Brandenburgische
Technische Universität
Cottbus - Senftenberg

**Faculty of Mathematics,
Natural Sciences and
Computer Science**

Institute of Computer Science

COMPUTER SCIENCE REPORTS

Report 01/16

November 2016

Fascination of Molecular Computing: Student's Contributions Convey Findings of Interdisciplinary Research

Henry Lieske
Daniel Noack
Christoph Schwalbe
Elisabeth Vogel
Thomas Hinze

Computer Science Reports
Brandenburg University of Technology Cottbus - Senftenberg
ISSN: 1437-7969

Send requests to: BTU Cottbus - Senftenberg
Institut für Informatik
Postfach 10 13 44
D-03013 Cottbus

Henry Lieske
Daniel Noack
Christoph Schwalbe
Elisabeth Vogel
Thomas Hinze
thomas.hinze@b-tu.de
https://www.b-tu.de/institut-fuer-informatik/

Fascination of Molecular Computing: Student's Contributions Convey Findings of Interdisciplinary Research

Computer Science Reports
01/16
November 2016

Brandenburg University of Technology Cottbus - Senftenberg

Faculty of Mathematics, Natural Sciences and Computer Science

Institute of Computer Science

Computer Science Reports
Brandenburg University of Technology Cottbus - Senftenberg
Institute of Computer Science

Head of Institute:
Prof. Dr. Petra Hofstedt hofstedt@b-tu.de
BTU Cottbus - Senftenberg
Institut für Informatik
Postfach 10 13 44
D-03013 Cottbus

Research Groups:	Headed by:
Computer Engineering	Prof. Dr. H. Th. Vierhaus
Computer Network and Communication Systems	Prof. Dr. H. König
Data Structures and Software Dependability	Prof. Dr. M. Heiner
Database and Information Systems	Prof. Dr. I. Schmitt
Programming Languages and Compiler Construction	Prof. Dr. P. Hofstedt
Software and Systems Engineering	Prof. Dr. C. Lewerentz
Theoretical Computer Science	Prof. Dr. K. Meer
Graphics Systems	Prof. Dr. D. Cunningham
Systems	Prof. Dr. R. Kraemer
Distributed Systems and Operating Systems	Prof. Dr. J. Nolte
Internet-Technology	Prof. Dr. G. Wagner

CR Subject Classification (1998): I.2.8, F.1.1, C.1.3

Printing and Binding: BTU Cottbus - Senftenberg

ISSN: 1437-7969

Computer Science Report 01/2016
Brandenburgische Technische Universität Cottbus-Senftenberg
Fakultät 1 MINT –
Mathematik, Informatik, Physik, Elektro- und Informationstechnik
Institut für Informatik

Henry Lieske
Daniel Noack
Christoph Schwalbe
Elisabeth Vogel
Thomas Hinze

Faszination Molekulares Rechnen

Studentische Beiträge vermitteln Erkenntnisse interdisziplinärer Forschung

Cottbus, 03. November 2016

Kontakt:
PD Dr.-Ing. habil. Thomas Hinze
Brandenburgische Technische Universität Cottbus-Senftenberg
Fakultät 1, Institut für Informatik
Postfach 10 13 44
03013 Cottbus

thomas.hinze@b-tu.de

Inhaltsverzeichnis

Vorwort

Biologische Computer nach dem Vorbild der Natur bieten eine interessante Alternative zu derzeit etablierten Rechnerarchitekturen, Programmierparadigmen und algorithmischen Konzepten. Die zugrunde liegenden Mechanismen der Informationsverarbeitung gelten als zuverlässig, anpassungsfähig und effizient. Sie beruhen größtenteils auf molekularen Interaktionen. Moleküle oder molekulare Systeme dienen hierbei als Speichermedium und übernehmen die Rolle des Datenträgers, auf dem feinabgestimmte biochemische Reaktionen operieren und gezielte Wechselwirkungen stattfinden. Daraus resultiert eine alternative Hardware im Nanometermaßstab, deren ingenieurtechnische Erschließung über die Bionik hinaus mannigfaltige Anwendungen verspricht, die die Informatik in vielerlei Hinsicht bereichern und weiterentwickeln können. Mit dem zunehmenden Verständnis molekularbiologischer Prozesse lässt sich die Idee, Biopolymere als Datenträger einzusetzen und gezielt zu verändern, immer besser verwirklichen. Darauf basierende biomolekulare Rechentechnik *in vitro* und *in vivo* verspricht hohe Speicherkapazität und -dichte, Miniaturisierung, Energieeffizienz sowie eine massiv datenparallele Informationsverarbeitung.

