sie verwenden Spiegel (Reflektoren), um das einfallende Sonnenlicht auf einem so genannten Absorber zu bündeln. Durch den Absorber strömt ein Wärmeträger (Wasser oder Thermoöl), welcher mit der eingefangenen Sonnenergie aufgeheizt wird. Die Erhitzung des Wärmeträgers durch die gebündelte Sonnenstrahlung bildet also die Wärmequelle des Wasser-Dampf-Kreislaufes. Wird Thermoöl als Wärmeträger verwendet, so wird an dieser Stelle durch einen Wärmetauscher die thermische Energie des Thermoöls dem Wasserdampfkreis-

lauf zugeführt.
Solarthermische Dampfkraftwerke bieten die Möglichkeit, im großen Stil Energie zu speichern. Somit kann der Strom zum Beispiel auch nachts zur Verfügung gestellt werden. Zum Speichern wird ein Speichermedium erhitzt (siehe Abbildung 2.8). Meist wird ein Flüssigsalzgemisch bestehend aus 60 % Natriumnitrat ($NaNO_3$) und 40 % Kaliumnitrat (KNO_3) verwendet. In einem Wärmetauscher gibt der Wärmeträger seine Wärme an eine flüssige Salzschmelze ab. Mit dem erhitzten flüssigen Salz kann dann bei Bedarf wieder der Wärmeträger erhitzt werden.

Die Reflektoren werden nach dem Stand der Sonne einachsig oder zweiachsig ausgerichtet. Dies bezeichnet man als Nachführen. Zur Bündelung der Sonnenstrahlung gibt es zwei Konzepte die in üblichen Kraftwerksblöcken (ab 10 MW_{el}) eingesetzt werden: Parabolrinnen- und Fresnel-Kollektor-Kraftwerke fokussieren die Sonnenstrahlung auf ein Absorberrohr, während Solarturm-Kraftwerke die Strahlung der Sonne mit Punktkonzentratoren auf einen Brennpunkt bündeln. In den folgenden Abschnitten werden diese drei solarthermischen Dampfkraftwerke näher beschrieben.

2.2.1 Parabolrinnen-Kraftwerke

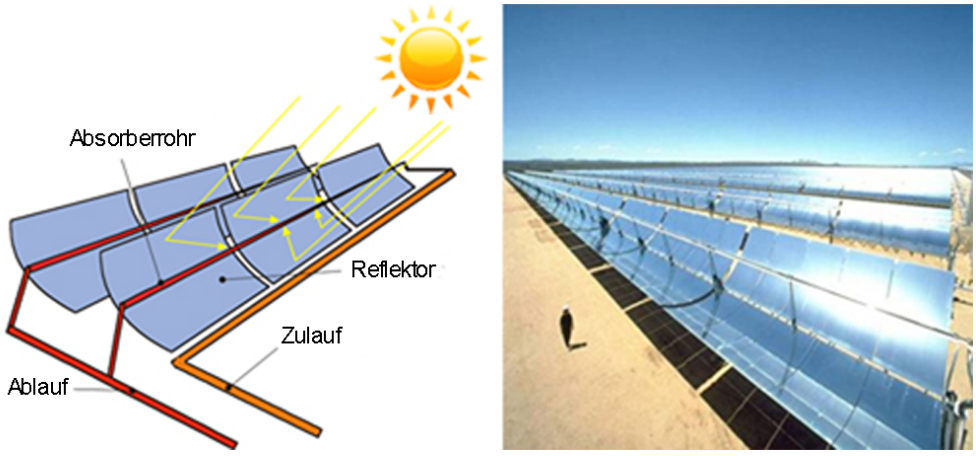


Abbildung 2.9: Parabolrinnenkraftwerk. Links schematische Darstellung, rechts eine kommerziell betriebene 50 MW-Anlage in der spanischen Provinz Granada.
Quelle: HTT energy systems (http://www.htt-energy-systems.biz/index.php?id=63)

Parabolrinnenkollektoren bestehen aus parabolisch gewölbten Spiegeln, die das Sonnenlicht auf ein in der Brennlinie verlaufendes Absorberrohr bündeln (siehe Abbildung 2.9). Die Parabolrinne gleicht im Querschnitt einer Parabel und hat damit die

Eigenschaft, parallel einfallende Strahlung in einem Brennpunkt zu fokussieren (siehe Abbildung 2.10).

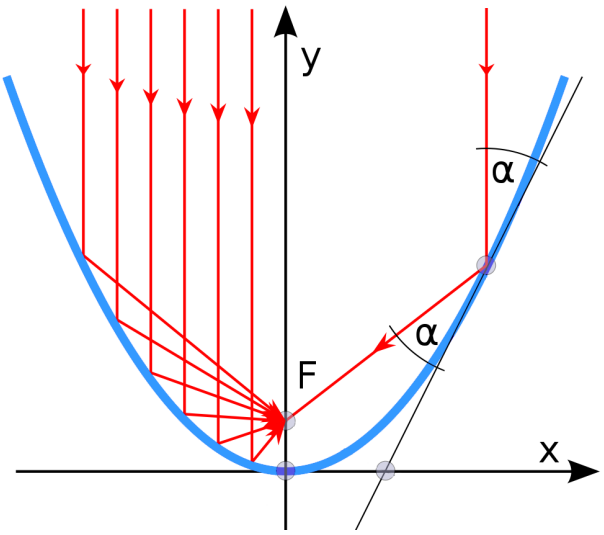


Abbildung 2.10: Eigenschaften einer Parabel - Strahlen die parallel zur Achse einfallen, werden durch die Spiegelung an der Parabel im Brennpunkt F gebündelt. Eine Parabel mit der Gleichung $y = ax^2$ für $a \in \mathbb{R}\backslash\{0\}$ hat den Brennpunkt $F(0, |\frac{1}{4a}|)$, vgl. [24].

Die Öffnung der Parabel (Apertur) misst in heutigen Kraftwerken zwischen 5 und 6 Metern. In der Brennlinie befindet sich ein hochselektiv beschichtetes Stahlrohr, das die Strahlung absorbiert. Hochselektiv bedeutet, dass es sowohl Sonnenstrahlung (kurzwelliges Licht) gut absorbiert als auch wenig Wärmestrahlung (langwelliges Licht) abgibt. Um Wärmeverluste zu verringern, ist das Absorberrohr in ein vakuumdichtes Glasrohr eingebettet.

Die vom Absorberrohr aufgenommene Wärmeenergie wird durch die Rohrwand an das Wärmeträgermedium übertragen. Hierbei handelt es sich meist um Thermoöl, das im Kollektorfeld auf maximal 400°C erhitzt wird. Durch die Verwendung von Wasser statt Thermoöl als Wärmeträger sind sogar noch höhere Dampftemperaturen von maximal 450°C erreichbar. Das Kollektorfeld setzt sich aus mehreren meist einigen hundert Meter langen Spiegelreihen zusammen. Die Spiegel werden einachsig der Sonne nachgeführt und in der Regel in Nord-Süd-Richtung ausgerichtet.

Abbildung 2.11: Fresnel-Kollektor-Kraftwerk. Links schematische Darstellung, rechts eine 1,4 MW Testanlage in Calasparra.
Quelle: Novatec-Biosol (http://www.novatec-biosol.com)

2.2.2 Fresnel-Kollektor-Kraftwerke

Fresnel-Kollektor-Kraftwerke bündeln ebenfalls das Sonnenlicht auf ein in der Brennlinie verlaufendes Absorberrohr. Im Unterschied zu den Parabolrinnenkollektoren setzen sich die Fresnel-Kollektoren aus mehreren parallelen, ungewölbten Spiegelreihen zusammen. Nach Morin [19] liegt die Spiegelbreite der einzelnen Spiegelfacetten zwischen 0,5 bis 2 Metern.

Das Sonnenlicht wird über mehrere einachsig gelagerte Spiegelreihen auf ein Absorberrohr gebündelt. Da die Optik der Reflektoren zu einer geweiteten Brennlinie führt, verfehlen einige der ankommenden Strahlen den Absorber. Mit Hilfe eines zusätzlich angebrachten Sekundärspiegel hinter dem Rohr, kann die Strahlung auf das Absorberrohr gelenkt werden, (siehe Abbildung 2.11).

2.2.3 Solarturm-Kraftwerke

Solarturm-Kraftwerke gehören zu den punktkonzentrierenden Systemen. Der Absorber ist auf einem Turm angebracht, um den herum mehrere Spiegel, die so genannten Heliostaten, angeordnet sind (siehe Abbildung 2.12). Die Heliostaten werden zweiachsig der Sonne nachgeführt und bündeln die Sonnenstrahlen auf den zentralen Absorber. Dieser absorbiert die Sonnenenergie und gibt sie an das Wärmeträgermedium ab. Das verwendete Wärmeträgermedium ist entweder flüssiges Nitratsalz, Wasserdampf oder

Heißluft. Durch die starke Konzentration der Sonneneinstrahlung können auch hohe Temperaturen von bis zu 1.000 °C erzeugt werden, was neben klassischen Dampfturbinen auch den Einsatz von Gasturbinen ermöglicht.

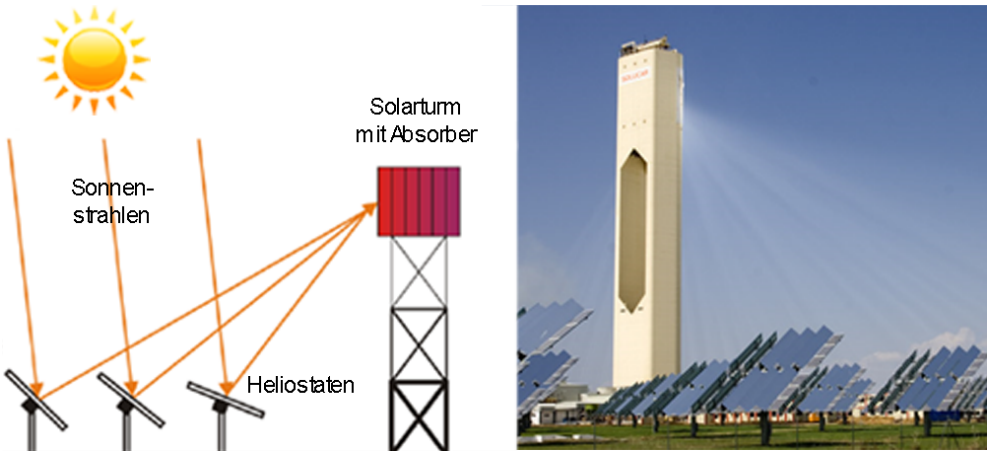


Abbildung 2.12: Solarturmkraftwerk. Links schematische Darstellung, rechts das 11 MW-Solarkraftwerk „PS10“ bei Sevilla in Spanien.
Quelle: ALTAC

Kapitel 3

Simulations- und Optimierungssoftware

In diesem Kapitel wird das Konzept und die Implementierung der Simulationssoftware vorgestellt. Die Software besteht aus mehreren Programmmodulen, die untereinander kommunizieren und durch einen Optimierungsalgorithmus, dem so genannten Optimierer gesteuert werden. Im Folgenden werden zunächst das Gesamtkonzept und die Kommunikation zwischen den Modulen beschrieben. Anschließend wird die Funktionsweise jedes einzelnen Moduls erläutert.

3.1 Konzept der Simulationssoftware

Die Software soll solarthermische Kraftwerke mit allen Subsystemen modellieren. Darüber hinaus soll in Abhängigkeit von den Kraftwerksparametern eine Zielfunktion optimiert werden, die die Stromgestehungskosten angibt.

Die Simulationssoftware umfasst die folgenden vier Komponenten (siehe Abbildung 3.1).

- **Optimierungsalgorithmus.** Mit dem Optimierer sollen gleichzeitig Kollektor- wie auch Kraftwerksblockvariablen optimiert werden (siehe Abschnitt 3.2). Die vom Optimierungsalgorithmus erstellten Belegungen der Variablen heißen Individuen. Ein Individuum i hat die Form

$$i = (\ \underbrace{t_1, \ldots, t_n}_{\text{Kraftwerksblock}}\ ,\quad \underbrace{c_1, \ldots, c_m}_{\text{Solarkollektor \& Speicher}}\ ) \in \mathbb{R}^{n+m}.$$

Der Zielfunktionswert eines jeden Individuums wird vom Kostenmodell geliefert.

- **Kraftwerksblock.** Um den Kraftwerksblock zu modellieren, wird die Software Thermoflex[1] verwendet. Mit $(t_1, \ldots, t_n) \in \mathbb{R}^n$ wird mit Hilfe von Thermoflex der Kraftwerksblock ausgelegt und mit dessen integrierten PEACE-Modul die Investitionskosten[2]

$$f_{K_{pb}}(t_1, \ldots, t_n) = K_{PBinvest} \in \mathbb{R}^+$$

bestimmt. Des Weiteren werden verschiedene Betriebszustände[3]

$$f_{B_{pb}}(t_1, \ldots, t_n) = (o_1, \ldots, o_\ell) \in \mathbb{R}^\ell$$

des Kraftwerksblocks berechnet (siehe Abschnitt 3.3).

- **Solarkollektor und Speicher.** Zur energetischen Abbildung des Kollektors und des thermischen Speichers wird das von Wittwer [28] am Fraunhofer ISE[4] entwickelte Programm ColSim verwendet, welches das Kraftwerksblockmodell von Thermoflex nutzt (siehe Abschnitt 3.4). Für die Simulation wird eine Variablenbelegung $(c_1, \ldots, c_m, o_1, \ldots, o_\ell) \in \mathbb{R}^{m+\ell}$ benötigt. Als Ausgabe liefert ColSim den Jahresenergieertrag des Kraftwerks[5]

$$f_P(c_1, \ldots, c_m, o_1, \ldots, o_\ell) = P_{el} \in \mathbb{R}.$$

- **Kostenmodell.** Mit dem von Morin [20] entwickelten Kostenmodell calculate_LEC wird auf Grundlage des Jahresenergieertrags des Kraftwerks eine Kostenrechnung aufgestellt, die auf den Investitionskosten und laufenden Kosten basiert. Als Resultat liefert das Kostenmodell die Stromgestehungskosten[6]

$$f_{LEC}(K_{PBinvest}, P_{el}) = LEC \in \mathbb{R}^+.$$

[1] Thermoflex ist ein kommerzielles Programm des Unternehmens Thermoflow Inc.
[2] $f_{K_{pb}}$ ist die Funktion, die die Kosten des Power Blocks bestimmt.
[3] $f_{B_{pb}}$ ist die Funktion, die die Betriebszustände des Power Blocks bestimmt.
[4] ISE - Institut für Solare Energiesysteme
[5] f_P ist die Funktion, die die elektrische Leistung P Kraftwerksblocks bestimmt.
[6] f_{LEC} ist die Funktion, die die Stromgestehungskosten (engl. Levelized Electricity Costs) bestimmt.

In das Kostenmodell ist die automatische Ausführung von ColSim für eine Variablenbelegung $(c_1, \ldots, c_m)$ und Kraftwerksblock-Betriebszustände $(o_1, \ldots, o_\ell)$ implementiert. Zusammen mit den Kraftwerksinvestitionskosten $K_{PBinvest}$ wird mit dem von ColSim errechneten Jahresenergieertrag P_{el} eine Kostenrechnung aufgestellt und die Stromgestehungskosten LEC bestimmt. Durch diese bereits vorhandene Implementierung des Kostenmodell-Programms, ändert sich der Simulationskreislauf gemäß der Abbildung 3.2.

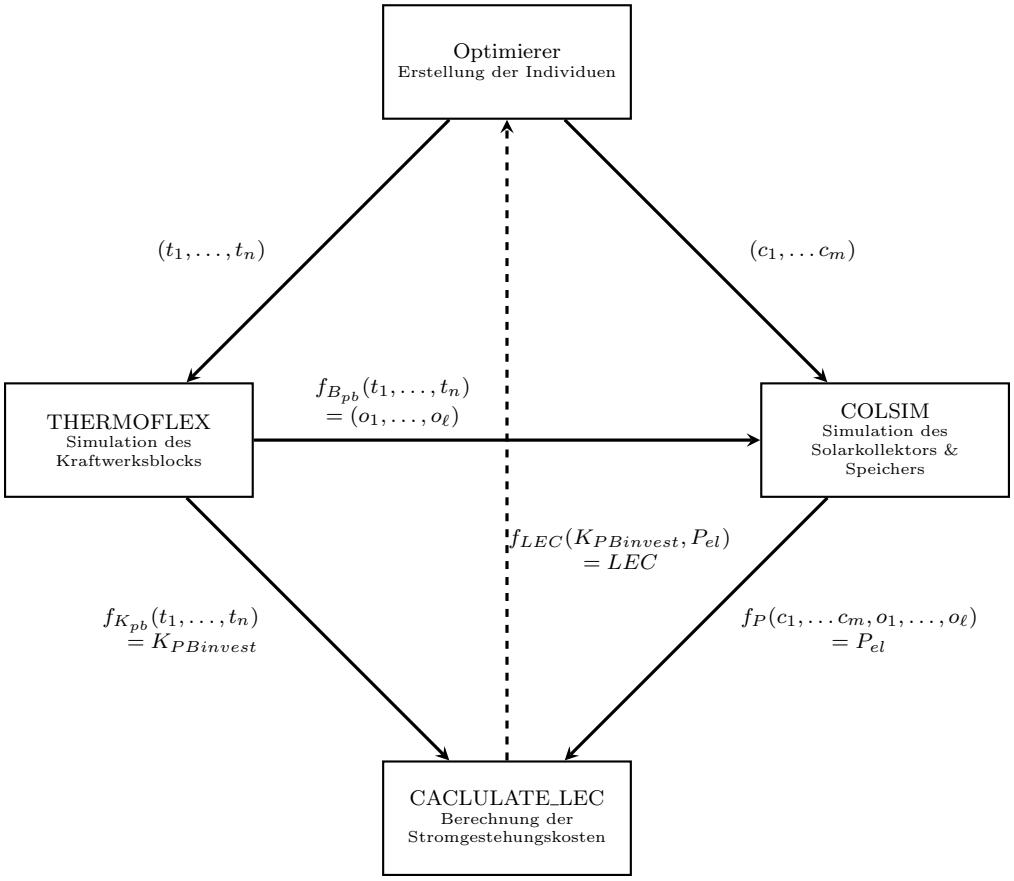


Abbildung 3.1: Kopplung der Komponenten Kraftwerksblock, Solarkollektor & Speicher und Kostenmodell mit einem Optimierungsalgorithmus.

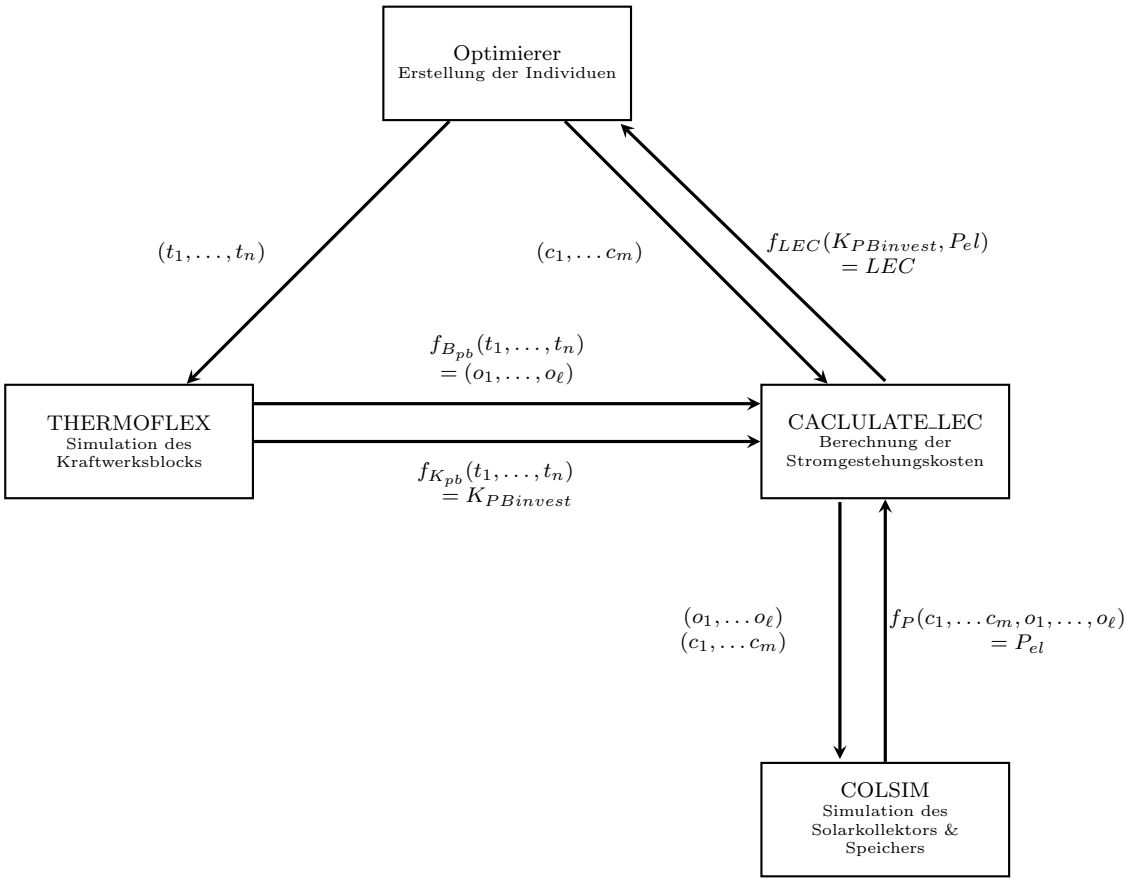


Abbildung 3.2: Kopplung der Komponenten Kraftwerksblock, Solarkollektor & Speicher und Kostenmodell-Programm mit einem Optimierungsalgorithmus. Das Kostenmodell-Programm führt automatisch Simulationen für Solarkollektor & Speicher aus und benutzt das Ergebnis zur Kostenberechnung.

Realisierung als Client-Server System mit zentralem Datenbankserver

Anhand der Abbildung 3.2 lässt sich erkennen, dass Daten zwischen den einzelnen Programmmodulen verschickt werden. Das Konzept der Simulationssoftware muss folglich den Austausch von Daten zwischen den einzelnen Programmmodulen unterstützen. Aus diesem Grund wurde hierfür im Rahmen dieser Arbeit ein Client-Server-System entwickelt, siehe Abbildung 3.3.

Ein Server ist ein Programm, das einen Dienst (Service) anbietet. Im Rahmen des Client-Server-Konzepts kann ein anderes Programm, der Client, diesen Dienst nutzen. Der Server ist in Bereitschaft, um jederzeit auf die Kontaktaufnahme eines Clients reagieren zu können. Im Unterschied zum Client, der aktiv einen Dienst anfordert, verhält sich der Server passiv und wartet auf Anforderungen. Durch ein Protokoll werden die Regeln der Kommunikation festgelegt.

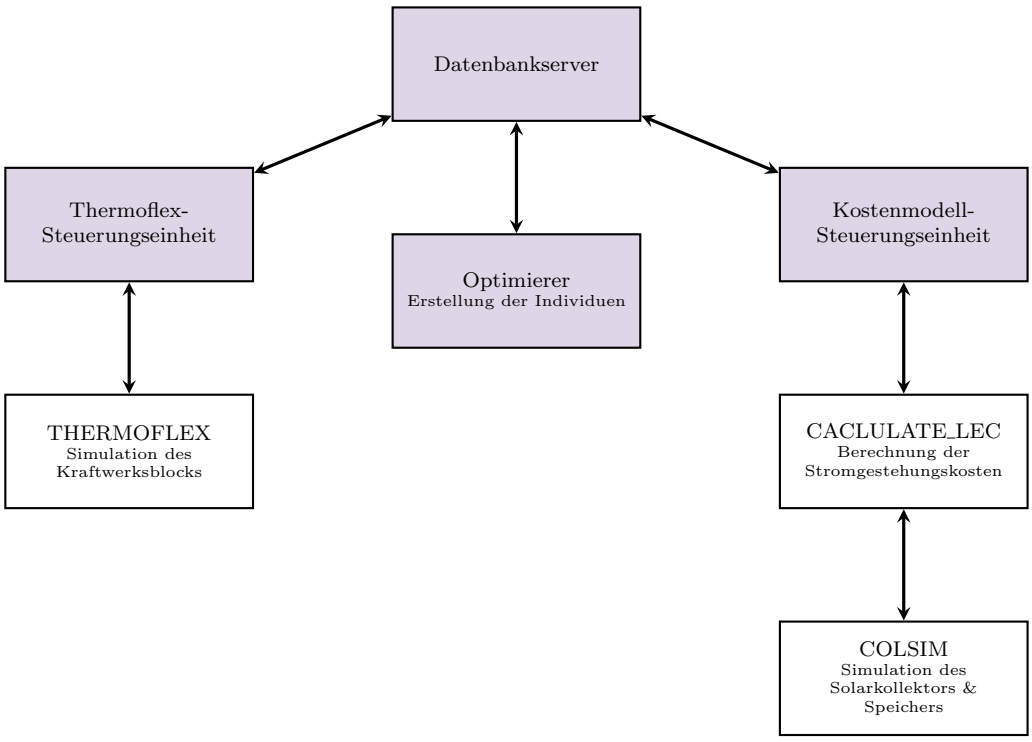


Abbildung 3.3: Client-Server-System mit einem zentralen Datenbankserver und den drei Steuerungseinheiten für Thermoflex, das Kostenmodell und dem genetischen Algorithmus als Clients. Der Server und die drei Clientprogramme (grau unterlegt) wurden im Rahmen der vorliegenden Arbeit erstellt.

Das Client-Server-System bildet eine Netzwerkstruktur, bestehend aus der zentralen Datenbank als Server-Komponente, Steuerungseinheiten für Thermoflex und das Kostenmodell sowie dem Optimierer als Client-Komponenten. Die Steuerungseinheiten steuern ihr jeweiliges Programm. Alle Clients können mit dem Datenbankserver kommunizieren. Dazu verfügen sie über eine Schnittstelle, um über das Netzwerk auf Ressourcen des Datenbankservers zu zu greifen.

Durch Senden von SQL[7]-Befehlen an den Server kann ein Client Daten aus der Datenbank auslesen oder aktualisieren. Der Server führt den SQL-Befehl aus und liefert das Ergebnis (Daten oder Fehlermeldung) als Antwort an den Client zurück. Mit dem Client-Server System sind die einzelnen Programmmodule autark und stehen nur mit dem Datenbankserver im direkten Kontakt. In den folgenden drei Abschnitten werden die einzelnen Clients vorgestellt und ihre Eigenschaften erläutert.

3.2 Optimierer

Die Optimiereinheit soll die Stromgestehungskosten eines solarthermischen Kraftwerks durch optimale Belegung der Kraftwerksparameter $(t_1, \ldots, t_n, c_1, \ldots, c_m) \in \mathbb{R}^{n+m}$, die so genannten Optimierungsvariablen, minimieren. Dabei umfassen die Kraftwerksparameter sowohl Kraftwerksblockparameter $(t_1, \ldots, t_n)$ als auch Solarkollektorparameter $(c_1, \ldots, c_m)$.

Die Zielfunktion des Optimierungsalgorithmus gibt in Abhängigkeit von den Optimierungsvariablen die Stromgestehungskosten an. Dabei hängt der Zielfunktionswert von den Simulationsergebnissen von Thermoflex, ColSim und dem Kostenmodell ab (siehe Abbildung 3.1). Die Hauptaufgabe des Optimierers besteht darin, zu den Optimierungsvariablen geeignete Belegungen zu finden, sog. Individuen.

Aufgrund der großen Anzahl der zu optimierenden Variablen stellt es eine besondere Herausforderung dar, ein Optimierungsverfahren zu implementieren, welches sowohl zeiteffizient ist, als auch hinreichend genaue Ergebnisse liefert. Wie bereits erwähnt, wurde in der Diplomarbeit von Gutjahr [9] zur Optimierung des Kraftwerksblocks erfolgreich ein genetischer Algorithmus verwendet. Aufbauend auf diesen Erfahrungen

[7]SQL - Structured Query Language. Datenbanksprache zur Definition, Abfrage und Manipulation von Daten in relationalen Datenbanken.

wird im Folgenden derselbe genetische Algorithmus verwendet, um ein solarthermisches Kraftwerk zu optimieren. Zunächst wird die Funktionsweise eines genetischen Algorithmus vorgestellt, bevor auf die speziellen Anforderungen an den Optimierer eingegangen wird.

3.2.1 Funktionsweise eines genetischen Algorithmus

Genetische Algorithmen sind heuristische Optimierungsverfahren die auf De Jong [4] und Holland [13] zurückgehen. Die Grundidee besteht darin, ähnlich wie bei der biologischen Evolution, eine Menge (Population) von Lösungskandidaten (Individuen) zufällig zu erzeugen und diejenigen auszuwählen, die einem bestimmten Gütekriterium am besten entsprechen (Selektion). Deren Eigenschaften (Parameterwerte) werden dann miteinander kombiniert (Rekombination) und teilweise verändert (Mutation), um eine neue Population von Lösungskandidaten (eine neue Generation) zu erzeugen. Auf diese wird wiederum iterativ die Selektion, Rekombination und Mutation angewandt, bis ein Stopkriterium erfüllt ist. Ein Stopkriterium ist beispielsweise das Erreichen einer maximalen Anzahl an Generationen. Im Folgenden werden die Operatoren gemäß Wall [26] beschrieben.

Bei den genetischen Algorithmen gibt es verschiedene Verfahren, die den Evolutionsprozess steuern. Diese unterscheiden sich vor allem dadurch, ob sie zulassen, Individuen einer älteren Generation in eine neue Generation zu übernehmen. Wall [26] unterscheidet grundsätzlich zwischen folgenden Ausprägungen:

- Der *Simple*-Algorithmus lässt alle Individuen der Vorgängergeneration austauschen. Nur das beste Individuum wird mit in die nächste Generation übernommen.
- Beim *Steady State*-Algorithmus wird die neue Generation gebildet, indem die schlechtesten Individuen der alten Generation durch neue Individuen ersetzt werden. Der Anteil der Ersetzungen an einer Generation wird vorher festgelegt.
- Der *Incremental*-Algorithmus wählt beim Übergang der Generationen nur ein oder zwei Individuen, die so genannten Eltern aus. Alle restlichen Individuen der neuen Generation werden aus diesem Elternpaar abgeleitet. Die Auswahl der

Elternpaare passiert wahlweise nach dem Zufallsprinzip oder nach der Güte der Individuen.

- Der *Deme*-Algorithmus bildet mehrere parallele Populationen, die einen Steady State-Algorithmus verwenden. Dabei können sich die Populationen untereinander austauschen, indem sie Wanderungen zwischen ihnen zulassen.

Evaluation

Für jedes Individuum der aktuellen Generation wird anhand einer Zielfunktion ein Wert, auch Fitnesswert genannt, bestimmt, der die Güte des Individuums angibt. Die Wahrscheinlichkeit, dass dieses Individuum in der nächsten Generation wieder vertreten ist, hängt von diesem Wert ab. Je nachdem wie gut seine Fitness ist, wird das Individuum überleben und sich fortpflanzen können. Der Wert ist also ein Maß für die Fortpflanzungswahrscheinlichkeit.

Selektion

Bei der Erstellung der neuen Generation werden nach einem Zufallsprinzip Individuen aus der aktuellen Generation ausgewählt. In der natürlichen Evolution entscheidet der Selektionsprozess über das Überleben und die Fortpflanzung eines Individuums. Die Individuen mit der besten Fitness haben dabei auch die besten Chancen zu überleben und sich fort zu pflanzen. Aber auch Lebewesen mit schlechterer Fitness können ausgewählt werden.

Als Vorbereitung auf eine Selektion werden alle Individuen $i \in P$ einer Population P gemäß ihrer Fitness mit einem Wert $w_i \in I := (0, 1]$ gewichtet, so dass stets $\sum_{i \in P} w_i = 1$ gilt. Für die Gewichtung gibt es verschiedene Strategien:

- Die *Uniform Random*-Selektion gewichtet jedes Individuum mit dem gleichen Wert, ohne Rücksicht auf die unterschiedlichen Fitnesswerte zu nehmen.

- Bei der *Roulette Wheel*-Selektion werden die Individuen gemäß ihres Fitnesswertes im Vergleich zu der Gesamtfitness der Population gewichtet. Somit erhalten bessere Individuen eine höhere Wahrscheinlichkeit ausgewählt zu werden als schlechtere Individuen.

- Die *Tournament*-Selektion ermittelt zwei Individuen nach der Roulette-Wheel Selektion und wählt anschließend das mit der besten Fitness aus.

Gemäß der Gewichtung eines Individuums i wird diesem eine Teilmenge $I_i \subseteq I$ zugeteilt, wobei für alle Individuen $i \in P$ gilt

$$I_i := (a_i, b_i] \text{ mit } a_i = \sum_{j=1}^{i-1} w_j \text{ und } b_i = \sum_{j=1}^{i} w_j,$$

so dass die Intervallgröße $b_i - a_i$ dem Gewicht w_i entspricht. Diese Teilmengen sind paarweise disjunkt und ihre Vereinigung überdeckt vollständig die Menge I. Somit ist sichergestellt, dass jeder Wert $x \in I$ eindeutig einem Teilintervall zugeordnet werden kann.

