

Rekonstruktion genregulatorischer Netzwerke mit Evolutionären Algorithmen

Hannes Planatscher

12. September 2003

Ich danke im Besonderen meinem Betreuer Felix Streichert für seine engagierte Betreuung und Geduld, Holger Ulmer, Christian Spieth, Dr. Markus Schwehm, Professor Dr. Andreas Zell und allen Mitarbeitern des Lehrstuhles für Rechnerarchitektur an der Fakultät für Kognitionswissenschaften in Tübingen, weiters meinen Eltern Annie und Karl Planatscher für die moralische und finanzielle Unterstützung, und natürlich Verena für alles andere.

Ich erkläre, daß ich die Studienarbeit 'Rekonstruktion genregulatorischer Netzwerke mit Evolutionären Algorithmen' selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und daß alle Stellen, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, durch Angabe der Quellen kenntlich gemacht worden sind.

Hannes Planatscher

Inhaltsverzeichnis

1 Zusammenfassung	4
2 Grundlagen genregulatorischer Netzwerke	5
2.1 Boolesche Netzwerke	6
2.2 Gewichtsmatrizen	7
2.3 Differentialgleichungen	8
2.3.1 S-Systeme	9
3 Evolutionäre Algorithmen	11
3.1 Evolutionsstrategien	12
3.1.1 Individuen	12
3.1.2 Selektion	12
3.1.3 Mutation	12
3.1.4 Rekombination	15
3.2 Genetische Programmierung	15
3.2.1 Individuen	15
3.2.2 Selektion	16
3.2.3 Mutation	16
3.2.4 Rekombination	17
4 Software	18
4.1 Problemstellung	18
4.1.1 Separierung des Problems	19
4.2 Verwendete Implementation	19
5 Vergleich der Verfahren ES und GP	20
5.1 Einfache Testsysteme	20
5.1.1 Eulerszillator	20
5.1.2 Räuber-Beute-Modell	21
5.1.3 S-System 2. Ordnung	23
5.2 Schwach besetzte S-Systeme	24
5.2.1 S-System 2. Ordnung	24
5.2.2 S-System 5. Ordnung	25
5.3 Permutationen schwach besetzter S-Systeme	27
5.4 Fazit Vergleich ES/GP	28
6 Vergleich der Verfahren GP und SGP	29
6.1 Eulerszillator	29
6.2 Räuber-Beute-Modell	30
6.3 S-System 2. Ordnung	31
6.4 Fazit Vergleich GP/SGP	32
7 Schlussfolgerungen	34

A	EDER	35
A.1	Implementation von Evolutionsstrategien	35
A.2	Implementation von GP	35
A.3	Implementation von GP mit Separierung	36
A.4	EDER-Einführung	36
B	Dynamiken	41
C	Verzeichnisse	43

1 Zusammenfassung

Die Microarraytechnik ermöglicht die exakte Messung der Expressionslevel von sehr vielen Genen der selben Zelle zum gleichen Zeitpunkt. Es können also Daten zum Konzentrationsverlauf von Genprodukten über einen bestimmten Zeitraum gesammelt werden. Diese Daten geben Aufschluss über Interaktionen zwischen den einzelnen Genen.

Ein Microarray ist ein kleiner Glaschip auf den tausende verschiedene eindeutig bestimmte DNA-Fragmente (Gene) jeweils an einer bestimmten Position aufgetragen werden. Um nun die mRNA-Konzentrationen einer Zelle zu messen, wird diese lysiert, die mRNA mit einem Fluoreszenzfarbstoff markiert, und auf den Chip aufgebracht. Die mRNA-Fragmente docken nun an Ihre komplementären DNA-Gegenstücke auf der Glasplatte. Es entstehen so genannte Spots, fluoreszierende Punkte auf der Platte. Jeder mehr mRNA-Fragmente an die DNA auf der Platte docken, desto heller leuchtet der entsprechende Spot. Dann wird die Glasplatte mit einem Laser gescannt, und durch spezielle Bildverarbeitungsalgorithmen werden die Intensitäten der einzelnen Fluoreszenzspots (vgl. Abb. 1) gemessen.

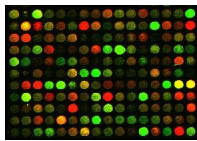


Abb. 1: Microarray-Scan (Falschfarben)

Aus den großen Datenmengen können durch Clusteringverfahren Teilmengen von Genen extrahiert werden, die in einem kausalen Zusammenhang zueinander stehen. Es ist auch für Biologen eine schwierige Aufgabe die tatsächlichen Zusammenhänge aus Experimentaldaten zu ergründen, was einen Bedarf an Hilfsmitteln zur Herleitung dieser erklärt.

In dieser Arbeit sollen Modelle für diese Zusammenhänge, und Verfahren zur Herleitung dieser Zusammenhänge verglichen werden.

Speziell wird die Modellierung mit Differentialgleichungssystemen behandelt. Für die Herleitung der Modelle werden evolutionäre Algorithmen verwendet. Genetische Programmierung und Evolutionsstrategien werden verglichen. Spezielle Stärken und Schwächen der Verfahren werden in verschiedenen Experimenten untersucht, und generelle Probleme bei der Inferenz beschrieben. Weiters wird ein Separationsverfahren vorgestellt das die Inferenz der Modelle erleichtern kann. Die Verfahren werden zunächst allgemein erläutert, bevor detaillierter auf die Anwendung eingegangen wird.

Im Rahmen der Studienarbeit wurden zwei Programme entwickelt Das Programm *EDER* stellt die Verfahren bereit, während *ReportViewer* zur Inspektion der Ergebnisse dient. Die Funktionsweise und Bedienung der Programme, sowie verschiedene Datenformate werden im Anhang erläutert.

2 Grundlagen genregulatorischer Netzwerke

Die Erbinformationen von Zellen sind in der Basensequenz ihrer DNS ¹ gespeichert. Funktionelle Abschnitte der DNS, die für vererbte Strukturen oder Funktionen codieren, heißen Gene. Viele Gene codieren für Proteine, die Grundbausteine für jeden Organismus, Proteine sind wiederum aus Aminosäuren zusammengesetzt. Ein Gen enthält also die Information in welcher Reihenfolge Aminosäuren zusammengesetzt werden müssen, um ein bestimmtes Protein zu bilden. Zur Synthese eines Proteins, der Expression des entsprechenden Gens, muss also die Sequenzinformation der DNS in eine Proteinsequenz umgesetzt werden. Dies geschieht im Wesentlichen in zwei Schritten. Bei der Transkription wird die genetische Information von der DNA abgelesen und eine Kopie der genetischen Information erstellt, die mRNA². Im Anschluß daran erfolgt die Translation wobei die mRNA in ein Protein übersetzt wird.

Jede menschliche Zelle enthält etwa 30.000 Gene. In keiner Zelle unseres Körpers werden alle Gene gleichzeitig exprimiert. Die Transskriptionskontrolle steuert welche Gene transkribiert werden und welche nicht. Die Promotor-Region eines Gens enthält bestimmte Sequenzabschnitte (Kontrollelemente) an die Transkriptionsfaktoren, spezielle Proteine, binden können und die Transkription beeinflussen. Dabei können die Kontrollelemente die Transkription verstärken oder hemmen. Diese Regulationsmechanismen ermöglichen es also, dass Gene die Stärke ihrer Expression gegenseitig über ihre Genprodukte in der Stärke ihrer Expression regeln können. Auch andere innere und äußere Faktoren (zB. UV-Licht) können die Genexpression beeinflussen. In einer Zelle existieren also komplexe genregulatorische Netzwerke von interagierenden Genen (vgl Abb. 2).

Die Analyse dieser Regulationsmechanismen ist wichtig, um die Funktionen und Unterschiede der verschiedenen Zellen zu verstehen. Moderne Verfahren wie Microarrays, können das Expressionsverhalten vieler Gene einer Zelle über die Zeit bestimmen.

Es wird versucht auf Basis dieser Messdaten ein Modell zu finden, das die Abhängigkeiten zwischen den Stoffen beschreibt, und letztlich eine Simulation der Vorgänge in einer Zelle ermöglicht.

¹Desoxyribonukleinsäure

²Messenger-RNA

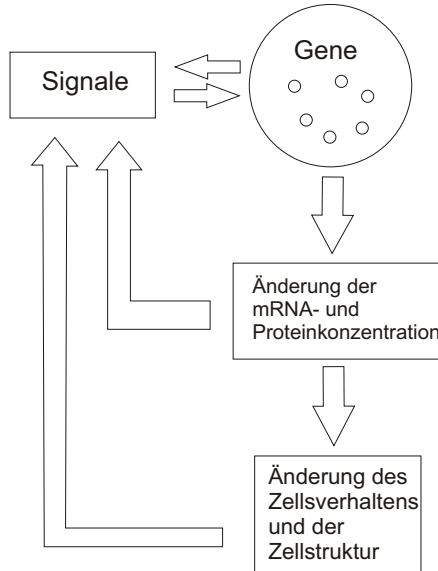


Abb. 2: schematische Darstellung eines genregulatorischen Netzwerks

Modelle für genregulatorische Netzwerke haben das Ziel Simulation zu ermöglichen, und Information über Funktion und Dynamik der Genregulation in Bezug auf intrazelluläre Vorgänge zu geben. Die verschiedenen Modelle lassen sich grob in zwei Klassen einteilen: qualitative Modelle und quantitative Modelle. Qualitative Modelle arbeiten mit diskreten Zuständen (zB: 'Gen 1 wird voll exprimiert'). Durch diese Diskretisierung sind diese Modelle etwas realitätsfern. Allerdings gibt es gute Verfahren diese Modelle aus Messdaten herzuleiten. Kontinuierliche Genexpression lässt sich mit quantitativen Modelle simulieren. Diese Modelle rechnen mit reellwertigen Expressionsleveln, und sind somit vom biologischen Standpunkt plausibler als qualitative Modelle. Leider ist die Inferenz aus Messdaten bei komplexen quantitativen Modellen oft schwierig, und für große Netzwerke bis dato nicht möglich.

2.1 Boolesche Netzwerke

Boolesche Netzwerke gehören zur Klasse der qualitativen Modelle. Sie basieren auf der Annahme, dass Gene nur ganz oder gar nicht exprimiert werden. Gegeben sei nun ein boolesches Netzwerk mit n Knoten. Jeder Knoten x_i in so einem booleschen Netzwerk hat einen booleschen Zustand $x_i(t)$. Die Zeit t wird als diskret betrachtet. Ein Knoten kann maximal n Eingabeknoten haben. Die Abhängigkeiten zwischen den Knoten werden durch boolesche Funktionen beschrieben:

$$x_i(t + 1) = f_i(X(t))$$

Die Funktionen für das Netzwerk in Abbildung 3 könnten zB. so aussehen:

$$\begin{aligned}x_1(t+1) &= x_3(t) \vee x_2(t) \\x_2(t+1) &= \neg x_3(t) \wedge x_1(t) \\x_3(t+1) &= x_2(t) \vee \neg x_1(t)\end{aligned}$$

Zur Simulation wird dann noch ein Startzustand benötigt. Wählt man für dieses System zB. den Startzustand in dem alle Gene nicht exprimiert werden ergibt sich der Expressionsverlauf in Abbildung 4.

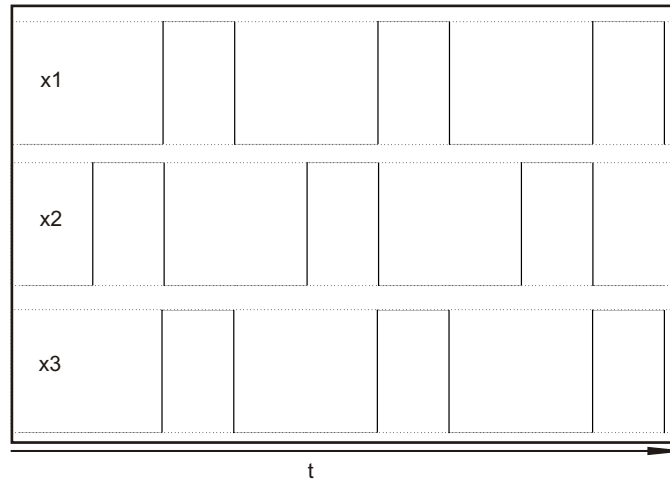
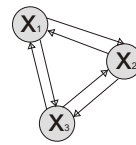


Abb. 4: Plot des Expressionsverlaufs durch Simulation eines booleschen Netzwerks

Es existieren verschiedene Algorithmen zur Rekonstruktion boolescher Netzwerke (zb. REVEAL [1][2]). Diese Algorithmen können als Eingabe oft Daten aus Knock-Out-Experimenten erhalten.³



2.2 Gewichtsmatrizen

Abb. 3: boolesches Netzwerk

In diesem Modell werden die Expressionslevel eines genregulatorischen Netzwerks von n Genen zum Zeitpunkt t durch einen Vektor $X(t) \in \mathbb{R}^n$ beschrieben. Die Interaktionen zwischen den Genen wird durch die Gewichtsmatrix W beschrieben in der jede Zeile die regulatorischen Eingaben eines Gens repräsentiert. Der regulatorische Einfluß von Gen j auf Gen i zum Zeitpunkt t ergibt sich aus dem Produkt des Expressionslevels von Gen j $x_j(t)$ mit dem Gewicht w_{ij} .