Die ersten erfolgreichen Experimente zum molekularen Rechnen liegen bereits mehr als 20 Jahre zurück. 1994 ist es einer Forschungsgruppe um den Informatiker Leonard Adleman erstmals gelungen, mithilfe von DNA-Strängen im Reagenzglas eine Instanz des Hamiltonkreis-Problems zu lösen. Diese Aufgabe gilt als besonders rechenintensiv. Es geht hierbei darum, mit einer gegebenen Landkarte aus Städten und Verbindungswegen zwischen den Städten herauszufinden, ob es eine Rundreise entlang von Verbindungswegen gibt, so dass jede Stadt genau einmal besucht wird und die Rundreise danach wieder am Ausgangspunkt endet. Benötigt man eine uneingeschränkt korrekte Lösung, so kennt die Informatik dafür derzeit kein qualitativ besseres Verfahren als alle potentiellen Reiserouten zu erzeugen und jede einzelne daraufhin zu überprüfen, ob sie die geforderten Kriterien einer Rundreise erfüllt. Der Algorithmus sucht folglich die Lösung durch Ausprobieren der möglichen Lösungskandidaten. Solche Aufgaben heißen kombinatorische Suchprobleme und stellen rechentechnisch eine große Herausforderung dar. Aus einer Landkarte mit n Städten ergeben sich bis zu $(n-1)\cdot(n-2)\cdot(n-3)\cdot\ldots\cdot 3\cdot 2\cdot 1$ abzuprüfende Reiserouten. Schon bei 20 Städten wären das mehr als 10^{17} (eine 1 gefolgt von 17 Nullen) Stück. Durch einige Kniffe lässt sich die Anzahl der Routen noch etwas reduzieren, ohne dass eine Rundreise übersehen wird, aber es bleibt ein hoher Aufwand. Elektronische Rechner benötigen sehr viel Zeit dafür. Erschwerend erweist sich, dass zusätzliche Städte den Suchraum besonders stark aufblähen. Nimmt man nur eine weitere

Stadt hinzu, so wird sich die Anzahl potentieller Reiserouten mehr als verdoppeln. Das heißt, dass auch leistungsfähigere Hardware schnell wieder an ihre Grenzen stößt. Molekulares Rechnen zeigt eine vielversprechende Idee auf, die Grenzen bei der praktischen Lösung kombinatorischer Suchprobleme ein deutliches Stück weiter zu schieben. Schafft man es, die Reiserouten jeweils als DNA-Stränge in einem gemeinsamen Reagenzglas zu repräsentieren, lässt sich die Zeit für die Problemlösung verkürzen. Während ein elektronischer Rechner alle Reiserouten nacheinander erzeugt und abprüft, geschieht dies im Reagenzglas gleichzeitig, also datenparallel, weil chemische Reaktionen gleichzeitig auf alle Moleküle wirken. Ausgehend vom Adleman-Experiment, das für eine Landkarte mit 7 Städten und 13 Verbindungswegen eine erfolgreiche Implementierung eines molekularen Algorithmus gezeigt hat, sind inzwischen zahlreiche weitere und verbesserte Ansätze entstanden, die eine Vielzahl von Aufgabenstellungen der Informatik in unkonventioneller Weise erstaunlich effizient zu lösen vermögen.

Weltweit arbeiten mehr als 5000 Wissenschaftler auf dem Gebiet des molekularen Rechnens nach dem biologischen Vorbild der Natur. Daraus haben sich im Laufe der Jahre mehrere Facetten gebildet wie zum Beispiel das evolutionäre Computing, Schwarmalgorithmen, künstliche Immunsysteme, chemisches Rechnen auf Basis von Stoffkonzentrationen, DNA-Computing oder das Rechnen mithilfe dynamischer Raumstrukturen (Membrane Computing). Eine Vielzahl von praktischen und theoretischen Forschungsbeiträgen sowie eine zunehmende Zahl von Patenten und vermarktungsfähigen Anwendungen flankiert die Entwicklung des Wissensgebietes. Beispielsweise lassen sich Wertgegenstände mit einem langlebigen künstlichen DNA-Fingerabdruck versehen, der eine zuverlässige Zuordnung – auch noch nach langer Zeit – ermöglicht. Das Wissensgebiet des molekularen Rechnens ist geprägt durch eine hohe *Interdisziplinarität*. Zusätzlich zur klassischen Informatik benötigt man weiteres Spezialwissen, zum Beispiel in der (Bio)Chemie, in der Molekular- und Systembiologie, in den Lebenswissenschaften, aber auch in der Gesundheits- und Verhaltensforschung. Fachartikel, die neue Erkenntnisse oder Verfahren für das molekulare Rechnen vorstellen, sind deshalb für fachfremde Leser oft schwer zu verstehen. Es wird ein hohes Maß an interdisziplinärem Wissen vorausgesetzt und darauf aufbauend werden neue Ergebnisse in kompakter Form unter Verwendung vieler spezifischer Fachbegriffe dargeboten. Dies verwundert nicht, denn die Hauptleserzielgruppe solcher Fachartikel besteht aus Wissenschaftlern, die mit dem vorausgesetzten Wissen vertraut sind. Darüber hinaus ist der Platz in hochrangigen wissenschaftlichen Fachzeitschriften knapp, was zu einem Schreibstil mit hoher Informationsdichte und mithin schwererer Allgemeinverständlichkeit zwingt.