Anschließend findet die Selektion statt, bei der sukzessiv eine gleichverteilte Zufallszahl $z \in I$ generiert wird. Liegt $z \in I_i \subseteq I$, so wird Individuum i ausgewählt. Die Wahrscheinlichkeit, dass ein Individuum i ausgewählt wird, hängt also von seiner Gewichtung w_i ab.

Eine andere Herangehensweise bei der Auswahl der Individuen bietet die sog. *Rank*-Selektion, welche die Individuen ihrem Zielfunktionswert nach sortiert. Die Individuen werden der erstellten Reihe nach ausgewählt.

Rekombination

Bei der Erstellung einer neuen Generation werden Individuen aus der alten Generation selektiert. Diese werden aber nicht unbedingt unverändert übernommen, sondern es wird zuvor mit einer Zufallsvariable überprüft, ob eine Rekombination zwischen den selektierten Individuen stattfinden soll. Dazu wird eine Zufallszahl generiert und mit einem zuvor festgelegten Schwellenwert (Rekombinationswahrscheinlichkeit) verglichen. Ist der Zufallswert niedriger als der Schwellenwert, so findet eine Rekombination statt.

Die Rekombination entspricht der geschlechtlichen Fortpflanzung der Lebewesen in der Natur. Zwei Eltern tauschen Teile ihrer Informationsketten nach einem bestimmten Schema aus oder mischen diese und erzeugen dadurch zwei Nachkommen. Für den genetischen Algorithmus bedeutet dies, dass die Variablenbelegungen der ausgewählten Individuen gemischt werden und daraus neue Individuen erzeugt werden. Zu

nennen seien hier die folgenden vier Ausprägungen, die in Abbildung 3.4 beispielhaft vorgestellt werden:

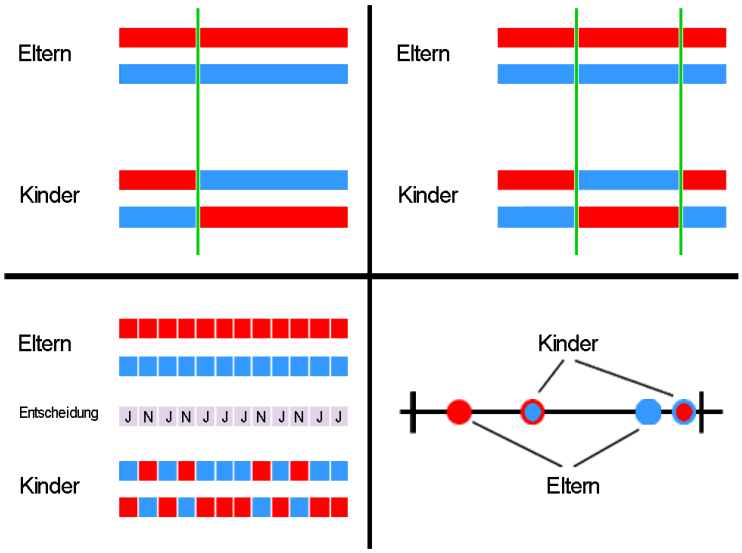


Abbildung 3.4: Beispiele für die vorgestellten Rekombinationsoperatoren genetischer Algorithmen. Oben sind die Einpunkt- und Zweipunkt-Rekombinationen beispielhaft dargestellt. Das Beispiel unten links stellt den Uniform-Rekombinationsoperator dar und unten rechts den Blend-Rekombinationsoperator.

- Der *Einpunkt*-Rekombinationsoperator bestimmt mittels Zufallszahl einen Punkt in den Variablenketten der Eltern-Individuen. Für die Nachkommen werden bis zu dem Punkt die Variablenbelegungen der Eltern behalten und ab dem Punkt die Variablenbelegungen der Eltern getauscht (vgl. Abbildung 3.4 links oben).

- Der *Zweipunkt*-Rekombinationsoperator bestimmt mittels Zufallszahl zwei Punkte in den Variablenketten der Eltern-Individuen. Für die Nachkommen werden bis vor dem ersten und ab dem zweiten Punkt die Variablenbelegung der Eltern beibehalten. Nur die Variablenbelegungen zwischen den Punkten werden ausgetauscht (vgl. Abbildung 3.4 rechts oben).

- Der *Uniform*-Rekombinationsoperator erzeugt Nachkommen, indem mittels idealisierten Münzwurf komponentenweise entschieden wird, ob die Variablenbelegungen der Eltern getauscht werden (vgl. Abbildung 3.4 links unten).

- Der *Blend*-Rekombinationsoperator erzeugt per Zufall Nachkommen, deren Belegungen innerhalb eines Definitionsbereichs liegen. Für jede Variable bilden die

Belegungen der Eltern-Individuen Grenzen eines Intervalls. Der Definitionsbereich der Nachkommen wird um einen festen Faktor (z.B. 0.5) über dieses Eltern-Intervall ausgedehnt (vgl. Abbildung 3.4 rechts unten). Bei der Ausdehnung wird der Definitionsbereich der Variable beachtet.

Mutation

Nachdem einige Individuen selektiert und eventuell rekombiniert wurden, werden einzelne Variablenbelegungen der Individuen noch verändert. Diesen Vorgang der zufälligen Veränderung der Individuen nennt man Mutation.

In der natürlichen wie in der simulierten Evolution dient die Mutation der Erzeugung einer gewissen Diversität in der Population. Durch die Mutation der Individuen kann der Suchprozess in noch nicht untersuchte Bereiche gelenkt werden.

Für jedes Individuum und jede einzelne Variablenbelegungen wird mit einem Zufallswert überprüft, ob eine Mutation für diese Variablenbelegung stattfinden soll. Ist dieser Wert kleiner als ein zuvor festgelegter Schwellwert (Mutationswahrscheinlichkeit), so findet eine Mutation statt. Die ursprüngliche Variablenbelegung wird durch einen neuen, mutierten Wert ersetzt. Der mutierte Wert kann durch die folgenden Mutationsoperatoren erstellt werden:

- Der *Flip*-Mutationsoperator bestimmt den mutierten Wert durch eine Gleichverteilung über den festgelegten Definitionsbereich der Variablen.
- Der *Gauss*-Mutationsoperator wählt den mutierten Wert gemäß einer um den aktuellen Wert der Variablenbelegung gelegte standardnormalverteilte Gauss-Glocke. Dabei önnen die Größe des Intervalls, über das die Verteilung erfolgen soll, und die Standardabweichung σ individuell an das vorliegende Problem angepasst werden.

3.2.2 Implementierung des Optimierers

Der Optimierer wurde im Rahmen dieser Arbeit entwickelt. Zur Implementierung des genetischen Algorithmus wurde die von Wall [26] am MIT geschriebene C++ Klassenbibliothek GAlib verwendet, deren Funktionsweise in Abschnitt 3.2.1 beschrieben wurde. Die Bibliothek stellt Werkzeuge zur Verwendung genetischer Algorithmen zur

Verfügung, um Optimierungen in C++-Programmen vorzunehmen. Ein großer Vorteil der GAlib ist der offen zugängliche Quellcode, so dass die Funktionsweisen der verwendeten Methoden nachvollzogen und auch erweitert werden können. So war es möglich, einige Änderungen im Code vorzunehmen, um den genetischen Algorithmus an die vorliegenden Gegebenheiten anzupassen.

Um GAlib in den Optimierer zu integrieren, wurde im Rahmen dieser Arbeit letzterer ebenfalls in der Programmiersprache C++ geschrieben. Für die Kommunikation mit dem Datenbankserver wird die standardisierte Datenbankschnittstelle ODBC[8] verwendet. Alle vom genetischen Algorithmus erstellten Individuen werden über diese Verbindung an den Datenbankserver übermittelt und die extern ermittelten Stromgestehungskosten ausgelesen (siehe Abbildung 3.5). Die Stromgestehungskosten entsprechen dem Fitnesswert.

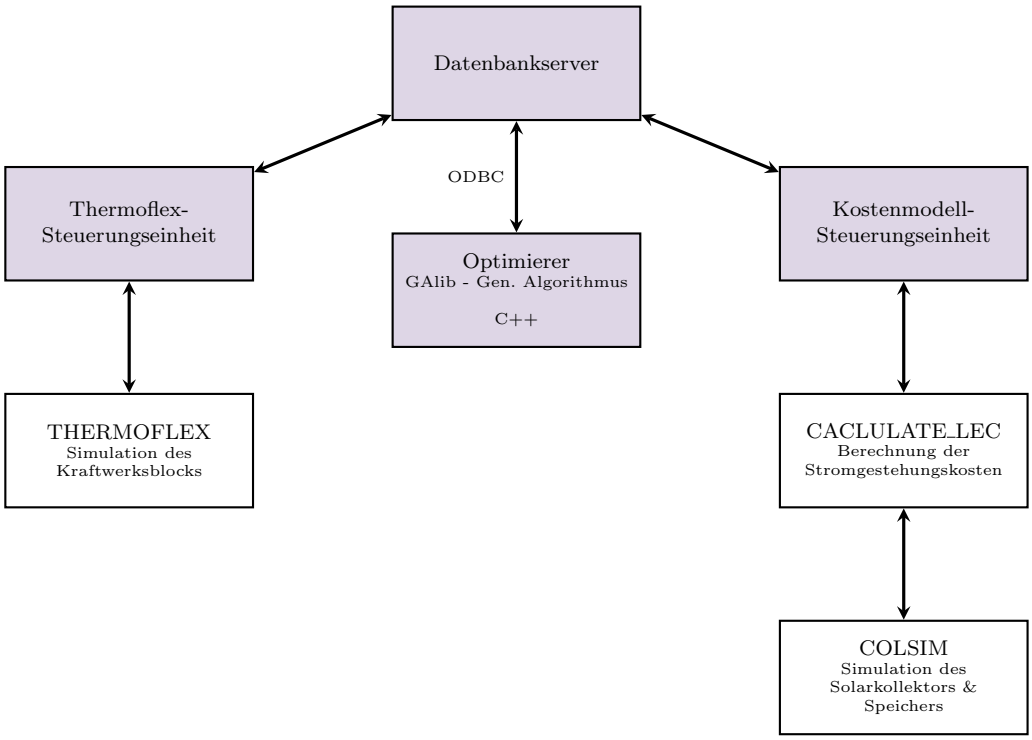


Abbildung 3.5: Simulationssoftware: Der optimierer hat GAlib integriert und kommuniziert mittels ODBC mit dem Datenbankserver.

GAlib bietet eine Vielzahl von Einstellungen. Zum einen müssen die Optimierungs-

[8]ODBC - Open Database Connectivity

variablen und ihre Definitionsbereiche bestimmt werden und zum anderen kann der genetische Algorithmus durch die Einstellungen wie Mutationswahrscheinlichkeit oder Rekombinationsoperator konfiguriert werden. Um diese Einstellungen vornehmen zu können, verbindet sich der Optimierer bei Programmstart mit dem Datenbankserver und fragt dort seine Einstellungen und die Definitionsbereiche der Optimierungsvariablen ab. Die wichtigsten Einstellmöglichkeiten sind:

- Typ des genetischen Algorithmus (Simple, Steady State, Incremental, Deme)
- Optimierungsrichtung (Minimum, Maximum)
- Art der Selektion (Uniform Random, Roulette Wheel, Tournament)
- Art der Rekombination (Einpunkt, Zweipunkt, Uniform, Blend)
- Art der Mutation (Flip, Gauss)
- Populationsgröße aus der Domäne $\mathbb{N}$
- Rekombinationswahrscheinlichkeit aus dem Intervall $[0, 1] \subset \mathbb{R}$
- Mutationswahrscheinlichkeit aus dem Intervall $[0, 1] \subset \mathbb{R}$
- Algorithmus-Terminierungskriterien

3.2.3 Eigene Anpassungen zur Rechenzeitreduktion

Wie bereits erwähnt, ist der Quellcode der GAlib offen zugänglich. Dies ermöglicht es die GAlib zu modifizieren und zu erweitern, um den genetischen Algorithmus an das vorliegende Problem anzupassen. Im Folgenden werden die Erweiterungen erläutert.

Generationsweise Parallelisierung

Der genetische Algorithmus der GAlib ist so konzipiert, dass die Individuen einer Generation einzeln erstellt werden. Erst wenn die Fitness des Individuums vorliegt wird ein neues Individuum erstellt. In unserem Fall würde dies bedeuten, dass nach Erstellung eines Individuums zunächst Thermoflex eine Simulation ausführt (siehe Abschnitt 3.3) und anschließend ColSim mit den Simulationsergebnissen den Zielfunktionswert des Individuums bestimmt. Erst wenn dieser von ColSim bestimmte Wert vorliegt erstellt der genetische Algorithmus der GAlib ein neues Individuum. Zum Erstellen eines

Individuums der aktuellen Generation sind lediglich die Individuen mitsamt Zielfunktionswert der vorigen Generationen wichtig. Der Zielfunktionswert von Individuen der gleichen Generation ist nicht relevant. Daher wurde der genetische Algorithmus der GAlib dahingehend abgeändert, dass zu Beginn einer neuen Generation alle Individuen erstellt und an den Datenbankserver gesendet werden. Erst dann wartet der genetische Algorithmus auf die Zielfunktionswerte der einzelnen Individuen. Mit dieser Änderung reduziert sich die Rechenzeit, da Thermoflex und ColSim parallel rechnen können. Hat beispielsweise Thermoflex zu Individuum i eine Simulation ausgeführt, so kann es sofort das nächste Individuum $i+1$ bearbeiten. Somit muss Thermoflex nun nicht mehr warten bis ColSim das Individuum i evaluiert hat und der genetische Algorithmus ein neues Individuum $i+1$ erstellt hat. Dies gilt aber nur solange das Ende einer Generation nicht erreicht ist. Der Effekt dieser Änderung ist eine generationsweise Parallelisierung der Berechnungen, wodurch Rechenzeit gespart wird.

Überprüfung von kraftwerksblockgleichen Individuen

Beim Erstellen neuer Individuen überprüft GAlib, ob die Individuen schon in einer früheren Generation existieren, so dass der Fitnesswert übernommen werden kann und damit Rechenzeit gespart wird. In unserem Fall besteht ein Individuum $i = (t_1, \ldots, t_n, c_1, \ldots, c_m)$ aus Kraftwerksblock- und Solarkollektorvariablen. Da die Simulation des Kraftwerksblocks mit Thermoflex der Rechenzeit bestimmende Faktor für die Berechnung eines Individuums ist, ist es beim Erstellen eines neuen Individuums von Interesse, auch zu überprüfen ob in einer früheren Generation ein Individuum mit gleichen Kraftwerksblockvariablen existiert. Ist dies der Fall so können die Thermoflex-Ergebnisse $f_{K_{pb}}(t_1, \ldots, t_n)$, $f_{pb}(t_1, \ldots, t_n)$ des älteren Individuums übernommen werden, so dass eine erneute Simulation in Thermoflex nicht mehr nötig ist. Um die Suche in der Datenbank zu beschleunigen, wird jedem Individuum eine Zahl zugeordnet, welche der Summe seiner Variablenwerte entspricht. Mit Vergleich dieser Summen kann schnell eine Vorauswahl von möglichen identischen Kraftwerksindividuen getroffen werden.

Setzen von Start-Individuen

Durch geeignete Wahl der Start-Individuen kann die Effizienz des genetischen Algorithmus gesteigert und damit die Rechenzeit reduziert werden. Beim Erstellen der

Initialisierungs-Generation benutzt die GAlib einen Zufallsgenerator, um für die Individuen eine Variablenbelegung innerhalb ihres Definitionsbereichs zu generieren. Durch Erfahrung bei der Auslegung von Kraftwerken können für eine Simulation schon vorweg Individuen genannt werden, die meist recht gute Ergebnisse erzielen. Daher wurde der genetische Algorithmus der GAlib noch dahingehend erweitert, dass Start-Individuen angegeben werden können. Werden gute Start-Individuen gesetzt, so erreicht der genetische Algorithmus mit hoher Wahrscheinlichkeit das Optimum früher, als wenn er mit Zufallszahlen initialisiert worden wäre. Der Effekt ist also auch hier die Einsparung von Rechenzeit. Das Setzen der Start-Individuen wird auch, wie die generellen Einstellungen des genetischen Algorithmus, als flexible Einstellmöglichkeit über den Datenbankserver geregelt.

Zusammenfassung Optimierer

Mit dem im Rahmen dieser Arbeit implementierten Optimierer liegt eine Programmeinheit im Client-Server System vor, die auf Grundlage eines genetischen Algorithmus Individuen erstellt, diese an den Datenbankserver sendet und gemäß eines durch die anderen Clients bestimmten Wertes, evaluiert. Eigenschaften des Optimierers lassen sich folgendermaßen zusammenfassend angeben:

- Client-Programmeinheit, die mittels ODBC mit dem Datenbankserver kommuniziert
- Konfiguration des genetischen Algorithmus bei Programmstart durch Abfragen der Einstellungen beim Datenbankserver
- Individuen werden gemäß einem genetischen Algorithmus erstellt
- Rechenzeit wurde durch folgende Merkmale reduziert:
  - Berechnungen sind innerhalb einer Generation parallelisiert
  - Überprüfung von kraftwerksblockgleichen Individuen
  - Setzen von Start-Individuen

3.3 Thermoflex-Steuerungseinheit

Zur Simulation der zu untersuchenden thermodynamischen Prozesse wird das kommerzielle Programm Thermoflex des Unternehmens Thermoflow Inc. verwendet. Thermoflex ist ein Simulationsprogramm für Kraftwerkskreisläufe, welches der technischen Abbildung von konventionellen Kraftwerksprozessen dient.

Laut einer vom VGB[9] in Auftrag gegebenen Studie zu dem Thema „Vergleich von Software zur thermodynamischen Prozessrechnung“ kommt Karl [15] zu dem Ergebnis, dass Thermoflex in allen Anwendungsgebieten auf den vordersten Plätzen vertreten ist.

Mit Thermoflex ist es möglich, Kraftwerkseigenschaften festzulegen was man als *Kraftwerksauslegung* oder Kraftwerksdesign bezeichnet, bei der alle Komponenten wie Turbinengröße und Pumpentyp festgelegt und dimensioniert werden. In Abbildung 3.6 ist ein mit Thermoflex erstelltes Schaltbild (vgl. [20]) eines 50 MW Dampfkreisprozesses mit sieben Speisewasservorwärmern und einem Zwischenüberhitzer dargestellt.

Ist ein Kraftwerk ausgelegt, so lässt sich dessen Verhalten unter verschieden Bedingungen untersuchen. Diese Untersuchungen nennen sich *Betriebszustandsberechnungen* oder off-design Fälle. Typische Betriebszustandsberechnungen sind beispielsweise die Variation der Umgebungstemperatur, die durch den Wechsel von Tag und Nacht natürlich ist und thermodynamisch auf den Wirkungsgrad des Prozesses einen enormen Einfluss hat. Weitere gängige Betriebszustandsberechnungen sind die sog. *Teillastfälle*. Dabei wird untersucht wie sich die erzeugte Strommenge bei Veränderung der Wärmeenergiezufuhr verhält, was bei einem Solarkraftwerk unterschiedlichen Solarstrahlungsbedingungen entspricht.

Mit Steigung der Umgebungstemperatur steigt auch die Kondensationstemperatur bzw. der Kondensationsdruck im Kondensator (vgl. Abbildung 2.3 Prozessschritt 2 → 3). In Abbildung 3.9 ist die Abhängigkeit der Kraftwerksleistung von der Umgebungstemperatur für den 50 Megawatt Clausius-Rankine Kraftwerksprozess dargestellt. Es lässt sich erkennen, dass mit höheren Umgebungstemperaturen die Kraftwerksleistung und damit auch der Wirkungsgrad sinkt. Der Einfluss der Umgebungstemperatur und der Teillast auf die elektrische Kraftwerksleistung wird *Umgebungstemperatur-Teillast*

[9] VGB - Technische Vereinigung deutscher Großkraftwerksbetreiber e.V.

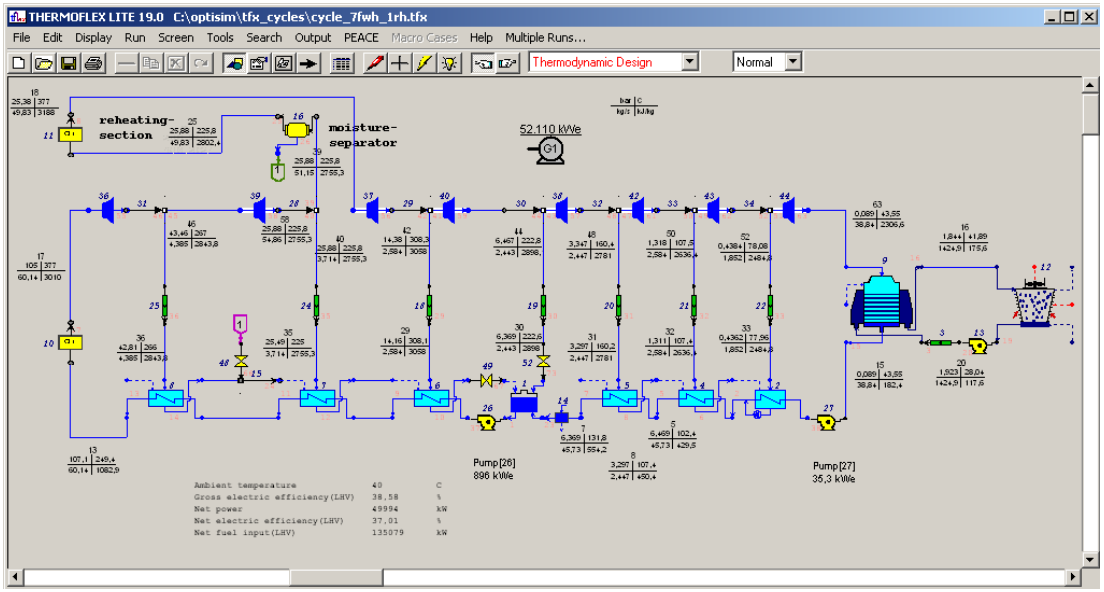


Abbildung 3.6: Grafische Benutzeroberfläche von Thermoflex, hier ein von Morin [20] erstellter 50 Megawatt Dampfkreisprozess mit sieben Speisewasservorwärmern und einem Zwischenüberhitzer: Im linken Bereich ist das Solarfeld als zwei Wärmequellen durch gelbe Rechtecke dargestellt. Im oberen Bereich lassen sich acht Turbinenstufen erkennen die durch blaue Trapeze abgebildet werden. Auf der rechten Seite sind der Kondensator und Kühlturm mit einer Pumpe im Kühlwasserkreis (gelb) zu sehen. Im unteren Bereich sind sieben Speisewasservorwärmer zu finden wobei der vierte Vorwärmer auch Speisewasserbehälter und Entgaser ist (türkise Rechtecke bzw. dunkelblaues Rechteck). Die Linien zwischen den Komponenten entsprechen den Rohrleitungen des Mediums.

Kennfeld genannt. In Untersuchungen zu mehreren Kraftwerksprozessen im Rahmen dieser Arbeit stellte sich heraus, dass die Leistungskurven einen recht ähnlichen Verlauf zeichnen. In Abbildung 3.7 ist für einen realistischen 50 Megawatt Kraftwerksprozess mit 5 Vorwärmern sowie für einen Basisprozess ohne Vorwärmer und ohne Zwischenüberhitzung (Clausius-Rankine Prozess) die Kraftwerksleistung in Abhängigkeit von der Umgebungstemperatur für verschiedenen Teillasten dargestellt. Die Umgebungstemperatur wurde in 2°C Schritten von 0 bis 50 °C variiert und die Teillast in 10% Schritten von 30 bis 100%. Auffällig ist nicht nur der ähnliche Kurvenverlauf der Lastkurven, sondern auch die nahezu konstanten Abstände zwischen den Lastkurven bezüglich einer festen Temperatur.

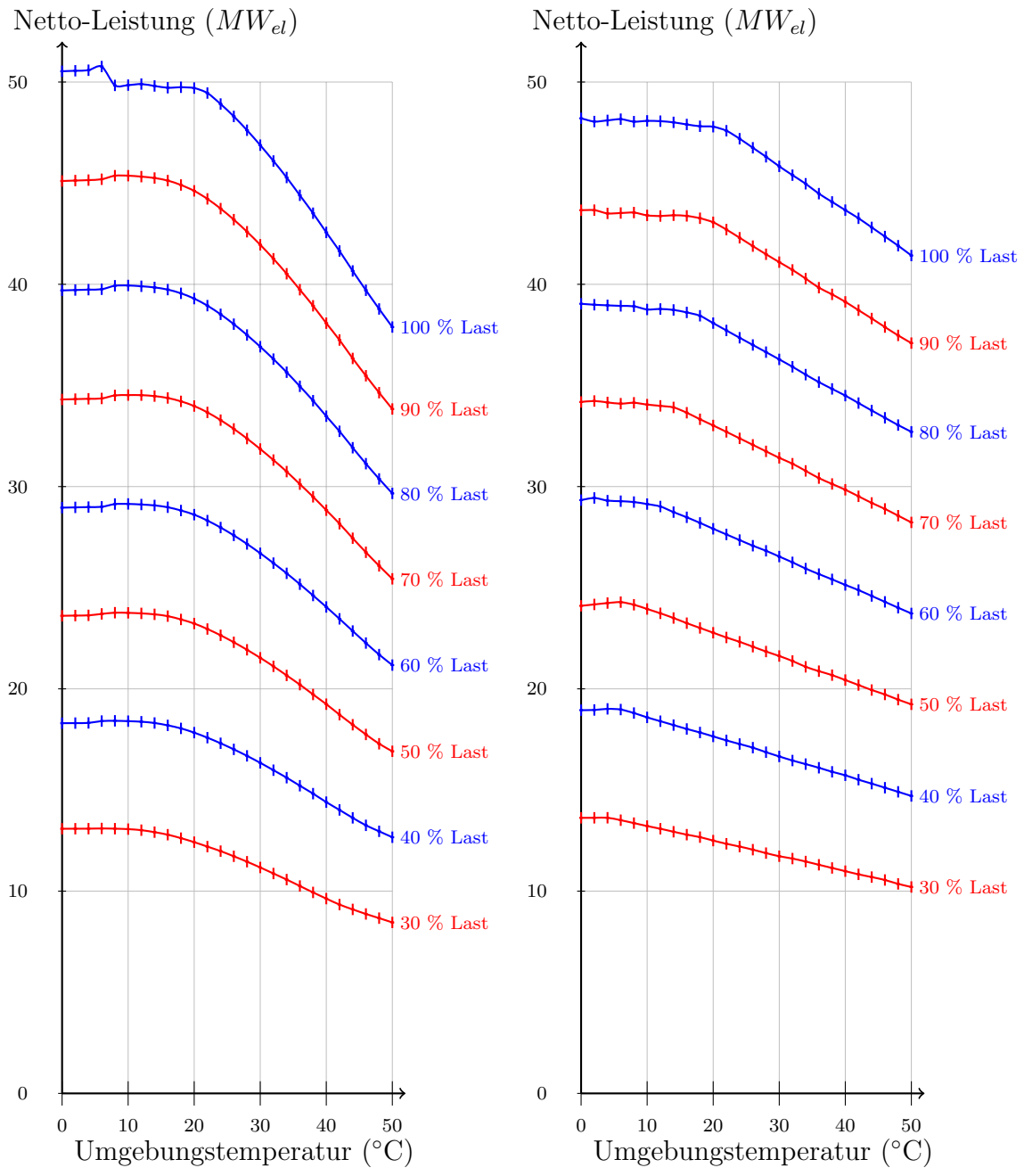


Abbildung 3.7: Umgebungstemperatur-Teillast Kennfeld eines 50 Megawatt Kraftwerksprozesses mit 5 Vorwärmern (links) und ohne Vorwärmer (rechts).

3.3.1 Implementierung der Thermoflex-Steuerungseinheit

Zu Thermoflex existiert eine VBA[10]-Schnittstelle namens *Elink*, die die Möglichkeit bietet, in Tabellenform festgelegte design oder off-design Fälle berechnen zu lassen. *Elink* kann Parameter von Kraftwerksprozessen verändern und Simulationen ausführen. Um diese Schnittstelle zu nutzen, wurde die im Rahmen dieser Arbeit entwickelte Thermoflex-Steuerungseinheit ebenfalls in der Programmiersprache VBA geschrieben.

Abhängig davon, ob ein unter Thermoflex erstellter Kraftwerksprozess im design- oder off-design Modus vorliegt, können über die *Elink*-Schnittstelle design-Berechnungen (Auslegung) oder off-design-Berechnungen (Betriebszustandsberechnungen) durchgeführt werden. Es gilt, *Elink* so zu verwenden, dass für ein Individuum i mit den Parametern $t_1, \ldots, t_n$ ein Kraftwerksprozess ausgelegt und dessen Investitionskosten $f_{K_{pb}}(t_1, \ldots, t_n) = K_{PBinvest}$ berechnet werden und anschließend durch Betriebszustandsberechnungen sein Verhalten $f_{B_{pb}}(t_1, \ldots, t_n) = (o_1, \ldots, o_\ell)$ bestimmt wird (siehe Abbildung 3.2). Eine Umstellung vom design-Modus in den off-design Modus ist über die *Elink*-Schnittstelle nicht möglich sondern muss von Hand über die Benutzeroberfläche von Thermoflex ausgeführt werden. Um dies zu automatisieren, wurde im Rahmen dieser Arbeit mit Hilfe von AutoIt[11] das Skript *changemode.exe* geschrieben, welches unter Windows Tastenanschläge und Mausklicks simuliert und somit in der Benutzeroberfläche von Thermoflex eine Modusänderung vornimmt.

Für die Kommunikation mit dem Datenbankserver wird die Datenbankschnittstelle ADO[12] verwendet. Alle von der Thermoflex-Steuerungseinheit benötigten Daten werden über diese Verbindung vom Datenbankserver angefordert, sowie alle simulierten Ergebnisse an den Datenbankserver übermittelt (siehe Abbildung 3.8).

3.3.2 Spezielle Eigenschaften der Thermoflex-Steuerungseinheit

Die Thermoflex-Steuerungseinheit wurde mit einigen Funktionen ausgestattet, um den speziellen Anforderungen für eine solarthermische Kraftwerksoptimierung gerecht zu werden.

[10] VBA - Visual Basic for Applications ist eine zu den Microsoft-Office-Programmen gehörende Skriptsprache.

[11] AutoIt ist eine Software zur Erstellung von Makros, mit denen Abläufe unter Microsoft Windows automatisiert werden können.

[12] ADO - ActiveX Data Objects

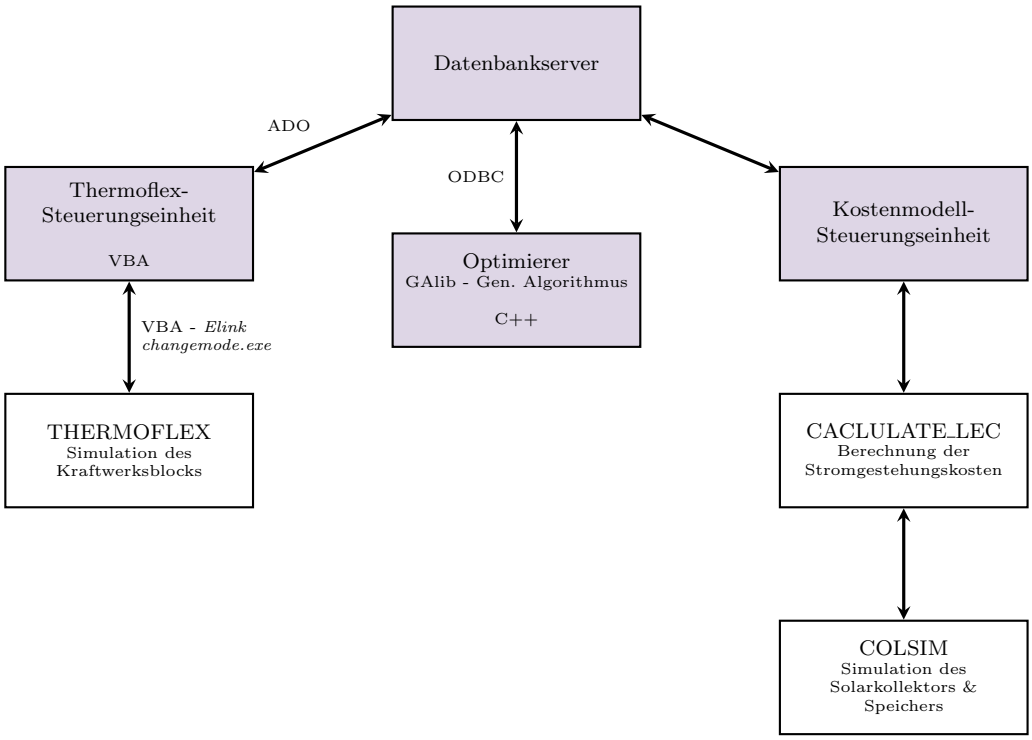


Abbildung 3.8: Simulationssoftware: Die Thermoflex-Steuerungseinheit steuert über VBA-*Elink* und *changemode.exe* das Programm Thermoflex und kommuniziert mittels ADO mit dem Datenbankserver.