³Bei Knockout-Experimenten wird jeweils ein Gen ausgeschaltet, und das Verhalten der restlichen Gene im Vergleich zum ungestörten genregulatorischen Netzwerk betrachtet.

Die gesamte regulatorische Eingabe $r_i(t)$ von Gen i ergibt sich also aus der Summe über alle Gene im System:

$$r_i(t) = \sum_{j=1}^n w_{ij} x_j(t)$$

Ist das Gewicht w_{ij} positiv hat Gen j einen aktivierenden, ist es jedoch negativ einen hemmenden, und wenn 0 gar keinen Einfluss auf Gen i . Die transkriptionelle Antwort $a_i(t)$ von Gen i wird nun eine Aktivierungsfunktion der Form

$$a_i(t+1) = \frac{1}{1 + \exp^{-(\alpha_i r_i(t) + \beta_i)}}$$

berechnet. α_i und β_i sind genspezifische Parameter. α beschreibt die Empfindlichkeit eines Gen auf regulatorische Einwirkungen. Gene mit großem α reagieren also stärker auf eine regulatorische Eingabe als Gene mit kleinem α . β ist als das Basisexpressionslevel eines Gens zu betrachten. a ist das relative Expressionslevel. Das absolute Expressionslevel zum Zeitpunkt $t+1$ kann nun so berechnet werden:

$$x_i(t+1) = m_i a_i$$

m_i ist das maximale Expressionslevel von Gen i .

Es existieren Verfahren wie REM (Reverse Engeneering of Matrices) die versuchen Gewichts-Matrix-Modelle, für genregulatorische Netzwerke zu bauen, für die es Messdaten gibt. Diese Verfahren führen das Inferenz-Problem auf lineare Gleichungssysteme zurück. [3]

Die Abhängigkeiten zwischen den Genen in diesem Modell sind linear. Vom biologischen Standpunkt aus ist diese Annahme wenig realitätsbezogen, da auf diese Weise nicht alle bekannten Reaktionsmechanismen modelliert werden können.

2.3 Differentialgleichungen

Differentialgleichungen sind eine sehr allgemeine Beschreibungsform für zeitdynamische Systeme, eignen sich also auch zur Modellierung genregulatorischer Netzwerke. Man betrachte ein Netzwerk von n Genen. x_i bezeichne das Expressionslevel von Gen i .

Nun gibt es eine Gleichung

$$\frac{\delta x_i}{\delta t} = f(X, t)$$

die die Änderung des Expressionslevels von x_i in Abhängigkeit von den anderen Genen über die Zeit beschreibt. Für n Gene ergibt sich ein Differentialgleichungssystem n -ter Ordnung. Gegeben seien nun die Expressionslevel zu einem Zeitpunkt X_0 . Gesucht werden die Expressionslevel X_t zu einem Zeitpunkt t . Zur Berechnung dieser Anfangswertprobleme werden numerische Integrationsmethoden wie das Runge-Kutta-Verfahren verwendet.

Es gibt parametrisierte Modelle für regulatorische Netzwerke, in der Form von Differentialgleichungssystemen, die zwar eine feste Struktur haben, aber durch Adaption ihrer Variablen trotzdem sehr flexibel sind. Zu diesen Differentialgleichungssystemen gehören zum Beispiel S-Systeme[4] [5]. Die Interpretation solcher Systeme ist einfacher für den Biochemiker, da die gefundenen Parameter als Reaktionsordnungen interpretiert werden können, wie sie aus der Physikalischen Chemie bekannt sind. Außerdem reduziert sich das Inferenz-Problem von solchen Gleichungen auf eine Parametersuche, für die bereits gute Heuristiken existieren. Durch die feste Struktur dieser parametrisierten Systeme, sind die durch sie darstellbaren Dynamiken begrenzt, das heißt es können möglicherweise nicht alle Reaktionsmechanismen modelliert werden die in der Natur vorkommen könnten.

Es wäre also vielleicht besser keine feste Struktur der DGL anzunehmen, und gegebenenfalls nach möglichst allgemeine DGL zu suchen, um unbekannte Vorgänge in der Natur abbilden zu können.

2.3.1 S-Systeme

S-Systeme sind eine nichtlineare kanonische Form gewöhnlicher Differentialgleichungssysteme. Jede Gleichung eines S-Systems beschreibt die Änderung eines Zustandes eines Systems über die Zeit. Eine solche Gleichung besteht aus zwei Termen wobei der eine für die aktivierenden Faktoren, und der andere für die hemmenden Faktoren steht.

Die Gleichungen soll nun durch Logarithmieren und Anwendung von Taylor-Reihen hergeleitet werden.

Sei die Synthesegeschwindigkeit v_i von Stoff i ausschließlich von der Konzentration desselben Stoffes x_i abhängig.

$$v_i = f(x_i)$$

Durch Logarithmieren ergibt sich:

$$\log v_i = \log(f(x_i))$$

Nun approximiert man $f(x_i)$ durch eine Taylor-Reihe, von der wir nur die ersten beiden Glieder betrachten:

$$\log v_i = \log(v_i)_0 + \frac{\log(v_i)_0}{\log x_i} (\log x_i - \log(x_i)_0)$$

oder

$$\log v_i = \log \alpha_i + g_{ii} \log x_i$$

Ist die Reaktionsgeschwindigkeit von n Stoffkonzentrationen X abhängig wird dieser Term einfach erweitert:

$$\log v_i = \log \alpha_i + g_{i1} \log x_1 + g_{i2} \log x_2 + \dots + g_{in} \log x_n$$

und durch Anwendung der Exponentialfunktion auf beiden Seiten

$$v_i = \alpha_i x_1^{g_{i1}} x_2^{g_{i2}} \dots x_n^{g_{in}}$$

g_{ij} ist die Reaktionsordnung der Synthese-Reaktion von Stoff i bezogen auf den Stoff j. Ein Term derselben Form lässt sich genauso für die Abbaugeschwindigkeit herleiten.

Also ergibt sich für die Änderung der Konzentration (Konzentrationszunahme durch Synthese + Konzentrationsabnahme durch Abbau) insgesamt folgender Term:

$$\frac{\delta x_i}{\delta t} = \underbrace{\alpha_i \prod_{j=1}^n x_j^{g_{ij}}}_{\text{Synthese}} - \underbrace{\beta_i \prod_{j=1}^n x_j^{h_{ij}}}_{\text{Abbau}}$$

. Durch die Form der Approxmierung erhält man ein allgemeines parametrisiertes Modell für biochemische Systeme. S-Systeme können beinahe alle kinetischen Phänomene in biochemischen Systemen zuverlässig beschreiben. Durch die Gleichsetzung der Exponenten als Reaktionsordnungen sind diese Modelle auch aus dem Blickwinkel der Biochemie einfach zu interpretieren.

Da die Anzahl der Parameter eines S-Systems mit der Anzahl der Anzahl der Stoffe quadratisch ($2n + 2n^2$) zunimmt, ist es für große Netzwerke ($n > 7$) sehr schwierig die richtigen Parameter zu finden, wenn man apriori nichts über über die Topologie des Netzwerks weiss. Ein weiteres Problem ist die Inferrierung von schwach verbundenen genregulatorischen Netzwerken mit S-Systemen.

Ein S-System könnte zB. so aussehen:

$$\begin{aligned} \frac{\delta x_1}{\delta t} &= 3.0 * x_1^0 * x_2^{-2.5} - 3.0 * x_1^{-1.0} * x_2^0 \\ \frac{\delta x_2}{\delta t} &= 3.0 * x_1^{2.5} * x_2^0 - 3.0 * x_1^0 * x_2^{2.0} \end{aligned}$$

Integriert man dieses System nun mit $x_1(t_0) = 1.5$ und $x_2(t_0) = 1$ ergibt sich folgender Expressionsverlauf:

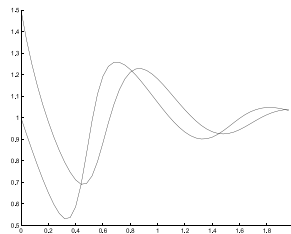


Abb. 5: einfaches regulatorisches Netzwerk

3 Evolutionäre Algorithmen

Die Herleitung oben genannter Modelle stellen den Bioinformatiker vor zwei Probleme: die Wahl es Modells, und gegebenenfalls das Finden der Parameter für das Modell. Im Allgemeinen Fall steht sogar die Struktur des Modells zur Diskussion. Da deterministische Verfahren meist zu schwach sind um diese Probleme zufriedenstellend zu lösen, werden Heuristiken angewandt.

Bei Evolutionären Algorithmen handelt es sich um stochastische Suchverfahren. Sie machen sich Methoden zu Nutze die in der Natur beobachtet wurden: Selektion, Rekombination und Mutation.

Gegeben ist ein Problem, für das eine möglichst optimale Lösung gesucht wird. Eine Hypothese zur Lösung des Problems wird Individuum genannt. Alle möglichen Hypothesen bilden zusammen den Suchraum. Gegeben ist ebenso eine Fitnessfunktion, die die Güte von Individuen bewertet. Nun sollen Individuen gefunden werden, die einen möglichst guten Fitnesswert haben. Zuerst wird im einfachen Evolutionären Algorithmus ein Menge von Individuen zufällig generiert. Die Individuen in dieser Anfangspopulation werden auf ihre Fitness getestet. Durch Selektion werden die besten Individuen ausgewählt, und einer Elternpopulation hinzugefügt. Aus dieser Elternpopulation wird durch Rekombination, das Vermischen zweier oder mehrerer Individuen, und Mutation, zufälliges Verändern eines Individuums, eine neue Population erzeugt. (vgl Abb. 6). In dieser neuen Generation kommen hoffentlich Individuen vor die einen besseren Fitnesswert besitzen, und somit bessere Lösungen für das Problem darstellen.

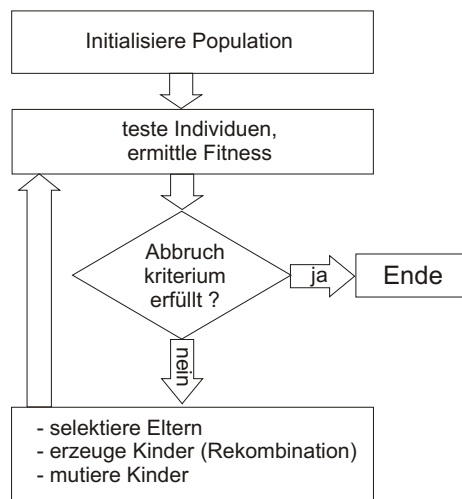


Abb. 6: schematische Darstellung des Basis-EA-Algorithmus

Evolutionäre Algorithmen unterscheiden sich oft in der Methode wie Individu-

en codiert werden. In einem Genetischen Algorithmus ist ein Individuum ein Bitvektor, der zB. in Gleitkommazahlen umgewandelt werden kann.

Evolutionsstrategien hingegen codieren ihre Individuen direkt in reellwertige Vektoren. Für ES gibt es sehr starke Evolutionsoperatoren, die dieses Verfahren zu einer guten Heuristik zur Optimierung reellwertiger Parameter machen. Genetische Algorithmen und Evolutionsstrategien wurden bereits erfolgreich zur Parametersuche bei S-Systemen eingesetzt. [6]

Genetische Programmierung verfolgt hingegen das Ziel Programme oder Gleichungen automatisch herzuleiten. Individuen werden oft als Programmbäume codiert. Gleichungen jedweder Struktur können dargestellt werden. Da vermutet wird, dass parametrisierte Modelle wie S-Systeme oder Gewichtsmatrizen, nicht alle möglichen Regulationsmechanismen in der Natur darstellen können, ist GP eine interessante Alternative um nach Gleichungssystemen ohne vorgegebene Struktur zu inferieren. Dies wurde zuerst von Iba und Sakamoto untersucht. [7][8] Auch Koza versuchte die Modellierung von GRN mit GP [9], indem er versuchte genregulatorische Netzwerke als analoge Schaltkreise zu modellieren, und deren Struktur mit Genetischer Programmierung zu optimieren.

3.1 Evolutionsstrategien

Evolutionstrategien gingen in den 70er-Jahren aus den Ingenieurwissenschaften hervor, und eignen sich besonders zur Optimierung reellwertiger Funktionen der Form $f : \mathbb{R}^n \rightarrow \mathbb{R}$. n ist die Dimension eines Problems.

3.1.1 Individuen

Individuen im Sinne der Evolutionsstrategien bestehen zum einen aus einem Vektor der Objektvariablen $A \in \mathbb{R}^n$, zum anderen aus Strategievariablen S . Strategievariablen werden zur Steuerung der Evolutionsoperatoren benötigt, auf die später näher eingegangen wird.

$$\text{Individuum } I = \{A, S\}$$

3.1.2 Selektion

Bei $(\mu + \lambda)$ -ES (Plus-Strategie) werden die Eltern bei der Selektion mitberücksichtigt. Aus μ Eltern werden zunächst λ Kinder erzeugt. Aus der Vereinigungsmenge von Eltern und Kindern werden dann μ Nachkommen selektiert.

Bei (μ, λ) -ES (Komma-Strategie) werden μ Nachkommen nur aus den λ Kindern selektiert.