Genau hier setzt das zentrale Anliegen des Seminars „Molecular Computing“ an: Interessante Erkenntnisse und Resultate auf dem Gebiet des molekularen Rechnens sollen systematisch erschlossen und allgemeinverständlich aufbereitet werden. Dies schließt eine tiefgründige Auseinandersetzung mit den jeweiligen Einzelaspekten der zugrunde liegenden Arbeiten sowie weitreichende wissenschaftliche Umfeldrecherchen ein, um die Thematik geistig zu durchdringen und ein in sich schlüssiges

Verständnis der Gesamtzusammenhänge zu erlangen. Letztlich entsteht eine Ausarbeitung im Stil eines motivierend-informativen wissenschaftsjournalistischen Artikels, der ohne Zuhilfenahme von Zusatzmaterial allein basierend auf weitverbreitetem Allgemeinwissen die wesentlichen Ergebnisse beschreibt, einordnet, auslotet und diskutiert. Eine solche sogenannte Sekundärverwertung des Erkenntnisfortschritts im molekularen Rechnen trägt wesentlich dazu bei, die Bekanntheit des Forschungsgebietes zu erhöhen und nicht zuletzt Studierende und Jungwissenschaftler zu begeistern, sich weiterführend damit zu beschäftigen und die Vernetzung mit anderen Disziplinen voranzutreiben. Der „Blick über den Tellerrand“ schärft zudem die eigene wissenschaftliche Kreativität und Innovationsfreudigkeit, denn fachübergreifendes Denken fördert frische und unkonventionelle Ideen, von denen nicht nur die Grundlagenwissenschaften profitieren. In diesem Sinne haben wir das Seminar als eine Plattform gestaltet, die dem Brainstorming Raum gibt und eine konstruktive Atmosphäre schafft, in der das gemeinsame Erschließen, Anwenden und Erweitern von Wissen Freude bereitet. Für den vorliegenden Technischen Report haben wir vier ganz verschiedene Aspekte des molekularen Rechnens ausgewählt und zu studentischen Beiträgen entwickelt.

Henry Lieske stellt einen Ansatz zum *ameisenbasierten Computing* vor. Ameisen übernehmen hierbei die Rolle des Datenflusses zur Ausführung von Berechnungen. Ein Ameisenstrom läuft dazu über eine Landschaft, die mit Wegen, Hindernissen, Quellen und Senken versehen ist. Diese Elemente bilden in ihrer Anordnung in der Landschaft Grundoperatoren nach, aus denen sich beliebige Berechnungen zusammensetzen lassen. Ameisen als molekulare Systeme gestatten den Aufbau eines besonders robusten, störungsunempfindlichen biologischen Computers, dem man bei seiner Arbeit direkt zusehen kann.

Daniel Noack hat sich mit dem Rechnen durch veränderbare Raumstrukturen inspiriert durch den Aufbau biologischer Zellen beschäftigt. Für dieses Teilgebiet des molekularen Rechnens hat sich der Begriff *Membrane Computing* etabliert. Membranen umschließen Reaktionsräume und schaffen in ihrem Inneren spezielle Bedingungen, unter denen bestimmte chemische Prozesse besonders gut ablaufen können. Zudem wirken Membranen selektiv als Filter: Moleküle mit bestimmten Eigenschaften durchdringen diese äußere Barriere und gelangen hinein, während andere Moleküle ferngehalten werden. Die Struktur von Membranräumen kann gezielt verändert werden, beispielsweise durch Zellteilung, durch die Aufnahme innerer Membranen (Endozytose) oder durch die Auflösung von Membranen. Hieraus lässt sich die Ausführung der Grundrechenarten (Addition, nichtnegative Subtraktion, Multiplikation, Division) auf beliebigen natürlichen Zahlen elegant nachbilden.

Christoph Schwalbe widmet sich in seinem Beitrag dem *chemischen Rechnen*, wo die Konzentration von Stoffen den zu verarbeitenden Daten entspricht. Durch die Ausführung chemischer Reaktionen verändern sich Stoffkonzentrationen in definierter Weise, so dass im Laufe der Reaktionszeit eine Ergebnis-Stoffkonzentration entsteht. Anhand des Beispiels Quadratwurzelberechnung wird diese Idee beleuchtet. Chemisches Rechnen findet man unter anderem in den Regelkreisen lebender

Organismen, die beim Menschen die Funktion einer inneren Uhr übernehmen oder Blutdruck bzw. Körperkerntemperatur steuern.

Elisabeth Vogel thematisiert einen anwendungsfähigen *DNA-Computer*, der das bisher größte erfolgreich laborpraktisch gelöste kombinatorische Suchproblem bewältigt hat, und zwar den Test, ob eine aussagenlogische Formel mit 20 Variablen erfüllbar ist oder nicht. Dabei müssen 2^{20}, also mehr als eine Million Lösungskandidaten durch DNA-Stränge parallel erzeugt und überprüft werden. Die speziell dafür entwickelte Labortechnik ist einfach und zuverlässig zugleich.