Parametrisierung der Kraftwerksprozesse

Die Thermoflex-Steuerungseinheit wurde nicht nur dahingehend ausgelegt, einen Kraftwerksprozess zu steuern, sondern sie ermöglicht auch, die Steuerung mehrerer Prozesse parallel zu führen. Dazu wurden die Kraftwerksprozesse parametrisiert, also als eine Optimierungsvariable des genetischen Algorithmus behandelt. Eine der vom Optimierer übergebenen Parameter aus $t_1, \ldots, t_n$ gibt die Kraftwerksprozessnummer an. Mit dieser Parametrisierung der Kraftwerksprozesse wird ein Vergleich von Prozessen unter gleichen Rahmenbedingungen ermöglicht.

Funktionale Abhängigkeiten und Funktionsparser

Für einen Kraftwerksprozess gibt es eine Reihe von *Konstanten* und *Optimierungsvariablen*. Es ist von Interesse, auch einige Konstanten in Abhängigkeit von den Optimierungsvariablen zu verändern, so dass ein dritter Parametertyp eingeführt wird, nämlich die *funktionalen Variablen*. Dazu wurde ein Funktionsparser geschrieben, der zu einer

Belegung der Optimierungsvariablen und einer vom Benutzer definierbaren Funktion den Funktionswert berechnet. Der Parser wurde neben den Grundrechenarten noch mit weiteren gängigen mathematischen und thermodynamischen Funktionen ausgestattet, wie beispielsweise der Exponentialfunktion oder der Umrechnungsfunktion $p_{sat}(T)$, die den Verdampfungs-Druck von Wasser bei einer gegebenen Temperatur T bestimmt[13]. Es erscheint sinnvoll, dass der Funktionsparser direkt nach Erstellung der Individuen (im genetischen Algorithmus) eingesetzt wird, so dass die funktionalen Variablen in Abhängigkeit von den Optimierungsvariablen ermittelt werden. So können auch funktionale Abhängigkeiten von ColSim-Variablen aufgenommen werden. Somit wird der Optimierer um diese Funktionalität erweitert, der über den Datenbankserver auf die zentral abgelegten Funktionsvorschriften zugreifen kann und die berechneten Funktionswerte in der Datenbank speichert. Von Roeckerath [22] existiert bereits ein unter Java geschriebener Funktionsparser. Der frei zur Verfügung gestellte Quellcode wurde im Rahmen dieser Arbeit in C++ übersetzt, durch thermodynamische Funktionen erweitert und in den Optimierer integriert.

Prüfeinheit zur Rechenzeitreduktion

In den Optimierungsrechnungen müssen bestimmte technische Randbedingungen beachtet werden. Einige der vom genetischen Algorithmus erzeugten Individuen sind im design oder off-design technisch nicht realisierbar und dadurch ungültig. Um diese ungültigen Lösungen zu identifizieren werden während der Simulation die von Thermoflex gelieferten Ergebnisse von einer Prüfeinheit überprüft. Liegt ein ungültiges Individuum vor, so wird dieses mit einem hohen Wert bestraft. Folgende Zustände können zu einer Bestrafung führen:

1. Thermoflex-Fehlermeldung
   Wurde eine Variablenbelegung gewählt, die zu thermodynamischen Widersprüchen führt, so kann Thermoflex seine Massen-Bilanzgleichungsmatrix nicht auflösen und erzeugt eine Fehlermeldung. Ist dies der Fall, so wird das Individuum bestraft.

2. Starke Abänderung der vorgegebenen Eingangsgrößen
   Durch eine Thermoflex-interne Fehlerprüfung macht Thermoflex bei bestimm-

[13] Bei beispielweise $T = 100°\text{C}$ ist $p_{sat}(T) = 1.0$ bar.

ten Parameterkonstellationen Verbesserungsvorschläge und rechnet mit diesen Werten weiter. Die Prüfeinheit bestimmt die relative Abweichung und vergleicht diesen mit einem zuvor definierten, zulässigen Maximalwert. Wird der Maximalwert überschritten, so wird das Individuum bestraft.

3. Dampfgehaltrestriktion
   Beim Entspannen des Dampfes in der Turbine wird ein Teil des Wasserdampfes wieder flüssig, so dass sich Wassertröpchen bilden. Ab einer gewissen Dampfnässe belasten die Wassertröpfchen durch ihr „Aufprallen“ die Turbinenschaufeln extrem (Gefahr von Tropfenschlagerosion). Laut Angaben von Turbinenhersteller wird ein Mindestdampfgehalt von 87 % gefordert (maximal 13 % flüssig), vgl. [19].
   In den thermodynamischen Modellen von Thermoflex ist der Zustand am Turbinenausgang theoretisch überall im Nassdampfgebiet möglich. Wird ein zuvor definierter Mindestdampfgehalt unterschritten, so wird das Individuum als nicht zulässig angesehen und bestraft.

Die Prüfeinheit sorgt dafür, dass bei einer unzulässigen design oder off-design Berechnung die Simulation des Individuums sofort abgebrochen wird. Unzulässige Individuen werden speziell markiert, so dass im weiteren Verlauf der Simulation dieses Individuum nicht weiter beachtet wird und damit keine weitere Rechenzeit unnötig verbraucht wird. Der Zielfunktionswert unzulässiger Individuen wird schlecht gewählt (Strafwert), um im Optimierer die Wahrscheinlichkeit für die Selektion und Reproduktion für dieses Strafindividuum zu minimieren.

Unterstützung iterativer Kontrollschleifen

Thermoflex bietet die Möglichkeit, für eine Größe p_1 einen zulässigen Wertebereich zu definieren. Durch Angabe eines zweiten Parameters p_2 iteriert Thermoflex diesen so lange, bis p_1 in dem definierten Wertebereich liegt. Diese sog. Kontrollschleife kann dazu benutzt werden, im off-design die Dampfgehaltrestriktion einzuhalten. Dazu wird p_1 als Dampfgehalt der Turbine gewählt, dessen zulässiger Wertebereich den Mindestdampfgehalt als untere Grenze hat. Als p_2 muss ein geeigneter Betriebsparameter gewählt werden, dessen Veränderung einen Einfluss auf den Dampfgehalt hat. In der Thermoflex-Steuerungseinheit wurde eine Funktion implementiert, die auf Kon-

trollschleifen zugreifen und sie steuern kann. Optional kann gewählt werden, ob die Kontrollschleife benutzt werden soll. Nachteil der Kontrollschleife ist die durch die Iterationen vergrößerte Rechenzeit. Jedoch wird die Akzeptanz der Prüfeinheit damit vergrößert, so dass mehr zulässige Individuen vorhanden sind und dass das Betriebsverhalten realitätsnah abgebildet wird.

Flexible Konfiguration der Thermoflex-Steuerungseinheit

Die Thermoflex-Steuerungseinheit verfügt über eine Vielzahl an Einstellungen. Es müssen die Kraftwerksprozesse mit ihren Optimierungsvariablen, Funktionsvariablen und Funktionen bestimmt werden. Auch kann man die Anzahl der off-design Fälle setzen und jeden off-design Fall separat definieren. Um diese Einstellungen wahrnehmen zu können, verbindet sich die Thermoflex-Steuerungseinheit bei Programmstart mit dem Datenbankserver und fragt dort die eigene Konfiguration ab.

Zusammenfassung der Thermoflex-Steurungseinheit

Die Aufgabe der Thermoflex-Steuerungseinheit ist es, die Simulationen eines Kraftwerksprozesses zu organisieren und zu überwachen. Diese Programmeinheit lässt ein vom Datenbankserver übermitteltes Individuum unter Thermoflex simulieren und sendet die Ergebnisse an den Datenbankserver zurück. Die Eigenschaften der Thermoflex-Steuerungseinheit lassen sich folgendermaßen zusammenfassend angeben:

- Client-Programmeinheit, die mittels ADO mit dem Datenbankserver kommuniziert.
- Konfiguration der Thermoflex-Steuerungseinheit bei Programmstart durch Abfragen der Einstellungen beim Datenbankserver.
- Kraftwerksprozesse können im design und off-design Modus simuliert werden.
- Einführung von funktionalen Variablen und eines Funktionsparsers zur konfigurationsspezifischen Einstellung eines Kraftwerksblocks.
- Vergleich von verschiedenen Kraftwerksprozessen ist in einer integrierten Optimierung möglich.

- Implementierung einer Prüfeinheit, die ungültige Individuen erkennt, abstraft und entsprechend Simulationen zur Rechenzeitreduktion vorzeitig abbricht.
- Funktion, die Kontrollschleifen zum Einhalten der Betriebsbedingungen (hier Dampfgehaltrestriktion) unterstützt.

3.4 Kostenmodell-Steuerungseinheit

Die Optimierungsvariablen beeinflussen nicht nur die Energieproduktion des solarthermischen Kraftwerks, sondern auch die Kosten. Das Kostenmodell von Morin [20] erfasst den monetären Aufwand eines solarthermischen Kraftwerks. Es berücksichtigt zum einen die einmal anfallenden Investitionskosten zu Projektbeginn und zum anderen die laufenden Betriebskosten. Das von Morin geschriebene Programm calculate_LEC lässt mit ColSim den Jahresenergieertrag errechnen und liefert als Resultat die Stromgestehungskosten.

3.4.1 ColSim

ColSim ist ein von Wittwer [28] am Fraunhofer ISE entwickeltes und von Mertins [18] erweitertes Linux-Simulationsprogramm mit dem die energetische Simulation eines Solarkollektorfeldes mit thermischer Speicherung möglich ist. Unter Angabe eines Kennfeldes eines Kraftwerksblocks ermöglicht ColSim darüber hinaus die Berechnung des jährlichen Stromertrags eines solarthermischen Dampfkraftwerks.

Im Gegensatz zu konventionellen Kraftwerken ist bei den solarthermischen Kraftwerken der „Brennstoff“ nicht immer zu 100 % verfügbar. Durch Wolkenbildung oder einen tageszeitbedingten, niedrigen Einstrahlwinkel der Sonne, kann sich die verfügbare Sonneneinstrahlung reduzieren, wodurch weniger Energie zur Verfügung steht, um Wasserdampf und damit elektrische Energie zu erzeugen. Diese Situation entspricht den in Abschnitt 3.3 erläuterten Teillastfällen.

ColSim verfügt über einen Sonnenstandsalgorithmus und eine Einleseroutine von standortspezifischen Wetterdaten. Der Sonnenstandsalgorithmus berechnet unter Berücksichtigung von Sonnenstand und Wetterdaten die durch die Sonnenstrahlung erzeugte thermische Energiezufuhr in das Dampfkraftwerk. Um dazu die von dem Kraftwerksblock erzeugte elektrische Energiemenge zu bestimmen, muss ColSim das Verhalten

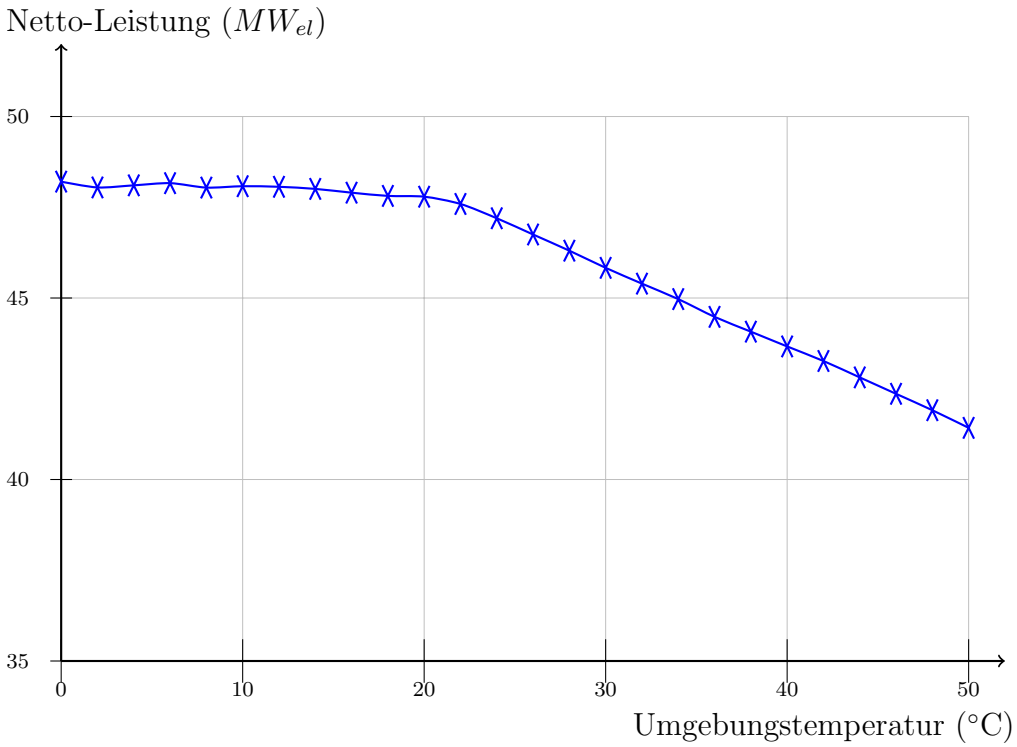


Abbildung 3.9: Abhängigkeit der Kraftwerksleistung von der Umgebungstemperatur für den 50 MW Clausius Rankine Kraftwerksprozess.

$o_1, \ldots, o_\ell$ des Kraftwerks bei Teillast und Umgebungstemperatur kennen. Die Simulation dieser Betriebszustände erfolgt in Thermoflex. Da die Berechnung eines Betriebszustandes recht zeitaufwendig ist (ca. 10 Sekunden auf einem quadcore PC), werden von Thermoflex nur einige Punkte des Umgebungstemperatur-Teillast Kennfeldes berechnet. Standardmäßig werden 16 Punkte bestimmt, für vier verschiedene Umgebungstemperaturen werden für vier verschiedene Lastfälle die erzeugten elektrischen Energiemengen bestimmt. Die restlichen Punkte werden von ColSim durch bilineare Interpolation zwischen den bekannten Betriebszuständen berechnet.

ColSim verfügt über Einstellungsparameter, deren Werte in der Konfigurationsdatei *sim.dek* stehen. Bei Programmstart von ColSim wird diese Datei zusammen mit der Kraftwerksblock-Kennfeld-Datei *powerblock.dat*[14] geladen. Die Eingabe zu ColSim bilden diese zwei Dateien, für die es gilt, sie bei Bedarf einzustellen.

Die Einstellungsparameter in *sim.dek* umfassen Standortdaten, Solarfeldparameter,

[14] *powerblock.dat* enthält die Thermoflex Ergebnisse $o_1, \ldots, o_\ell$.

optische Kollektorparameter und thermische Kollektorparameter. Die wichtigsten Parameter sind:

- Wetterdaten und Standortdaten
- Kollektortyp (Parabolrinnen, Fresnel)
- Kollektorreihenlänge bzw. Kollektorfläche
- Optische Effizienz des Kollektors (sonnenstandsabhängig)
- Solarfeld-Eintrittstemperatur

Einige der Einstellungsparameter werden zur Auslegungsoptimierung des Solarkollektors verwendet, vgl. [20]. Diese insgesamt fünf Optimierungsvariablen $c_1, \ldots, c_m$ lauten:

- Solarfeld-Spiegelfläche
- Speichervolumen
- Solarfeld-Austrittstemperatur
- Abstand der parallelen Reihen
- Solarfeld-Orientierung

3.4.2 Implementierung der Kostenmodell-Steuerungseinheit

Im Rahmen dieser Arbeit wurde die Kostenmodell-Steuerungseinheit auf dem Windows Betriebssystem entwickelt. Das von Morin erstellte Kostenmodellprogramm calculate_LEC ist ein unter Linux geschriebenes awk-Skript[15]. Die beiden Programmme werden somit auf unterschiedlichen Betriebssystemen ausgeführt. Die Kommunikation wurde dadurch gelöst, dass die Kostenmodell-Steuerungseinheit mittels SSH[16] Linux-Befehle auf einem anderen Rechner ausführen. Ergebnisse werden auf einem Netzlaufwerk abgelegt, auf das beide Programme Schreib- und Leserechte besitzen. Es stellt sich zu Recht die Frage, weshalb die Kostenmodell-Steuerungseinheit den Umweg über SSH geht, anstatt direkt unter Linux zu operieren. Der Grund ist, dass mittelfristig

[15] awk ist eine Programmiersprache (Skriptsprache) unter Linux zur Bearbeitung und Auswertung einfacher Textdateien.

[16] SSH - Secure Shell bezeichnet ein Netzwerkprotokoll, mit dem man auf eine sichere Art und Weise eine verschlüsselte Netzwerkverbindung mit einem entfernten Computer herstellen kann.

hier ein Softwarepaket entstehen soll, das nur noch auf dem Windows-System läuft (da Thermoflex nur unter Windows läuft). Mit der vorliegenden Lösung wurde bereits ein Schritt in diese Richtung gemacht.

Um calculate_LEC zu steuern, wurde die Kostenmodell-Steuerungseinheit implementiert. Diese soll für calculate_LEC die benötigten Dateien erstellen, calculate_LEC ausführen und die berechneten Ergebnisse in die Datenbank schreiben.

Damit die Kostenmodell-Steuerungseinheit mit dem Datenbankserver kommunizieren kann, wird die standardisierte Datenbankschnittstelle ODBC verwendet. Alle von der Steuerungseinheit benötigten Daten werden über diese Verbindung vom Datenbankserver angefordert, sowie alle simulierten Ergebnisse an den Datenbankserver übermittelt. Um die schon für den Optimierer verwendete ODBC-Schnittstelle zu nutzen, wurde die im Rahmen dieser Arbeit entwickelte Kostenmodell-Steuerungseinheit in der Programmiersprache C++ geschrieben (siehe Abbildung 3.10).

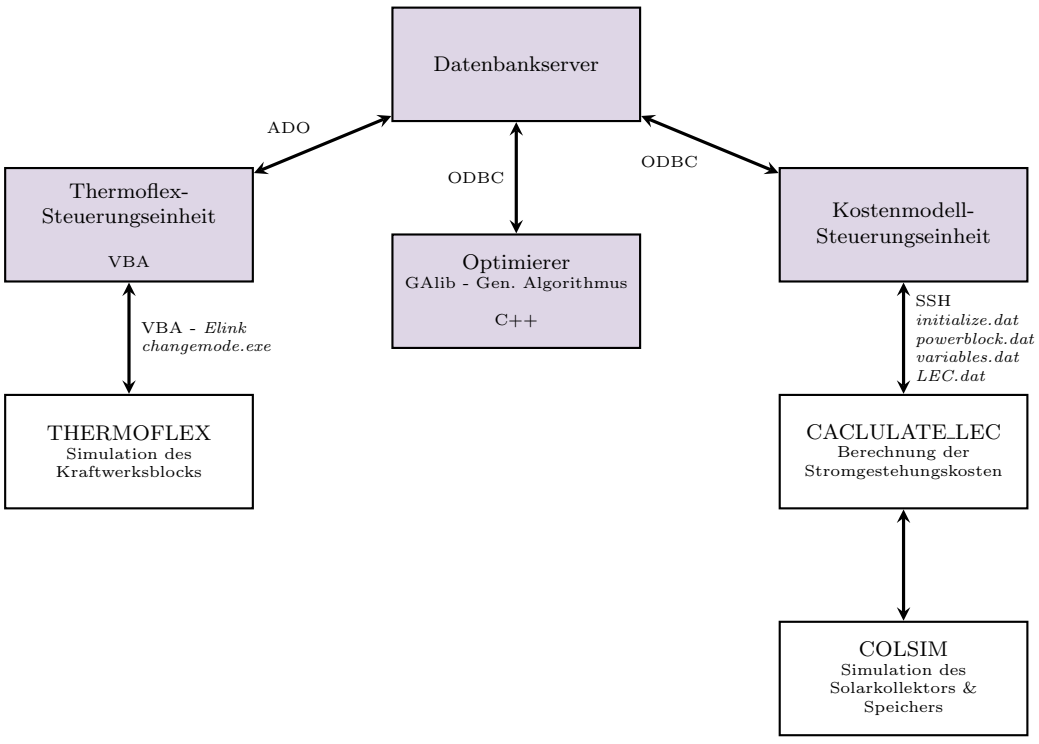


Abbildung 3.10: Simulationssoftware: Die Kostenmodell-Steuerungseinheit kommuniziert mit dem Kostenmodell-Programm über mehrere Dateien und steuert es mittels SSH. Die Kommunikation mit dem Datenbankserver erfolgt über ODBC.

Bei Programmstart verbindet sich die Kostenmodell-Steuerungseinheit mit dem Datenbankserver und fragt dort die Konfiguration für ColSim (*sim.dek*) und calculate_LEC ab. Es wird die Datei *initialize.dat* auf dem Netzlaufwerk abgelegt und anschließend per SSH das awk[17]-Skript initialize_colsim aufgerufen. Dieses Konfigurationsskript liest Einstellungswerte aus der Datei *initialize.dat* aus und schreibt diese in die Kostenberechnungs-Konfigurationsdatei *sim.dek*.

Zur Berechnung der Stromgestehungskosten bezüglich eines vom genetischen Algorithmus erzeugten Individuums werden zunächst die zu optimierenden ColSim-Variablen des Individuums vom Datenbankserver angefordert und in der Datei *variables.dat* auf dem Netzwerklaufwerk abgelegt. Auch werden die von Thermoflex berechneten Teillastfälle in der Kraftwerksblock-Kennlinienfeld-Datei *powerblock.dat* gespeichert. Anschließend wird mittels SSH das Kostenmodell-Programm calculate_LEC ausgeführt. Dieses Simulationsskript liest die Belegungen der Optimierungsvariablen aus der erstellten Datei *variables.dat* aus und schreibt diese in die ColSim-Konfigurationsdatei *sim.dek*. calculate_LEC führt anschließend ColSim aus, das den Jahresenergieertrag liefert.

Die Kraftwerksblock-Investitionskosten $K_{PBinvest}$ für eine Kraftwerksauslegung werden von den in Thermoflex hinterlegten Kostenmodellen PEACE geliefert und von der Kostenmodell-Steuerungseinheit als Variable in *variables.dat* an calculate_LEC übergeben. Nachdem das Kostenmodell die Stromgestehungskosten[18] berechnet hat, werden diese in die Datei *LEC.dat* gespeichert und auf dem Netzwerklaufwerk abgelegt. Die Kostenmodell-Steuerungseinheit liest den Wert der Stromgestehungskosten aus und übermittelt diesen an den Datenbankserver. Da die Stromgestehungskosten direkt dem Fitnesswert des Individuums entsprechen, ist an dieser Stelle auch der Simulationskreislauf geschlossen, da der Optimierer auf diesen Wert wartet.

Die Eigenschaften der Kostenmodell-Steuerungseinheit lassen sich folgendermaßen zusammenfassend angeben:

- Client-Programmeinheit, die mittels ODBC mit dem Datenbankserver kommu-

[17] awk ist eine Programmiersprache (Skriptsprache) unter Linux zur Bearbeitung und Auswertung einfacher Textdateien.

[18] Die Kostenaufstellung würde für die vorliegende Arbeit zu weit gehen. Die interessierten Leser werden an [20] verwiesen

niziert

- Flexible Konfiguration der Kostenmodell-Steuerungseinheit bei Programmstart durch Abfragen der Einstellungen beim Datenbankserver
- Automatischer Aufruf des Kostenmodell-Programms calculate_LEC mittels SSH

3.5 Datenbankserver

Im Rahmen dieser Arbeit wurde ein Datenbankserver entwickelt. In dem Client-Server System bildet eine relationale Datenbank die Server-Komponente. Diese empfängt und bearbeitet SQL Befehle der Clients. Der Datenbankserver besitzt die Aufgaben, die Daten der Clients aufzubewahren, zu aktualisieren und auszugeben. Der Austausch der Daten zwischen den einzelnen Clients geschieht somit über den Datenbankserver.

Als Datenbankverwaltungssystem wird ein MySQL Server[19] verwendet, dessen Transaktionsverwaltung die von Härder und Reuter [14] aufgestellten ACID[20]-Eigenschaften erfüllt:

- Atomarität: Eine Transaktion wird entweder ganz oder gar nicht ausgeführt.
- Konsistenz: Nach Ausführung der Transaktion muss der Datenbestand in einer konsistenten Form sein, wenn er es bereits zu Beginn der Transaktion war.
- Isolation: Bei gleichzeitiger Ausführung mehrerer Transaktionen dürfen sich diese nicht gegenseitig beeinflussen.
- Dauerhaftigkeit: Die Auswirkungen einer Transaktion müssen im Datenbestand dauerhaft bestehen bleiben.

Diese Eigenschaften garantieren, dass SQL-Befehle als logische Einheit betrachtet und ausgeführt werden.

Um Redundanzen der Daten zu vermeiden, die bei Änderung von Daten zu Inkonsistenzen führen können, wurde das Datenbankschema in die dritte Normalform überführt, vgl. [16]. Jede Tabelle modelliert nur einen Sachverhalt und innerhalb der Tabellen gibt

[19] Der MySQL Server ist ein relationales Datenbankverwaltungssystem in Open-Source.
[20] ACID - Atomicity, Consistency, Isolation, Durability

es keine transitiven Abhängigkeiten. Es finden sich nur noch funktional zusammengehörige Informationen in einer Tabelle. So hat jede Tabelle nur Spalten (so genannte Attribute) die, wie folgend beschrieben wird, genau einer Funktion dienen.

Abbildung 3.11 zeigt eine UML[21]-Darstellung der implementierten Datenbank. Die Datenbank lässt sich in den Simulationsbereich und einen Einstellungsbereich gliedern:

- Im Einstellungsbereich sind die Einstellungstabellen dargestellt, die vor einer Simulation gefüllt werden.
  - Die obere Tabelle *t_simulation_settings* zeigt alle generellen Einstellungen einer Optimierung.
  - Der linke Block mit den drei Tabellen entspricht den Einstellungen für die Thermoflex-Steuerungseinheit, unterteilt nach generellen Einstellungen (*t_tfx_settings*), Informationen zu den Kraftwerksprozessen (*t_tfx_cycles*) und off-design Einstellungen (*t_tfxoffdesign_casedef*).
  - Die Tabellen *t_colsim_settings* und *t_optimizer_settings* zeigen alle Einstellungen für die Kostenmodell-Steuerungseinheit bzw. den Optimierer.
  - Die rechte Tabelle *t_variables_settings* speichert alle Optimierungs- und funktionalen Variablen.
- Im Simulationsbereich sind die Simulationstabellen dargestellt, die während einer Simulation gefüllt werden.
  - In der oberen Tabelle *t_individuals* werden alle generellen Informationen der Individuen und in *t_individuals_var* alle Variablenbelegungen eines Individuums abgespeichert. Dazu zählen die Optimierungsvariablenbelegungen sowie davon abhängige funktionale Variablen von Thermoflex

    $$(t_1, \ldots, t_n, c_1, \ldots, c_m).$$

  - In der Tabelle *t_tfx_individuals* werden die design und in *t_tfxoffdesign* alle off-design Thermoflex-Simulationsergebnisse eines Individuums abgespei-

[21] UML - Unified Modeling Language ist eine standardisierte Sprache für die Modellierung von Software.

chert, was den Werten

$$f_{B_{pb}}(t_1, \ldots, t_n) = (o_1, \ldots, o_\ell)$$

entspricht.

– In der Tabelle *t_colsim_individuals* werden die Simulationsergebnisse der Kostenmodell-Steuerungseinheit gespeichert, was den Werten

$$f_P(c_1, \ldots, c_m, o_1, \ldots, o_\ell) = P_{el}, \quad f_{LEC}(K_{PBinvest,P_{el}}) = LEC$$

entspricht.

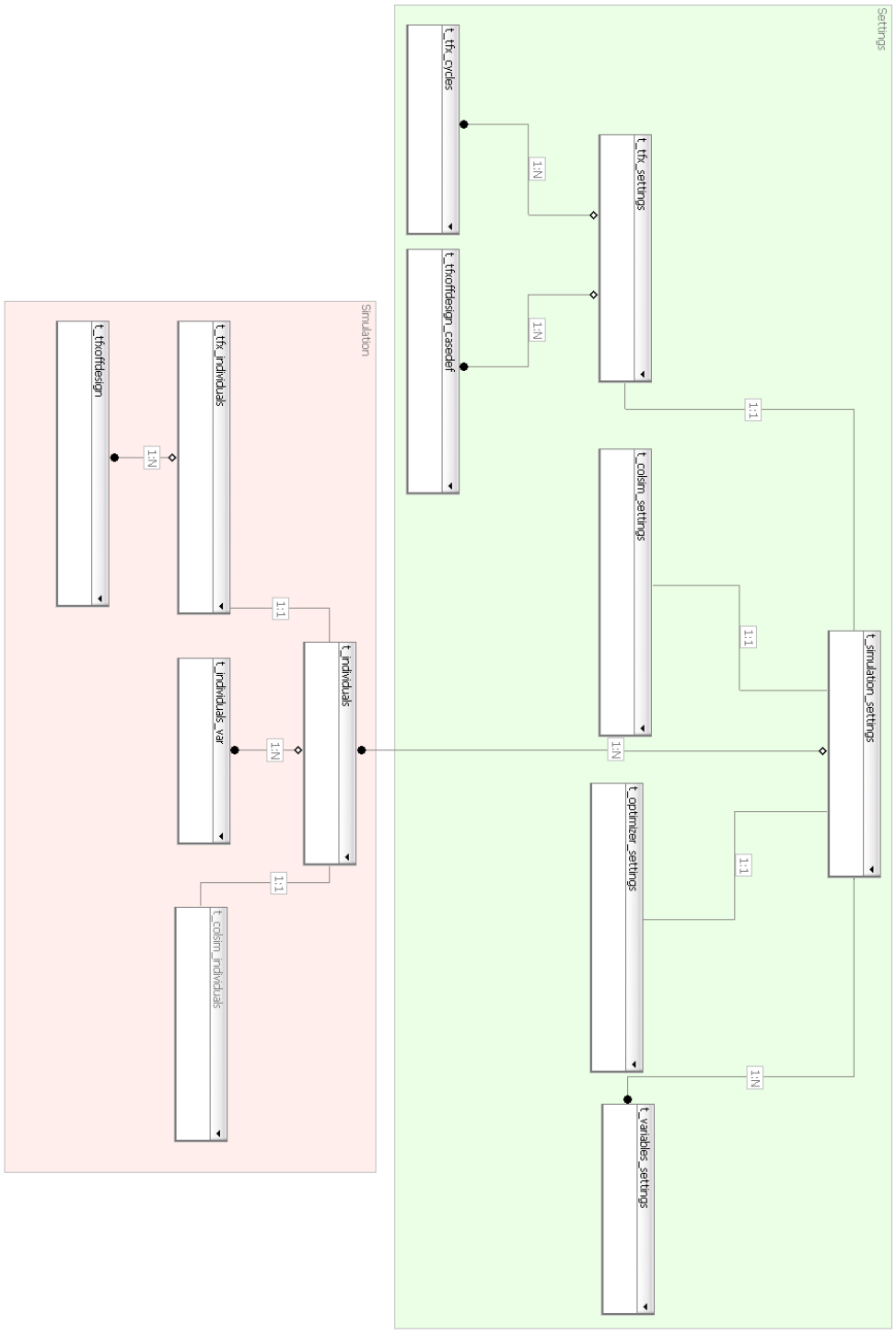


Abbildung 3.11: UML-Darstellung der implementierten Datenbank. Im oberen Bereich befinden sich die Einstellungstabellen und im unteren die Simulationstabellen.

3.6 Graphische Benutzeroberfläche

Im Rahmen dieser Arbeit wurde eine graphische Benutzeroberfläche entwickelt. Wie bereits oben beschrieben, verfügen die drei vorgestellten Clients über eine automatische Konfiguration. Bei Programmstart verbinden sich diese mit dem Datenbankserver und fragen Ihre Einstellungsdaten ab. Somit erfolgt die Einstellung aller Clients über eine zentrale Belegung der Datenbankdatensätze.