3.1.3 Mutation

Die Mutation ist der primäre Operator bei Evolutionsstrategien. Bei der ES-Mutation wird zu jeder Objektvariable eines Individuums ein zufälliger Wert addiert.

$$a'_i = a_i + m_i$$

Die Zufallszahlen werden gemäß einer Normalverteilung $m_i \sim N(0, \sigma)$ mit der Dichte

$$\phi(x) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp^{-\frac{1}{2\sigma_i^2}x^2}$$

generiert. Die Standardabweichung σ_i wird als Mutationsschrittweite bezeichnet. Existiert nur eine globale Mutationsschrittweite σ werden alle Individuen damit mutiert. Durch eine geschickte Adaption der Schrittweite kann der Erfolgsfaktor einer Mutation verbessert werden.

- Die einfachste Adaptionsstrategie ist die 1/5-Erfolgsregel. Hierbei wird überprüft bei wie vielen Individuen durch eine Mutation eine Verbesserung der Güte eingetreten ist. Ist dies bei über 1/5 der mutierten Individuen der Fall wird die Schrittweite vergrößert, da angenommen wird, dass man noch weit vom gesuchten Optimum entfernt ist. Liegt die Erfolgswahrscheinlichkeit unter 1/5 wird die Schrittweite verkleinert um eine lokale Suche durchzuführen. Der Wert 1/5 für eine optimale Erfolgswahrscheinlichkeit wurde empirisch ermittelt.
- Eine andere Möglichkeit zur Verbesserung der Evolutionsstrategie ist die Selbst-Adaption der Schrittweite. Die Schrittweite wird als Strategieparameter für jedes Individuum separat mitgeführt, und ebenso dem Evolutionsprozess unterworfen. $\sigma \in S$ wird nun wie folgt mutiert:

$$\sigma' = \sigma \exp \frac{1}{\sqrt{n}} z$$

wobei z eine $N(0, 1)$ -verteilte Zufallsvariable ist. Individuen mit geeigneter Mutationsschrittweite erzeugen bessere Nachkommen, und so setzt sich diese Mutationsschrittweite durch. Bei dieser uniformen Schrittweitenanpassung ist die erwartete Schrittweite eine Hyperkugel (siehe Abbildung 7 links) im Suchraum. Dies kann nachteilig sein, wenn das Individuum im Suchraum einem Grat entlang wandern muss.

- Dem kann entgegenwirkt werden, indem für jede einzelne Objektvariable a_i eine eigene Schrittweite σ_i als Strategievariable definiert wird. Diese wird so mutiert:

$$\sigma'_i = \sigma_i \exp \frac{1}{\sqrt{2n}} z + \frac{1}{\sqrt{2}\sqrt{n}} z'_i$$

z_i ist eine zusätzliche $N(0, 1)$ -verteilte Zufallsvariable für jede Schrittweite. Durch diese Modifikation ist die erwartete Schrittweite eine Hyperellipse im Suchraum (siehe Abbildung 7 links) die Gegebenheiten des Suchraums schon besser angepasst ist.

- Weiter können n Drehwinkel mitgeführt werden, die Drehung der Hyperellipse im Raum festlegen. Diese Variation ergibt gedrehte Hyperellipsen als erwartete Schrittweiten im Suchraum (siehe Abbildung 7 rechts).

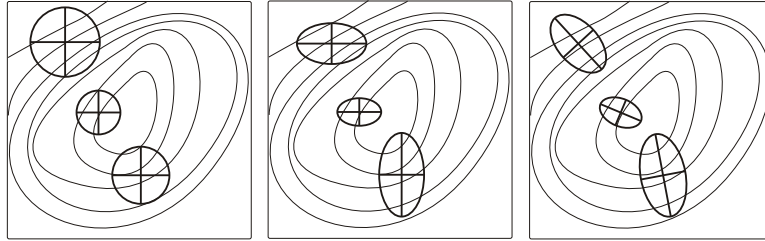


Abb. 7: ES-Mutation

- Die Kovarianzmatrixadaption ist eines der mächtigsten Adaptionsverfahren. Die Idee die dahinter steckt ist, dass die verschiedene Objektvariable miteinander korreliert sein können. Deswegen ist es sinnvoll Zufallsverteilungen die bei Mutationen der korrelierten Variablen auch zu korrelieren. Die Korrelation von zwei Zufallsvariablen nennt man Kovarianz.

Z_1, Z_2, \dots, Z_n seien nun unabhängig $N(0,1)$ -verteilte Zufallsvariablen.
 $z_1, z_2, \dots, z_n \in \mathbb{R}^n$ Vektoren und $\sigma_1, \sigma_2, \dots, \sigma_n \in \mathbb{R}^+$ Skalare.

$$Z = \sum_{i=1}^n Z_i \sigma_i z_i$$

$$C = \sum_{i=1}^n Z_i \sigma_i^2 z_i z_i'$$

Z ist ein normalverteilter Zufallsvektor mit Mittelwert 0 und Kovarianzmatrix C . Ist C gegeben kann $N(0, C)$ erzeugt werden indem in der Summe für Z die z_i auf die Eigenvektoren und σ_i^2 auf die entsprechenden Eigenwerte von C setzt. Beim CMA-ES wird zusätzlich zu C noch der Mutationspfad p_m als Strategievariable mitgeführt. p_m wird zur Adaption von C benötigt.

$$p'_m = (1 - \kappa_m)p'_m + \kappa_m^u \sqrt{\sigma} (x' - x)$$

$$0 < \kappa_m < 1$$

$$\kappa_m^u = \sqrt{\kappa_m(2 - \kappa_m)}$$

Die Kovarianzmatrix wird nun wie folgt adaptiert:

$$C' = (1 - \kappa_{cov})C + \kappa_{cov}p'_m(p'_m)^T$$

Nun müssen noch die Eigenvektoren und Eigenwerte von C ermittelt werden um die Objektvariable mit $N(0, \sigma^2 C')$ zu mutieren. σ ist eine globale Schrittweite die mit den oben erwähnten Adaptionsverfahren adaptiert wird.

3.1.4 Rekombination

Auch von diesem Evolutionsoperator, der gewöhnlich bei Evolutionsstrategien nur eine sekundäre Rolle einnimmt, gibt es verschiedene Realisierungen. Bei der diskreten Rekombination wird eine Variable entweder aus Elter 1 oder aus Elter 2 übernommen. Intermediäre Rekombination bedeutet, dass ein Mittelwert zwischen den korrespondierenden Variablen der Eltern gebildet wird. Von diesen beiden Möglichkeiten, gibt es jeweils noch die globale Variante, womit gemeint ist, dass für jede Variable ein neues Elternteil aus der Population gewählt wird. Meist werden die Objektvariablen diskret, die Strategieparameter jedoch intermediär rekombiniert.

3.2 Genetische Programmierung

Genetische Programmierung ist ein evolutionärer Algorithmus zur Suche und Optimierung von Programm- oder Gleichungsstrukturen. Die ersten konkreten Ansätze entwickelte John Koza 1989 [10][11]. Genetische Programmierung wurde in dieser Arbeit zur Suche nach allgemeinen Differentialgleichungssystemen angewandt.

3.2.1 Individuen

In der Genetischen Programmierung sind Individuen Programmstrukturen. Im klassischen Ansatz von Koza wurde die Programme in einer Untermenge der funktionalen Programmiersprache LISP codiert. Programme dieser Art lassen sich in einer Baumstruktur darstellen und speichern.

Die Blätter des Baumes sind Terminale, die entweder aus der Menge der Variablen gewählt oder Konstanten sind. Die Variablen entsprechen den Daten die vom Programm verarbeitet werden sollen. Die Konstanten werden meist am Anfang des Evolutionslaufs in einer festen Anzahl K generiert und nicht mehr direkt verändert. In den Knoten stehen primitive Funktionen wie Addition, Multiplikation, Division, trigonometrische Funktionen, logische Operatoren etc. . Die Menge der Terminale und der primitiven Funktionen F muss vor der genetischen Optimierung festgelegt werden.

Die primitiven Funktionen müssen so implementiert sein, dass sie jede Anzahl, jede Kombination und jeden Typ von Eingaben verarbeiten können. Das bedeutet zum Beispiel, dass die Division durch Null keinen Fehler hervorrufen darf, sondern für diesen Spezialfall eine bestimmte Ausgabe wie z.B. 0 vorsieht. Sind

diese Eigenschaften erfüllt und die Menge der Terminale und Funktionen abgeschlossen, resultiert eine Programmiersprache in der alle möglichen Programme ausführbar sind. Diese Eigenschaft nennt man Closure-Eigenschaft.

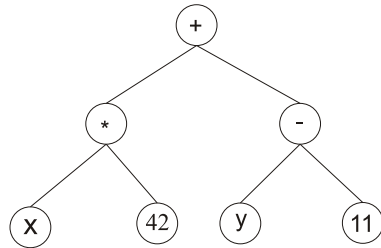


Abb. 8: LISP-Programmbaum: Dieser Baum beschreibt die LISP - Funktion $(+ (* 23 42) (- 47 11))$ oder in Infix-Notation $(x, y) = x * 42 + y * 11$.

Eine weitere Forderung die an ein solches Programm gestellt wird, dass es eine Lösung zu dem gestellten Problem repräsentieren kann (Sufficiency-Eigenschaft). Meistens sind die Mengen der Terminale und primitiven Funktionen auf das zu lösende Problem zugeschnitten. Im Zuge der Auswahl der primitiven Funktionen, muss ein vernünftiger Kompromiss gefunden werden. Wird die Menge der primitiven Funktionen zu gross gewählt, wird der Suchraum unrealistisch gross, wird sie zu klein gewählt wird überhaupt keine Lösung gefunden, da die Lösung mit den vorhandenen Funktionen gar nicht erst repräsentiert werden kann.

In der Praxis zeigen Programme im Evolutions-Lauf den so genannten Bloat-Effekt. Er besteht darin dass sich die Programme durch fortschreitende Mutation und Rekombination durch nicht berechnenden Code aufblähen, der die Programme unlesbar und langsam macht. Aus diesem Grund wird oft ein maximale Baumtiefe T^4 festgelegt. Ist die maximale Baumtiefe zu gross gewählt, wird die Laufzeit der Programme unverträglich lang und die Lesbarkeit der Programme schlecht. Eine zu kleine maximale Baumtiefe wiederum führt dazu, dass Lösungen nicht repräsentiert werden können.

3.2.2 Selektion

Ein gern bei GP verwendeter Selektionsmechanismus ist 'Tournament Selektion'. Die Idee besteht darin, eine Untermenge der Größe γ (je nach Populationsgrösse) aus der Population zufällig auszuwählen, und aus dieser kleinen Untergruppe, das beste Individuum zu bestimmen und zu selektieren. Dies wird so oft wiederholt, bis genügend Eltern selektiert sind.

3.2.3 Mutation

Bei den Programm bäumen wie sie oben beschrieben sind, ist Mutation wie folgt definiert: Es wird zufällig einen Teilbaum ausgesucht, und dieser durch einen neuen zufälligen neuen Teilbaum ersetzt.

⁴Um die Tiefe zu ermitteln werden die Kanten von der Wurzel des Baumes bis zum Blatt das untersten Ende des Baumes hängt gezählt

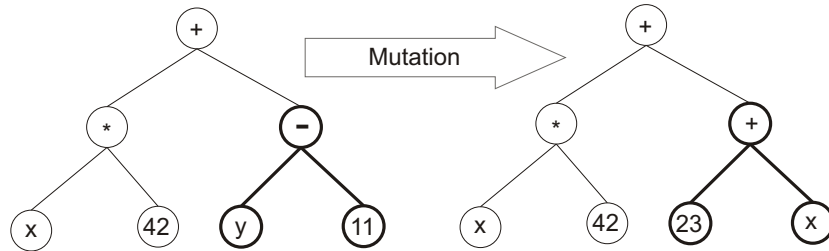


Abb. 9: Baum-Mutation

Ist eine maximale Tiefe für Bäume definiert um Bloat zu vermeiden, muss das bei der Mutation berücksichtigt werden.

3.2.4 Rekombination

Die Rekombination (oder Crossover) erzeugt aus zwei Programmen neue Programme durch Vermischen (Rekombination) der Programmbäume. Dieser Operator gründet auf der Hoffnung, dass zwei gute Programme gewisse Eigenschaften (bei LISP-Programmen also Unterbäume) auszeichnen, und dass durch eine Kombination dieser Eigenschaften ein besseres Programm resultiert. Bei den Programmbäumen wie sie oben beschrieben sind, ist Rekombination wie folgt definiert: Jeweils ein Unterbaum in den Programmbäumen wird zufällig ausgewählt, und dann miteinander vertauscht. Auch bei der Mutation gibt es eine tiefenbewahrende Variante.