Allen Leserinnen und Lesern wünschen wir eine spannende Lektüre.

Thomas Hinze

[1]

Henry Lieske

Ameisenbasiertes Computing

Reflexion des Fachartikels

Loizos Michael.
Ant-Based Computing.
In M. Capcarrere et al. (eds.). ECAL 2005. Series *Lecture Notes in Artificial Intelligence* **3630**:572-583, Springer Verlag, 2005

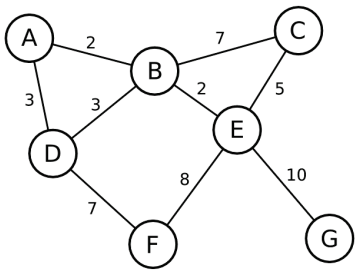
1 Einleitung

Auf einer Straßenlandkarte den kürzesten oder schnellsten Weg von einem Ort A zu einem Ort B zu finden, gehört in der Informatik zu denjenigen Aufgaben, die häufig gelöst werden müssen und für eine Vielzahl verschiedener Anwendungen anfallen. Im Laufe der Zeit wurden mehrere Verfahren zur *Wegfindung* entwickelt, die aber mitunter recht lange brauchen, um eine perfekte Lösung zu errechnen. Will man Rechenzeit sparen, erscheint es lohnenswert, nach alternativen Lösungsideen, zum Beispiel in der Natur, zu suchen. Im Folgenden wollen wir einen Ansatz dazu näher betrachten.

Unter der Wegfindung versteht man in der Informatik die Suche mit Hilfe eines Algorithmus nach dem besten, also zumeist dem kürzesten oder schnellsten Weg auf einer Landkarte mit Orten und Verbindungswegen zwischen den Orten. Der beste Weg muss nicht zwangsläufig der geradlinigste oder direkte Weg sein, denn es ist nicht immer möglich, die Luftlinie zu nehmen. Mitunter ist ein Umweg über einen oder mehrere Zwischenorte günstiger. Darüber hinaus können auch weitere Bedingungen an den Weg gestellt werden, zum Beispiel, dass ein bestimmter Ort vorher noch angefahren werden muss - wie beim Paketbringdienst, wenn noch ein Paket abzuholen ist, bevor es weitertransportiert werden kann. Es kommt auch vor, dass es von einem Ort zu einem anderen gar keine direkte Wegeverbindung gibt, sondern erst eine Route über existierende Wegeverbindungen und Zwischenorte bestimmt werden muss. Es kann auch sein, dass manche Wegeverbindungen besser ausgebaut sind als andere oder häufige Staus, Baustellen oder Engstellen ein schnelles Vorankommen ausbremsen. In solchen Fällen ist es vorteilhaft, eine alternative Route zu wählen, die vielleicht nicht die kürzeste ist, dafür aber schneller und leichter passiert werden kann, so dass letztlich wertvolle Zeit - eine wichtige Ressource - gespart wird.

Relevante Anwendungen für die Wegfindung sind, wie es gerade schon angedeutet wurde, zum Beispiel die Routenplanung für den Lieferverkehr, Neuplanung von Verkehrswegen, die Netzwerk-Flussanalyse für das Internet, um eine schnellere Datenübertragung zu erreichen oder für die Künstliche Intelligenz innerhalb von Computerspielen, so dass diese dem Spieler möglichst schnell geschickt positionierte Gegner entgegenschicken kann.

Die herausfordernde Problematik, die der Wegfindung innewohnt, ist die hohe Komplexität, also der Zeit- bzw. Speicherplatzbedarf, um zu einer perfekten Lösung zu kommen. Steht nur ein Prozessor (oder nur wenige Prozessoren) zur Verfügung, stauen sich in gewissem Sinne die nacheinander verarbeiteten Daten. Die Wegfindung ist gekennzeichnet durch aufwendige Berechnungen auf großen Datenmengen, die die Bearbeitung benötigt. Der damit einhergehende hohe Ressourcenbedarf entsteht üblicherweise dann, wenn in einem riesigen Pool möglicher Routen nach der insgesamt besten Lösung gesucht wird. Stellen wir uns vor, auf unserer Landkarte befinden sich n Orte. Von jedem dieser Orte kann es eine Wegeverbindung definierter Länge oder Passierdauer zu jedem anderen Ort geben, insgesamt bis zu $n \cdot n$ einzelne Wegeverbindungen (siehe Abbildung 1). Die Anzahl von Routen, also Ketten aus hintereinander durchlaufenen Wegeverbindungen, kann unermesslich groß werden. Sie liegt üblichwerweise in der Größenordnung $2^{n \cdot n}$. All diese Routen als Lösungskandidaten zu erzeugen und



Weg(x,y)		A	B	C	D	E	F	G
	A	0	2		3			
	B	2	0	7	3	2		
	C		7	0		5		
y	D	3	4		0		7	
	E		2	5		0	8	10
	F				7	8	0	
	G					10		0

x

Abbildung 1: Wegenetz mit Entfernungsmatrix

durchzuprobieren, kostet den Computer viel Rechenzeit und oft auch viel Speicherplatz. Zwar lassen sich häufig etliche Routen von vornherein als ungeeignet ausschließen, aber gemessen an der Gesamtzahl bleiben zumeist noch sehr viele Routen übrig, die einzeln getestet werden müssen, um die beste davon zu finden.