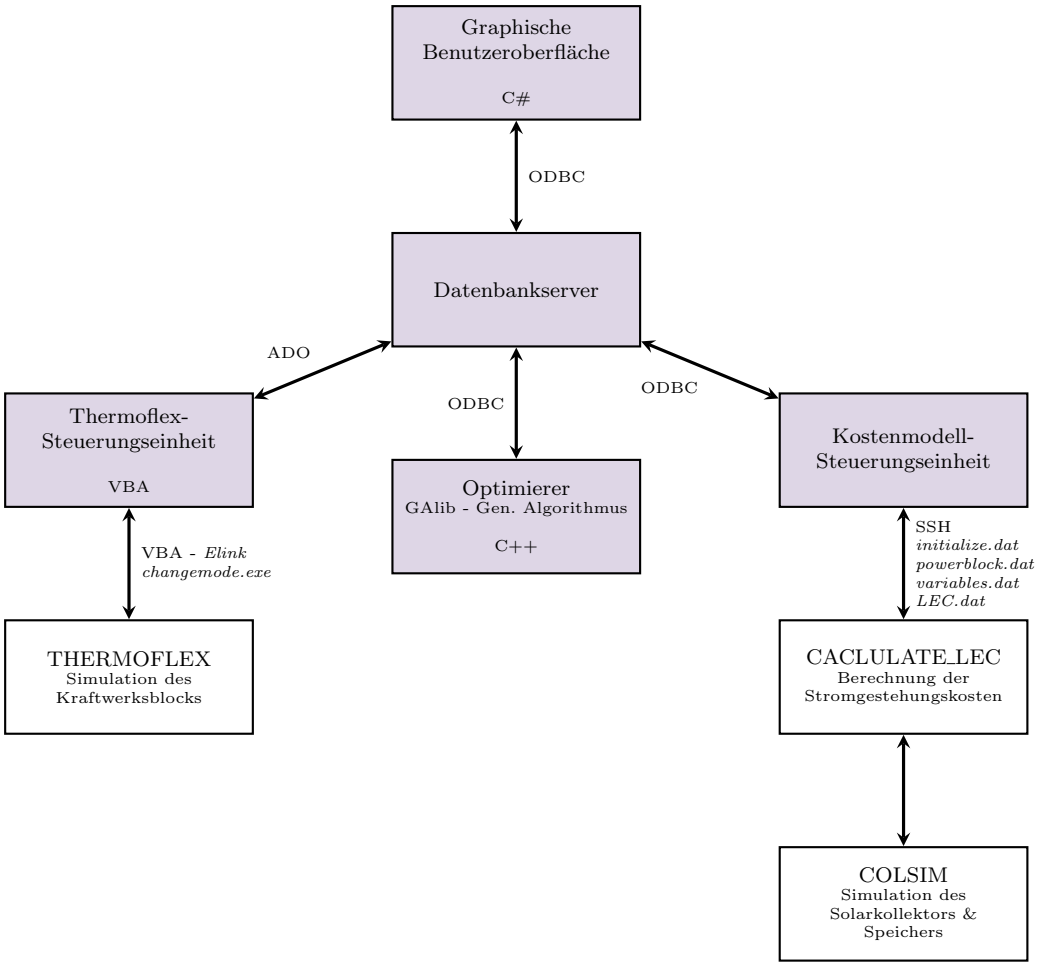


Abbildung 3.12: Simulationssoftware: Die Steuerungseinheiten Thermoflex-Steuerungseinheit, Kostenmodell-Steuerungseinheit und Optimierer sowie die graphische Benutzeroberfläche sind als Client mit dem Datenbankserver über ODBC oder ADO verbunden.

Um die Datenbank komfortabel mit den gewünschten Werten zu füllen, wurde eine graphische Benutzeroberfläche entwickelt. Mit dieser lassen sich nicht nur die Vielzahl an

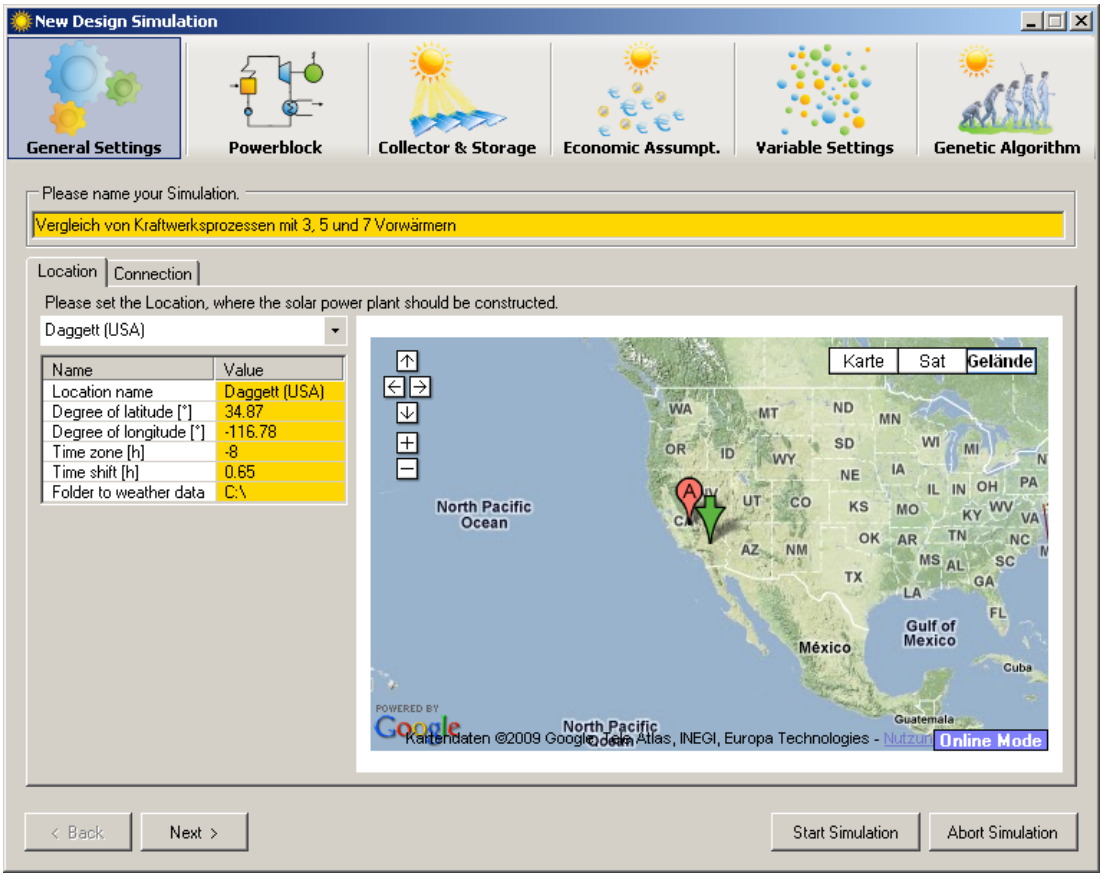


Abbildung 3.13: Generelle Einstellungen: Auswahl des Ortes mitsamt seiner Eigenschaften wie geographische Koordinaten, Zeitzone und Wetterdaten.

Einstellungsparameter für Solarkollektor, Kraftwerksblock und Optimierungsalgorithmus übersichtlich festlegen, sondern auch während der Simulation der aktuelle Stand und Auswertungen anzeigen. Der Simulationskreislauf mit graphischer Oberfläche ist im Client-Server System in Abbildung 3.12 dargestellt.

Die graphische Benutzeroberfläche ist in der Programmiersprache C# geschrieben. Das Design wurde so konzipiert, dass die Einstellmöglichkeiten dem Auslegungskonzept eines solarthermischen Kraftwerks folgen und nicht der softwareseitigen Lösung mit der dreigliedrigen Unterteilung in Kraftwerksblocksimulation, Solarkollektorsimulation und genetischen Algorithmus.

Die Einstellungen werden in einem Fenster mit verschiedenen Kategorien vorgenommen, zwischen denen frei gewechselt werden kann. Auf den nachfolgenden Seiten werden die Einstellmöglichkeiten vorgestellt.

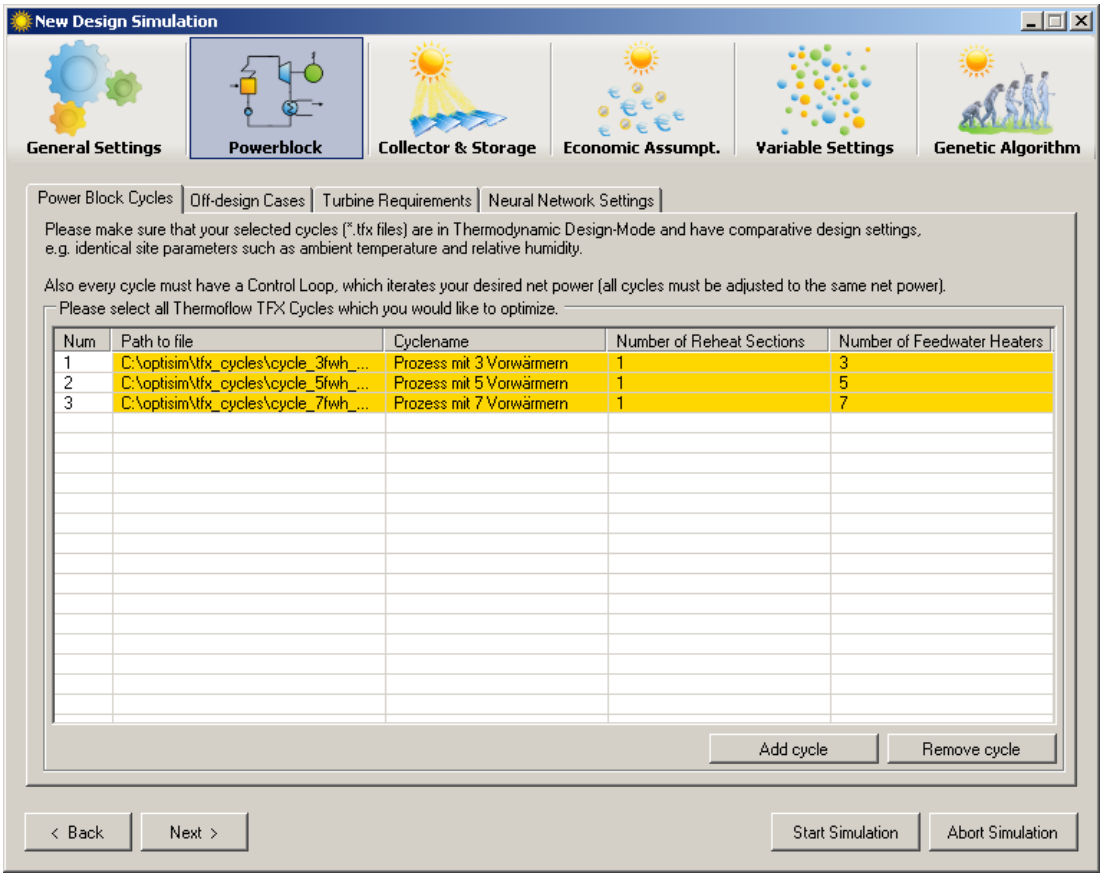


Abbildung 3.14: Kraftwerksblock Einstellungen: Auswahl der Kraftwerksblockprozesse.

Generelle Einstellungen

In der Kategorie *General Settings* werden die allgemeinen Einstellungen für die Simulation vorgenommen. Im Reiter *Location* werden Standortdaten mit jährlichen Wetterdaten ausgewählt (siehe Abbildung 3.13). Diese Einstellungen sind für die Kostenmodell-Steuerungseinheit bestimmt, vgl. Abschnitt 3.4.

Im Reiter *Connection* werden die SSH- und Datenbankverbindungen sowie ein temporärer Ordner festgelegt, in dem die Simulationsdaten gespeichert werden sollen. Beim Aufruf der Clientprogramme werden diese Datenbank-Einstellungen als Parameter übergeben. Die SSH-Einstellungen sind für die Kostenmodell-Steuerungseinheit bestimmt, die mittels SSH ColSim aufruft, vgl. Abschnitt 3.4.2.

Kraftwerksblock Einstellungen

In der Kategorie *Powerblock* werden die Einstellungen für den Kraftwerksblock vorgenommen. Im Reiter *Power Block Cycles* werden die Kraftwerksblockprozesse (*.tfx Datei) ausgewählt (siehe Abbildung 3.14). Diese Einstellungen sind für die Thermoflex-Steuerungseinheit bestimmt, die die ausgewählten Kraftwerksprozesse verknüpft und steuert, vgl. Abschnitt 3.3.2. Im Reiter *Off-design Cases* werden die verschiedenen Be-

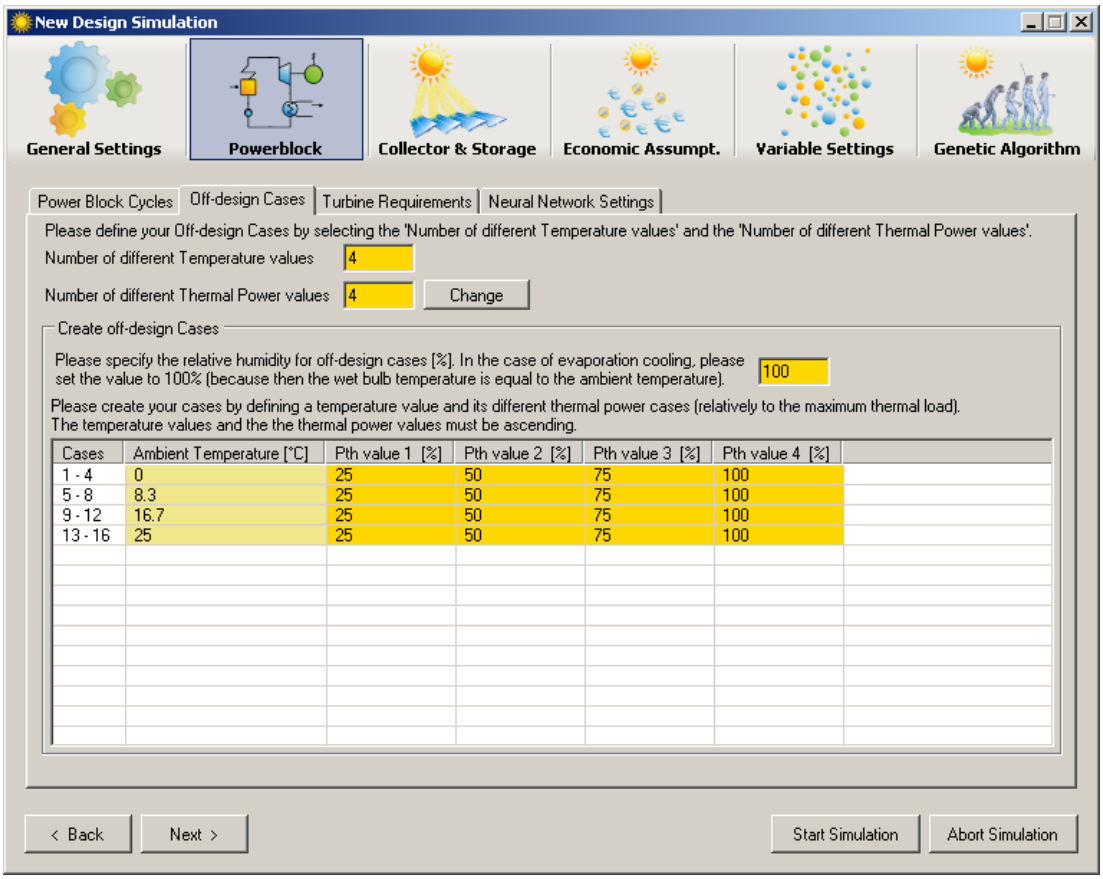


Abbildung 3.15: Kraftwerksblock Einstellungen: Auswahl der verschiedenen Betriebszustände eines Kraftwerksblocks.

triebszustände eingestellt, die für einen ausgelegten Kraftwerksprozess simuliert werden. Dazu wird die Anzahl der verschiedenen Temperaturen und die Anzahl der thermischen Teillastfälle je Temperatur festgelegt (siehe Abbildung 3.15). Die Einstellungen werden für die Thermoflex-Steuerungseinheit benötigt um die einzelnen off-design Fälle zu berechnen, vgl. Abschnitt 3.3. Die Ergebnisse der Simulationen werden von der Kostenmodell-Steuerungseinheit benötigt, vgl. Abschnitt 3.4.

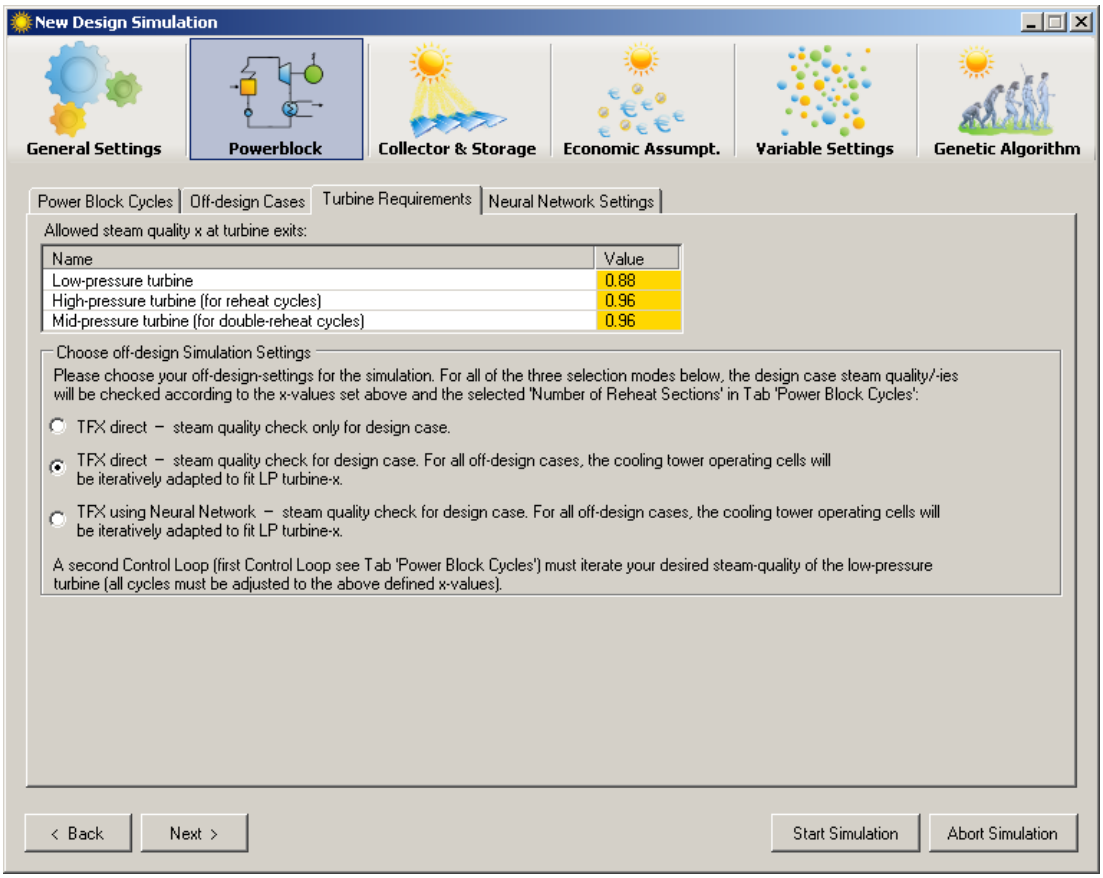


Abbildung 3.16: Kraftwerksblock Einstellungen: Einstellung der Dampfgehaltrestriktionen der verschiedenen Turbinenstufen, sowie Aktivierung / Deaktivierung der Dampfgehaltrestriktions-Kontollschleife.

Im Reiter *Turbine Requirements* werden die Dampfgehaltrestriktionen der verschiedenen Turbinenstufen eingestellt (siehe Abbildung 3.16). Des Weiteren kann eingestellt werden, ob die Thermoflex-interne Kontrollschleife aktiviert werden soll, die für eine Erfüllung der Dampfgehaltrestriktion sorgt, vgl. Abschnitt 3.3.2.

Eine weitere Einstellmöglichkeit ist die Verwendung von neuronalen Netzen, auf die nicht an dieser Stelle, sondern im Kapitel 4 eingegangen wird.

Solarkollektor Einstellungen

In der Kategorie *Collector & Storage* werden die Einstellungen für den Solarkollektor vorgenommen. Diese lassen sich unterteilen in Solarfeldparameter (Reiter *Solarfield*), optische Kollektorparameter (Reiter *Collector Optics*) und thermische Kollektorpara-

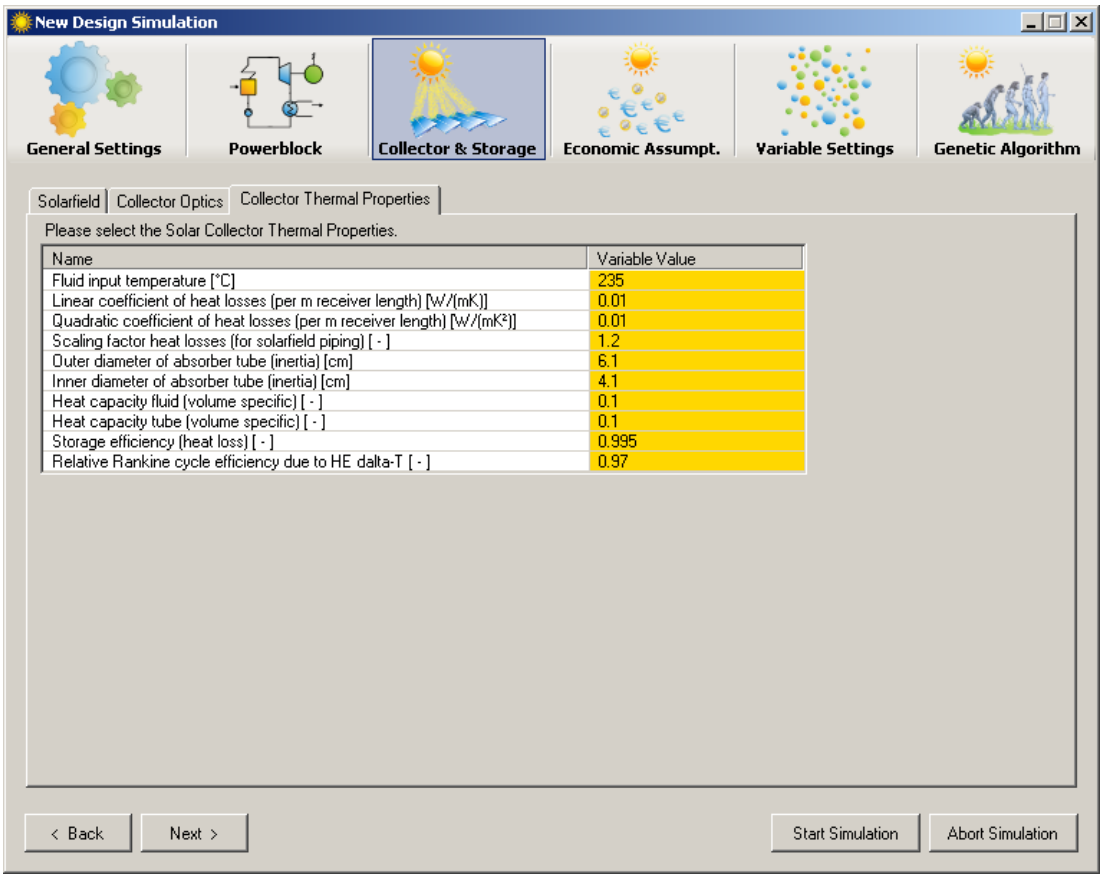


Abbildung 3.17: Solarkollektor Einstellungen: Einstellung der thermische Kollektorparameter.

meter (Reiter *Collector Thermal Properties*). In Abbildung 3.17 sind die Einstellmöglichkeiten der thermische Kollektorparameter dargestellt. Diese Parameter werden von der Kostenmodell-Steuerungseinheit als Einstellparameter für ColSim verwendet.

Ökonomische Einstellungen

In der Kategorie *Economic Assumptions* werden die ökonomischen Annahmen festgelegt, siehe Abbildung 3.18. Diese Parameter werden von der Kostenmodell-Steuerungseinheit als Einstellparameter für calculate_LEC verwendet.

Auswahl und Einstellung der Variablen

In der Kategorie *Variable Settings* werden alle Einstellungen zu den Variablen gemacht. Im Reiter *Variables* werden die Optimierungsvariablen und funktionalen Variablen ein-

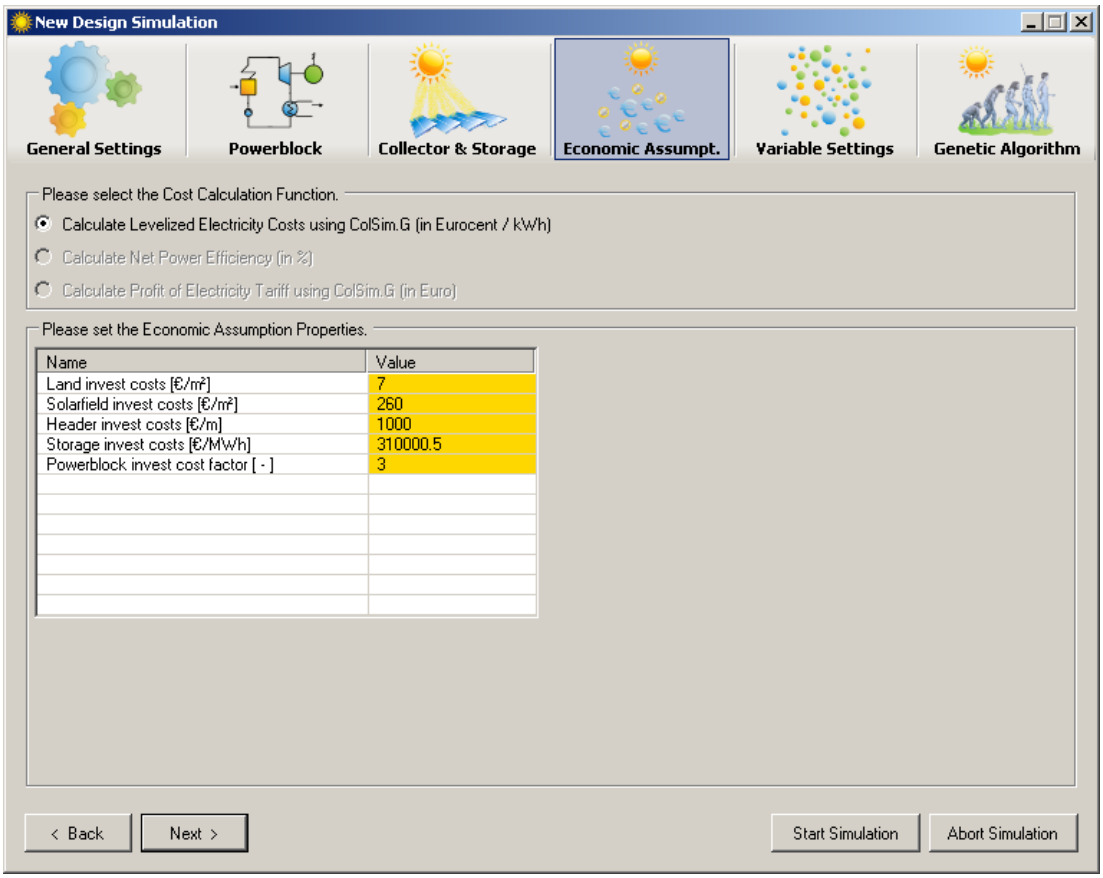


Abbildung 3.18: Ökonomische Einstellungen zur Berechnung der Stromgestehungskosten.

gestellt. Zu den festgelegten Solarkollektor-Optimierungsvariablen können wahlweise Kraftwerksblock-Optimierungsvariablen hinzugefügt oder entfernt werden und deren Definitionsbereich festgelegt werden. Zusätzlich können hier für jeden Kraftwerksprozess die funktionalen Variablen definiert werden (siehe Abbildung 3.19). Die Definitionsbereiche der Optimierungsvariablen und die Funktionsvorschriften der funktionalen Variablen werden für die Konfiguration des genetischen Algorithmus benötigt, vgl. Abschnitt 3.3.2. Im Reiter *Start Population* können Individuen kreiert werden (siehe Abbildung 3.20), die zu der Start-Population des genetischen Algorithmus gehören werden, vgl. Abschnitt 3.2.2. Es ist möglich, eine Menge von Individuen aus einer Leerzeichen-separierten Datei zu laden.

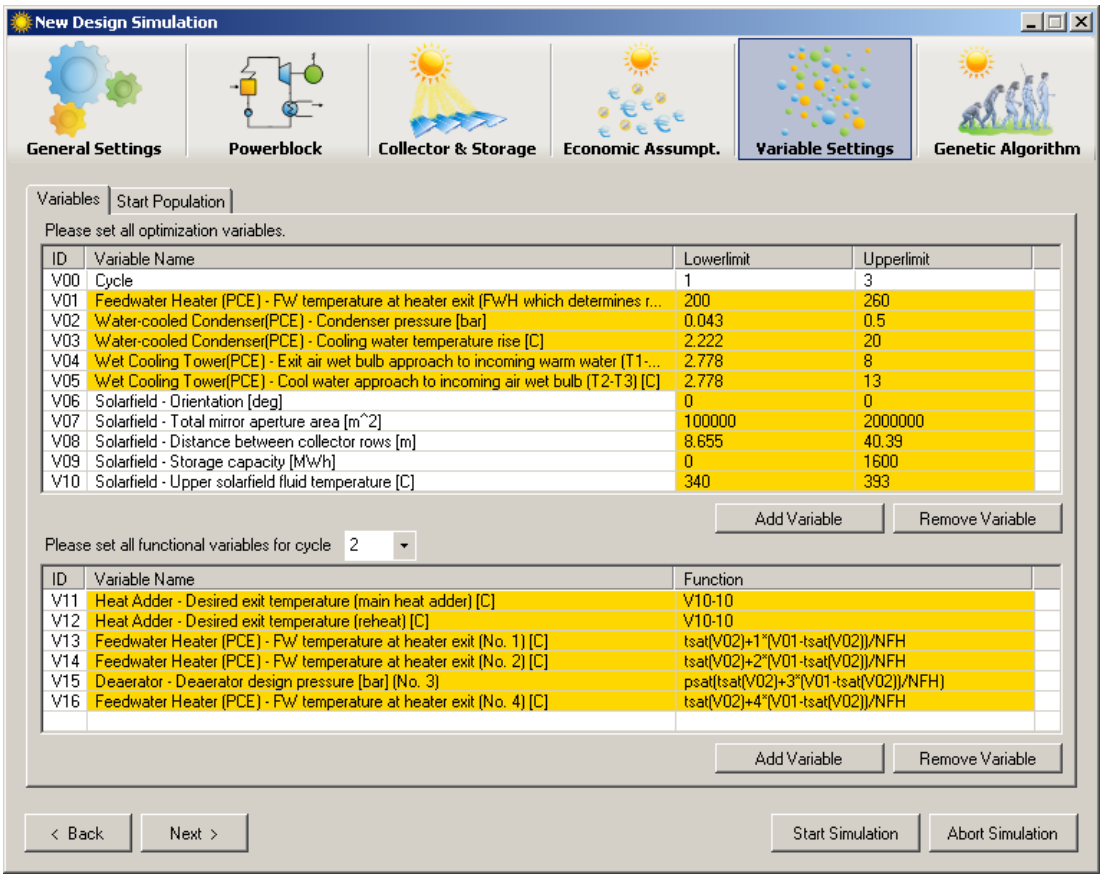


Abbildung 3.19: Variablen Einstellungen: Definition des Gültigkeitsbereichs von Optimierungsvariablen und Definition der funktionalen Variablen.

Einstellung des genetischen Algorithmus

In der Kategorie *Genetic Algorithm* wird der genetische Algorithmus eingestellt, siehe Abbildung 3.21. Es werden die Funktionsweise und Einstellungsparameter gesetzt. Diese Einstellungen werden vom Optimierer verwendet, vgl. Abschnitt 3.2.2.

3.7 Starten einer Simulation

Der Simulationskreislauf sieht wie folgt aus. Für eine zuvor fest definierte Anzahl von Optimierungsvariablen für den Kraftwerksblock und Solarkollektor werden mittels genetischen Algorithmus Belegungen (Individuen) erstellt. Die Kraftwerksblockvariablen der Individuen werden nach und nach als Parameter an Thermoflex übergeben, welches die technische Auslegung des Kraftwerksblocks bestimmt. Nach Bestimmung der Auslegung und der Kraftwerksblock-Investitionskosten werden von Thermoflex noch

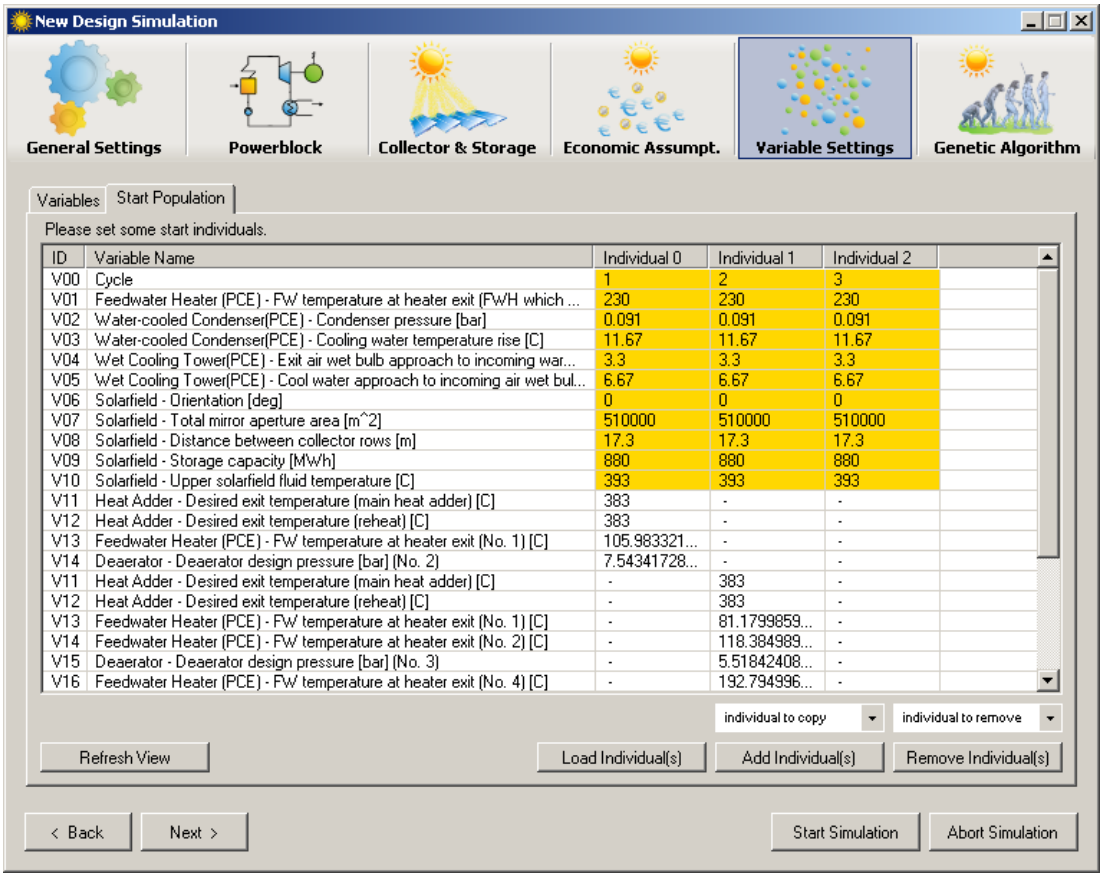


Abbildung 3.20: Variablen Einstellungen: Setzen von Start-Individuen.