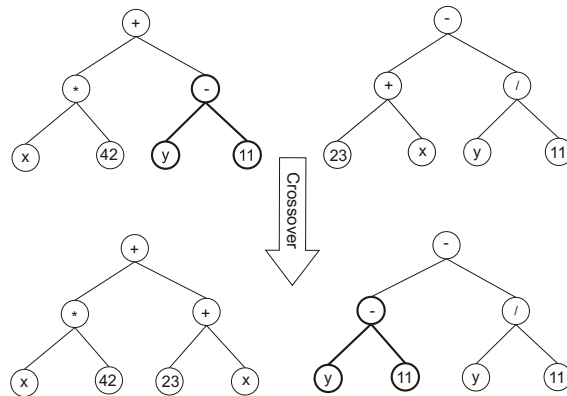


Abb. 10: Baumrekombination

4 Software

4.1 Problemstellung

Gegeben sei eine $(n \times m)$ -Matrix $M \in \mathfrak{R}^{n \times m}$, und ein Vektor $T \in R^m$. T enthält m sortierte Zeitpunkte ($t_i < t_{i+1}$), und m_{ij} ist das Expressionslevel $x_i(t_j)$ von Gen i zum Zeitpunkt t_j .

Das Inferenz-Problem besteht nun darin ein gewöhnliches Differentialgleichungssystem

$$\dot{X} = \begin{pmatrix} \dot{x}_1(x_1, x_2, \dots, x_n) \\ \dot{x}_2(x_1, x_2, \dots, x_n) \\ \vdots \\ \dot{x}_n(x_1, x_2, \dots, x_n) \end{pmatrix}$$

zu finden, das die Konzentrationsverläufe aus M möglichst gut beschreibt, also ein Fehlermaß zwischen realen Daten und Modellausgabe minimiert.

Es gibt unter anderen das Fehlermaß des quadratischen Abstands:

$$MSE(X') = \sum_{i=1}^n \sum_{j=1}^m (x'_i(t_j) - x_i(t_j))^2$$

Es bietet sich an die quadratischen Abstände zu relativieren um den Einfluss stärker exprimierter Gene auf den Gesamtfehler einzuschränken:

$$RSE(X') = \sum_{i=1}^n \sum_{j=1}^m \frac{x'_i(t_j) - x_i(t_j)^2}{x_i(t_j)}$$

Die Modellausgabe $X'(t)$ wird durch numerische Integration mit dem Runge-Kutta-Verfahren vierter Ordnung mit den Anfangswerten $x_i(0) = m_{i1}$ berechnet

Das Runge-Kutta-Verfahren funktioniert so:

$$X_{t+1} = X_t + h\left(\frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4\right)$$

$$k_1 = f(X_t, t)$$

$$k_2 = f\left(X_t + h\frac{k_1}{2}, t + \frac{h}{2}\right)$$

$$k_3 = f\left(X_t + h\frac{k_2}{2}, t + \frac{h}{2}\right)$$

$$k_4 = f(X_t + hk_3, t + h)$$

wobei h ein frei wählbare Schrittweite ist. Der verfahrensbedingte Fehler der Runge-Kutta-Methode ist von der Ordnung h^5 .

In dieser Arbeit soll untersucht werden, wie sich bekannte Verfahren aus dem Bereich der Evolutionären Algorithmen eignen, dieses Inferenz-Problem zu lösen

und sie zu vergleichen. Untersucht wurden Evolutionsstrategien zur Parametereoptimierung von S-Systemen, und Genetische Programmierung zur Herleitung allgemeinerer Differentialgleichungssysteme.

4.1.1 Separierung des Problems

Das oben genannte Problem lässt sich in Teilprobleme kleinerer Dimensionen zerlegen. Die Idee ist zuerst die einzelnen Gleichungen des Gesamtsystem zu finden, und diese wieder zu einem großen System zusammensetzen.

Da die von uns verwendeten Integrationsverfahren Schritte zwischen den einzelnen Messpunkten machen, und jeweils die Konzentrationen zwischen diesen Messpunkten benötigen, müssen Annahmen über den Konzentrationsverlauf zwischen den Messpunkten getroffen werden.

Zunächst werden für alle gemessenen Konzentrationsverläufe Splinefunktionen erstellt. $S_j(t)$ bezeichne die Splinefunktion für Gen j.

Wir suchen nun eine wieder eine Differentialgleichung für den Konzentrationsverlauf von Gen i der Form:

$$\frac{\delta x_i}{\delta t} = f(X, t)$$

wobei

$$x_j(t) = \begin{cases} S_j(t) & \text{für } j \neq i \\ x_i & \text{sonst} \end{cases}$$

gilt.

Diese Gleichung kann mit einem numerischen Differentialgleichungslöser integriert werden. Das Teilproblem ist also eine derartige Differentialgleichung zu finden die den Konzentrationsverlauf eines Gens unter den gegebenen Voraussetzungen möglichst gut beschreibt.

Auch hier kann der relative quadratische Fehler zur Bewertung herangezogen werden:

$$RSE(X') = \sum_{j=1}^m \frac{x'_i(t_j) - x_i(t_j))^2}{x_i(t_j)}$$

Sind einzelnen Teilgleichungen gefunden, können diese zu vollständigen Systemen zusammengesetzt werden. Diese Systeme bilden dann eine Startpopulation für eine gewöhnliche Optimierung, um die Gleichungen aufeinander abzustimmen.

4.2 Verwendete Implementation

In den folgenden Untersuchungen wurde das Programm *EDER* verwendet, dessen Funktionen und Anwendung in Anhang A erläutert werden.

5 Vergleich der Verfahren ES und GP

In diesem Abschnitt soll die Leistungsfähigkeit und das Konvergenzverhalten der verschiedenen Verfahren getestet und dann verglichen werden. Im Vordergrund steht der Vergleich zwischen Genetischer Programmierung zur Modellierung allgemeiner Differentialgleichungssysteme und Evolutionsstrategien zur Parametersuche bei S-Systemen.

5.1 Einfache Testsysteme

5.1.1 Eulerszillator

Der Eulerszillator ist ein Differentialgleichungssystem 3. Ordnung der Form

$$\begin{aligned}\frac{\delta x_1}{\delta t} &= x_2 x_3 \\ \frac{\delta x_2}{\delta t} &= -x_1 x_3 \\ \frac{\delta x_3}{\delta t} &= -0.51 x_1 x_2\end{aligned}$$

In Abbildung 11 erkennt man das Schwingungsverhalten dieses Systems. Dieses System ist ein bekanntes Beispiel für allgemeine Differentialgleichungssysteme und wurde deshalb als Referenzsystem ausgewählt. Dazu wurde das System zunächst integriert und 50 äquidistante Messpunkte im Zeitintervall zwischen 0 und 20 genommen. Als Anfangswerte wurden $(x_1)_{t_0} = 1, (x_2)_{t_0} = 2, (x_3)_{t_0} = 1$ gewählt. Nun wurde ES und GP angewandt, um das System aus den Messdaten zu rekonstruieren. Die Parametrisierung der Verfahren wurde wie folgt vorgenommen:

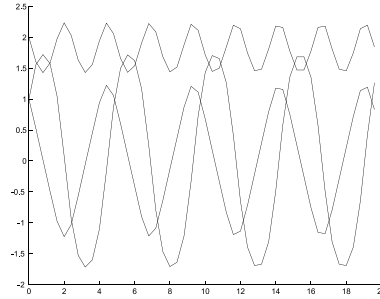


Abb. 11: Eulerszillator

μ	λ	G	Selektion	Mutation	P_{mut}	P_{cross}
10	500	40	Komma	CMA	1	0

Tabelle 1: Parameter für ES angewandt auf Eulerszillator-Inferenz

μ	λ	G	Selektion	F	K	T	P_{mut}	P_{cross}
40	1000	10	TS $\gamma = 100$	+, -, *	1000	4	0.3	0.7

Tabelle 2: Parameter für GP angewandt auf Eulerszillator-Inferenz

Es wurde jeweils 10 Läufe pro Verfahren durchgeführt. ES erzielte einen durchschnittlichen RSE des besten gefundenen Systems pro Lauf von 97,8189 ($\sigma = 0,5002$), Genetische Programmierung 97,0304 ($\sigma = 2,1923$) (vgl. Abb.12).

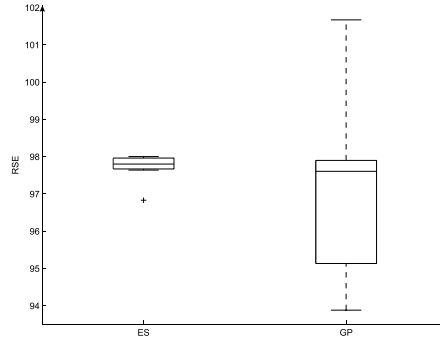


Abb. 12: Vergleich ES/GP Eulerszillator

Beide Verfahren versagen bei der Inferenz des Systems, und erreichen etwa die gleiche Leistung. Auch zahlreiche Wiederholungen brachten keine Verbesserung. Es ist anzunehmen, dass die oszillierende Dynamik des Zielsystems die Verfahren überfordert.

5.1.2 Räuber-Beute-Modell

Über 90 Jahre führte die Hudson-Bay-Company Protokoll über den Eingang von Fellen von Luchsen und Schneehasen. Der Luchs ist der wichtigste natürliche Feind des Schneehasen. Auffällig in diesen Daten waren starke Schwankungen im Abstand von jeweils 7 Jahren.

Vito Volterra (1860-1940) und Alfred James Lotka (1880-1949) entwickelten ein Modell, das diese Schwankungen erklären konnte. In dieses Modell wurden folgende Gesetzmäßigkeiten einbezogen: Innerhalb einer Räuber-Beute-Beziehung schwanken die Populationsgrößen des Räubers und der Beutetiere periodisch und im zeitlichen Wechsel zueinander. Die Populationsdichte der Räuber und der Beutetiere schwanken immer um einen Mittelwert.

Nimmt die Population von Beute und Räuber gleichmäßig ab, so steigt zunächst die Anzahl der Beutetiere schneller an als die der Räuber.

Das Räuber-Beute-Modell ist ein Differentialgleichungssystem 2. Ordnung der Form :

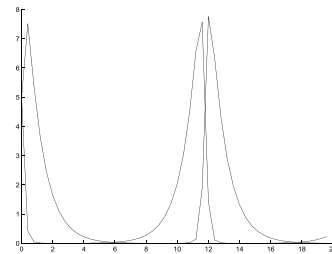


Abb. 13: Räuber-Beute-Modell

$$\begin{aligned}\frac{\delta x_1}{\delta t} &= x_1(1 - x_2) \\ \frac{\delta x_2}{\delta t} &= x_2(x_1 - 1)\end{aligned}$$

Das System wurde integriert und 50 äquidistante Messpunkte im Intervall zwischen 0 und 20 genommen. Als Anfangswerte wurden $(x_1)_{t_0} = 5, (x_2)_{t_0} = 5$ gewählt. Nun wurden ES und GP angewandt, um das System aus den Messdaten zu rekonstruieren. Die Parametrisierung der Verfahren wurde wie folgt vorgenommen:

μ	λ	G	Selektion	Mutation	P_{mut}	P_{cross}
80	1000	40	Komma	CMA	1	0

Tabelle 3: Parameter für ES angewandt auf Räuber-Beute-Modell-Inferenz

μ	λ	G	Selektion	F	K	T	P_{mut}	P_{cross}
20	1000	40	TS $\gamma = 200$	+, -, *	1000	4	0.8	0.7

Tabelle 4: Parameter für GP angewandt auf Räuber-Beute-Modell-Inferenz

Es wurden jeweils 20 Läufe pro Verfahren durchgeführt. Die Evolutionsstrategie erzielte einen durchschnittlichen RSE des besten gefundenen Systems pro Lauf von 58.7989 ($\sigma = 40.5454$), GP 46.3388 ($\sigma = 60.2622$) (vgl. Abb.14).

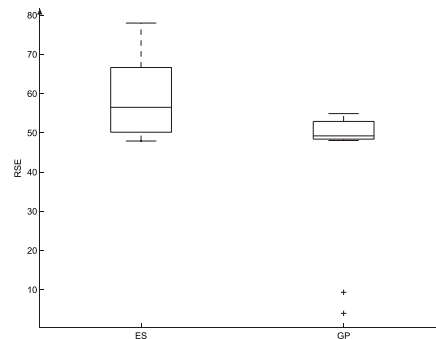


Abb. 14: Vergleich ES/GP Räuber-Beute-Modell

Also führen auch bei diesem System beide Verfahren nicht zum Erfolg. Erwähnenswert ist jedoch, dass GP zu geringfügig besseren, wenn auch unbrauchbaren, Ergebnissen führte. Vermutlich ist der Suchraum des ES stark multimodal und verfügt über zahlreiche lokale Minima.

5.1.3 S-System 2. Ordnung

Es handelt sich hier um das System, das in Abschnitt 2.3.1 vorgestellt wurde:

$$\begin{aligned}\frac{\delta x_1}{\delta t} &= 3.0 * x_1^0 * x_2^{-2.5} - 3.0 * x_1^{-1.0} * x_2^0 \\ \frac{\delta x_2}{\delta t} &= 3.0 * x_1^{2.5} * x_2^0 - 3.0 * x_1^0 * x_2^{2.0}\end{aligned}$$

Das System wurde integriert und 50 äquidistante Messpunkte im Zeitintervall zwischen 0 und 2 genommen. Nun wurden ES und GP angewandt, um das System aus den Messdaten zu rekonstruieren. Die Parametrisierung der Verfahren wurde wie folgt vorgenommen:

μ	λ	G	Selektion	Mutation	P_{mut}	P_{cross}
20	500	40	Komma	CMA	1	0

Tabelle 5: Parameter für ES angewandt auf die Inferenz eines S-Systems 2. Ordnung

μ	λ	G	Selektion	F	K	T	P_{mut}	P_{cross}
20	500	40	TS $\gamma = 50$	+,-,*	100	4	0.8	0.7

Tabelle 6: Parameter für GP angewandt auf die Inferenz eines S-Systems 2. Ordnung

Es wurden jeweils 20 Läufe pro Verfahren durchgeführt. Die Evolutionsstrategie erzielte einen durchschnittlichen RSE des besten gefundenen Systems pro Lauf von 0.2294 ($\sigma = 1.8630$), GP dagegen GP von 0.6125 ($\sigma = 0.4890$) (vgl. Abb. 15, zu beachten ist die Skalierung der y-Achse) .