Die Natur steht vor der gleichen Herausforderung. Ameisenstaaten sind zum Beispiel auf gute Transportwege von Nahrung angewiesen. Offenbar scheinen sich hier erstaunlich elegante Lösungsverfahren entwickelt zu haben, die mit deutlich weniger Ressourcen effizient und schnell ein brauchbares Ergebnis liefern. Der 'Trick' der Natur besteht nun darin, nicht die allerbeste Lösung gezielt zu suchen, sondern sich auch mit einer sehr guten Lösung zufriedenzugeben, die nur wenig vom globalen Optimum entfernt ist. Ein zweiter Vorteil eines biologischen Verfahrens liegt in einer massiven Parallelverarbeitung. Statt nur einen oder wenige Prozessoren zu nutzen, setzt die Natur beim Ameisenstaat auf eine Vielzahl einfacher Verarbeitungseinheiten, nämlich zahlreiche Ameisen, die die Rolle von Prozessoren übernehmen. Dadurch hat man ein Computersystem mit sehr vielen Prozessoren, die zeitgleich und weitgehend unabhängig voneinander Routen generieren und prüfen können. Für den außenstehenden Betrachter wird so eine Lösung deutlich schneller erzielt, da sich die Arbeit auf viele Schultern verteilt.

Wie im Rechner die Prozessoren miteinander kommunizieren, um Daten auszutauschen, müssen die Ameisen auch Informationen mit ihren Artgenossen auszutauschen, um ihre Aufgabe effektiv zu bearbeiten. Da die Ameisen bekanntlicherweise nicht verbal kommunizieren können, greifen sie auf eine Technik zurück, die sich über Jahrtausende entwickelt hat, nämlich die Kommunikation mithilfe von chemischen Botenstoffen, den *Pheromonen*. Ameisen setzen diese ein, um anzuzeigen, wo eine geeignete Route zur nächsten Nahrungsquelle entlangführt. Dies geschieht, indem jede Ameise Pheromone beim Laufen ausschüttet und auf dem belaufenen Weg platziert.

Wenn nun Ameisen eine Route stärker frequentieren, entsteht dort eine höhere Pheromonenanreicherung als auf Routen, die seltener belaufen werden. Höhere Pheromonenanreicherungen locken weitere Ameisen an, die noch weiteres Pheromon hinzufügen, was einen selbstverstärkenden Effekt auslöst. Die Abbildung 2 zeigt anhand der Dicke der Wegeverbindungen, wie sich die von den Ameisen ausgeschütteten Pheromone verteilt haben. Die bevorzugte Route verläuft demnach von A nach G über B und E. Hier hat sich besonders viel

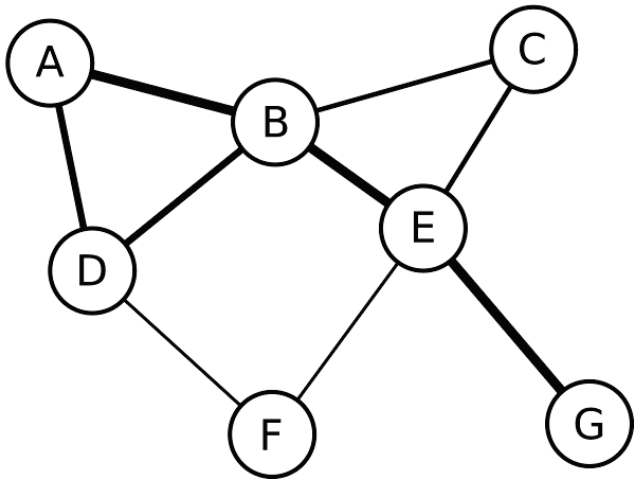


Abbildung 2: verstärktes Wegenetz

Pheromon angereichert. Einige Ameisen haben alternativ den Weg von A über D nach B und dann weiter auf der Hauptroute nach G gewählt, der zwar etwas länger ist, aber dennoch sinnvoll genutzt werden kann. Neben Pheromonen können Ameisen auch andere Botenstoffe – wie Undecan oder spezifische Hormone, ausschütten und wahrnehmen, so dass ein umfangreiches chemisches Repertoire für die Kommunikation zwischen Ameisen zur Verfügung steht. Während Pheromone überwiegend als Lockstoff fungieren, Ameisen sich also zumeist zum Pheromon hingezogen fühlen, dienen andere Botenstoffe eher dazu, Gefahr zu signalisieren, zum Beispiel, weil sich in der Nähe ein Fressfeind befindet und andere Ameisen davor gewarnt werden, sich ihm zu nähern. Auf gleiche Weise schützen Ameisen auch ihre Artgenossen vor bestimmten giftigen Pflanzen. Die chemische „Sprache“ der Ameisen kennt noch einen weiteren Ruf, und zwar die Aufforderung, einen eindringenden Feind in gemeinsamer Aktion zu vertreiben bzw. unschädlich zu machen.