Teillastfälle durchgeführt (siehe Abschnitt 3.3), die das Verhalten eines Kraftwerksprozesses beschreiben.

Diese Ergebnisse und die Kollektorvariablen der Individuen werden ausgelesen und als Parameter in Dateien auf dem Netzlaufwerk an calculate_LEC übergeben. Auf Grundlage der in Thermoflex bestimmten Investitionskosten des Kraftwerksblocks und des mit ColSim bestimmten Jahresenergieertrags, berechnet calculate_LEC die Stromgestehungskosten, welche die Bewertung des Individuums darstellen.

Jedes Individuum durchläuft diesen Zyklus, so dass der Optimierer über eine Bewertung für jedes einzelne Individuum verfügt. Auf Grundlage dieser Bewertungen erzeugt der Algorithmus neue Individuen mittels der üblichen Werkzeuge, die einen genetischen Algorithmus auszeichnen.

Sind alle Einstellungen gemacht, so wird mit drücken der *Start Simulation* Schaltflä-

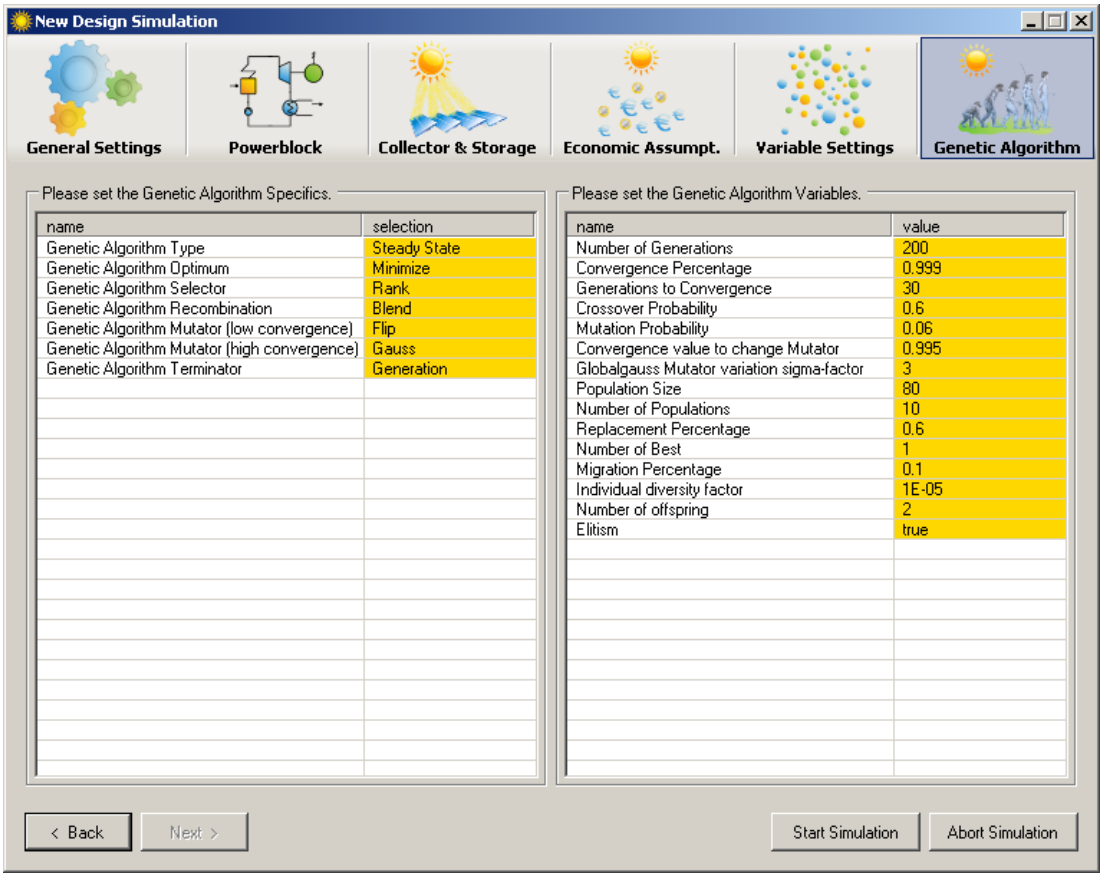


Abbildung 3.21: Einstellung des genetischen Algorithmus.

che die Simulation gestartet. Dies bedeutet dass zunächst einmal alle Einstellungen überprüft werden, wie beispielsweise ob das obere Limit einer Optimierungsvariable tatsächlich größer ist als ihr unteres Limit. Sind alle Einstellungen überprüft, wird eine Verbindung zur Datenbank aufgebaut und alle vom Benutzer eingegebenen Parameter mittels ODBC übertragen.

Anschließend wird die Thermoflex-Steuerungseinheit automatisch gestartet. Diese verbindet sich zunächst mittels ADO mit der Datenbank und fragt seine Konfiguration ab. Anschließend werden die Kostenmodell-Steuerungseinheit und der Optimierer automatisch gestartet, die sich mittels ODBC mit der Datenbank verbinden und ihre Konfiguration abfragen. Sind alle Client-Steuerungseinheiten fertig initialisiert, startet der genetische Algorithmus mit der Erstellung der Initialisierungsgeneration. Wurden vom Benutzer in der graphischen Benutzeroberfläche Start-Individuen gesetzt, so sind diese in der Start-Population vertreten.

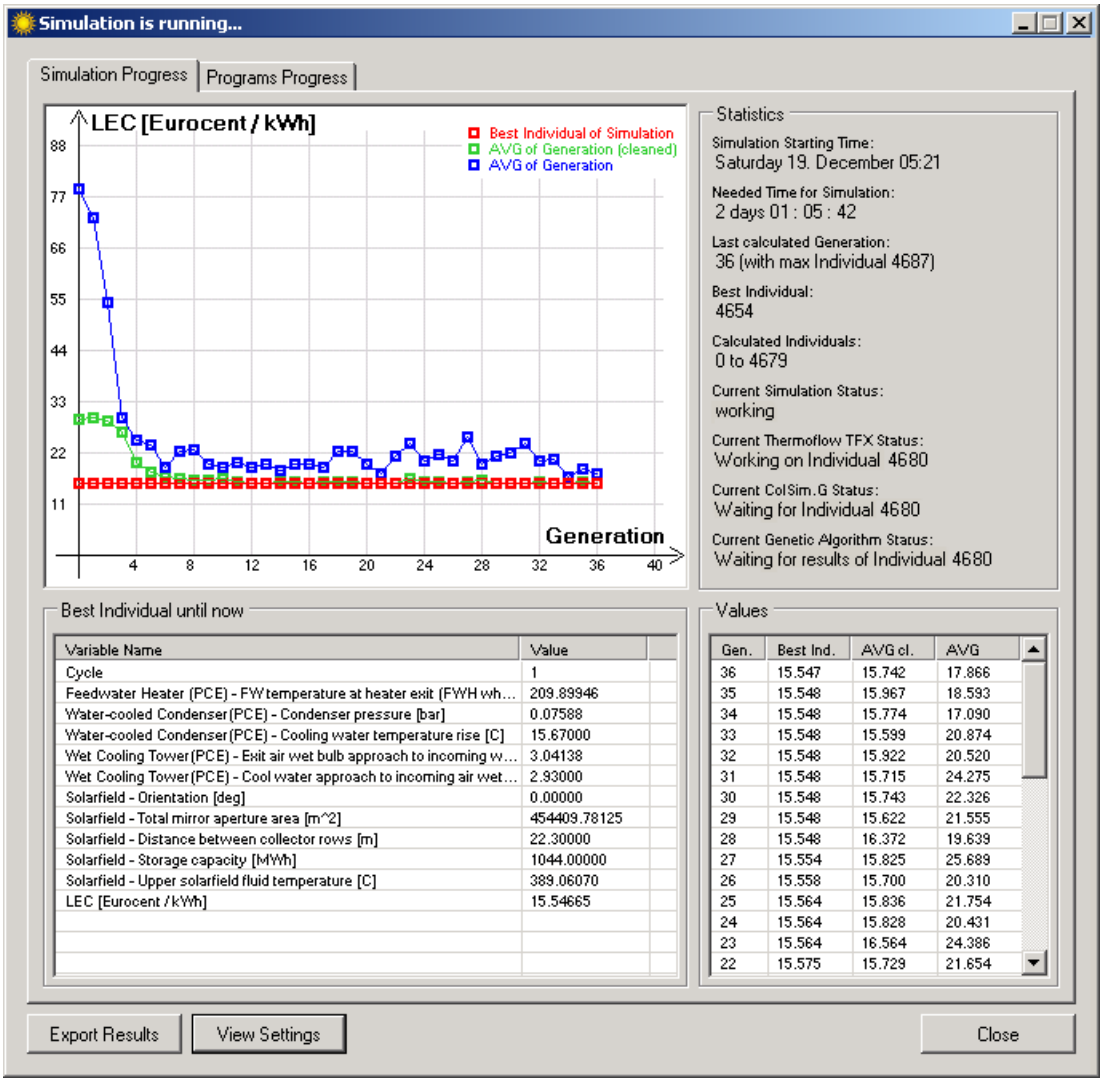


Abbildung 3.22: Graphische Benutzeroberfläche während der Simulation.

Die vom genetischen Algorithmus erstellten Individuen entsprechen den Optimierungsvariablen des Kraftwerksblocks und des Solarkollektors. Wurden funktionale Variablen gesetzt, so berechnet der Optimierer deren Funktionswerte. Anschließend werden alle Individuen in der Datenbank abgelegt. Existiert zu einem erstellten Individuum i bereits ein Individuum j mit gleichen Optimierungsvariablen in der Datenbank, so werden für i die Ergebnisse von j kopiert. Sind nicht alle Optimierungsvariablen gleich aber alle Kraftwerksblock-Optimierungsvariablen gleich, so werden alle Kraftwerksblock Ergebnisse kopiert, vgl. Abschnitt 3.2.2. Nachdem der Optimierer alle Individuen einer Generation in die Datenbank geschrieben hat, wartet er auf den Fitnesswert, der von

der Thermoflex-Steuerungseinheit und der Kostenmodell-Steuerungseinheit ermittelt wird. Liegen alle Fitnesswerte vor, erstellt der genetische Algorithmus eine neue Generation.

Die Thermoflex-Steuerungseinheit betrachtet sequentiell alle in der Datenbank vorhandenen Individuen. Liegt ein noch nicht bearbeitetes Individuum vor, wird zunächst überprüft, ob dieses Individuum schon in einer früheren Generation existierte. Ist dies der Fall, so ist die Bearbeitung fertig und das nächste Individuum wird betrachtet. Andernfalls werden die Variablenbelegungen des Individuums geladen. Gemäß der Kraftwerksprozessnummer werden der entsprechende Kraftwerksprozess geladen, die Variablenwerte gesetzt und der Designfall simuliert. Das Skript *changemode.exe* wird gestartet um den Kraftwerksprozess über automatische Mausklicks in den off-design Modus umzustellen. Nach und nach werden die off-design Teillastfälle durchgeführt, sofern die Prüfeinheit der Thermoflex-Steuerungseinheit keine Fehler erkennt. Die Ergebnisse werden in der Datenbank abgelegt.

Die Kostenmodell-Steuerungseinheit betrachtet sequentiell alle in der Datenbank vorhandenen Individuen. Liegt ein schon von Thermoflex-Steuerungseinheit bearbeitetes Individuum vor, wird zunächst überprüft, ob dieses Individuum schon in einer früheren Generation existierte. Ist dies der Fall, so ist die Bearbeitung fertig und das nächste Individuum wird betrachtet. Andernfalls werden dessen Solarkollektorvariablen und zugehörigen Thermoflex-Simulationsergebnisse ausgelesen, diese in die Dateien *variables.dat* und *powerblock.dat* geschrieben und auf dem Netzlaufwerk hinterlegt. Mittels SSH wird calculate_LEC aufgerufen, dessen Ergebnis in der Datei *LEC.dat* geschrieben steht. Die Kostenmodell-Steuerungseinheit liest den Wert aus und schreibt ihn in die Datenbank, so dass der Optimierer dem Individuum seinen Fitnesswert zuteilen kann.

Während der Simulation kann sich der Benutzer ständig ein Bild vom aktuellen Fortschritt der Optimierungsrechnung machen. Die graphische Benutzeroberfläche zeigt generationsweise den Verlauf der Simulation graphisch an, siehe Abbildung 3.22. Der aktuelle Status der einzelnen Steuerungseinheiten wird angezeigt, sowie die Parameter des bis dahin berechneten, besten Individuums. Durch Betätigen von Schaltflächen kann eine Simulation pausiert oder abgebrochen werden, sowie die zuvor gesetzten Einstellungen in einem PDF-Dokument angesehen werden. Man kann sich auch eine

bestimmte Auswahl von Daten in separate Dateien ausgegeben lassen. Beispielsweise eine Auflistung aller Individuen mit ihren Einstellungen oder aber die Trainingsmenge des neuronalen Netzes (siehe Kapitel 4).

Kapitel 4

Anwendung von neuronalen Netzen zur Verbesserung der Simulationszeit

In Kapitel 3 wurde die Implementierung des Optimierungskreislaufs erläutert. Die weitaus längste Rechenzeit benötigt Thermoflex, das für jedes Individuum einen design Fall und mehrere off-design Teillastfälle berechnet. Gerade hier besteht das größte Potential, um Rechenzeit zu reduzieren. Im Folgenden wird untersucht, inwiefern sich neuronale Netze eignen, um effizienter als Thermoflex off-design Fälle zu bestimmen.

In Abschnitt 4.1 wird zunächst auf die Grundlagen neuronaler Netze eingegangen. Abschnitt 4.2 beschäftigt sich mit der Implementierung des neuronalen Netzes, und dessen Eingliederung in das Gesamtkonzept wird erläutert. Der Abschnitt 4.3 beschäftigt sich mit der emprischen Untersuchung von neuronalen Netzen für die solarthermische Kraftwerkssimulation.

4.1 Theorie der neuronalen Netze

Laut Bishop [2] können künstliche neuronale Netze beliebig komplexe Funktionen approximieren, Aufgaben erlernen und Probleme lösen, bei denen eine explizite Modellierung schwierig bis nicht durchführbar ist.

Der Ursprung *künstlicher neuronaler Netze* liegt in der Biologie. Man stellt sie den natürlichen neuronalen Netzen gegenüber, wobei es sich um Nervenzellvernetzungen

im Gehirn handelt. Ein künstliches neuronales Netz (siehe Abbildung 4.1) entspricht einem gerichteten und gewichteten Graphen, wobei die Knoten als *künstliche Neuronen* bezeichnet werden. Die Kanten entsprechen der Vernetzung von Neuronen. Die Basis für die neuronalen Netze bilden die künstlichen Neuronen, die als Modell aus dem biologischen Vorbild der Nervenzelle entstanden sind.

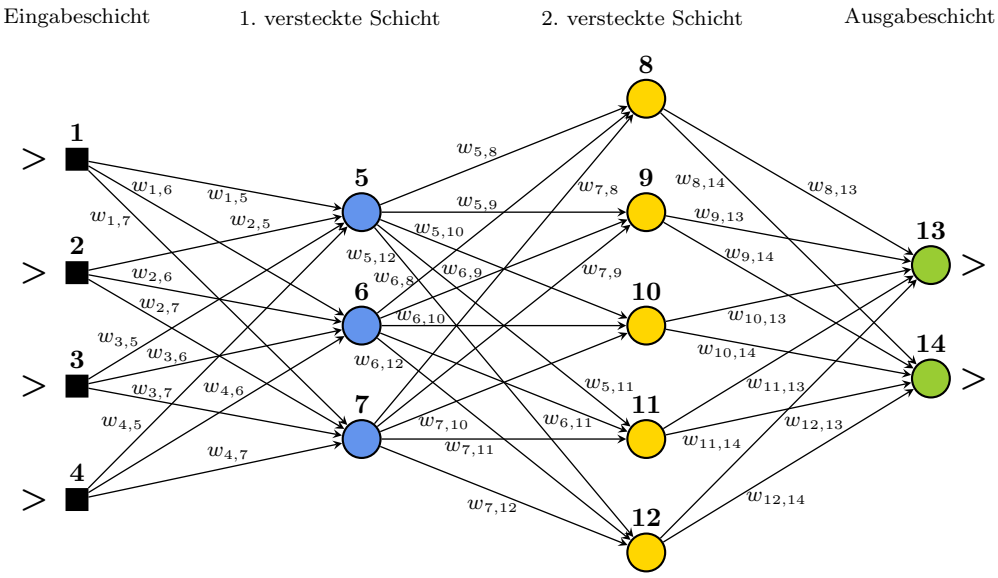


Abbildung 4.1: Vereinfachte Darstellung eines künstlichen neuronalen Netzes. Die Neuronen sind über Kanten verbunden. Die Eingänge bzw. die Ausgänge des Netzes werden durch die Pfeile dargestellt.

Ein künstliches neuronales Netz erlernt eine Funktion anhand von Trainingspunkten. Die prinzipielle Arbeitsweise besteht darin, einen Eingabevektor in einen Ausgabevektor umzuwandeln. Der an den Netzeingang angelegte Eingabevektor erzeugt einen Informationsfluss im neuronalen Netz. Die Informationen werden über mehrere Verbindungen von Neuron zu Neuron übertragen. Die Neuronen verarbeiten die Informationen und schicken sie über ihre Verbindungen weiter zu anderen Neuronen. Die Verbindungen sind unterschiedlich stark gewichtet, so dass sich dies verstärkend oder hemmend auf die geschickten Informationen auswirkt. Die am Netzausgang ankommenden Informationen stellen den Ausgabevektor dar. Ein neuronales Netz wird „trainiert“, indem es zu einer Menge von Eingabevektoren interne Parameter so ändert, dass sich die Ausgabevektoren den Zielvektoren annähern. In einem neuronalen Netz können beispielsweise die Gewichte der Verbindungen oder die informationsver-

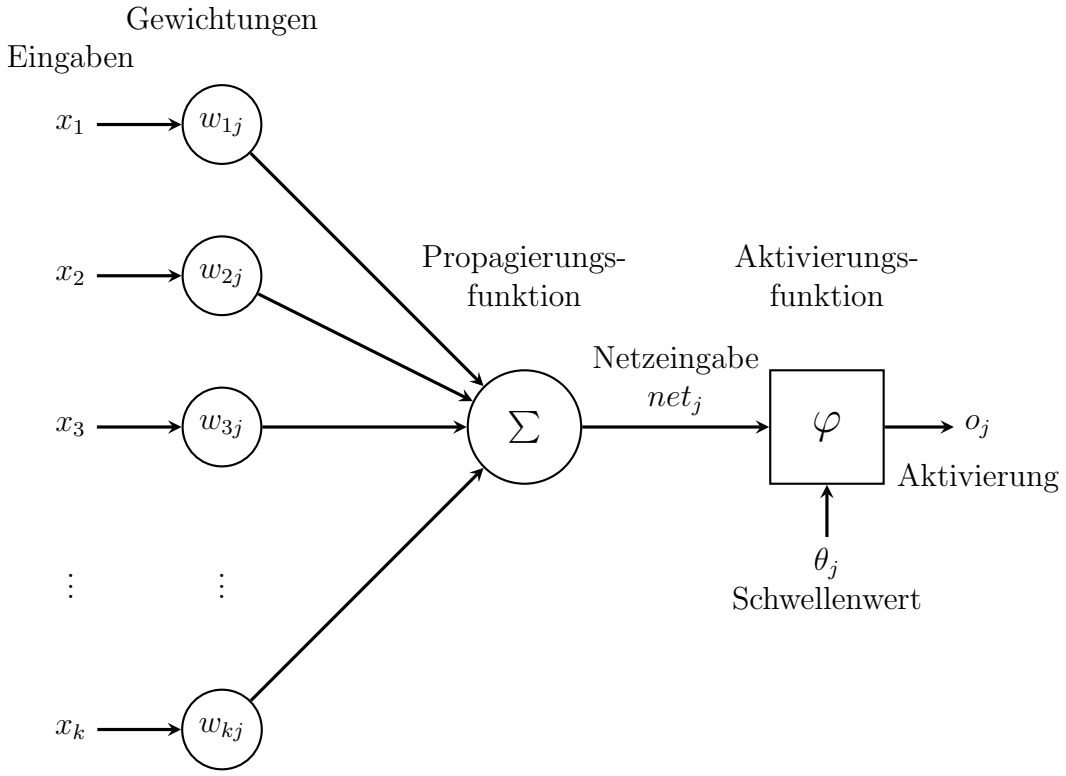


Abbildung 4.2: Schema eines künstlichen Neurons j: Die Eingaben werden über die Verbindungsgewichte gehemmt oder verstärkt und durch die Propagierungsfunktion zu einem Wert, der Netzeingabe, zusammengefasst. Zusammen mit dem Schwellenwert bestimmt die Aktivierungsfunktion die Ausgabe, die sog. Aktivierung des Neurons.

arbeitenden Funktionen der Neuronen angepasst werden. Nach dem Training wird die Güte des neuronalen Netzes an einer von der Trainingsmenge unabhängigen Validierungsmenge gemessen.

Nach dem Vorbild der Natur reagiert jedes Neuron auf seine Eingaben unterschiedlich stark. Die in einem Neuron ankommenden Informationen werden zunächst gemäß dem *Verbingungsgewicht* verstärkt oder gehemmt und anschließend mit Hilfe einer sog. *Propagierungfunktion* zu einem Wert, der *Netzeingabe*, verarbeitet (siehe Abbildung 4.2). Eine interne informationsverarbeitende Funktion verarbeitet die Netzeingabe zu einer Ausgabe, der sog. *Aktivierung*. Dabei ist diese *Aktivierungsfunktion* von einem *Schwellenwert* abhängig. Der Schwellenwert stellt biologisch gesehen die Reizschwelle dar, ab der ein Neuron aktiv wird. Diese sind neben den Verbindungsgewichtungen die Parameter, die vom neuronalen Netz während des Trainings verändert werden können.

Auf die genaue Definition der Begriffe wird in Definition 4.3 eingegangen.

In der vorliegenden Arbeit werden nur die auf McClelland et al. [12] zurückgehenden *mehrlagigen Perzeptrone* betrachtet. Dies sind Netze, in denen Neuronen in Schichten angeordnet sind und die Verbindungen von den Eingängen bis zu den Ausgängen vorwärtsgerichtet sind (siehe Abbildung 4.1). Alle Neuronen jeder Schicht sind nur mit Neuronen der nächst höheren Schicht verbunden. Die Neuronen der Eingabeschicht dienen nur zur Eingabe der Werte in das neuronale Netz und sind demnach keine Neuronen im eigentlichen Sinne, da der Schwellenwert konstant Null ist und die Aktivierungsfunktion der Identitätsfunktion entspricht.

Definition 4.1 (Mehrlagiges Perzeptron)**.** *In einem* mehrlagigen Perzeptron *sind für ein* $c \in \mathbb{N}$ *Neuronen in einer Sequenz von* $c+1$ *Schichten* $L^{(0)}, \ldots, L^{(c+1)}$ *gruppiert. Alle Neuronen einer Schicht* $L^{(k)}$ *sind mit allen Neuronen der nächsten Schicht* $L^{(k+1)}$ *für alle* $k = 0, \ldots, c$ *verbunden. Die Zahl* $h_k := |L^{(k)}| \in \mathbb{N}$ *gibt die Anzahl der Neuronen in der Schicht* $L^{(k)}$ *an. Es heißt* $L^{(0)}$ Eingabeschicht *und* h_0 Eingabedimension *des Netzes, sowie* $L^{(c+1)}$ Ausgabeschicht *und* h_{c+1} Ausgabedimension*. Die Schichten* $L^{(1)}, \ldots, L^{(c)}$ *werden versteckte Schichten genannt.*

Die Anzahl der Neuronen in der Eingabe- und Ausgabeschicht wird durch das Problem definiert. Um eine Funktion $f : \mathbb{R}^m \to \mathbb{R}^n$ abzubilden, haben die zugehörigen neuronalen Netze eine Eingabedimension m und Ausgabedimension n. Die Anzahl der Neuronen in den versteckten Schichten sowie die Anzahl der versteckten Schichten werden von der Komplexität der Aufgabe bestimmt. Laut Rojas [23] kann schon ein einlagiges Perzeptron mit genügend vielen Neuronen in der versteckten Schicht alle Aufgaben erfüllen, die auch eine Turing Maschine berechnen kann. Es werden jedoch trotzdem Netzwerke mit mehreren internen Schichten verwendet, da diese unter Umständen leichter zu trainieren sind.

Die Kommunikation zwischen den Neuronen geschieht Schicht für Schicht von der Eingangsschicht über die versteckten Verarbeitungsschichten bis hin zu der Ausgangsschicht. Die Aktivierung der Neuronen in der Ausgangsschicht entspricht den Ergebnissen der Berechnung. Diese lassen sich für eine beliebige Eingabe laut Bishop [2] in einer deterministischen Berechnungsweise bestimmen: Schicht für Schicht wird die Netzeingabe eines Neurons bestimmt und mit dessen Aktivierungsfunktion die Aktivierung ermittelt. Durch diese sukzessive Vorgehensweise lässt sich eine explizite Funktion für

alle Neuronen der Ausgangsschicht ermitteln, die von den Gewichten und Schwellenwerten der Neuronen der anderen c Schichten abhängt.

Ein mehrlagiges Perzeptron kann man einstellen, indem man die Verbindungsgewichte zwischen den Neuronen oder die Schwellenwerte der Neuronen variiert. Die nächste Definition gibt eine Formel an, die die Anzahl dieser *Einstellungsparameter* bestimmt:

Definition 4.2 (Einstellungsparameter)**.** *Für ein mehrlagiges Perzeptron mit $c+1$ Schichten ergibt sich aus der Anzahl der Verbindungsgewichte zwischen den Neuronen und der Schwellenwerte der Neuronen die Anzahl der* freien Einstellungsparameter λ *des neuronalen Netzes. Zu beachten ist, dass die Schwellenwerte der Neuronen in der Eingangsschicht nicht frei sind:*

$$\begin{aligned} \lambda &:= \sum_{i=0}^{c} \left(\underbrace{h_i \cdot h_{i+1}}_{Verbindungsgewichte} + \underbrace{h_{i+1}}_{Schwellenwerte} \right) \\ &= \sum_{i=0}^{c} (h_i + 1) \cdot h_{i+1} \end{aligned}$$

Sei $\alpha \in \mathbb{R}^\lambda$, so nennt man α Parameterbelegung des mehrlagigen Perzeptrons.

Im Folgenden wird eine formale Definition eines neuronalen Netzes gegeben.

Definition 4.3 (Neuronales Netz)**.** *Ein* neuronales Netz *ist definiert als ein sechs-Tupel $\mathcal{N} = (N, V, w, f_{prop}, \theta, \varphi)$.*

- *N ist eine endliche Menge von* Neuronen*.*
- *$V \subseteq N \times N$ repräsentiert die Menge von* Verbindungen *zwischen den Neuronen.*
- *Die Funktion $w : V \to \mathbb{R}$ ordnet jeder Verbindung ein* Gewicht *zu. Das Gewicht $w((i,j))$ von Neuron i zu Neuron j wird mit w_{ij} bezeichnet.*
- *f_{prop} ordnet zu jedem $j \in N$ eine* Propagierungsfunktion *f_{prop_j} zu. Sei $I = \{i \mid (i,j) \in V\}$. Für $I \neq \emptyset$ (d.h. j ist kein Neuron der Eingabeschicht) und $k = |I|$ heißt eine Funktion*

$$f_{prop_j} : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R},\ (o, w) \mapsto net_j$$

Propagierungsfunktion des Neurons j. Wir nennen net_j die Netzeingabe *des Neurons j unter* Eingabe o *und* Gewichtung w. *Für $I = \emptyset$ (d.h. j ist Neuron der Eingabeschicht) ist $f_{prop_j} : \mathbb{R} \to \mathbb{R}$ die Identitätsfunktion.*

- $\theta : N \to \mathbb{R}$ *ordnet jedem $j \in N$ einen* Schwellenwert θ_j *zu.*
- $\varphi : N \to (\mathbb{R} \times \mathbb{R} \to \mathbb{R})$ *ordnet jedem $j \in N$ eine* Aktivierungsfunktion φ_j *zu, die definiert ist als:*

$$\varphi_j : \mathbb{R} \times \mathbb{R} \to \mathbb{R},\ (net_j, \theta_j) \mapsto o_j$$

Das Zielelement o_j wird Aktivierung *des Neurons j genannt.*

Bemerkung 4.4. *Als Propagierungsfunktion für alle Neuronen $j \in N$ mit mit $I = \{i \mid (i,j) \in V\} \neq \emptyset$ (d.h. j ist kein Neuron der Eingabeschicht), wird in der Regel die gewichtete Summe verwendet:*

$$net_j = \sum_{i \in I} o_i \cdot w_{ij},$$

wobei o_i die Aktivierungen der Neuronen aus I und w_{ij} die entsprechenden Verbindungsgewichte sind.

Als Aktivierungsfunktion φ_j können verschiedene Funktionstypen verwendet werden. Im Allgemeinen sind Aktivierungsfunktionen monoton steigend. Laut Haykin [11] sind die binäre Schwellenwertfunktion, die lineare Identitätsfunktion, die logistische Funktion und die Tangens Hyperbolicus Funktion die am häufigsten benutzten Aktivierungsfunktionen. In Abbildung 4.3 sind alle vier Aktivierungsfunktionen abgebildet.

Die Schwellenwertfunktion (siehe Gleichung 4.1) nimmt nur zwei Werte an. Sie wechselt am Schwellenwert von einem Wert auf den anderen, ist aber ansonsten konstant. Dies impliziert, dass sie am Schwellenwert nicht differenzierbar ist und die Ableitung ansonsten gleich Null ist.

$$\varphi(x) = \begin{cases} -1 & , x < 0 \\ 1 & , x \geq 0 \end{cases} \tag{4.1}$$

Die lineare Identitätsfunktion (siehe Gleichung 4.2) ist stetig differenzierbar.

$$\varphi(x) = x \tag{4.2}$$

Die logistische Funktion (siehe Gleichung 4.3) und die Tangens Hyperbolicus Funktion (siehe Gleichung 4.4) sind beschränkte, stetig differenzierbare, reelle Funktion, die zu der Klasse der Sigmoidfunktionen gehören.

$$\varphi(x) = \frac{1}{1 + e^{-x}} \tag{4.3}$$

$$\varphi(x) = \tanh(x) \tag{4.4}$$

Das interessanteste Merkmal neuronaler Netze besteht in ihrer Fähigkeit, sich Problemen durch Training vertraut zu machen und, nach ausreichendem Training, auch zu untrainierten Werten der zu lernenden Funktion die Lösung bestimmen zu können. Als „Training“ eines neuronalen Netzes wird der Prozess beschrieben, bei dem die freien Einstellungsparameter des neuronalen Netzes, also Verbindungsgewichte oder Schwellenwerte von Neuronen, eingestellt werden. Für das Training werden eine *Trainingsmenge*, eine *Fehlerfunktion* und ein *Lernverfahren* benötigt.

Definition 4.5 (Trainingsmenge)**.** *Für ein neuronales Netz $\mathcal{N}$ mit Eingabedimension m und Ausgabedimension n ist eine* Trainingsmenge *eine Menge $P \subseteq \mathbb{R}^m \times \mathbb{R}^n$. Ein $p = (e_p, t_p) \in P$ heißt* Trainingsdatensatz*, bestehend aus einer Eingabe $e_p \in \mathbb{R}^m$ und einem Ziel $t_p \in \mathbb{R}^n$. Die von einem neuronalen Netz berechnete Ausgabe zu der Eingabe e_p wird mit $\mathcal{N}(e_p) \in \mathbb{R}^n$ bezeichnet.*

Die Anzahl der Trainingsdatensätze in einer Trainingsmenge sollte größer sein als die Anzahl der freien Einstellungsparameter λ, da sonst der Effekt des sogenannten „auswendig Lernens“ auftreten kann. Gibt es weniger Gleichungen als Parameter, so liegt ein unterbestimmtes Gleichungssystem vor. Das neuronale Netz kann auf die Trainingsmenge perfekt eingestellt werden, so dass zu jedem Trainingsdatensatz (e_p, t_p) das Ziel korrekt berechnet wird, $\mathcal{N}(e_p) = t_p$. Das neuronale Netz hat aber nur die Trainingspunkte erlernt, anstatt die Systematik zu erlernen. Der Test mit einer anderen Menge von Datensätzen würde zeigen, dass das neuronale Netz möglicherweise schlecht eingestellt ist.