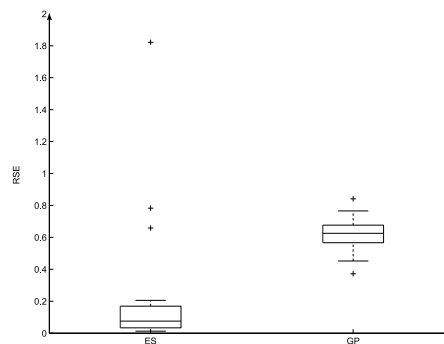


Abb. 15: Vergleich ES/GP SSystem 2.Ordnung

Beide Verfahren liefern bei diesem Inferenzproblem vernünftige Ergebnisse, doch der ES ist geringfügig besser. Die inferrierten Parameter unterscheiden sich jedoch massgeblich von denen des Zielsystems.

System	α_1	α_2	β_1	β_2	g_{11}	g_{12}	g_{21}	g_{22}	h_{11}	h_{12}	h_{21}	h_{22}
Orginal	3	3	3	3	0	-2.5	2.5	0	-1.0	0	0	2.0
Inferenz	3.2	0.9	3.1	0.9	0.4	1.9	6.5	-3.0	-0.7	0.3	-2.2	3.2

Tabelle 7: Abweichung der Parameter vom Orginalsystem (S-System 2.ter Ordnung)

Die ersten Experimente zeigten, dass die Leistung eher vom Problem abhängt als vom Verfahren. Deshalb soll untersucht werden welches die Parameter sind, die die Leistung beeinflussen.

5.2 Schwach besetzte S-Systeme

In den folgenden Versuchen soll untersucht werden, ob die Anzahl der Interaktionen in einem regulatorischen Netzwerk Auswirkungen auf die Inferrierbarkeit hat. Hierzu werden wiederum S-Systeme verwendet, um künstliche Daten zu generieren. Die Tableaus der S-Systeme werden unterschiedlich stark mit Nullen gefüllt. Die Anzahl der Einträge, die nicht Null sind, wird im Folgenden *Kardinalität* genannt.

5.2.1 S-System 2. Ordnung

Zunächst wurde ein S-System 2. Ordnung untersucht. Die Parameter der S-Systeme können der Tabelle 8 entnommen werden.

Kardinalität	g_{11}	g_{12}	g_{21}	g_{22}	h_{11}	h_{12}	h_{21}	h_{22}
0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
2	1	0	0	1	0	0	0	0
3	1	0	0	1	-1	0	0	0
4	1	0	0	1	-1	0	0	-1
5	1	0	0	1	-1	-2	0	-1
6	1	0	0	1	-1	-2	-2	-1
7	1	-1	0	1	-1	-2	-2	-1
8	1	-1	-2	1	-1	-2	-2	-1

Tabelle 8: Kardinalität

α und β wurden jeweils auf $[1, 1]$ gesetzt.

Es wurden jeweils 20 äquidistante Messpunkte zwischen den Zeitpunkten 0 und 2 genommen. Nun wurden wiederum ES und GP zur Inferrierung der Systeme angewandt.

μ	λ	G	Selektion	Mutation	P_{mut}	P_{cross}
20	500	20	Komma	CMA	1	0

Tabelle 9: Parameter ES Kardinalitätsversuch S-Systeme 2. Ordnung

μ	λ	G	Selektion	F	K	T	P_{mut}	P_{cross}
40	1000	10	TS $\gamma = 100$	+, -, *	1000	4	0.8	0.7

Tabelle 10: Parameter GP Kardinalitätsversuch S-Systeme 2. Ordnung

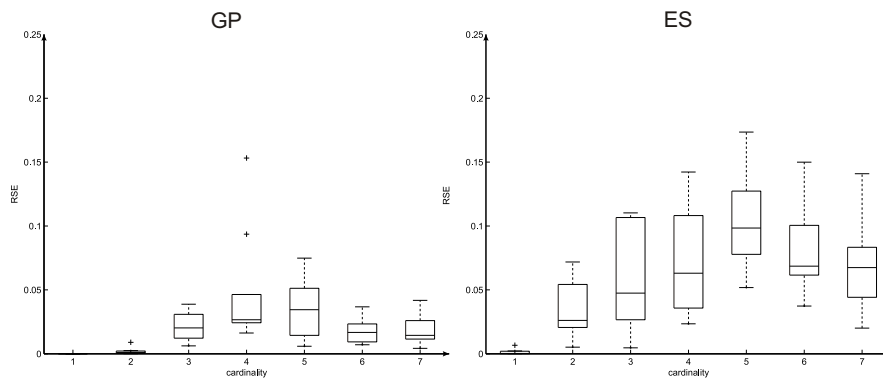


Abb. 16: Vergleich ES/GP Kardinalitätsabhängigkeit der Inferierbarkeit bei S-Systemen 2. Ordnung

Es wurden jeweils 20 Wiederholungen durchgeführt. Anscheinend sind die Problemstellungen (vgl. Abb. 16)) für beide Verfahren einfach zu lösen. Da beide Verfahren alle Systeme jeder Kardinalität gut ($RSE < 0.15$) inferieren können, wobei GP geringfügig im Vorteil zu sein scheint, liegt es nahe Systeme höherer Ordnung zu untersuchen.

5.2.2 S-System 5. Ordnung

Im Folgenden wurden S-Systeme 5. Ordnung untersucht. Als Grundlage wurde ein System aus [6] modifiziert und pro Schritt jeweils 5 Nullen mit anderen Werten ersetzt. Die Dynamiken der modifizierten Systeme können dem Anhang B entnommen werden. Das ursprüngliche System war folgendermaßen parametrisiert:

$$\alpha = 15.0 \quad 10.0 \quad 10.0 \quad 8.0 \quad 10.0 \quad \beta = 10.0 \quad 10.0 \quad 10.0 \quad 10.0 \quad 10.0$$

$$G = \begin{matrix} & \begin{matrix} 0.0 & 0.0 & 1.0 & 0.0 & -0.1 \end{matrix} \\ \begin{matrix} 2.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -0.1 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 2.0 & 0.0 & -1.0 \\ 0.0 & 0.0 & 0.0 & 2.0 & 0.0 \end{matrix} & \begin{matrix} 2.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -0.1 & 2.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 2.0 \end{matrix} \\ H = & \end{matrix}$$

Insgesamt wurden 9 Systeme unterschiedlicher Kardinalität erstellt. Dabei wurde darauf geachtet, dass die Systeme kein chaotisches Verhalten zeigen. ES und GP wurden wie folgt parametrisiert:

μ	λ	G	Selektion	Mutation	P_{mut}	P_{cross}
20	500	20	Komma	CMA	1	0

Tabelle 11: Parameter ES Kardinalitätsversuch S-Systeme 5. Ordnung

μ	λ	G	Selektion	F	K	T	P_{mut}	P_{cross}
40	1000	10	TS $\gamma = 100$	+,-,*	1000	4	0.8	0.7

Tabelle 12: Parameter GP Kardinalitätsversuch S-Systeme 5. Ordnung

Es wurden jeweils 20 Wiederholungen durchgeführt.

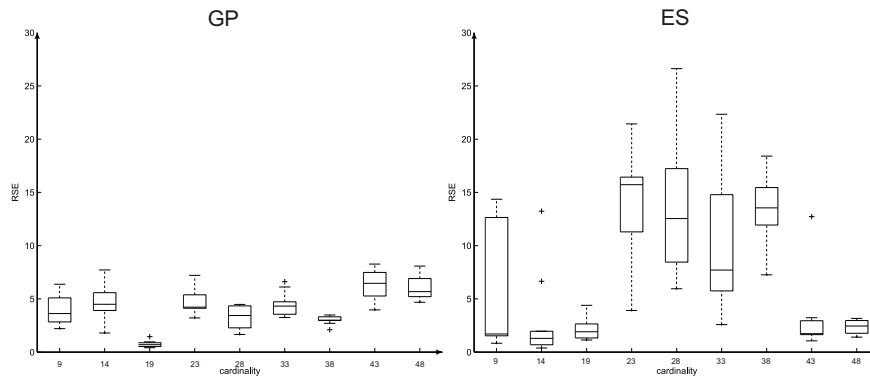


Abb. 17: Vergleich ES/GP Kardinalitätsabhängigkeit der Inferrierbarkeit bei SSystemen 5. Ordnung

Auch hier scheint kein direkter Zusammenhang zwischen Kardinalität und Inferrierbarkeit feststellbar zu sein. Die Schwierigkeit, ein regulatorisches Netzwerk zu inferrieren, scheint nicht proportional von der Kardinalität abzuhängen sondern von anderen Faktoren. Dies führt zur These, dass die Inferrierbarkeit eines Systems von anderen schwierig bestimmbar Faktoren abhängt und somit zum nächsten Versuch.

5.3 Permutationen schwach besetzter S-Systeme

In diesem Versuch galt es die These zu überprüfen, dass die Inferrierbarkeit eines Systems von anderen Faktoren konditioniert wird. Dazu wurden die Parameter des bereits bekannten S-System 5. Ordnung aus dem vorhergehenden Versuch permutiert und so 7 Systeme erstellt, die zwar die gleiche Kardinalität aufweisen, aber untereinander ein völlig unterschiedliches Verhalten zeigen. Bei der Erstellung wurde wieder darauf geachtet, dass Systeme ohne Probleme integrierbar sind und kein chaotisches Verhalten zeigen.

ES und GP wurden wie folgt parametrisiert:

μ	λ	G	Selektion	Mutation	P_{mut}	P_{cross}
20	500	40	Komma	CMA	1	0

Tabelle 13: Parameter ES Permutationsversuch S-Systeme 2. Ordnung
ES wurden jeweils 20 Wiederholungen durchgeführt.

μ	λ	G	Selektion	F	K	T	P_{mut}	P_{cross}
20	500	40	TS $\gamma = 100$	+, -, *	1000	4	0.8	0.7

Tabelle 14: Parameter GP Permutationsversuch S-Systeme 2. Ordnung

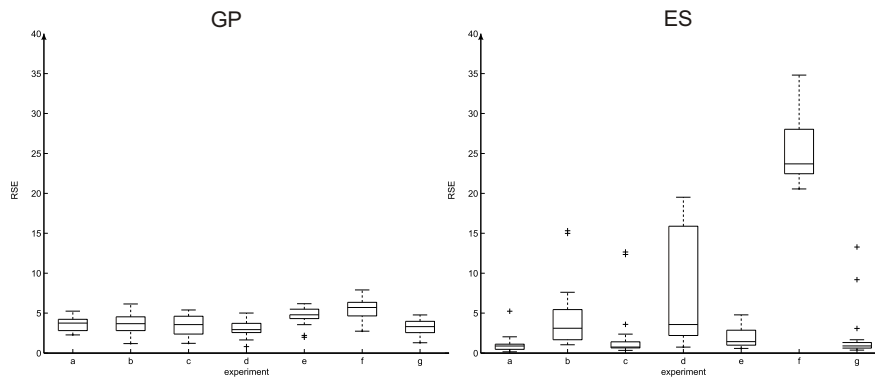


Abb. 18: Vergleich ES/GP Permutationsabhängigkeit der Inferrierbarkeit bei S-Systemen 5. Ordnung

Wie vermutet ist die Inferrierbarkeit von regulatorischen Netzwerken eine schwer vorauszusagende Größe. Für die verschiedenen Systeme wurde Ergebnisse sehr unterschiedlicher Güte erzielt. Dabei erreichten ES in 4 von 7 Fällen bessere Ergebnisse als GP, in den übrigen 3 Fällen jedoch wesentlich schlechtere, sogar absolut unbrauchbare Ergebnisse. GP zeigt sich jedoch über alle untersuchten Permutationen sehr viel stabiler als ES.

5.4 Fazit Vergleich ES/GP

Beide Verfahren zeigten ähnliche Probleme bei der Inferenz des Eulerszillators und des Räuber-Beute-Modells, aber auch vergleichbare Erfolge bei der Inferenz des S-Systems 2. Ordnung. Die Benachteiligung von GP durch den relativ großen Suchraum und die schwache Kausalität von Mutation und Rekombination, tritt bei dieser Problemstellung nicht in Erscheinung (vgl. Abb. 23).