Das Kommunikationssystem und das Kolonieverhalten von Ameisen liefert ein beeindruckendes Vorbild für biologisch inspirierte Problemlösungsverfahren. Aus ingenieurtechnischer Sicht bieten Ameisenalgorithmen eine Reihe von Vorteilen gegenüber konventionellen Suchverfahren und zeichnen sich durch verschiedene interessante Eigenschaften aus, die nachfolgend kurz thematisiert werden.

Niedriger Ressourcenverbrauch: Ein Ameisenstaat braucht, um zu überleben, Nahrung und ein passendes Klima. Bei der Nahrung handelt es sich je nach Ameisenart um Insekten, Ausscheidungen verschiedener Pflanzensäftesauger, Sekrete aus floralen und extrafloralen Nektarien, Samen, Pollen, Früchte und verschiedene Pflanzenteile. Da die Nahrung normalerweise frei in der Natur verfügbar ist, entstehen bei Ausführung von Ameisenalgorithmen im lebenden System kaum Energiekosten.

Robustheit: Alle bisher bekannten Ameisen sind in Ameisenstaaten organisiert. Ein solcher Staat hat den Vorteil, auf Grund seiner Größe von einigen hundert bis mehreren Million Individuen sehr robust zu sein gegenüber dem Ausfall oder dem Verlust einzelner Ameisen. Dies bedeutet, es kön-

nen einzelne Ameisen aus dem System entfernt werden, ohne dass das Funktionieren des gesamten Ameisenstaates beeinträchtigt wird.

Einfache und zuverlässige Kommunikationsfähigkeit: Der Fernaustausch von Informationen funktioniert bei den Ameisen hauptsächlich über Pheromone, also Botenstoffe, die der unbewussten Informationsübertragung innerhalb einer Spezies dienen. Diese werden bei den Ameisen eingesetzt, um einen Weg zu markieren, der aktuell am besten geeignet ist, um zu einem Nahrungsplatz zu gelangen. Die meisten Ameisen folgen der Pheromonspur, die am stärksten ist. Einige wenige Ameisen lassen sich jedoch weniger vom Pheromon locken und erkunden neue, bisher kaum gegangene Wege. Dadurch können möglicherweise noch kürzere Routen gefunden und dann durch viele Ameisen genutzt werden. Mitunter erweisen sich neue Routen aber auch als länger oder gefährlicher, so dass sie verworfen werden.

Kollektive Intelligenz: Jede Ameise als einzelnes Individuum ist recht einfach und verfügt nur über sehr begrenzte geistige Fähigkeiten. Im Grunde beschränkt sich das Verhaltensmuster im wesentlichen auf Reaktionen, die durch Botenstoffe direkt ausgelöst werden. Darüber hinaus reagieren Ameisen auf unmittelbar in der Nähe befindliche Artgenossen, indem sie deren Verhalten nachahmen. Ameisen besitzen kein inneres Langzeitgedächtnis.

Stellen Sie sich nun vor, Sie haben einen ameisenbasierten Rechner, der optimiert ist für die Wegfindung, also effizient kurze Wege finden kann und dabei wenig bis gar keinen Strom benötigt, sondern nur wenig chemische Energie. Wie könnte ein solcher Rechner konkret aussehen, das heißt, wie müsste man das Wegenetz ameisengerecht gestalten? Informatiker denken noch weiter und stellen die Frage, ob es möglich ist, mithilfe von Ameisen auch das Verhalten gewöhnlicher elektronischer Rechner zu imitieren. Wenn dies gelingt, hat man gezeigt, dass ein ameisenbasierter Rechner prinzipiell alle Aufgaben erledigen kann, die auch ein elektronischer Rechner zu lösen vermag. Dann könnte man bekannte Algorithmen recht leicht auf ameisenbasierte Rechner übertragen.

Um einen ameisenbasierten Rechner detailliert verstehen zu können, benötigen wir einige weitere Informationen zur Biologie der Ameisen und zu Eigenschaften von Pheromonen. Im nächsten Abschnitt wird darauf eingegangen. Danach stellen wir Grundelemente ameisenbasierter Rechner vor, aus denen sich Wegenetze zweckbestimmt aufbauen lassen. Mithilfe dieser Grundelemente lassen sich dann Wegenetze aufbauen, die die elementaren Rechenoperationen elektronischer Computer nachbilden. Mit diesen können dann kompliziertere Schaltungen zusammengesetzt werden, zum Beispiel ein Addierer. Auf Basis des Fachartikels [1] zeigen wir diesen faszinierenden Aspekt biologisch-inspirierten Rechnens auf.