Die Topologie des neuronalen Netzes (Anzahl der Neuronen und Anzahl der Schichten) sollte der Größe der Trainingsmenge angepasst werden, und umgekehrt. In Beispiel 4.6 wird anhand einer Sinus-Funktion gezeigt wie sich bzgl. einer festen Trainingsmenge die Veränderung der Topologie auf die Güte des neuronalen Netzes auswirkt.

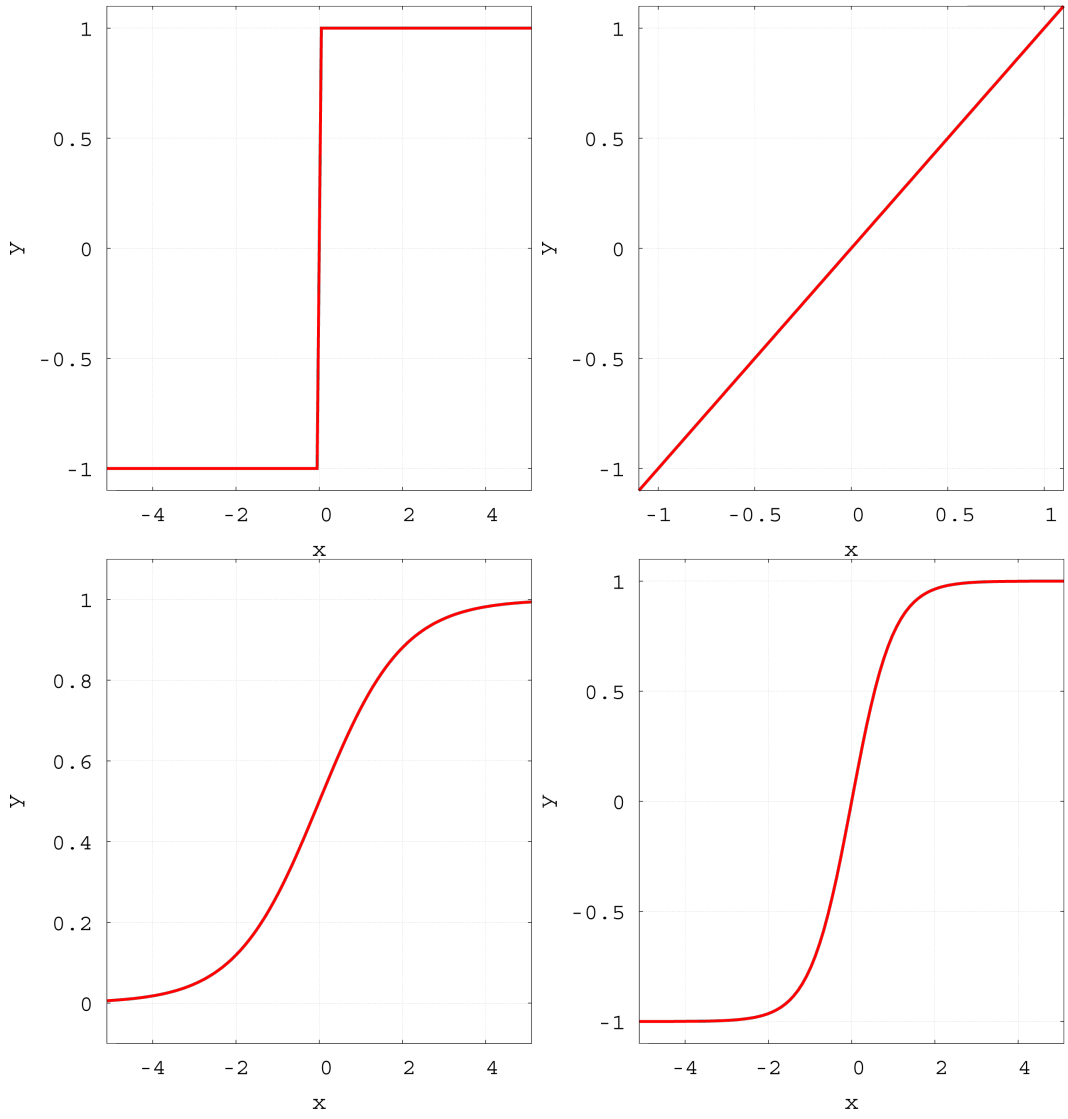


Abbildung 4.3: Graphen zu den vier verschiedenen Aktivierungsfunktionen. Oben sind die Schwellenwertfunktion (links) und die lineare Identitätsfunktion (rechts) abgebildet. Unten sind die logistische Funktion (links) und die Tangens Hyperbolicus Funktion dargestellt (rechts).

Beispiel 4.6 (Sinusproblem aus [2])**.** *Für die Funktion*

$$h : \mathbb{R} \to \mathbb{R}, x \mapsto 0.5 + 0.4\sin(2\pi x).$$

soll ein neuronales Netz gefunden werden, welches h bestmöglich annähert. Dazu wird für 21 äquidistante $x_i \in [0, 1] \subset \mathbb{R}$ eine Trainingsmenge generiert, indem die Funktionswerte $h(x_i)$ bestimmt und mit Gaußscher Normalverteilung ($\sigma = 0.5$) verrauscht werden, um Messwerte zu simulieren. Abbildung 4.4 zeigt die Trainingsmenge und die Funktion h. Es werden drei verschiedene zweischichtige Perzeptrone trainiert und deren

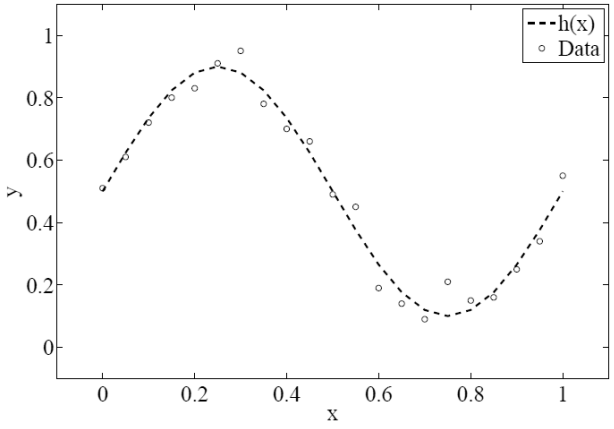


Abbildung 4.4: Datenmenge des Sinusproblems und die zugehörige Sinusfunktion h.
Abbildung aus [17]

Güte verglichen. Da die Funktion h zu einem eindimensionalem Eingabewert $x \in \mathbb{R}$ einen eindimensionalen Ausgabewert $h(x) = y \in \mathbb{R}$ erstellt, haben die zugehörigen neuronalen Netze eine Eingangs- und Ausgangsdimension von eins. Die Anzahl der Neuronen in der versteckten Schicht werden von 1 auf 2 und weiter auf 10 erhöht, so dass sich die Anzahl der freien Parameter von 4 auf 7 und weiter auf 31 erhöht.

Die Neuronen der versteckten Schichten erhalten die Tangens Hyperbolicus Funktion sowie die Neuronen der Ausgabeschicht haben die lineare Identitätsfunktion als Aktivierungsfunktion. Die neuronalen Netze wurden mit der Quasi-Newton Methode und der Mean Squared Fehlerfunktion trainiert (Definition siehe weiter unten). In Abbildung 4.5 sind die neuronalen Netze und ihre durch die Trainingsmenge antrainierte Funktionen abgebildet.

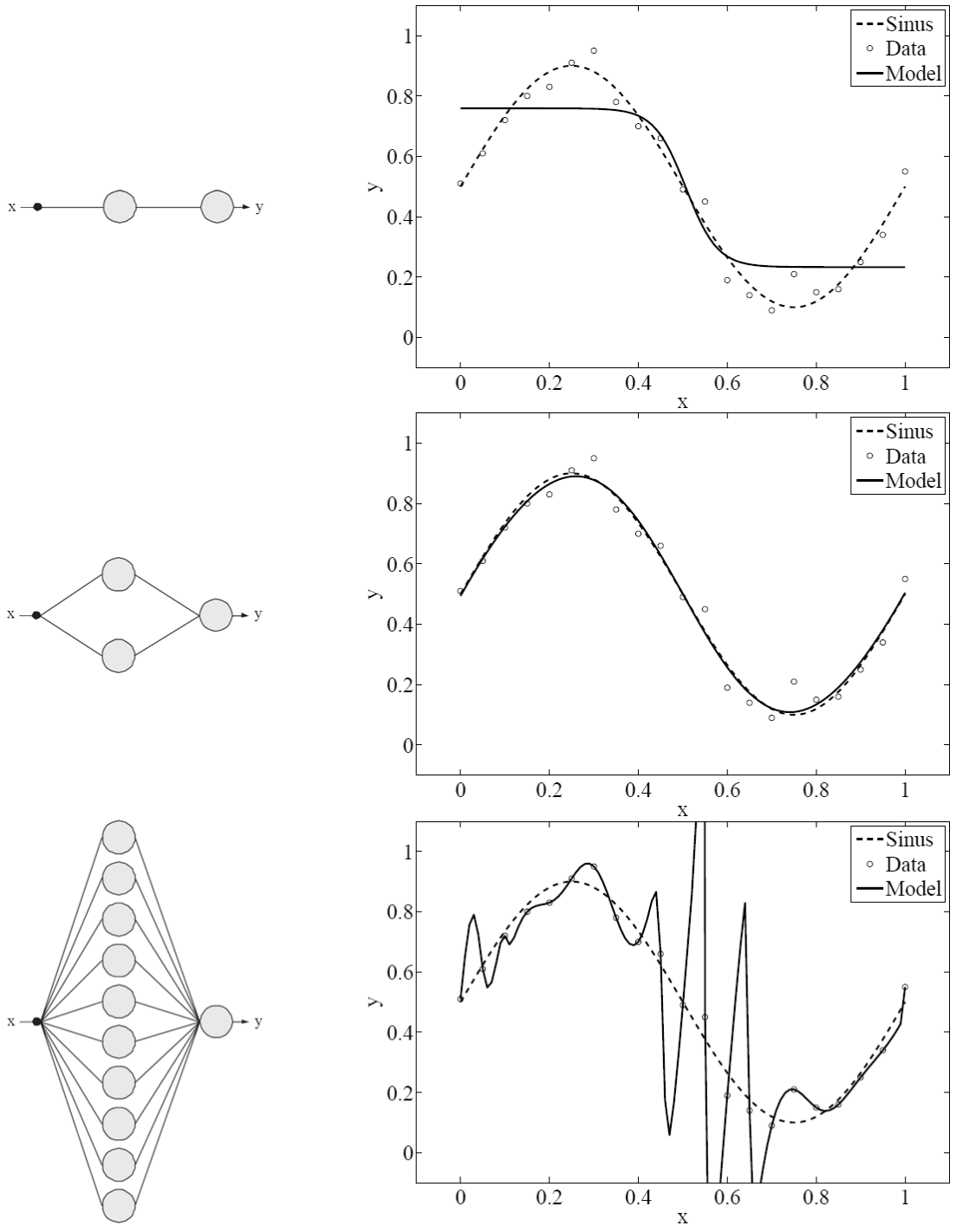


Abbildung 4.5: Sinusproblem: Für eine fest definierte Trainingsmenge von 21 Datensätzen wurden die abgebildeten neuronalen Netze mit 1, 2 und 10 Neuronen in der versteckten Schicht trainiert. Das von den neuronalen Netzen gelieferte Ergebnis wird in den Abbildungen rechts dargestellt.

Abbildungen aus [17]

Das erste neuronale Netz (Abbildung 4.5 oben) ist zu klein gewählt um eine adäquate Annäherung zu erhalten, was sich an dessen Funktionsverlauf erkennen lässt. Das zweite neuronale Netz (Abbildung 4.5 mitte) hat genau die richtige Größe, da es die Sinusfunktion h sehr gut approximiert. Das dritte neuronale Netz (Abbildung 4.5 unten) ist zu groß gewählt. Die erstellte Funktion verläuft zwar mit nur geringer Abweichung durch alle Trainingspunkte, aber für neue x-Werte ist der Fehler sehr groß, was darauf hindeutet, dass dieses Netz nur „auswendig gelernt" hat.

Für jeden Trainingsdatensatz (e_p, t_p) kann die Ausgabe des neuronalen Netzes (Aktivierung der Ausgabeneuronen) $\mathcal{N}(e_p)$ direkt mit dem Ziel t_p verglichen werden. Um für eine Trainingsmenge den entstehenden Fehler zwischen den Ausgaben $\mathcal{N}(e_p)$ und Zielen t_p zu bestimmen, wird eine Fehlerfunktion verwendet. Anhand des errechneten „Fehlers" lässt sich die aktuelle Einstellung des neuronalen Netzes bewerten.

Definition 4.7 (Fehlerfunktion)**.** *Für ein neuronales Netz $\mathcal{N}$ und eine Trainingsmenge P ist die* Fehlerfunktion *E eine Funktion vom Typ*

$$E : \mathcal{N} \times P \to \mathbb{R}^+.$$

Bemerkung 4.8. *Wir betrachten die folgenden sechs stetig differenzierbaren Fehlerfunktionen für ein mehrschichtiges Perzeptron $\mathcal{N} = (N, V, w, f_{prop}, \theta, \varphi)$ bezüglich einer Trainingsmenge $P \subseteq \mathbb{R}^m \times \mathbb{R}^n$: Für einen Vektor $x \in \mathbb{R}^m$ bezeichne $x^{(k)}$ für ein $k \in \{1, \ldots, m\}$ den k-ten Eintrag des Vektors, sowie $|x|$ den Betrag des Vektors x und $|P|$ die Anzahl der Trainingsdatensätze in P. Bezeichne $\bar{t} \in \mathbb{R}^m$ den Durchschnittszielvektor über P mit $\bar{t}^{(k)} = \frac{\sum_{p \in P} t_p^{(k)}}{|P|}$. Sei $F(\mathcal{N}) = \{\theta_i \mid i \in N \backslash L^{(0)}\} \cup \{w_{ij} \mid (i,j) \in V\}$ die Menge der Werte der freien Parameter in $\mathcal{N}$, sowie $|F(\mathcal{N})|$ die Anzahl der freien Parameter.*

- *Sum Squared Fehlerfunktion*

$$E_{SSE}(\mathcal{N}, P) := \sum_{p \in P} |\mathcal{N}(e_p) - t_p|^2$$

- *Mean Squared Fehlerfunktion*

$$E_{MSE}(\mathcal{N}, P) := \frac{E_{SSE}(\mathcal{N}, P)}{|P|} = \frac{\sum_{p \in P} |\mathcal{N}(e_p) - t_p|^2}{|P|}$$

- *Root Mean Squared Fehlerfunktion*

$$E_{RMSE}(\mathcal{N}, P) := \sqrt{E_{MSE}(\mathcal{N}, P)} = \sqrt{\frac{\sum_{p \in P} |\mathcal{N}(e_p) - t_p|^2}{|P|}}$$

- *Normalized Squared Fehlerfunktion*

$$E_{NSE}(\mathcal{N}, P) := \frac{E_{SSE}(\mathbb{N}, P)}{\sum_{p \in P} |t_p - \bar{t}|^2} = \frac{\sum_{p \in P} |\mathcal{N}(e_p) - t_p|^2}{\sum_{p \in P} |t_p - \bar{t}|^2}$$

- *Minkowski Fehlerfunktion für ein* $r \in \mathbb{R}_{\geq 1}$

$$E_{ME}(\mathcal{N}, P) := \sum_{p \in P} \sum_{k=1}^{m} \left| \mathcal{N}(e_p)^{(k)} - t_p^{(k)} \right|^r$$

- *Regularized Minkowski Fehlerfunktion für* $r, s \in \mathbb{R}^+$

$$E_{RME}(\mathcal{N}, P) := r \cdot E_{MSE}(\mathcal{N}, P) + s \cdot \frac{\sum_{w \in F(\mathcal{N})} w^2}{|F(\mathcal{N})|}$$

Für $r = 2$ *entspricht die Minkowski Fehlerfunktion der Sum Squared Fehlerfunktion.*

Mit der Fehlerfunktion liegt ein Maß für die Bewertung der Parameterbelegung des neuronalen Netzes vor. Nach Lopez [17] lässt sich eine Fehlerfunktion E zu einer Funktion f reduzieren, die den Fehlerfunktionswert in Abhängigkeit der Parameterbelegung des mehrlagigen Perzeptrons bestimmt.

Definition 4.9. *Gegeben seien* N, V, f_{prop}, φ*, eine Trainingsmenge* P *und eine Fehlerfunktion* E*. Sei* $\lambda_1 = |N \backslash L^{(0)}|$ *sowie* $\lambda_2 = |V|$*. Die zu* E *assoziierte Funktion* f *ist definiert als*

$$f : \mathbb{R}^{\lambda_1} \times \mathbb{R}^{\lambda_2} \to \mathbb{R}, \ (\theta^*, w^*) \mapsto E((N, V, w^*, f_{prop}, \theta^*, \varphi), P)$$

Aus der Definition ist ersichtlich, dass das Minimum einer Fehlerfunktion E dem Minimum der zu E assoziierten Funktion f entspricht. Mit einem Lernverfahren soll nun die Parameterbelegung $(\theta^*, w^*) \in \mathbb{R}^{\lambda_1} \times \mathbb{R}^{\lambda_2}$ gefunden werden, bei der der Fehlerfunktionswert $f(\theta^*, w^*)$ minimal wird. Das Lernverfahren ist also ein Optimierungsproblem,

bei der durch Variation der freien Parameter des neuronalen Netzes das Optimum der Fehlerfunktion gesucht ist.

Das Training startet mit einer zufälligen Initialisierungsbelegung $(\theta^{(0)}, w^{(0)})$. Im i-ten Iterationsschritt, auch *Epoche* genannt, wird die nächste Belegung $(\theta^{(i+1)}, w^{(i+1)})$ durch die Bildung eines Erweiterungsvektors $\Delta(\theta^{(i)}, w^{(i)})$ durch das Lernverfahren bestimmt, der zu der aktuellen Belegung addiert wird:

$$(\theta^{(i+1)}, w^{(i+1)}) = (\theta^{(i)}, w^{(i)}) + \Delta(\theta^{(i)}, w^{(i)})$$

Der Erweiterungsvektor $\Delta(\theta^{(i)}, w^{(i)})$ wird so gewählt, dass sich der Fehlerfunktionswert in jedem Schritt reduziert

$$f(\theta^{(i+1)}, w^{(i+1)}) \leq f(\theta^{(i)}, w^{(i)}).$$

Ausgehend von der Initialisierungsbelegung wird so eine Sequenz von Parameterbelegungen $(\theta^{(0)}, w^{(0)}), (\theta^{(1)}, w^{(1)}), (\theta^{(2)}, w^{(2)}), \ldots \in \mathbb{R}^{\lambda_1} \times \mathbb{R}^{\lambda_2}$ erzeugt. Das Training kann nach zuvor definierten Bedingungen beendet werden, wie beispielsweise einer maximalen Anzahl von Epochen. Es gibt eine Vielzahl an Lernverfahren, die sich nach Press [21] entsprechend der benutzten Fehlerfunktionsordnung unterteilen lassen.

- Lernverfahren der 0. Ordnung sind globale Optimierungsmethoden, die nur die Fehlerfunktion benutzen. Die bekanntesten Lernverfahren dieser Klasse sind laut Goldberg [8] und Fogel [6] *Random Search* und *Evolutionäre Algorithmen.*

- Lernverfahren der 1. Ordnung sind lokale Optimierungsmethoden, die die Fehlerfunktion und ihren Gradienten verwenden. Laut Battiti [1] gehören zu diesen Verfahren *Gradient Descent*, *Conjugate Gradient* und die *Quasi-Newton Methode.*

- Lernverfahren der 2. Ordnung sind lokale Optimierungsmethoden, die die Fehlerfunktion, ihren Gradienten und ihre Hesse-Matrix verwenden. Zu diesen Verfahren gehören die *Newton Methode* und der *Levenberg-Marquardt Algorithmus*, vgl. [10].

Auf die einzelnen Lernverfahren wird an dieser Stelle nicht weiter eingegangen, sondern auf die angegebene Literatur verwiesen.

Nachdem neuronale Netze trainiert wurden, wird noch ein Referenzwert benötigt um die neuronalen Netze untereinander zu vergleichen. Wie im Sinusproblem-Beispiel 4.6 gezeigt, kann ein neuronales Netz auf der Trainingsmenge sehr gute Ergebnisse erzielen. Aber andere Werte außerhalb der Trainingsmenge werden nicht so gut approximiert (in dem Beispiel galt dies für das neuronale Netz mit 10 Neuronen in der versteckten Schicht). Um die Güte eines trainierten neuronalen Netzes zu bestimmen, bedarf es also einer zweiten, von der Trainingsmenge verschiedenen, Datenmenge, einer sog. *Validierungsmenge* $P' \subseteq \mathbb{R}^m \times \mathbb{R}^n$. Wie die Trainingsmenge besteht P' aus Datensätzen $p = (e_p, t_p) \in P'$ mit Eingabe $e_p \in \mathbb{R}^m$ und Ziel $t_p \in \mathbb{R}^n$.

Definition 4.10 (Validierungsfehler)**.** *Für ein neuronales Netz $\mathcal{N}$, eine Fehlerfunktion E und einer Validierungsmenge P' wird $e := E(\mathcal{N}, P') \in \mathbb{R}^+$ als* Validierungsfehler *bezeichnet.*

Mit dem Validierungsfehler können trainierte neuronale Netze verglichen werden und damit eine Aussage über die Güte eines neuronalen Netzes gemacht werden. Wir betrachten die folgenden zwei Validierungs-Fehlerfunktionen bezüglich einer Validierungsmenge P':

Bemerkung 4.11. *Für einen Vektor $x \in \mathbb{R}^m$ bezeichne $|x|$ den Betrag des Vektors x, sowie $|P'|$ die Anzahl der Trainingsdatensätze.*

- *Relativer Validierungsfehler* $\quad E_{rel}(\mathcal{N}, P') := \dfrac{\sum_{p \in P'} |\mathcal{N}(e_p) - t_p|}{|P'|}$
- *Maximaler relativer Validierungsfehler* $\quad E_{max}(\mathcal{N}, P') := \max_{p \in P'} |\mathcal{N}(e_p) - t_p|$

4.2 Implementierung des neuronalen Netzes

Mit dem neuronalen Netz sollen off-design Fälle des Kraftwerksblocks berechnet werden, vgl. Kapitel 3.3. Da die off-design Fälle ausschließlich für ColSim benötigt werden, liegt es nahe, das neuronale Netz in die Kostenmodell-Steuerungseinheit zu integrieren.

Die Kostenmodell-Steuerungseinheit wurde um diese Funktion erweitert. Dazu wurde die C++ Klassenbibliothek Flood 2 der Universitat Politècnica de Catalunya Barcelona verwendet, in der Lopez [17] mehrlagiges Perzeptrone mitsamt verschiedener Fehlerfunktionen und Lernverfahren implementiert hat.

Es gibt verschiedene Parameter mit denen sich ein neuronales Netz einstellen lässt:

- Anzahl der Neuronen in den verstecken Schichten
- Aktivierungsfunktion der versteckten Schichten und der Ausgangsschicht (lineare Identitätsfunktion, logistische Funktion, Tangens Hyperbolicus Funktion)
- Lernverfahren (Random Search, Gradient Descent, Conjugate Gradient, Quasi-Newton Methode, Evolutionäre Algorithmen)
- Fehlerfunktion (Sum Squared, Mean Squared, Root Mean Squared, Normalized Squared, Minkowski)
- Trainingsstop-Kriterien
- Größe der Trainingsmenge (als Faktor $\beta > 1$ bezüglich der freien Parameter λ des eingestellten neuronalen Netzes)

Anhand der graphischen Benutzeroberfläche (siehe Abbildung 4.6) können die Einstellungen des neuronalen Netzes gewählt werden. Die in der Datenbank gespeicherten Einstellungen werden bei Programmstart der Kostenmodell-Steuerungseinheit automatisch geladen.

Die implementierte neuronale-Netz-Einheit der Kostenmodell-Steuerungseinheit stellt sicher, dass von Thermoflex nur noch so lange off-design Fälle berechnet werden, bis die zuvor bestimmte Größe der Trainingsmenge an off-design Fällen vorhanden ist. Die Einheit stellt sicher, dass alle off-design Fälle der Trainingsmenge verschieden sind, und gemäß den energietechnischen Vorgaben (wie Dampffeuchterestriktion) berechnet wurden. Ist die Trainingsmenge vollständig, so wird das neuronale Netz mit diesen Datensätzen trainiert und die Kostenmodell-Steuerungseinheit übernimmt die Berechnung der off-design Fälle, was die Rechenzeit auf einen Bruchteil reduziert.

4.3 Empirische Untersuchungen

Das neuronale Netz soll dazu eingesetzt werden, off-design Fälle des Kraftwerksblocks zu berechnen. Diese Maßnahme wird die Rechenzeit enorm reduzieren. Die Berechnung eines off-design Falls unter Thermoflex benötigt auf einem derzeit aktuellen quadcore PC eine Simulationsdauer von ca. 10 Sekunden. Wie in Kapitel 3.3 beschrieben,

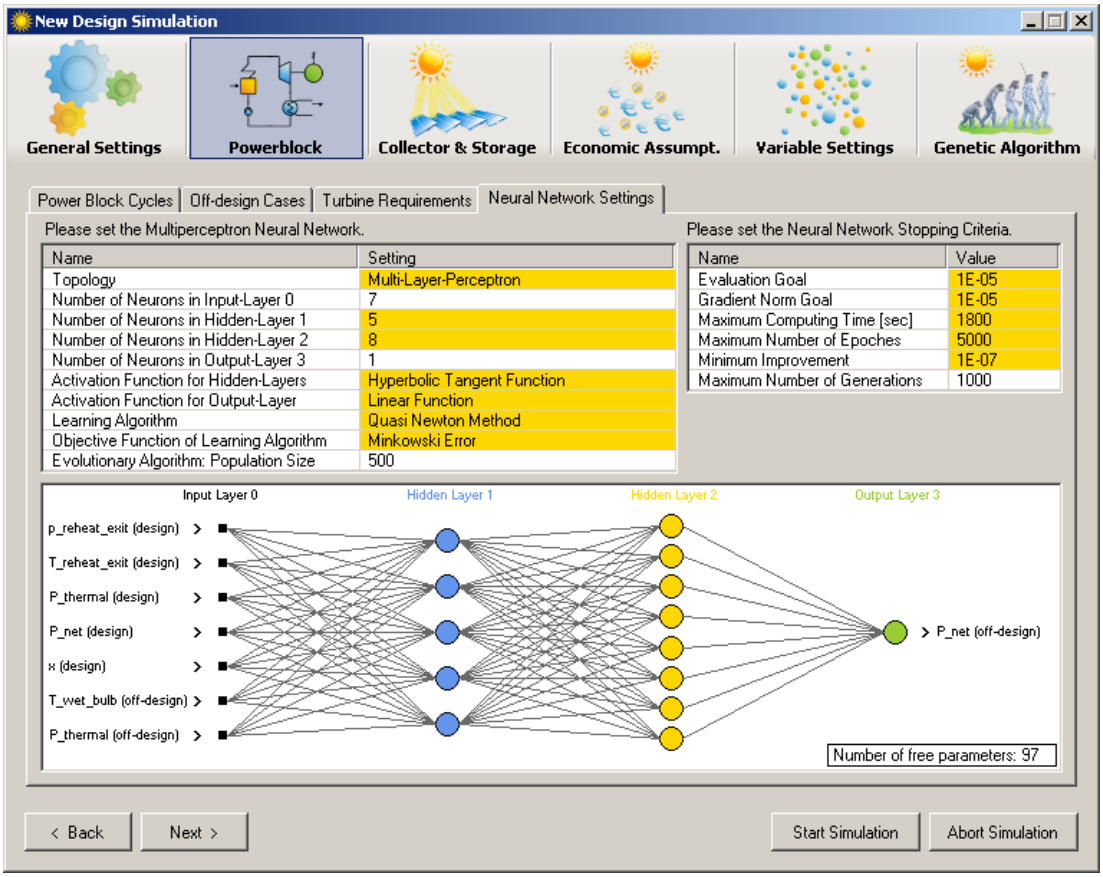


Abbildung 4.6: Einstellung des neuronalen Netzes.

werden derzeit 16 off-design Fälle je Individuum durchgeführt, was einer Rechenzeit von ca. 160 Sekunden je Individuum nur für die off-design Fälle entspricht. Mit einem trainierten neuronalen Netz würde sich diese Rechenzeit stark reduzieren auf weniger als eine Sekunde je Individuum.

Es gilt aber dabei zu untersuchen, wie gut das neuronale Netz die thermodynamischen Zusammenhänge erkennt. Zunächst wird kurz auf die Eingabegrößen des neuronalen Netzes (Abschnitt 4.3.1) eingegangen, bevor die empirischen Untersuchengen vorgestellt werden.

4.3.1 Eingabegrößen des neuronalen Netzes

Das trainierte neuronale Netz soll das Verhalten des ausgelegten Kraftwerks bei verschiedenen Betriebszuständen bestimmen und somit die elektrische Leistung in Abhängigkeit von Umgebungstemperatur und thermischer Last als Ausgabewert berechnen,

vgl. Abbildung 3.7. Es wurden fünf thermodynamische Variablen ausgewählt, bei denen davon ausgegangen wird, dass sie das Verhalten eindeutig beschreiben:

- Zwischenüberhitzerdruck
- Zwischenüberhitzertemperatur
- Thermische Leistung
- Netto-Leistung
- Dampffeuchte

Diese fünf Variablen bilden zusammen mit der Umgebungstemperatur und der thermischen Last die Eingabegrößen in das neuronale Netz (siehe Abbildung 4.6).

4.3.2 Vergleich der Kombinationen aus Lernverfahren und Fehlerfunktion

Wie in Abschnitt 4.1 beschrieben, kann ein neuronales Netz mit einer Trainingsmenge, einer Fehlerfunktion und einem Lernverfahren trainiert werden. Um anschließend trainierte Netze zu validieren und damit untereinander vergleichen zu können, bedarf es einer zweiten Datenmenge, der Validierungsmenge. Mit dem aus der Validierungsmenge bestimmten Validierungsfehler lassen sich die neuronale Netze bewerten.

Laut Lopez [17] muss bei jeder neuen Problemstellung die Kombination aus Lernverfahren und Fehlerfunktion überprüft werden, da sich je nach Kombination unterschiedlich gute Ergebnisse erzielen lassen. In diesem Abschnitt wird in einem Beispiel für das „solarthermische Kraftwerksproblem“ die beste Kombination gesucht. Dazu wurde für einen Kraftwerksprozess eine Simulation erstellt, bei der über 12000 Werte für die Trainingsmenge und Validierungsmenge berechnet wurden. Unter gleichen Bedingungen wie Trainingsmengen-Faktor β und Trainingsstop-Kriterien, wurden 20 zweilagige Perzeptrone mit 1 bis 20 Neuronen in der versteckten Schicht zu je einer Kombination aus Lernverfahren und Fehlerfunktion trainiert. Dabei wurden ihr relativer Validierungsfehler sowie ihr relativer maximaler Fehler bestimmt. Zur Bewertung der Kombination wurde das arithmetische Mittel über die 20 Fehler gebildet. In Abbildung 4.7 ist diese Auswertung dargestellt.

Es lässt sich erkennen, dass hierbei das Lernverfahren einen größeren Einfluss auf die

	Sum Squared Error	Mean Squared Error	Root Mean Squared Error	Normalized Squared Error	Minkowski Error ($r = 1.5$)	Regularized Minkowski ($s = 1.0, r = 0.1$)
Random Search	Validierungsfehler 42.3% Max. Fehler 318.5%	Validierungsfehler 42.3% Max. Fehler 318.5%	Validierungsfehler 42.3% Max. Fehler 318.5%	Validierungsfehler 42.3% Max. Fehler 318.5%	Validierungsfehler 42.3% Max. Fehler 318.5%	Validierungsfehler 42.3% Max. Fehler 318.5%
Evolutionärer Algorithmus	Validierungsfehler 32.1% Max. Fehler 237.0%	Validierungsfehler 32.1% Max. Fehler 237.0%	Validierungsfehler 32.1% Max. Fehler 237.0%	Validierungsfehler 32.1% Max. Fehler 237.0%	Validierungsfehler 31.7% Max. Fehler 229.4%	Validierungsfehler 35.0% Max. Fehler 227.3%
Gradient Descent	Validierungsfehler 10.7% Max. Fehler 107.5%	Validierungsfehler 10.7% Max. Fehler 108.2%	Validierungsfehler 10.7% Max. Fehler 106.1%	Validierungsfehler 10.7% Max. Fehler 108.0%	Validierungsfehler 10.2% Max. Fehler 104.2%	Validierungsfehler 5.5% Max. Fehler 83.6%
Conjugate Gradient	Validierungsfehler 15.4% Max. Fehler 124.7%	Validierungsfehler 20.1% Max. Fehler 124.3%	Validierungsfehler 14.8% Max. Fehler 117.7%	Validierungsfehler 21.8% Max. Fehler 139.0%	Validierungsfehler 23.8% Max. Fehler 139.9%	Validierungsfehler 3.9% Max. Fehler 64.9%
Quasi-Newton Methode	Validierungsfehler 33.5% Max. Fehler 223.0%	Validierungsfehler 21.3% Max. Fehler 141.7%	Validierungsfehler 25.2% Max. Fehler 173.8%	Validierungsfehler 14.8% Max. Fehler 130.8%	Validierungsfehler 40.3% Max. Fehler 297.7%	Validierungsfehler 17.2% Max. Fehler 126.0%

Abbildung 4.7: Vergleich der Lernverfahren-Fehlerfunktion-Kombinationen. Für jede Kombination wurden der relative Validierungsfehler und der relative maximale Fehler ermittelt. Die Kombinationen mit einem Validierungsfehler von unter 25% und einen maximalen Fehler von unter 150% sind grün umrandet.