System	ES	GP
Eulerszillator	–	–
Räuber-Beute-Modell	–	–
Tominaga Ordnung 2	+	+

Tabelle 15: Vergleich ES/GP

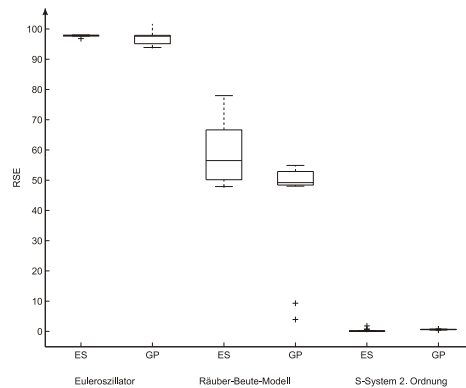


Abb. 19: Vergleich ES/GP über verschiedene einfache Testsysteme

Überraschenderweise zeigt GP sogar oft ein stabileres Konvergenzverhalten als ES. Grundsätzlich scheint GP bezüglich der Inferenzstärke durchaus mit ES konkurrieren zu können, was wahrscheinlich eher an der Schwierigkeit des Problems, als an der Stärke der Verfahren liegt.

Die Untersuchungen zur Komplexität des Inferenzproblems verschiedener Systeme (siehe 5.2 und 5.3) lassen nur den Schluss zu, dass diese sehr problemabhängig und grundsätzlich schwer zu spezifizieren ist. Die Kardinalität, also die Verbindungsdichte in einem Netzwerk, ist wenn überhaupt nicht als einziges Kriterium zu beachten.

Hier sind weitere Untersuchungen nötig um Klarheit zu schaffen.

6 Vergleich der Verfahren GP und SGP

In diesem Kapitel soll die Nützlichkeit von Separationsstrategien am Beispiel von GP untersucht werden. Der Autor möchte darauf hinweisen, dass GP mit Separationsstrategie in keinem Fall in Konkurrenz zu ES steht, da die in der Arbeit untersuchten ES keine Separationsstrategien unterstützen.

6.1 Euleroszillator

Dieses System wurde bereits in 5.1.1 beschrieben. Die Parametrisierung der Verfahren wurde wie folgt vorgenommen:

μ	λ	G	Selektion	F	K	T	P_{mut}	P_{cross}
40	1000	10	TS $\gamma = 100$	+, -, *	1000	4	0.8	0.7

Tabelle 16: Parameter für GP angewandt auf Euleroszillator-Inferenz

μ	λ	G	Selektion	F	K	T	P_{mut}	P_{cross}
20	500	10	TS $\gamma = 100$	+, -, *	1000	4	0.8	0.7

Tabelle 17: Parameter für SGP angewandt auf Euleroszillator-Inferenz

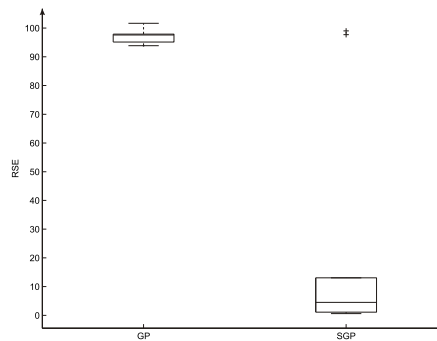


Abb. 20: Vergleich GP/SGP Euleroszillator

Es wurden jeweils 20 Wiederholungen durchgeführt. Die Separationsstrategie schlägt bei diesem Inferenzproblem sehr gut an. Während GP ohne Ausnahme unbrauchbare Ergebnisse erzielt, liefert SGP durchgehend gute Ergebnisse, also Systeme die der Dynamik des Euleroszillators entsprechen.

Nun wurde noch überprüft, ob die inferierten Gleichungen auch in etwa den gesuchten entsprechen. Dazu wurde das beste System, das mit SGP gefunden

wurde schrittweise vereinfacht, wobei vorher alle Konstanten auf die erste Komma-
 mastelle gerundet wurden.

$$\begin{aligned}
 \frac{\delta x_1}{\delta t} &= (((0.4 - 0.4) * (0.4 - 0.5) + x_2) * x_3 = x_2 * x_3 \\
 \frac{\delta x_2}{\delta t} &= (0.3 + (0.2 * (0.8 * 0.5))) * (((x_3 * 0.1) + (0.9 + 0.6)) * ((0.3 - x_2) * (x_3 * x_1))) \\
 &= x_1 x_3 (0.03 x_3 + 0.45 - 0.1 x_2 x_3 - 1.5 x_2) \\
 &= x_1 x_3 (0.45 - 1.5 x_2) \\
 &\approx x_1 x_3 (0.45 - 2.7) \\
 &= -0.63 x_1 x_3 \\
 \frac{\delta x_3}{\delta t} &= (((x_2 + 0.2) * (0.8 - 0)) - (x_2 * 0.7 + 0.5)) \\
 &= 0.8 x_2 + 0.16 - (1.2 x_2) \\
 &= -0.4 x_2 + 0.16
 \end{aligned}$$

Vergleicht man das Ausgangssystem

$$\begin{aligned}
 \frac{\delta x_1}{\delta t} &= x_2 x_3 \\
 \frac{\delta x_2}{\delta t} &= -x_1 x_3 \\
 \frac{\delta x_3}{\delta t} &= -0.51 x_1 x_2
 \end{aligned}$$

wir die eigentliche Ähnlichkeit des Inferenzergebnisses deutlich. Das gesuchte System wurde zwar nicht exakt getroffen, die funktionalen Abhängigkeiten stimmen aber im Großen und Ganzen.

6.2 Räuber-Beute-Modell

Dieses System wurde bereits in 5.1.2 beschrieben. Die Parametrisierung der Verfahren wurde wie folgt vorgenommen:

μ	λ	G	Selektion	F	K	T	P_{mut}	P_{cross}
20	4000	40	TS $\gamma = 100$	+, -, *	1000	4	0.8	0.7

Tabelle 18: Parameter für GP angewandt auf Räuber-Beute-Modell-Inferenz

μ	λ	G	Selektion	F	K	T	P_{mut}	P_{cross}
20	4000	15	TS $\gamma = 100$	+, -, *	1000	4	0.8	0.7

Tabelle 19: Parameter für SGP angewandt auf Räuber-Beute-Modell-Inferenz

Es wurden jeweils 20 Wiederholungen durchgeführt. Wie schon beim Euleroszillator beobachtet erzielt SGP deutlich bessere Ergebnisse. Im Allgemeinen ist bei der Inferenz eine Multistart-Ansatz zu empfehlen.

Hier ein Beispiel für ein Inferenz-Ergebnis von SGP beim Räuber-Beute-Modell:

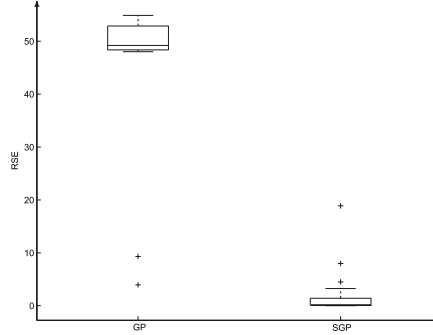


Abb. 21: Vergleich GP/SGP Räuber-Beute-Modell

$$\begin{aligned}
 \frac{\delta x_1}{\delta t} &= x_1 - (x_1 * (x_2 + ((0.4 * x_1) * (0.7 - 0.7)))) \\
 &= x_1 - (x_1 * x_1) \\
 &= x_1(1 - x_2) \\
 \frac{\delta x_2}{\delta t} &= x_2((x_1 - ((0.4 + 0.4) * (0.4 + 0.6))) + (((0.9 + x_1) - (x_1 + 0.9)) - (0.4 * (0.3 * 0.6)))) \\
 &= x_2(x_1 - 0.8 + 0 - 0.1) \\
 &= x_2(x_1 - 0.9)
 \end{aligned}$$

Hier wird die Ähnlichkeit zum Ausgangssystem

$$\begin{aligned}
 \frac{\delta x_1}{\delta t} &= x_1(1 - x_2) \\
 \frac{\delta x_2}{\delta t} &= x_2(x_1 - 1)
 \end{aligned}$$

noch deutlicher als beim Eulerszillator. Die funktionalen Zusammenhänge wurden fast exakt wiedergegeben und die Konstanten entsprechen auch beinahe denen des Ausgangssystems.

6.3 S-System 2. Ordnung

Dieses System wurde bereits in 2.1.3 beschrieben. Die Parametrisierung der Verfahren wurde wie folgt vorgenommen:

μ	λ	G	Selektion	F	K	T	P_{mut}	P_{cross}
20	500	40	TS $\gamma = 50$	+, -, *	100	4	0.8	0.7

Tabelle 20: Parameter für GP angewandt auf die Inferenz eines S-Systems 2. Ordnung

μ	λ	G	Selektion	F	K	T	P_{mut}	P_{cross}
20	500	15	TS $\gamma = 200$	+, -, *	100	4	0.8	0.7

Tabelle 21: Parameter für SGP angewandt auf die Inferenz eines S-Systems 2. Ordnung

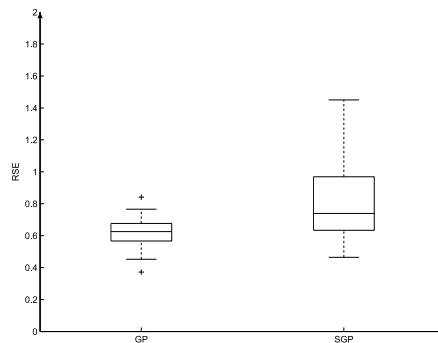


Abb. 22: Vergleich GP/SGP SSystem 2.Ordnung

Es wurden jeweils 20 Wiederholungen durchgeführt. Es ist anzumerken, dass dieses System wohl im allgemein als leicht inferierbar (vgl. 5.1.3) einzustufen ist. SGP performt auch gut, wenn GP gut performt.

GP liefert hier ein Ergebnis das zwar schwer nachvollziehbar ist, jedoch die Dynamik des Zielsystems hat:

$$\begin{aligned} \frac{\delta x_1}{\delta t} &= -x_2^2 + 0.9x_1^2 - 2,7x_1x_2 - 1.4x_1 - 4.2x_2 - 2.9 \\ \frac{\delta x_2}{\delta t} &= 2x_1^2 - x_2^2 + 2.0x_1 - 0.4x_2^2 + 1 \end{aligned}$$

6.4 Fazit Vergleich GP/SGP

SGP schlägt sich bei schwierigen Dynamiken wesentlich besser als GP. Wo GP überhaupt keine Anzeichen von Konvergenz zeigt, konnte durch die Voroptimierung bei SGP meist ein guter Startpunkt für die globale Suche gefunden werden.

System	GP	SGP
Euleroszillator	-	+
Räuber-Beute-Modell	-	++
Tominaga Ordnung 2	+	+

Tabelle 22: Vergleich GP/SGP

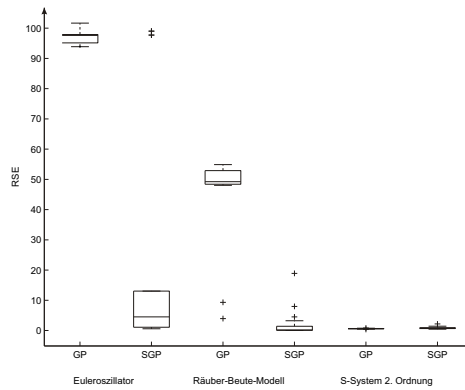


Abb. 23: Vergleich GP/SGP über verschiedene einfache Testsysteme

GP zeigt zum Beispiel beim Euleroszillator und dem Räuber-Beute-Modell überhaupt keine Anzeichen für Konvergenz. SGP jedoch gelingt durch die Voroptimierung eine geeignete Positionierung im Suchraum und somit die meist erfolgreiche Inferenz der zugrundeliegenden Systeme, wie auch die Analyse der erzeugten Systeme bestätigen konnte. Das zeigt, dass Separationsstrategien einen vielversprechenden Ansatz darstellen. Hier sei noch einmal ausdrücklich darauf hingewiesen, dass das Prinzip der Separierung nicht nur für GP anwendbar ist, sondern auch für ES und andere geeignete Optimierungsverfahren.

7 Schlussfolgerungen

Es wurde bestätigt, dass mit Evolutionären Algorithmen wie Genetischer Programmierung und Evolutionsstrategien regulatorische Netzwerke inferriert werden können, die in ihrer Dynamik gegebenen Zielsystemen ähnlich sind, jedoch nicht immer in ihrer Struktur. Im Vergleich der Verfahren konnten Unterschiede in Kovergenzverhalten und Stabilität festgestellt werden.

GP tritt zwar mit einem großen Suchraum und schwach kausalen Evolutionsoperatoren an, zeigt aber im Vergleich eine ähnliche Inferenzstärke wie Evolutionsstrategien, angewandt auf S-Systeme. Diese verzeichnen zwar einen leichten Vorteil in der Inferenz von S-Systemen, aber überraschenderweise eine schlechtere Stabilität als GP. Weiter scheint der Lösungsraum in dem sich der ES bewegt, stark multimodal zu sein und führt deshalb zu oft schwer interpretierbaren Ergebnissen. Also bringt die Flexibilität von GP durchaus die erhofften Vorteile.

Interessant ist weiterhin, dass eine Eigenschaft erkennbar ist, die als spezielle Komplexität des Inferenzproblems bezeichnet werden. Verschiedene Systeme gleicher Ordnung sind verschieden schwer zu inferieren. Die Untersuchungen zeigten, dass die Inferierbarkeit, unabhängig von der Kardinalität, ein schwer vorher zu bestimmende Größe darstellt. GP zeigte sich auch hier in der Untersuchungsreihe etwas unempfindlicher als ES.