2 Ameisen und Ameisenstaaten als biologisches Vorbild

Ameisen gehören zu den staatenbildenden Insekten. Sie können in verschiedene Arten unterteilt werden. Diese Unterteilung kann anhand der Staatengröße

bzw. Staatenbildung sowie deren Lebens- und Ernährungsweise vorgenommen werden[1].

Die Ernährung der Ameisen kann in Allesfresser, Räuber und Aasfresser, Nutzung von an Pflanzen saugenden Insekten, Samenfresser, Samensammler, Diebe sowie Pilzzüchter unterschieden werden.

Ameisen bilden sogenannte Staaten. Diese Staaten können einen Umfang von wenigen hundert bis zu mehreren Millionen Tiere umfassen. Wenn ein Ameisenstaat von nur einer Königin regiert wird, entspricht sein Alter im wesentlichen dem Alter der Königin. Dies können nach bisherigem Wissensstand bis zu etwa 29 Jahre werden. Manche Ameisenstaaten haben mehr als eine Königin. Man hat schon Staaten mit bis zu ungefähr 5000 beobachtet. Solche Ameisenstaaten haben theoretisch eine unbegrenzte Existenzdauer, aber durch Krankheiten oder Aussterbewellen wird selten ein Alter oberhalb von 80 Jahren erreicht.

Ein Ameisenstaat ist im Inneren erstaunlich gut organisiert. Man findet darin üblicherweise verschiedene Kasten, je nach Autor wird eine Einteilung in zwei bis drei Kasten vorgenommen. Zu der ersten Kaste gehören die weiblichen Geschlechtstiere, zu denen auch die Königin gezählt wird. Die zweite Kaste umfasst die immer weiblichen Hilfstiere bzw. die Arbeiterinnen, die aber keine Nachkommen hervorbringen. Darüber hinaus werden die männlichen Geschlechtstiere zumeist einer dritten Kaste zugeordnet.

Wie jedes Lebewesen, so haben auch die Ameisen Sinnesorgane, die es ihnen ermöglichen, sich in ihrer Umgebung zurecht zu finden.
Zu den Sinnesorganen, die die visuelle Wahrnehmung realisieren, zählen das Komplexauge und die drei Stirnaugen, welche jedoch von Art zu Art und von Kaste zu Kaste unterschiedlich sensitiv ausgeprägt sein können. So ist das Komplexauge und die drei Stirnaugen bei den Geschlechtstieren am besten entwickelt. Bei den Arbeiterinnen sieht das schon ganz anders aus. Bei diesen sind die drei Stirnaugen nicht vorhanden oder inaktiv und das Komplexauge nicht so stark ausgeprägt wie bei den Geschlechtstieren.

Die Antennen bilden ein weiteres markantes Sinnesorgan bei den Ameisen. Es erlaubt das Tasten, Riechen und Schmecken. Darüber hinaus können sie damit Luftströmungen und den Kohlendioxidgehalt der Luft sowie Temperaturänderungen wahrnehmen. Außerdem wird vermutet, dass auf den Antennen auch der Feuchtesinn zu verorten ist. Der zur Verfügung stehende Tastsinn in den beweglichen Antennen ist für die Ameisen eine der wichtigsten Möglichkeiten, sich mit anderen Ameisen per Berührung zu verständigen.

Eine weitere Art der Verständigung läuft als Fernkommunikation über größere räumliche Distanzen ab. Hierfür nutzen Ameisen eine chemische Kommunikation mithilfe von Pheromonen. Pheromone sind Signalstoffe, die nur artbezogen wirken und somit von Ameisen als Kommunikationsbotschaften wahrgenommen und auch nur von Ameisen als Sinnessignale gedeutet werden können. Sie werden in speziellen Pheromondrüsen produziert und bei Bedarf an die Umgebung abgegeben. Andere Ameisen nehmen diese über ihre Pheromonrezeptoren auf. Bereits eine geringe Menge dieses Signalstoffes löst bei den Ameisen einen Nervenreiz aus, so dass diese mit einem bestimmten Verhalten reagieren. Dabei entscheidet die chemische Struktur und die Anzahl der Pheromon-Moleküle dar-

[1] Quelleninformationen in diesem Abschnitt aus `https://de.wikipedia.org/wiki/Ameisen`, abgerufen am 18.11.2015

über, welche Information die Ameisen erhalten – beispielsweise, ob ein Fressfeind in der Nähe ist. Sogenannte Spurpheromone markieren die Größe eines Territoriums oder kennzeichnen die Position von Wegen.

Die von den Ameisen eingesetzten Botenstoffe, die Pheromone, sind oft leichtflüchtig und werden somit durch Diffusion schnell verbreitet, zersetzen sich aber auch schnell wieder. Das heißt, sie sind schon nach kurzer Zeit nicht mehr existent.