Güte des neuronalen Netzes hat, als die Fehlerfunktion. Die globalen Verfahren (Random Search und Evolutionärer Algorithmus) haben die theoretisch geringste Wahrscheinlichkeit vorzeitig ein Optimum zu finden, so dass diese im Vergleich zu den lokalen Lernverfahren (Gradient Descent, Conjugate Gradient, Quasi-Newton Methode) eher ungeeignet sind. Es sei an dieser Stelle aber noch einmal darauf hingewiesen, dass es sich hier nur um ein Beispiel handelt, und man keineswegs verallgemeinern darf. Diese Untersuchung soll für das solarthermische Kraftwerksproblem lediglich eine Idee oder Tendenz für eine gute Einstellung zum Trainieren neuronaler Netze liefern.

Bezüglich dieser Beispielsimulation erzielen die Kombination aus dem Conjugate Gradient Lernverfahren mit der Regularized Minkowski Squared Fehlerfunktion die besten

Ergebnisse.

4.3.3 Neuronale Netze für einen Prozess

In diesem Abschnitt wird bezüglich der besten Kombination (siehe Abschnitt 4.3.2) die beste Netztopologie, durch Variation der Neuronenanzahl gesucht. In der Untersuchung werden zwei- und dreischichtige Perzeptrone berücksichtigt, sowie die Trainingsmenge variiert um auch diesen Einfluss zu betrachten. Der Trainingsdatensatz besteht nur aus Daten von *einem* Kraftwerksprozess. Im nächsten Abschnitt 4.3.4 wird auch noch dieser Einfluss überprüft.

Mit Thermoflex wurde zu einem Prozess insgesamt eine Menge von $n \approx 12000$ Datensätzen erstellt, die zu den in Abschnitt 4.3.1 angegebenen Eingabegrößen die elektrische Leistung als Ziel haben. Auf diesem Datensatz sollen zwei- und dreilagige Perzeptrone trainiert werden. Die neuronalen Netze unterscheiden sich durch die Anzahl der Neuronen in den versteckten Schichten. Für die erste versteckte Schicht wurde die Anzahl der Neuronen von 1 bis 20 variiert, für die zweite versteckte Schicht von 0 bis 20. Dabei entsprechen die Fälle von 0 Neuronen zweilagigen Perzeptronen. Jedes neuronale Netz erhielt in Abhängigkeit seiner freien Parameter λ zunächst die 1.1-fache, dann zweifache und anschließend dreifache Menge an Trainingsdatensätzen. Für $\beta \in \{1.1, 2, 3\}$ besteht somit die Trainingmenge aus $\beta \cdot \lambda$ Datensätzen. Die restliche Menge von $(n - \beta \cdot \lambda)$ Datensätzen wurde zum Validieren benutzt. Durch die Bestimmung des Validierungsfehlers sind die neuronalen Netze untereinander vergleichbar. Als Lernverfahren wurde, aufbauend auf den Ergebnissen in Abschnitt 4.3.2 das Conjugate Gradient Lernverfahren mit der Regularized Minkowski Squared Fehlerfunktion verwendet.

Für das Training der neuronalen Netze wurde ein C++ Programm geschrieben. Nach eigenen Hochrechnungen würde die Rechenzeit auf einem 1.3 GHz Computer mehrere Monate dauern. Daher wurden die Rechnungen auf dem Cluster des Rechenzentrums der RWTH Aachen ausgeführt, so dass mit der Rechenleistung von 18 Prozessorkernen die Ergebnisse innerhalb von 3 Tagen zur Verfügung standen.

In den Abbildungen 4.8, 4.9 und 4.10 sind die Ergebnisse für die 1.1-fache, zweifache und dreifache Trainingsmenge (bzgl. der freien Parameter λ) dargestellt. Zwischen den

diskreten Punkten der Neuronenanzahl wurde interpoliert, um mit einer Einfärbung der entstehenden Flächen den Validierungsfehler deutlicher zu machen.

	Trainingsmengenfaktor $\beta = 1.1$					
Beste Netze	(14,10)	(14,18)	(15,17)	(17,17)	(19,18)	(20,18)
Validierungsfehler	1.23%	1.25%	1.21%	1.22%	**1.21%**	1.21%
Max. Validierungsfehler	16.81%	**12.9%**	16.3%	14.8%	14.4%	20.2%

	Trainingsmengenfaktor $\beta = 2$					
Beste Netze	(16,18)	(17,19)	(18,20)	(19,18)	(20,14)	(20,19)
Validierungsfehler	**1.07%**	1.19%	1.20%	1.19%	1.22%	1.15%
Max. Validierungsfehler	12.3%	15.4%	14.4%	15.0%	13.8%	**12.8%**

	Trainingsmengenfaktor $\beta = 3$					
Beste Netze	(8,19)	(13,17)	(15,15)	(16,20)	(19,18)	(20,18)
Validierungsfehler	**1.24%**	1.27%	1.25%	1.26%	1.44	1.21%
Max. Validierungsfehler	15.7%	17.6%	14.1%	13.9%	**11.1%**	20.2%

Tabelle 4.1: Die besten Netze bei einem Kraftwerksprozess mit niedrigsten Validierungsfehlern und niedrigsten maximalen Fehlern für $\beta = 1.1$ (oben), für $\beta = 2$ (mitte), sowie für $\beta = 3$ (unten). Der jeweils beste Validierungsfehler und maximale Fehler sind hervorgehoben.

Erwartungsgemäß stellt sich heraus, dass Netze mit wenigen Neuronen in der ersten oder zweiten versteckten Schicht schlechtere Ergebnisse erzielen, als es für größere Netze der Fall ist. Dies gilt sowohl für die zweilagigen (0 Neuronen in der zweiten Schicht) als auch dreilagigen Perzeptrone. Von wenigen „Ausreißern“ abgesehen, lässt sich die Tendenz erkennen, dass das neuronale Netz mit seiner Größe besser wird. Verwunderlich sind die „Ausreißer“ mit schlechtem Validierungsfehler zwischen recht akzeptablen Ergebnissen. Eine Analyse der Log-Datei, die beim Training der neuronalen Netze Zwischenergebnisse mitgeschrieben hat, lieferte als Erklärung einen vorzeitigen Abbruch des Lernprozesses da ein lokales Optimum gefunden wurde. Das bedeutet, dass an diesen Stellen das lokale Lernverfahren für unser Problem versagt hat.

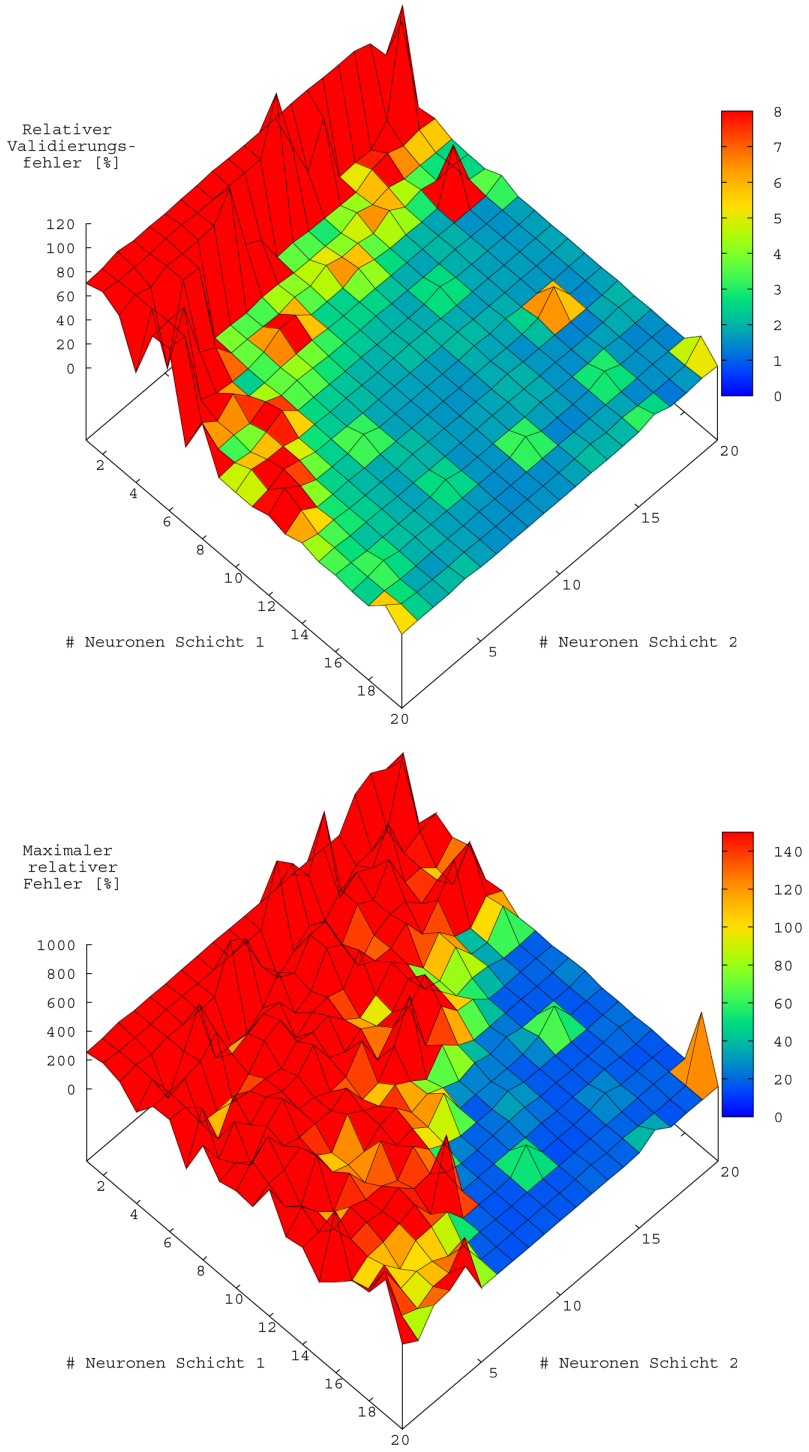


Abbildung 4.8: Relativer Validierungsfehler (oben) und maximaler relativer Validierungsfehler (unten) der neuronalen Netze mit der variierten Anzahl der Neuronen in der ersten und zweiten Schicht bei einem Kraftwerksprozess. Die Größe der Trainingsmenge für jedes neuronale Netz entspricht der 1.1-fachen Menge an freien Parametern.

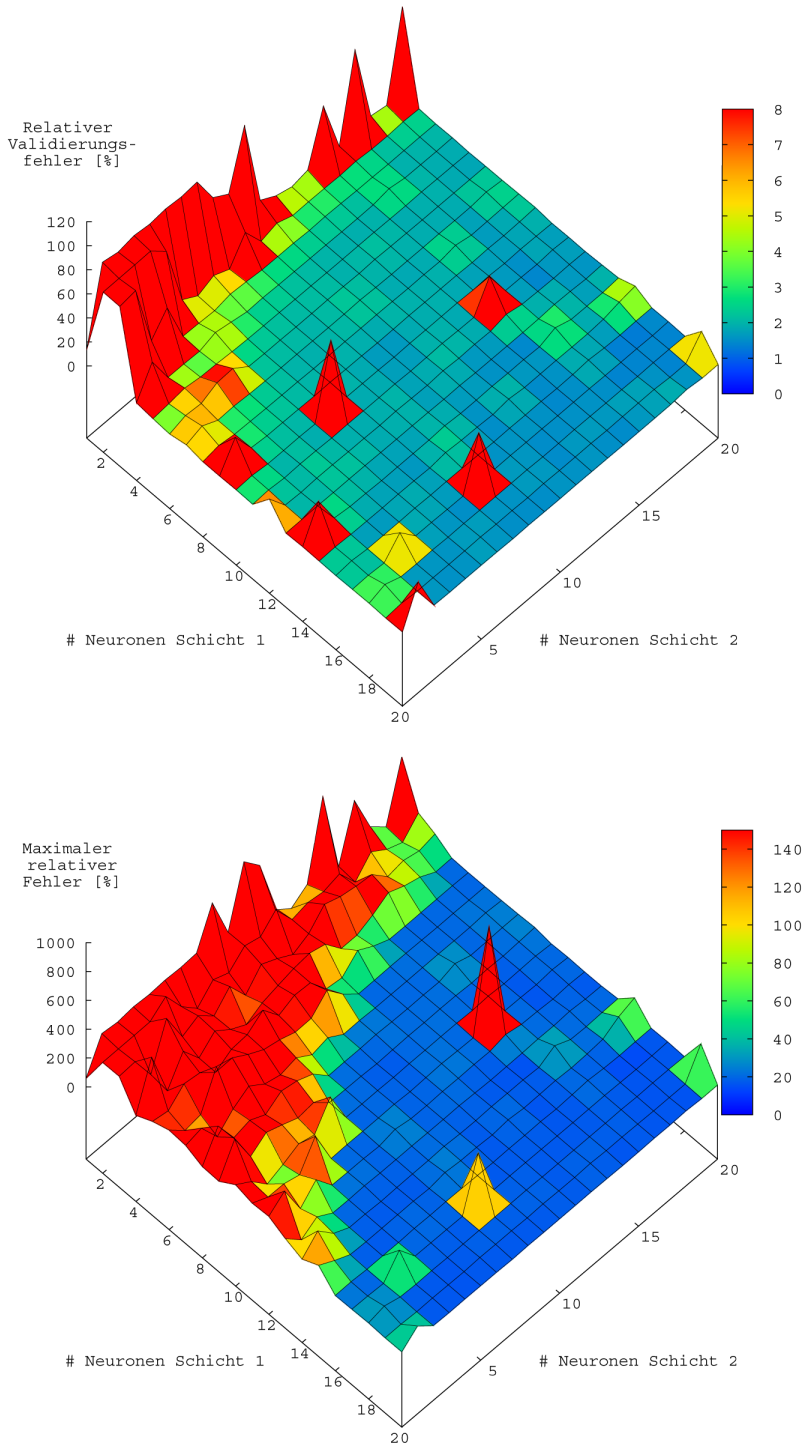


Abbildung 4.9: Relativer Validierungsfehler (oben) und maximaler relativer Validierungsfehler (unten) der neuronalen Netze mit der variierten Anzahl der Neuronen in der ersten und zweiten Schicht bei einem Kraftwerksprozess. Die Größe der Trainingsmenge für jedes neuronale Netz enpricht der zweifachen Menge an freien Parametern.

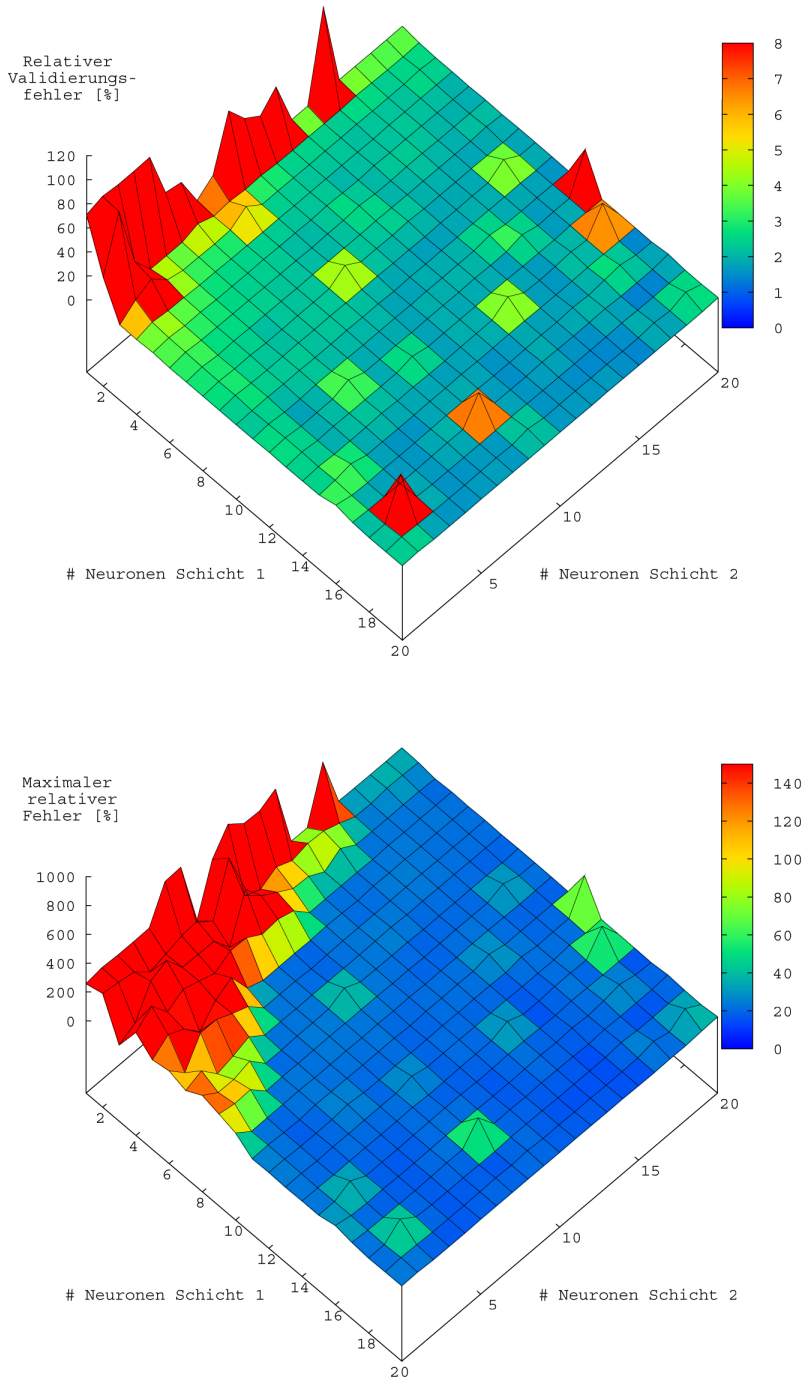


Abbildung 4.10: Relativer Validierungsfehler (oben) und maximaler relativer Validierungsfehler (unten) der neuronalen Netze mit der variierten Anzahl der Neuronen in der ersten und zweiten Schicht bei einem Kraftwerksprozess. Die Größe der Trainingsmenge für jedes neuronale Netz enpricht der dreifachen Menge an freien Parametern.

Trainingsmengenfaktor β	1.1	2	3
Validierungsfehler	1.5%	1.8%	2.3%
Max. Validierungsfehler	18.3%	19.4%	16.0%

Tabelle 4.2: Validierungsfehler für das neuronale Netz mit 14 Neuronen in der ersten und zweiten versteckten Schicht in Abhängigkeit vom Trainingsmengenfaktor β.

Trainingszeit	30 Minuten			60 Minuten		
Trainingsmengenfaktor β	1.1	2	3	1.1	2	3
Validierungsfehler	1.22%	1.33%	1.28%	1.22%	1.22%	1.11%
Max. Validierungsfehler	14.83%	16.38%	16.29%	14.83%	16.26%	14.33%

Tabelle 4.3: Validierungsfehler für das neuronale Netz mit 11 Neuronen in der ersten und 18 Neuronen in der zweiten versteckten Schicht in Abhängigkeit vom Trainingsmengenfaktor β. Den Netzen wurde unterschiedlich viel Zeit zum trainieren gegeben.

Es lässt sich erkennen, dass die Vergrößerung der Trainingsmenge mit $\beta \in \{1.1, 2, 3\}$ einen Einfluss auf die Güte der Netze hat. Mit steigendem β wird der Validierungsfehler zwar teilweise sogar minimal schlechter (siehe Tabelle 4.2), aber dies liegt vermutlich daran, dass Netze mit größerer Trainingsmenge mehr Zeit zum Trainieren benötigen als es für Netze mit kleineren Trainingsmengen der Fall ist. Da die Trainingszeit in den Trainingsstop-Kriterien auf 30 Minuten limitiert wurde, konnten sich die Netze mit großer Trainingsmenge aufgrund der limitierten Zeit nicht weiter verbessern. Um dies zu überprüfen, wurde das neuronale Netz mit 11 Neuronen in der ersten und 18 Neuronen in der zweiten versteckten Schicht für $\beta \in \{1.1, 2, 3\}$ doppelt so lange wie vorher (60 Minuten) trainiert. In Tabelle 4.3 ist zu erkennen, dass das Netz mit dem größten β bei ausreichender Trainingszeit das beste Ergebnis erzielt.

Trainingsmengenfaktor β	1.1	2	3
Validierungsfehler	2.5%	2.0%	1.8%
Max. Validierungsfehler	183.4%	128.0%	14.4%

Tabelle 4.4: Validierungsfehler für das neuronale Netz mit 6 Neuronen in der ersten versteckten Schicht und 9 Neuronen in der zweiten versteckten Schicht in Abhängigkeit vom Trainingsmengenfaktor β.

Bei Betrachtung der maximalen Validierungsfehler in den Abbildungen 4.8, 4.9 und 4.10 lässt sich am besten der Einfluss von β auf die Güte der neuronalen Netze beobachten. Je größer β ist, desto niedriger ist der maximale Fehler, wie es beispielsweise beim

Punkt (6,9) der Fall ist. In Tabelle 4.4 wird für das neuronale Netz mit 6 Neuronen in der ersten versteckten Schicht und 9 Neuronen in der zweiten versteckten Schicht der Fehler in Abhängigkeit von β dargestellt. Die Erhöhung des Faktors β bewirkt zwar nur eine geringe Veränderung des Validierungsfehlers, allerdings eine deutliche Verbesserung des maximalen Fehlers.

Bei näherer Betrachtung der Abbildungen 4.8, 4.9 und 4.10 zum maximalen Validierungsfehler, lässt sich an den Rändern ein „Gebirge“ erkennen, welches entlang einer Grenzkurve abrupt in eine „Tiefebene“ bricht. Die Grenzkurve trennt folglich die schlechteren Ergebnisse ($> 60\%$) von den besseren Ergebnissen. Mit steigendem β nähert sich diese Grenzkurve dem Ursprung. Da die Güte eines neuronalen Netzes maßgeblich durch die Größe der Trainingsmenge bestimmt wird, liegt die Vermutung nahe, dass dies für den Verlauf der Grenzkurve ebenfalls der Fall ist. Die Größe der Trainingsmenge für jedes neuronale Netz wird durch den Trainingsmengenfaktor β und die Anzahl der freien Parameter λ bestimmt. Nach Definition 4.2 ist λ abhängig von der Größe seiner versteckten Schichten. Da die Eingabeschicht mit 7 Neuronen und die Ausgabeschicht mit 1 Neuron schon fest gewählt sind, gilt für ein dreischichtiges Perzeptron (mit also $h_1, h_2 \neq 0$)

$$\begin{aligned} \lambda &= \sum_{i=0}^{2} (h_i + 1) \cdot h_{i+1} \\ &= (7+1) \cdot h_1 + (h_1 + 1) \cdot h_2 + (h_2 + 1) \cdot 1 \\ &= 8 \cdot h_1 + (h_1 + 2) \cdot h_2 + 1. \end{aligned}$$

Für $h_1 \in [1, 20] \subset \mathbb{R}$ sowie $h_2 \in [1, 20] \subset \mathbb{R}$ ist die Funktion

$$\lambda(h_1, h_2) := 8 \cdot h_1 + (h_1 + 2) \cdot h_2 + 1$$

in Abbildung 4.11 dargestellt. In der Darstellung sind drei Höhenlinien zu verschiedenen Parameteranzahlen eingezeichnet. Der Verlauf der Höhenlinie für 250 freie Parameter entspricht in etwa dem Verlauf der Grenzkurve in Abbildung 4.8. Alle neuronalen Netze auf der „Gebirgsseite“ der Grenzlinie, haben für $\beta = 1.1$ weniger als $1.1 \cdot 250 = 275$ Datensätze zum trainieren erhalten. Gleiches gilt für die 135er und 90er Höhenlinie bezüglich der Grenzkurven in Abbildung 4.9 und 4.10. Dort haben die neu-

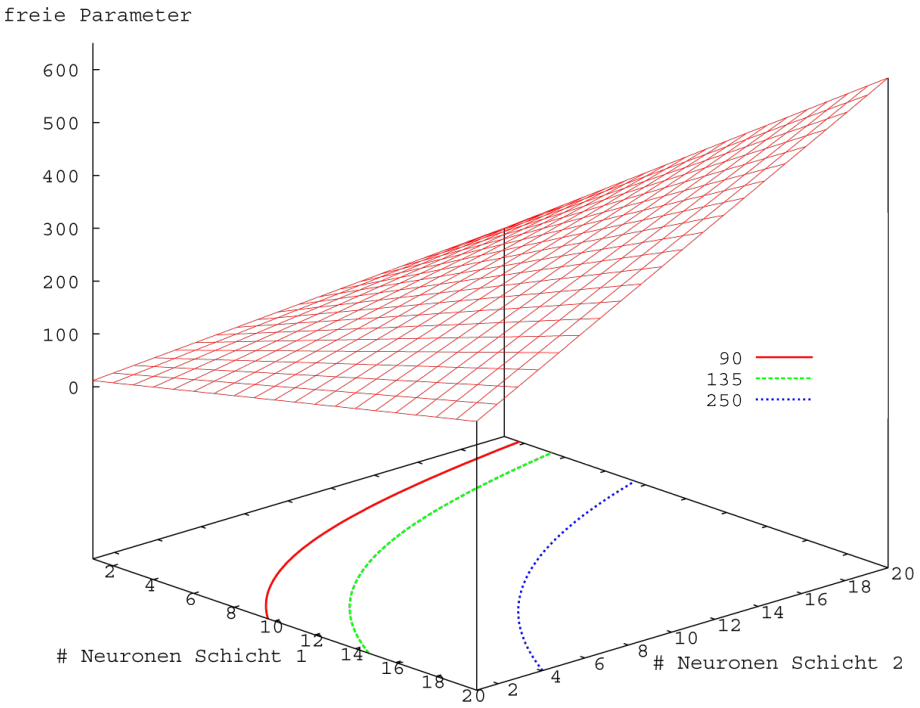


Abbildung 4.11: Darstellung der freien Parameter-Funktion $\lambda(h_1, h_2)$ für verschiedene neuronale Netze mit h_1 Neuronen in der ersten und h_2 Neuronen in der zweiten versteckten Schicht.

ronalen Netze auf der „Gebirgsseite“ der Grenzlinie, für $\beta = 2$ weniger als $2 \cdot 135 = 270$, sowie für $\beta = 3$ weniger als $3 \cdot 90 = 270$ Datensätze zum trainieren erhalten. Alle anderen Netze in der „Tiefebene“ haben eine größere Trainingmenge. Somit lässt sich sagen, dass die Güte eines neuronalen Netzes nicht nur von dem Trainingsmengenfaktor β, sondern auch von der Größe der Datenmenge abhängt. Damit sich ein positiver Effekt auf das neuronale Netz auswirken kann, muss die Trainingsmenge eine gewisse Mindestgröße haben. In dem vorliegenden Fall liegt die nötige Mindestgröße der Trainingsmenge bei ca. 270 Trainingsdaten.

In Tabelle 4.1 sind die besten Netze mit ihrem Fehler für jedes untersuchte β abgebildet. Dass die trainierten Netze nicht noch besser sind, kann verschiedene Gründe haben. Zum einen kann es sein, dass die Netze die verrauschten Daten nicht besser bestimmen können und somit an ihre machbaren Grenzen stoßen. Ein weiterer Grund könnte eine ungünstige Wahl der thermodynamischen Eingabegrößen sein.

4.3.4 Neuronale Netze für mehrere Prozesse

In diesem Abschnitt wird untersucht, ob ein neuronales Netz auch fähig ist, das Verhalten von *mehreren* Kraftwerksprozessen gleichzeitig zu erlernen. Im Unterschied zu Abschnitt 4.3.3 besteht hier der Trainingsdatensatz aus Daten von *drei* verschiedenen Kraftwerksprozessen.

Diese Untersuchung ist deswegen von Interesse, da der in Kapitel 3 vorgestellte Simulationskreislauf auch mehrere Kraftwerksprozesse parallel optimieren kann. Daher ist es wichtig zu wissen, ob und wie gut ein neuronales Netz mehrere Kraftwerksprozesse gleichzeitig abbilden kann. Als achte Eingabegröße wurde die Prozessnummer hinzugefügt, wodurch sich gemäß Definition 4.2 die Anzahl der freien Parameter λ um die Anzahl der Neuronen in der ersten versteckten Schicht erhöht (hier variabel zwischen 1 und 20).

	Trainingsmengenfaktor $\beta = 1.1$					
Beste Netze	(15,19)	(16,19)	(18,9)	(18,19)	(19,20)	(20,14)
Validierungsfehler	1.03%	1.06%	1.16%	**1.01%**	1.07%	1.07%
Max. Validierungsfehler	16.0%	14.1%	**13.2%**	14.6%	14.2%	14.1%

	Trainingsmengenfaktor $\beta = 2$					
Beste Netze	(15,20)	(16,18)	(17,14)	(18,15)	(19,10)	(19,12)
Validierungsfehler	**0.97%**	1.03%	1.27%	0.99%	1.03%	1.05%
Max. Validierungsfehler	16.8%	15.8%	**13.4%**	14.4%	14.0%	14.4%

	Trainingsmengenfaktor $\beta = 3$					
Beste Netze	(17,16)	(17,18)	(19,9)	(19,16)	(20,7)	(20,10)
Validierungsfehler	1.10%	1.10%	1.06%	1.11%	1.17%	**1.06%**
Max. Validierungsfehler	14.5%	13.2%	14.9%	20.2%	**13.1%**	14.1%

Tabelle 4.5: Die besten Netze bei drei Kraftwerksprozessen mit niedrigsten Validierungsfehlern und niedrigsten maximalen Fehlern für $\beta = 1.1$ (oben), für $\beta = 2$ (mitte), sowie für $\beta = 3$ (unten). Der jeweils beste Validierungsfehler und maximale Fehler sind hervorgehoben.

Als Kraftwerksprozesse wurden Prozesse mit einem Zwischenüberhitzer und drei, fünf bzw. sieben Vorwärmern gewählt. Mit Thermoflex wurde zu diesen drei Prozessen insgesamt eine Menge von $n \approx 6000$ Datensätzen erstellt. Auf diesen Datensätzen wurden wie in Abschnitt 4.3.3 zwei- und dreilagige Perzeptrone trainiert, wobei wieder jedes neuronale Netz in Abhängigkeit seiner freien Parameter λ die 1.1-fache, zweifache und

dreifache Menge an Trainingsdatensätzen erhielt. Die restliche Menge von $(n - \beta \cdot \lambda)$ Datensätzen wurde zum Validieren benutzt. Als Lernverfahren wurde, aufbauend auf den Ergebnissen in Abschnitt 4.3.2, das Conjugate Gradient Lernverfahren mit der Regularized Minkowski Squared Fehlerfunktion verwendet.

In den Abbildungen 4.12, 4.13 und 4.14 sind die Ergebnisse für jedes untersuchte β dargestellt. Zwischen den diskreten Punkten der Neuronenanzahl wurde interpoliert, um mit einer Einfärbung der entstehenden Flächen den Validierungsfehler deutlicher zu machen. Für diese neuronalen Netze zeigt sich ein ähnliches Fehlerverhalten, wie für die neuronalen Netze im vorigen Abschnitt 4.3.3. Es gilt, dass Netze mit wenigen Neuronen in den versteckten Schichten schlechtere Ergebnisse erzielen, als es für größere Netze der Fall ist. Die im vorangegangenen Abschnitt erwähnten „Ausreißer“ finden sich auch hier und lassen sich mit der gleichen Begründung erklären. Ebenfalls lassen sich mit steigendem Trainingsmengenfaktor β geringe Veränderungen für den Validierungsfehler und starke Veränderungen für den maximalen Fehler erkennen. In Tabelle 4.5 sind die besten Netze mit ihrem Fehler für jedes untersuchte β abgebildet.

Die Ergebnisse zeigen, dass die Abbildung mehrerer Kraftwerksprozesse mit einem neuronalem Netz gut funktioniert. Es bleibt zu untersuchen, wie gut sie das solarthermische Kraftwerksproblem lösen. Diese Untersuchung wird im folgenden Abschnitt 4.3.5 vorgestellt.