In dieser Arbeit wurden Separationsstrategien auf ihre Nützlichkeit getestet. Die beschriebene Voroptimierung brachte große Vorteile, und erlaubte die Inferierung von Systemen die ohne Separation, wohl gar nicht zu inferieren gewesen wären. Der Autor sieht dieses Ergebnis als besonders wichtig an. Es ist zu vermuten, dass Separationsstrategien in zukünftigen Inferenzansätzen eine große Rolle spielen werden und müssen. Letzendlich scheitert die Inferenz von realen genregulatorischen Netzwerken bis heute an der Dimensionalität des Problems.

A EDER

EDER (Evolutionary Differential Equation Inferrer) ist ein Programmpaket, das im Rahmen dieser Studienarbeit entwickelt wurde. Als Programmiersprache wurde Java gewählt. Folgende Methoden zur Inferenz von Differentialgleichungssystemen aus Messdaten wurden implementiert:

- Parametersuche von S-Systemen durch Evolutionsstrategien
- Inferenz allgemeiner Differentialgleichungssysteme mit Genetischer Programmierung
- Inferenz allgemeiner Differentialgleichungssysteme mit Genetischer Programmierung durch Separation

EDER erhält als Eingabe zwei XML-Dateien: eine enthält die Messdaten aus den Experimenten, die andere die Konfiguration. Konfigurierbar ist die Art des Verfahrens und die verfahrensbedingten Parameter wie: Größe der Elterngeneration, Größe der Nachkommenpopulation, maximale Generationsanzahl, Evolutionsoperatoren, etc.. Am Ende der Berechnungen schreibt *EDER* die Messdaten, die Konfiguration, und die besten gefundenen Systeme in einen Report im XML-Format. Dieser kann dann mit einem zweiten Programm dem *ReportViewer* betrachtet und ausgewertet werden.

Bei der Implementierung wurde darauf geachtet, dass ein Vergleich zwischen den Verfahren ermöglicht werden soll. Inkonsistenzen in der Problemstellung sollten vermieden werden. Da die Konfiguration in einer Datei festgelegt wird, können Fehler in der Parametrisierung von Wiederholungsexperimenten vermieden werden. *EDER* ist ein Kommandozeilen-Programm und erlaubt so eine unkomplizierte Automatisierung von Experimenten. *EDER* verwendet zum Teil das Framework *JavaEva* das am Lehrstuhl für Rechnerarchitektur an der Universität Tübingen von Holger Ulmer und Felix Streichert entwickelt wurde.

A.1 Implementation von Evolutionsstrategien

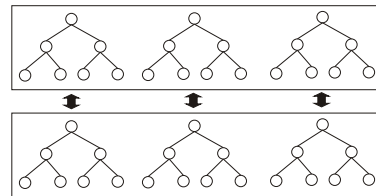
EDER verwendet Evolutionsstrategien zur Inferenz von S-Systemen. Zur Auswahl stehen Komma- und Plus-Selektion. Als Mutationsoperatoren stehen Kovarianzmatrixadaptation [12] und Hauptvektoradaptation zur Verfügung. Ferner können die Größe der Elternpopulation und der Nachkommenpopulation festgelegt werden. Ansonsten entspricht die Implementierung der Verfahren der Beschreibung in Abschnitt 4.1.

A.2 Implementation von GP

Zur Inferenz von allgemeinen Differentialgleichungssystemen mit Genetischer Programmierung, mussten einige Überlegungen getroffen werden. Individuen mussten so ausgelegt werden, dass n Gleichungen gleichzeitig optimiert werden können. Ein Individuum in der GP-Implementierung von *EDER* besteht also aus n Programmbäumen.

Rekombination findet jeweils nur zwischen den korrespondierenden Programmbäumen statt, da davon ausgegangen werden kann, dass nur die zusammenpassenden Gleichungen, Strukturen enthalten die für genau diese Teilgleichung wichtig sind. Zwei Rekombinationsoperatoren wurden implementiert. Die naive Rekombination tauscht einfach zwei Programmbäume aus. Wie erwartet führt das zum unkontrollierten Wachstum der Programmbäume. Ein tiefenbewahrender Rekombinationsoperator wurde implementiert, durch den eine maximale Baumtiefe bei der Rekombination erhalten bleibt. Als Mutationsoperatoren stehen auch eine naive und eine tiefenbewahrende Variante zur Verfügung.

Die primitiven Funktionen in dieser Implementation haben alle feste Stelligkeit. Ein ganzer Satz von Funktionen steht zur Verfügung: Addition, Subtraktion, Multiplikation, geschützte Division, Potenzen, Wurzel, Logarithmus, Sinus, Cosinus, Maximumsbildung und eine Fallunterscheidung. Diese Bibliothek kann einfach erweitert werden.



Konstanten werden aus einem beliebig großen Pool von Zufallswerten ausgewählt, deren allen Individuen gemein ist. Die Zufallswerte am Anfang der Evolutionslaufs generiert und bleiben über den ganzen Evolutionslauf fest.

A.3 Implementation von GP mit Separierung

Um die Performance der Suche mit Genetischer Programmierung zu erhöhen wurde die oben beschriebene Separierung implementiert. Zunächst werden einzelne Gleichungen gesucht. Zwischen den Messpunkten wird mit kubischen Splines interpoliert.

A.4 EDER-Einführung

Das Programm EDER stellt 3 Verfahren zu Inferrierung von Differentialgleichungssystemen aus Messdaten zur Verfügung. Das Program ist ein Kommandozeilen-Programm im Framework JavaEva, das am Lehrstuhl für Rechnerarchitektur an der Universität Tübingen von Holger Ulmer, Felix Streichert und mehreren studentischen Mitarbeitern entwickelt wurde.

Um den Aufruf der Software zu erleichtern wurden 2 kleine Skripte erstellt, jeweils eins für UNIX-Umgebungen (eder.sh) und eins für Windows (wineder.sh). Im folgenden wird der Aufruf der Software unter Unix-Umgebungen erklärt. Soll die Software unter Windows eingesetzt werden muss statt eder.sh wineder.sh aufgerufen werden.

Ein Aufruf von EDER sieht so aus:

```
./eder.sh <Konfigurationsdatei> <Messdaten-Datei> <Logdatei> [Seed]
```

EDER entnimmt der Konfigurationsdatei, welches Verfahren mit welchen Parametern eingesetzt werden soll. Des Messdaten-Datei entnimmt EDER die Zeitverläufe für die ein passendes Modell gefunden werden soll. In die Logodatei werden dann die Ergebnisse geschrieben. Der Parameter Seed ist eine natürliche Zahl, wird er angegeben, wird der Zufallszahlengenerator mit diesem Wert initialisiert.

Eine typische Konfiguration für das Verfahren ES sieht so aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<experiment expid="1">
  <eamethod name="ES">
    <mu intvalue="80" />
    <lambda intvalue="1000" />
    <maxgenerations intvalue="40" />
    <selectionoperator stringvalue="Komma" />
    <mutationoperator stringvalue="CMA" />
  </eamethod>
</experiment>
```

mu steht für die Elternpopulationsgrösse, lambda für die Nachkommenpopulationsgrösse. Die Anzahl der Generationen wird durch maxgenerations gesetzt. Als Selektionsoperator kann Komma- oder Plus-Selektion ausgewählt werden. Mit Kovarianzmatrixadaption (CMA) und Hauptvektotradaption (MVA) stehen zwei Mutationsoperatoren zur Verfügung.

Für GP/SGP kommen noch einige Optionen dazu:

```
<?xml version="1.0" encoding="UTF-8"?>

<experiment expid="1">
  <eamethod name="GP">
    <mu intvalue="20" />
    <lambda intvalue="4000" />
    <maxgenerations intvalue="40" />
    <pcross doublevalue="0.7" />
    <pmut doublevalue="0.8" />

    <maxdepth intvalue="5" />
    <fuzzyness doublevalue="0.2" />
    <constantno intvalue="1000" />

    <populationtype stringvalue="KozaStylePopulation" />
    <selectionoperator stringvalue="TournamentSelection">
      <gamma intvalue="200" />
    </selectionoperator>
    <mutationoperator stringvalue="MutationDepthPreserving" />
    <crossoveroperator stringvalue="CrossoverDepthPreserving" />

  </eamethod>
</experiment>

<functionset>
  <primitivefunction
  stringvalue="javaeva.server.oa.gp.lisp.basicfunctions.MultFunction" />
  <primitivefunction
  stringvalue="javaeva.server.oa.gp.lisp.basicfunctions.SumFunction" />
  <primitivefunction
  stringvalue="javaeva.server.oa.gp.lisp.basicfunctions.SubtractionFunction" />
</functionset>
```

```
</eamethod>
</experiment>
```

Die Konfiguration für GP und SGP sind weitgehend gleich, nur der Parameter `eamethod` ändert sich. SGP verwendet für jeden Optimierungsschritt, dann die spezifizierten Parameter. `pcross` und `pmut` geben dem Anwender die Möglichkeit die Rekombinations bzw. Mutationswahrscheinlichkeit einzustellen. `maxdepth` ist die maximale Baumtiefe. `fuzzyness` ist die Wahrscheinlichkeit, dass bei der zufälligen Konstruktion eines Baumes ein Terminal (Konstante, Variable) eingebaut, bevor die maximale Baumtiefe erreicht ist. Der Parameter `populationtype` und `selectionoperator` sollten nicht verändert werden, der Parameter `gamma` gibt an, wie viele Stichproben bei der Tournament-Selektion gezogen werden sollen. Als Mutationsoperatoren bzw. Rekombinationsoperatoren stehen jeweils eine tiefenbewahrende Variante ('MutationDepthPreserving', 'CrossoverDepthPreserving') und eine nicht tiefenbewahrende ('MutationNoDepthPreserving', 'CrossoverNoDepthPreserving') zur Wahl. `functionset` enthält die primitiven Funktionen, die zur Verfügung stehen sollen. Es muss der Klassenname angegeben werden, um die Funktion einzubinden. Jede Klasse kann eingebunden werden, die das Interface `javaeva.server.oa.gp.lisp.PrimitiveFunction` implementiert:

```
package javaeva.server.oa.gp.lisp;
import java.io.*;

public interface PrimitiveFunction extends Serializable {

    public double evaluate(double[] params, double[] cons);
    public String toString(double[] params, double[] cons);
    public int arity();
    public String getSymbol();
    public String toLatex(String[] fromchilds);

}
```

Implementiert wurden:

Funktion	Klassenname
Addition	javaeva.server.oa.gp.lisp.basicfunctions.SumFunction
Subtraktion	javaeva.server.oa.gp.lisp.basicfunctions.SubtractionFunction
Multiplikation	javaeva.server.oa.gp.lisp.basicfunctions.MultFunction
Sinus	javaeva.server.oa.gp.lisp.basicfunctions.SinFunction
Cosinus	javaeva.server.oa.gp.lisp.basicfunctions.CosFunction
Logarithmus	javaeva.server.oa.gp.lisp.basicfunctions.LogFunction
Division	javaeva.server.oa.gp.lisp.basicfunctions.DivFunction
Maximum	javaeva.server.oa.gp.lisp.basicfunctions.MaxFunction
Quadratwurzel	javaeva.server.oa.gp.lisp.basicfunctions.SqrtFunction
Potenz	javaeva.server.oa.gp.lisp.basicfunctions.PowerFunction

Tabelle 23: Implementierte Primitiva

mit jeweils festen Aritäten.

Die Messdaten sollen im folgenden Format vorliegen:

```
<?xml version="1.0" encoding="UTF-8"?>
<timeseries>
  <timestamp doublevalue="0.0">
    <yval doublevalue="5.0" id="1" />
    <yval doublevalue="5.0" id="2" />
  </timestamp>
  <timestamp doublevalue="0.8">
    <yval doublevalue="3.238191322753876" id="1" />
    <yval doublevalue="-0.04728690089707359" id="2" />
  </timestamp>
  <timestamp doublevalue="1.6">
    <yval doublevalue="1.6897212452718868" id="1" />
    <yval doublevalue="-1.7914120342656212" id="2" />
  </timestamp>
  <timestamp doublevalue="2.4000000000000004">
    <yval doublevalue="-0.0914898333228086" id="1" />
    <yval doublevalue="-2.0872101698442864" id="2" />
  </timestamp>
  <timestamp doublevalue="3.2">
    <yval doublevalue="-1.3913805466449964" id="1" />
    <yval doublevalue="-1.237650942581827" id="2" />
  </timestamp>
  <timestamp doublevalue="4.0">
    <yval doublevalue="-1.702706439870367" id="1" />
    <yval doublevalue="0.12453455480031694" id="2" />
  </timestamp>
</timeseries>
```

Eine Zeitreihe (timeseries) enthält mehrere Zeitpunkte (timestamps) an denen mehrere Werte (yval) gemessen wurden. Werte mit gleicher id sind dem selben Stoff zuzuordnen.

Nach dem EDER den Evolutionslauf beendet hat, kann die Logdatei, welche die Ergebnisse enthält, mit dem Programm ReportViewer betrachtet werden. Gestartet wird dieser entweder mit ./rp.sh oder ./winrp.sh.

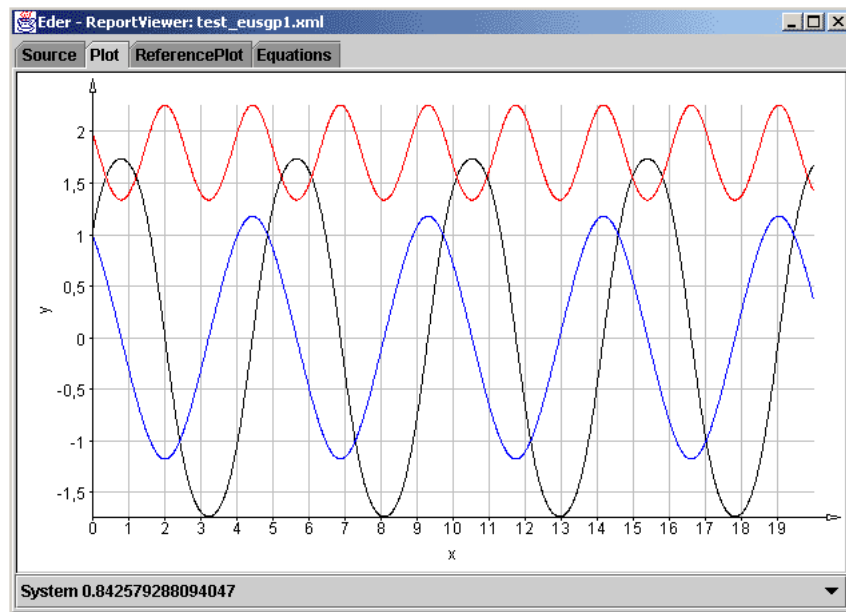


Abb. 25: Reportviewer

Mit 'Load Report' kann nun eine Logdatei geladen werden. Im Reiter 'Source' wird die Datei als einfach als Text dargestellt, in 'Plot' können die Plot der besten gefundenen Lösungen betrachtet werden, in 'Reference-Plot' können nochmal die Messwerte verglichen werden. In Reiter 'Equations' finden sich entweder, die generierten Funktion bei den Methoden GP und SGP, oder die Tableaus für das S-System bei der Methode ES.

B Dynamiken

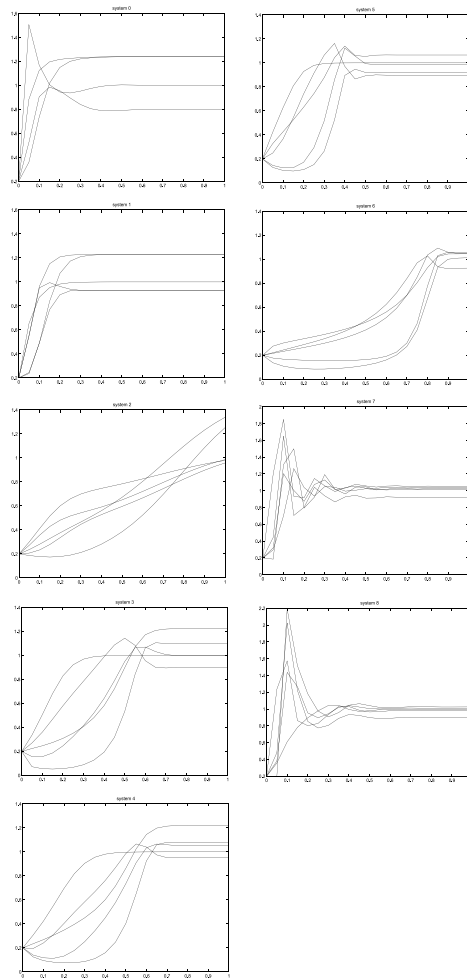


Abb. 26: Dynamiken Kardinalitätsversuch S-System 5. Ordnung

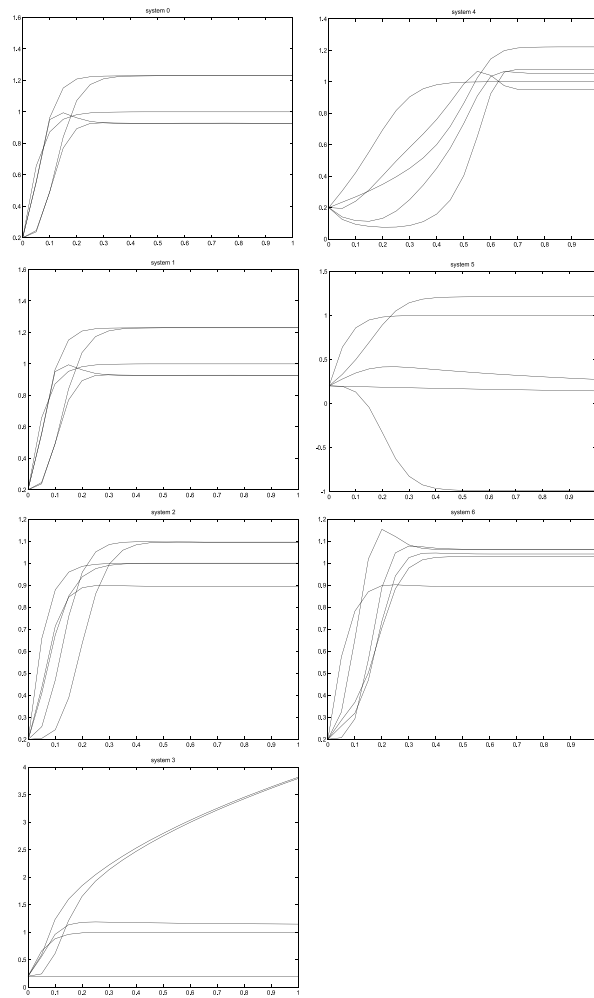


Abb. 27: Dynamiken Permutationsversuch S-System 5. Ordnung

C Verzeichnisse

Abbildungsverzeichnis

1	Microarray-Scan (Falschfarben)	4
2	schematische Darstellung eines genregulatorischen Netzwerks . .	6
4	Plot des Expressionsverlaufs durch Simulation eines booleschen Netzwerks	7
3	boolesches Netzwerk	7
5	einfaches regulatorisches Netzwerk	10
6	schematische Darstellung des Basis-EA-Algorithmus	11
7	ES-Mutation	14
8	LISP-Programmbaum: Dieser Baum beschreibt die LISP - Funktion (+ (* 23 42) (- 47 11)) oder in Infix-Notation $(x, y) = x * 42 + y * 11$. .	16
9	Baum-Mutation	17
10	Baumrekombination	17
11	Euleroszillator	20
12	Vergleich ES/GP Euleroszillator	21
13	Räuber-Beute-Modell	21
14	Vergleich ES/GP Räuber-Beute-Modell	22
15	Vergleich ES/GP SSystem 2.Ordnung	23
16	Vergleich ES/GP Kardinalitätsabhängigkeit der Inferrierbarkeit bei SSystemen 2. Ordnung	25
17	Vergleich ES/GP Kardinalitätsabhängigkeit der Inferrierbarkeit bei SSystemen 5. Ordnung	26
18	Vergleich ES/GP Permutationsabhängigkeit der Inferrierbarkeit bei SSystemen 5. Ordnung	27
19	Vergleich ES/GP über verschiedene einfache Testsysteme	28
20	Vergleich GP/SGP Euleroszillator	29
21	Vergleich GP/SGP Räuber-Beute-Modell	31
22	Vergleich GP/SGP SSystem 2.Ordnung	32
23	Vergleich GP/SGP über verschiedene einfache Testsysteme . . .	33
24	Rekombination von Individuen	36
25	Reportviewer	40
26	Dynamiken Kardinalitätsversuch S-System 5. Ordnung	41
27	Dynamiken Permutationsversuch S-System 5. Ordnung	42

Tabellenverzeichnis

1	Parameter für ES angewandt auf Euleroszillator-Inferenz	20
2	Parameter für GP angewandt auf Euleroszillator-Inferenz	20
3	Parameter für ES angewandt auf Räuber-Beute-Modell-Inferenz . .	22
4	Parameter für GP angewandt auf Räuber-Beute-Modell-Inferenz . .	22
5	Parameter für ES angewandt auf die Inferenz eines S-Systems 2. Ordnung	23

6	Parameter für GP angewandt auf die Inferenz eines S-Systems 2. Ordnung	23
7	Abweichung der Parameter vom Orginalsystem (SSystem 2.ter Ordnung)	24
8	Kardinalität	24
9	Parameter ES Kardinalitätsversuch S-Systeme 2. Ordnung	25
10	Parameter GP Kardinalitätsversuch S-Systeme 2. Ordnung	25
11	Parameter ES Kardinalitätsversuch S-Systeme 5. Ordnung	26
12	Parameter GP Kardinalitätsversuch S-Systeme 5. Ordnung	26
13	Parameter ES Permutationsversuch S-Systeme 2. Ordnung	27
14	Parameter GP Permutationsversuch S-Systeme 2. Ordnung	27
15	Vergleich ES/GP	28
16	Parameter für GP angewandt auf Eulerszillator-Inferenz	29
17	Parameter für SGP angewandt auf Eulerszillator-Inferenz	29
18	Parameter für GP angewandt auf Räuber-Beute-Modell-Inferenz	30
19	Parameter für SGP angewandt auf Räuber-Beute-Modell-Inferenz	30
20	Parameter für GP angewandt auf die Inferenz eines S-Systems 2. Ordnung	31
21	Parameter für SGP angewandt auf die Inferenz eines S-Systems 2. Ordnung	32
22	Vergleich GP/SGP	32
23	Implementierte Primitiva	38

Literatur

- [1] AKUTSU, T. ; KUHARA, S. ; MARUYAMA, O. ; MIYANO, S.: Identification of gene regulatory networks by strategic gene disruptions and gene overexpressions. In: ASAI, K. (Hrsg.) ; MIYANO, S. (Hrsg.) ; TAKAGI, T. (Hrsg.): *Proceedings of the 9th annual ACM-SIAM symposium on Discrete algorithms*. San Francisco, California, USA : ACM Press, 1998. – ISBN 0-89871-410-9, S. 695–702
- [2] LIANG, S. ; FUHRMAN, S. ; SOMOGYI, R.: Reveal, a general reverse engineering algorithm for inference of genetic network architectures. 3 (1998), S. 18–29
- [3] WEAVER, D. C. ; WORKMAN, C. T. ; STORMO, G. D.: Modeling regulatory networks with weight matrices. In: ALTMAN, R. B. (Hrsg.) ; DUNKER, A. K. (Hrsg.) ; HUNTER, L. (Hrsg.) ; KLEIN, T. E. (Hrsg.): *Pacific Symposium on Biocomputing* Bd. 4. Singapore : World Scientific Press, 1999, S. 112–123
- [4] IRVING, D.H. ; SAVAGEAU, M.A.: Efficient solution of nonlinear ordinary differential equations expressed in S-systems canonical form. In: *SIAM Journal of Numerical Analysis* 27 (1990), S. 704–735

- [5] SAVAGEAU, M.A.: 20 years of S-systems. In: VOIT, E.O. (Hrsg.): *Canonical Nonlinear Modeling. S-systems Approach to Understand Complexity*. New York : Van Nostrand Reinhold, 1991, S. 1–44
- [6] TOMINAGA, D. ; OKAMOTO, M. ; MAKI, Y. ; WATANABE, S. ; EGUCHI, Y.: Nonlinear Numerical Optimization Technique Based on Genetic Algorithm for Inverse Problem: Towards the Inference of Genetic Networks. In: *Proceedings of Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, 2000, S. 251–258
- [7] SAKAMOTO, Erina ; IBA, Hitoshi: Inferring a System of Differential Equations for a Gene Regulatory Network by using Genetic Programming. In: *Proceedings of Congress on Evolutionary Computation*. COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea : IEEE Press, 27-30 2001. – ISBN 0–7803–6658–1, S. 720–726
- [8] ANDO, Shin ; IBA, Hitoshi ; SAKAMOTO, Erina: Modeling Genetic Network by Hybrid GP. In: FOGEL, David B. (Hrsg.) ; EL-SHARKAWI, Mohamed A. (Hrsg.) ; YAO, Xin (Hrsg.) ; GREENWOOD, Garry (Hrsg.) ; IBA, Hitoshi (Hrsg.) ; MARROW, Paul (Hrsg.) ; SHACKLETON, Mark (Hrsg.): *Proceedings of Congress on Evolutionary Computation*, IEEE Press, 2002. – ISBN 0–7803–7278–6, S. 291–296
- [9] KOZA, J.R. ; MYDLOWEC, W. ; LANZA, G. ; YU, J. ; KEANE, M.A.: Reverse Engineering of Metabolic Pathways from Observed Data Using Genetic Programming. In: *Pacific Symposium on Biocomputing* Bd. 6, 2001, S. 434–445
- [10] KOZA, John R.: Genetic programming. In: WILLIAMS, James G. (Hrsg.) ; KENT, Allen (Hrsg.): *Encyclopedia of Computer Science and Technology* Bd. 39, Marcel-Dekker, 1998, S. 29–43
- [11] KOZA, John R.: Evolution of emergent cooperative behavior using genetic programming. In: PATON, Ray (Hrsg.): *Computing with Biological Metaphors*. London, UK : Chapman & Hall, 1994, S. 280–297
- [12] HANSEN, N. ; OSTERMEIER, A.: Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In: *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, 1996, S. 312–317