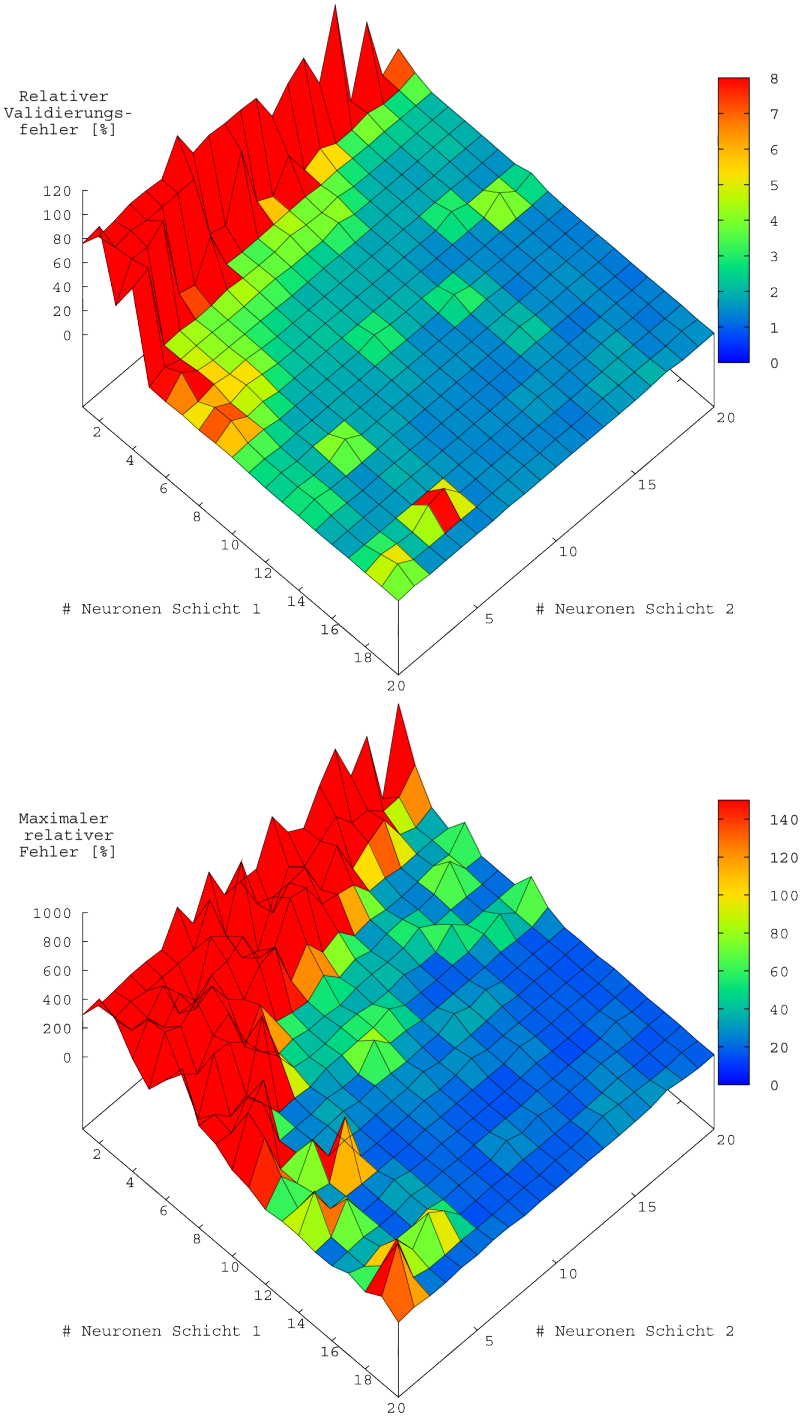


Abbildung 4.12: Relativer Validierungsfehler (oben) und maximaler relativer Validierungsfehler (unten) der neuronalen Netze mit der variierten Anzahl der Neuronen in der ersten und zweiten Schicht bei drei Kraftwerksprozessen. Die Größe der Trainingsmenge für jedes neuronale Netz entspricht der 1.1-fachen Menge an freien Parametern.

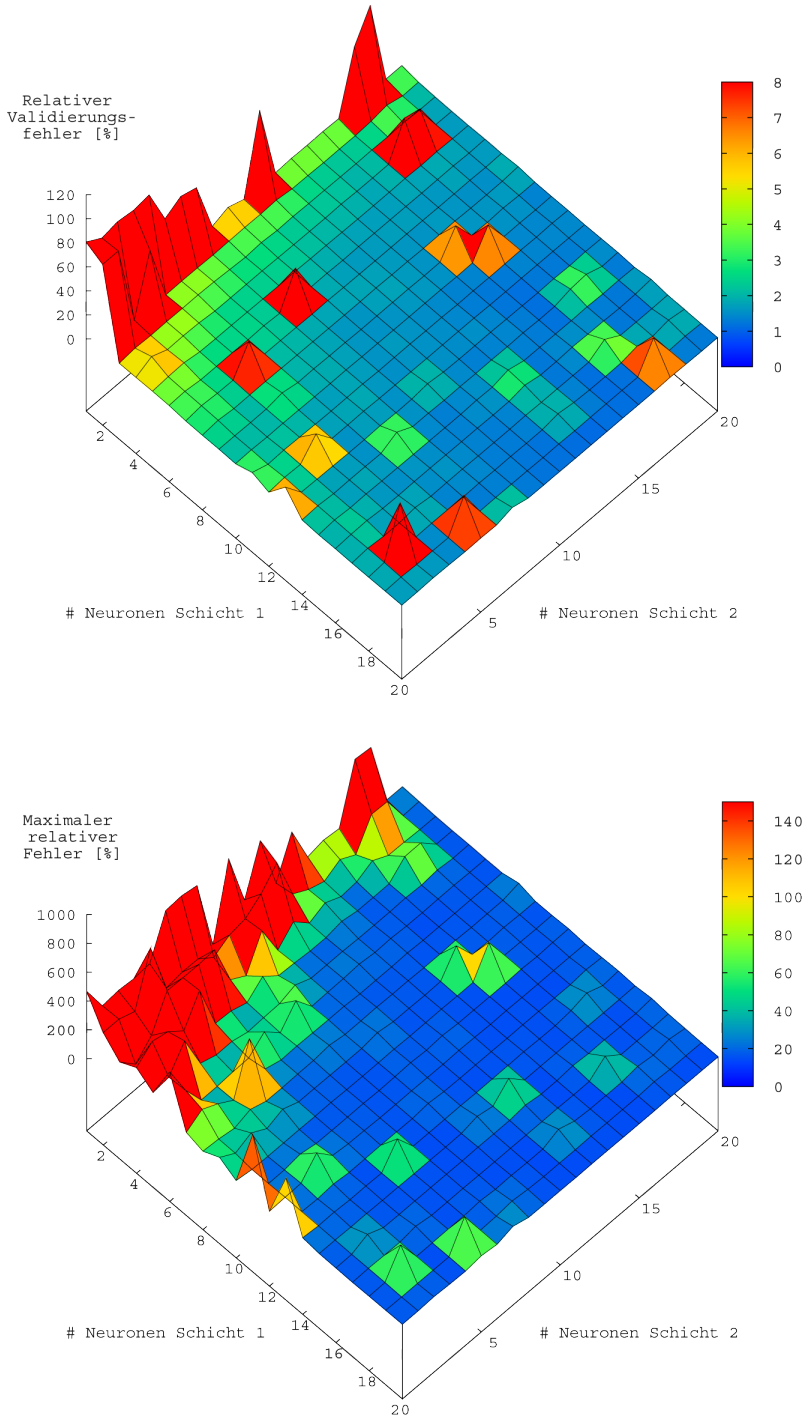


Abbildung 4.13: Relativer Validierungsfehler (oben) und maximaler relativer Validierungsfehler (unten) der neuronalen Netze mit der variierten Anzahl der Neuronen in der ersten und zweiten Schicht bei drei Kraftwerksprozessen. Die Größe der Trainingsmenge für jedes neuronale Netz entspricht der zweifachen Menge an freien Parametern.

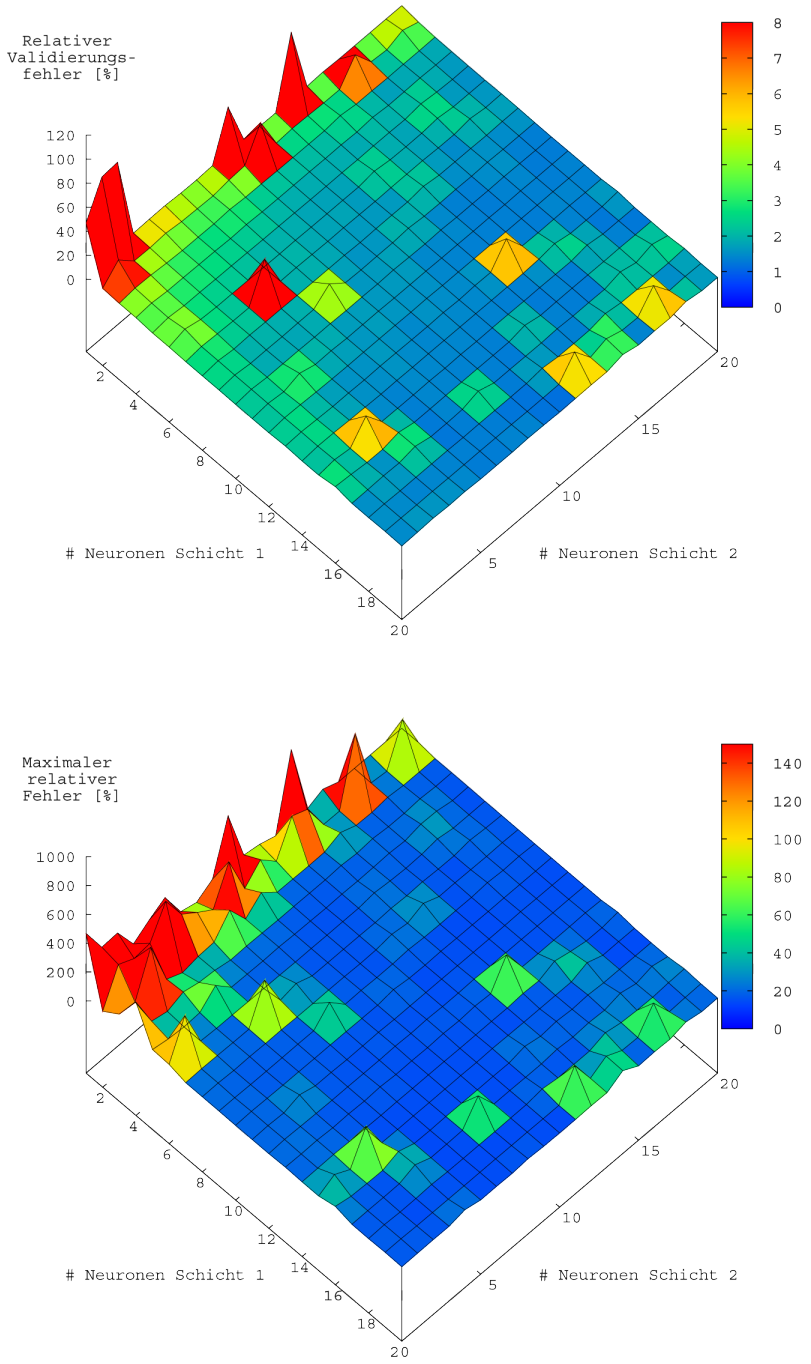


Abbildung 4.14: Relativer Validierungsfehler (oben) und maximaler relativer Validierungsfehler (unten) der neuronalen Netze mit der variierten Anzahl der Neuronen in der ersten und zweiten Schicht bei drei Kraftwerksprozessen. Die Größe der Trainingsmenge für jedes neuronale Netz entspricht der dreifachen Menge an freien Parametern.

4.3.5 Güte neuronaler Netze in der Anwendung

Die off-design Fälle werden benötigt, um das Verhalten des Kraftwerks bei Teillast in Abhängigkeit von der Umgebungstemperatur, zu untersuchen. Da die Berechnung der Fälle unter Thermoflex sehr rechenzeitaufwendig sind (ca. 10 Sekunden je Fall) werden nur einige wenige ausgewählte Fälle in Thermoflex berechnet und anschließend wird von ColSim zwischen diesen bilinear interpoliert, um das gesamte Kennfeld zu erhalten.

In diesem Abschnitt wird untersucht, wie gut dieses Verfahren im Vergleich zu einer Berechnung mit dem neuronalen Netz ist. Um möglichst gut das "echte" Kennfeld abbilden zu können, wurden mit Thermoflex zu drei ausgelegten Kraftwerksprozessen jeweils 520 off-design Fälle berechnet. Die Temperatur wurde in 1 Grad Schritten von 0 auf 25°C und die Last in 3.75% Schritten von 25 auf 100% variiert. Dieses Kennfeld bildet die Referenz, an der wir die Güte anderer Kennfelder messen.

Aufgrund der hohen Rechenzeit für off-design Fälle, erhält ColSim - als Kompromiss zwischen Abbildungsgenauigkeit und Rechenzeit - für eine Simulation die Ergebnisse von nur 16 off-design Fällen. Die Temperaturen und Lasten werden äquidistant in den Intervallgrenzen ausgewählt, so dass für die Umgebungstemperatur $T \in \{0, 8.33, 16.67, 25\}$ und die Last $P_{th} \in \{25, 50, 75, 100\}$ die zugehörige elektrische Leistung $P_{el}(T, P_{th})$ vorliegt. Aus diesen Werten erzeugt ColSim mittels bilinearer Interpolation das Kennfeld.

Das neuronale Netz wird mit off-design Fällen trainiert, die während einer Simulation für ColSim erstellt wurden. Somit besteht die Trainingsmenge aus äquidistanten off-design Fällen. Aufbauend auf den Ergebnissen der vorigen Abschnitte, wird als neuronales Netz ein dreischichtiges Perzeptron mit 16 Neuronen in der ersten und 18 Neuronen in der zweiten versteckten Schicht verwendet. Mit dem Conjugate Gradient Lernverfahren und der Regularized Minkowski Squared Fehlerfunktion wird das neuronale Netz trainiert.

Die Kennfelder der Thermoflex-Daten, der bilinearen ColSim Interpolation und der berechneten Daten des neuronalen Netzes sind in den Abbildungen 4.15 und 4.16 abgebildet.

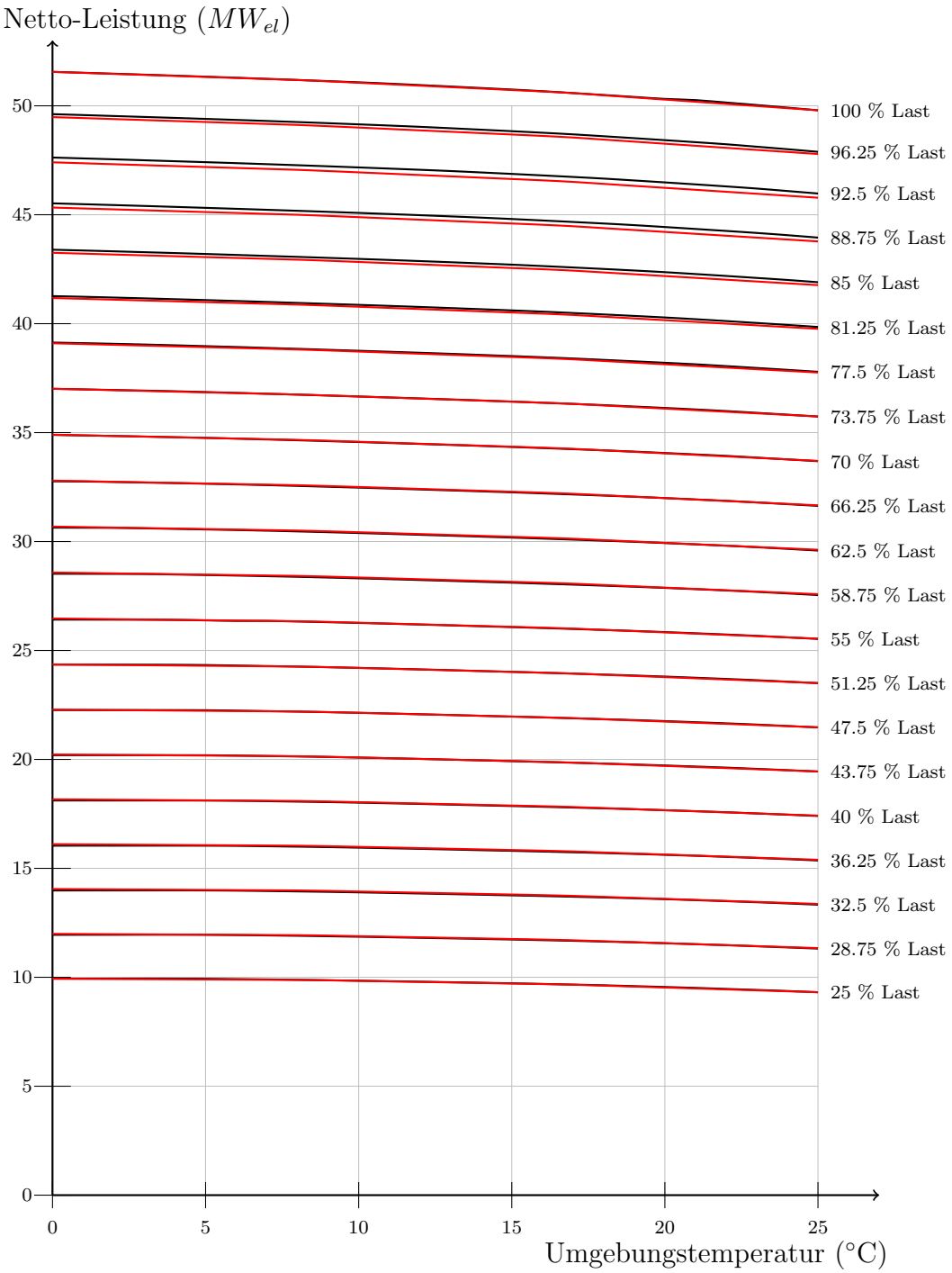


Abbildung 4.15: Von Thermoflex ermitteltes Umgebungstemperatur-Teillast Kennfeld (schwarz), sowie das mittels bilinearer Interpolation durch 16 Punkte ermittelte Kennfeld (rot).

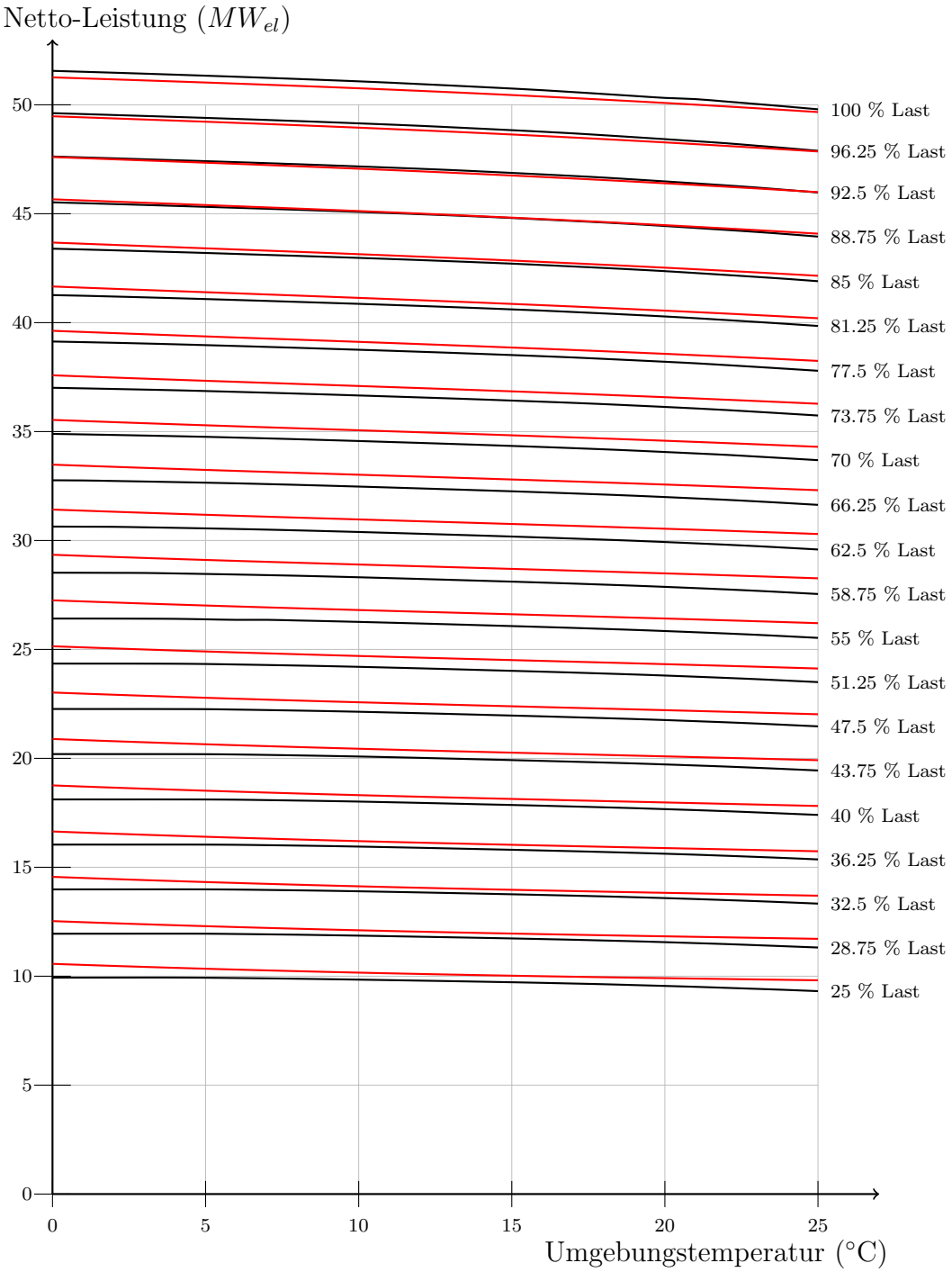


Abbildung 4.16: Von Thermoflex ermitteltes Umgebungstemperatur-Teillast Kennfeld (schwarz), sowie das mittels neuronalen Netzes ermittelte Kennfeld (rot). Die Abweichung bei mittleren Lasten ist höher als bei kleineren oder größeren Lasten.

	Neuronales Netz			ColSim_{16}		
Individuum	1	2	3	1	2	3
Validierungsfehler	1.63%	1.39%	0.88%	0.16%	0.15%	0.13%
Max. Validierungsfehler	6.36%	8.15%	8.59%	0.54%	0.38%	0.56%

Tabelle 4.6: Relativer Validierungsfehler und maximaler relativer Validierungsfehler des neuronalen Netzes und der ColSim-Interpolation für die Berechnung der 520 off-design Fälle je ausgelegtem Kraftwerksprozess.

Die Kennfeldabweichungen der ColSim-Interpolation und neuronalen Netz-Berechnung bzgl. der Thermoflex-Daten, sind in Tabelle 4.6 dargestellt. Es lässt sich erkennen, dass die bilineare Interpolation von ColSim die „echten“ Werte sehr gut approximiert. Auch das neuronale Netz erzielt sehr gute Ergebnisse, die aber die Qualität der bilinearen Interpolation nicht erreichen. Dafür aber liefert das neuronale Netz einen ganz entscheidenden Rechenzeitvorteil. Um die 16 off-design Fälle für ColSim zu generieren, bedarf es ca. 160 Sekunden, während ein bereits trainiertes neuronales Netz für die Berechnung eines off-design Falles weniger als eine Millisekunde benötigt.

Aufgrund dieser sehr guten Approximation durch ColSim, kann die bilineare Interpolation als Referenz für die Güte des neuronalen Netzes betrachtet werden. Das oberste Ziel der Berechnung der off-design Fälle, ist die möglichst genaue Bestimmung der Stromgestehungskosten. In einer weiteren Untersuchung wurden für 806 verschiedene Individuen die Stromgestehungskosten mit Hilfe des neuronalen Netzes ermittelt und mit bilinearen ColSim-Interpolationsergebnissen verglichen. Die Simulation mit Hilfe des neuronalen Netzes liefert sehr gute Ergebnisse. Der relative Fehler bei der Bestimmung der Stromgestehungskosten liegt bei 0.67% und der maximale Fehler bei 1.33%.

Diese recht guten Ergebnisse lassen sich möglicherweise zugunsten des neuronalen Netzes noch weiter verbessern, wenn nicht immer mit den gleichen Punkten für Teillast und Umgebungstemperatur trainiert wird.

Kapitel 5

Zusammenfassung und Ausblick

Im Rahmen der vorliegenden Arbeit wurde auf der Basis bestehender Einzel-Programme eine integrierte Simulationssoftware entwickelt, welche solarthermische Kraftwerke modellieren und deren Stromgestehungskosten optimieren kann. Das Konzept der Software basiert auf einem Client-Server Modell, bestehend aus einem Datenbankserver und drei Clients. Dabei handelt es sich um zwei Steuerungseinheiten für die externen Simulationsprogramme Thermoflex, ColSim und dem Kostenmodellprogramm calculate_LEC, sowie einer Optimierungseinheit. Die Optimierungseinheit erstellt anhand eines genetischen Algorithmus Variablenkonfigurationen, sog. Individuen, die von Thermoflex, ColSim und calculate_LEC über Simulationen bewertet werden. Themoflex simuliert den Kraftwerksblock und ColSim den Solarkollektor und den Speicher. Das Kostenmodell berechnet aus den energetischen Simulationsergebnissen und Kostenmodellen die Stromgestehungskosten. Der Datenaustausch zwischen den Programmmodulen wird über den Datenbankserver organisiert. Mit der im Rahmen dieser Arbeit entstandenen Software kann für einen bestimmten Standort das kostenoptimale solarthermische Kraftwerk ermittelt werden.

Um die Simulationsgeschwindigkeit zu erhöhen, wurde als Erweiterung der Simulationssoftware die Verwendung neuronaler Netze erfolgreich untersucht. Mit einer Testsimulation ließen sich die Stromgestehungskosten mit einer relativen Abweichung von nur 0.67% bestimmen. Die Simulationszeit für die Betriebszustandsberechnung eines Individuums reduziert sich mit dieser Vorgehensweise von ca. 160 Sekunden auf nur noch wenige Millisekunden. Die leicht reduzierte Abbildungsgenauigkeit kompensiert also mit deutlicher Steigerung der Geschwindigkeit. Dies eröffnet wiederum neue Perspektiven zur Simulation neuer komplexerer Sachverhalte.

In der vorliegenden Arbeit wurden exemplarische Untersuchungen zur Einstellung des neuronalen Netzes gemacht, die zum einen auf empirischen Untersuchungen und zum anderen auf Annahmen basieren. Gerade hier besteht noch einiges Optimierungspotential. Beispielsweise wurde die Topologie des neuronalen Netzes von vorneherein auf das mehrlagige Perzeptron festgelegt, bei dem zu Beginn des Trainings die Anzahl der Neuronen fest gewählt wird. Gerade in diesem Bereich gibt es viele Alternativen. Zu nennen seien die so genannten selbstorganisierten Netzwerkmodelle, wie beispielsweise das von Fritzke [7] entwickelte Growing Neural Gas: Hier existieren zu Beginn zwei Startneuronen. Im Laufe der Trainingsphase des Netzes werden neue Neuronen eingefügt, sodass eine adäquate Anpassung des Netzes an die Eingabedaten stattfinden kann. Dabei werden selbstständig Verbindungen zwischen den Neuronen eingefügt und gelöscht.

Als weitere Verbesserungsmaßnahme sei die Anpassung der Trainingsmenge zu nennen. In den bisherigen Untersuchungen wurde das neuronale Netz mit den für ColSim bestimmten 16 off-design Fällen (vier Umgebungstemperaturen zu vier thermischen Lasten) trainiert. Es ist zu erwarten, dass durch ein dichteres Netz der off-design Fälle (beispielsweise 7 mal 7 off-design Fälle), das Kennfeld durch das neuronale Netz noch besser erkannt werden kann. Auch die Kombination des Lernverfahrens mit der Fehlerfunktion wurde nur exemplarisch für wenige neuronale Netze untersucht. Mögliche Einstellungsparamter, wie beispielsweise der Exponent r der Minkoswki Fehlerfunktion, die Populationsgröße des evolutionären Algorithmus oder die Trainingsstopkriterien, wurden nicht benutzt, so dass hier noch weiteres Untersuchungspotential besteht.

Die Simulationsgeschwindigkeit lässt sich weiter erhöhen, wenn man die trainierten neuronalen Netze zu den Kraftwerksprozessen abspeichern würde. Bei späteren Simulationen mit ähnlicher Problemstellung kann dieses neuronale Netz wieder geladen werden. Die Trainingphase, in der Thermoflex off-design Fälle für die Trainingsmenge des neuronalen Netzes berechnet, wäre nicht mehr nötig, so dass die Simulation mit dem fertig trainierten neuronalem Netz startet.

ColSim bestimmt den jährlichen Energieertrag, indem es für jede der 8760 Stunden des Jahres mittels bilinearer Interpolation der 16 gegebenen off-design Fälle den stündlichen Energieertrag berechnet. Diese Stundenwerte kann auch das neuronale Netz

liefern. Es erscheint sinnvoll, das neuronale Netz direkt in ColSim einzubetten. Der Umweg über die bilineare Interpolation wäre nicht mehr nötig, was zu einer Verbesserung der Kennfeldbestimmung führen würde.

Die Stabilität der Simulationssoftware wird durch das Arbeiten auf zwei Betriebssystemen, bei dem der Datenaustausch über das Ablegen von Dateien auf dem Netzlaufwerk geregelt ist, beeinträchtigt. Diese Lösung sollte definitiv nur eine Zwischenlösung darstellen. Zum einen ist es recht unkomfortabel, für das Ausführen einer Software zwei verschiedene Systeme zu benötigen. Zum anderen ist es auch leicht störanfällig und instabil. Als Lösung könnte eine Portierung des Kostenmodellprogramms calculate_LEC sowie des Kollektorsimulationsprogramms ColSim auf das Windows System forciert werden.

Literaturverzeichnis

[1] R. Battiti. First and second order methods for learning: Between steepest descent and newton's method. *Neural Computation*, 4(2):141–166, 1992.

[2] C. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[3] G. Cerbe and H.-J. Hoffmann. *Einführung in die Thermodynamik. Von den Grundlagen zur technischen Anwendung, 13. Auflage.* Hanser Verlag, 2002.

[4] K. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, 1975.

[5] K. Ehrenberg. *Solarthermische Kraftwerke (Teil VI der Reihe Regenerative Energien).* VDI-Gesellschaft Energietechnik, 1997.

[6] D. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1994.

[7] B. Fritzke. Growing cell structures–A self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994.

[8] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, 1988.

[9] T. Gutjahr. Solarthermische Kraftwerke: Simulation und Optimierung von Kraftwerksprozessen unter Verwendung eines geeigneten Optimierungsverfahrens und eines kommerziellen Kreislaufsimulationsprogramms. Diplomarbeit, Fachbereich Maschinen- und Energietechnik, FH Leipzig, 2009.

[10] M. Hagan and M. Menhaj. Training feedforward networks with the marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6):989–993, 1994.

[11] S. Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall, 1999.

[12] G. Hinton, J. McClelland, and D. Rumelhart. *Parallel distributed processing: Explorations in the microstructure of cognition*, volume 1, chapter 3. MIT Press, 1986.

[13] J. Holland. *Adaptation in natural and artificial systems*. MIT Press, 1992.

[14] T. Härder and A. Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys (CSUR)*, 15(4):3–17, 1983.

[15] J. Karl, M. Nixdorf, M. Pogoreutz, and I. Giglmayr. Vergleich von Software zur Thermodynamischen Prozessrechnung. *Technisch wissenschaftlicher Bericht. TU Graz und TU München*, 1999.

[16] A. Kemper and A. Eickler. *Datenbanksysteme. Eine Einführung*. Oldenbourg Wissenschaftsverlag, 2004.

[17] R. Lopez. Flood: An open source neural networks c++ library. www.cimne.com/flood, Universitat Politècnica de Catalunya, Barcelona, 2008.

[18] M. Mertins. *Technische und wirtschaftliche Analyse von horizontalen Fresnel-Kollektoren*. PhD thesis, Universität Karlsruhe, 2009.

[19] G. Morin. Auslegung und Wirtschaftlichkeitsanalyse eines solarthermischen Kraftwerks auf der Basis von linearfokussierenden Fresnel-Kollektoren. Diplomarbeit, Fraunhofer Institut für Solare Energiesysteme, 2003.

[20] G. Morin, T. Gutjahr, W. Platzer, and R. Leithner. Techno-economic design optimisation of solar thermal power plants. 2009.

[21] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical recipes: the art of scientific computing*. Cambridge University Press, 2007.

[22] C. Roeckerath. Populationstool. `http://www.matha.rwth-aachen.de/~roeckerath/Populationstool/start.html`, 2009.

[23] R. Rojas. *Theorie der neuronalen Netze: eine systematische Einführung*. Springer-Verlag, 1993.

[24] W. Thom and F. Klement. Im Brennpunkt: Die Parabel. *Praxis Schule 5-10*, Heft 3:38–41, 1997.

[25] P. Waas, M. Kollera, and D. Castorph. *Skript zur Vorlesung Technische Thermodynamik*. FH München, 1999.

[26] M. Wall. GAlib: A C++ library of genetic algorithm components. `http://lancet.mit.edu/ga/`, 1996.

[27] S. Wittig. *Skript zur Vorlesung Strömungsmaschinen für Wirtschaftsingenieure (WS 2000/2001)*. Universität Karlsruhe, 2000.

[28] C. Wittwer. *ColSim Simulation von Regelungssystemen in aktiven Solarthermischen Anlagen*. PhD thesis, Universität Karlsruhe, 1998.

[29] R. Zahoransky. *Energietechnik, 3.* Aufl., Vieweg, Wiesbaden, 2007.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich diese Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen meiner Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Dasselbe gilt sinngemäß für Tabellen und Abbildungen. Diese Arbeit hat in dieser oder einer ähnlichen Form noch nicht im Rahmen einer anderen Prüfung vorgelegen.

Aachen, im Dezember 2009

Pascal Richter