Loizos Michael, ein Wissenschaftler der „Division of Engineering and Applied Sciences“ der Harvard Universität hat sich überlegt, wie man sich Ameisen gemeinsam mit Pheromonen nutzbar machen und dadurch die Bewegung von Ameisen auf vorher bestimmten Bahnen steuern könnte. Auf diese Weise erscheint es möglich, elektrische Leitungsstrukturen eines Rechners per Simulation nachzubilden. Für diese Art der Simulation musste eine mathematische Beschreibung entwickelt werden, die das Verhalten von Pheromonen ausdrückt, da diese ein wesentlicher Bestandteil der Ameisen-Manipulation und somit der Simulation darstellt.

Wichtige Parameter für diese Funktion zur Bestimmung der Pheromonenkonzentration sind die Position im Raum, der aktuelle Pheromonengehalt an dieser Position sowie in deren unmittelbarer Umgebung, die Ausbreitungsgeschwindigkeit, mit der sich Pheromone verteilen bzw. auflösen sowie die Herkunftsquelle der Pheromone, also entweder eine Ameise oder eine Pumpe.

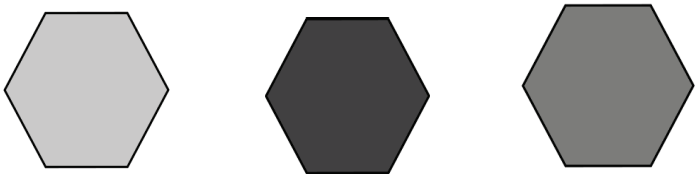
$$P^t(L) = (1-d) * [(1-f) * P^{t-1}(L) + f * W^{t-1}(L)] + s^{t-1}(L) + p(L)$$

So beschreibt die obenstehende Formel $P^t(L)$ den Pheromongehalt zum Zeitpunkt t an der Position L. Die Funktion $W^{t-1}(L)$ bildet den durchschnittlichen Pheromonengehalt zum vorherigen Zeitpunkt $(t-1)$ an der Position L und seiner Nachbarn ab. Die Konstanten d und f erfassen zum einen den Anteil der Pheromonmenge, die an die Umgebung abgegeben wird und zum anderen wie gleichmäßig das geschieht. Weitere Faktoren, die in diese Formel einbezogen werden, sind die Funktion $s^{t-1}(L)$ und $p(L)$. Diese beschreiben die zugeführten Pheromone seitens der Ameisen (s) und der gepumpten, also künstlich hinzugegebenen Pheromone (p).

Anhand dieser Funktion und unter Zuhilfenahme weiterer Algorithmen konnte eine Simulation erstellt werden, die in den folgenden Abschnitten näher beleuchtet werden soll.

3 Grundelemente eines ameisenbasierten Computers

Bestandteile für die Darstellung und funktionelle Berechnung dieser Simulation sind folgende Strukturelemente. Aus diesen setzen sich die im späteren betrachteten Bauelemente zusammen.



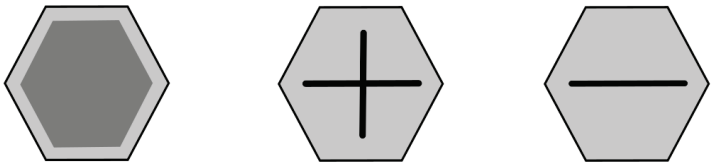
Grundelement undurchlässige Wand Wegelement

Das Grundelement ist eine sechseckig begrenzte Fläche. An ihren Außenkanten können Grundelemente zu zweidimensionalen Strukturen aneinandergefügt und angeordnet werden, so dass die modellhafte Nachbildung einer Landschaft möglich wird, auf der sich Ameisen dann bewegen können. Man unterscheidet verschiedene Typen von Grundelementen, die jeweils eine bestimmte Bedeutung haben und als strukturbildende Elemente fungieren:

Wegelement: Auf diesem können sich die Ameisen ohne Richtungsänderung fortbewegen. Pheromone verteilen sich über dieses Element uneingeschränkt.

Wand: Diese kann weder von Pheromonen noch von Ameisen durchdrungen werden.

Die nächsten drei Elemente dienen der Steuerung sowie der Verteilung der Ameisen:



Pheromonenpumpe Quelle Senke

Pumpe: Dieses Element sorgt dafür, dass künstlich Pheromone in die Simulation eingebracht werden können, so dass Ameisen gezielt dorthin, wo sich die Pumpe befindet, gelockt werden.

Quelle: Hierüber betreten Ameisen die Bewegungsebene.

Senke: Hierüber verlassen Ameisen die Bewegungsebene.

Die Funktion von Quelle und Senke kann man sich anhand eines Billardtisches veranschaulichen. Die darauf befindlichen Kugeln symbolisieren die Ameisen. Fällt eine Kugel in ein Loch, verlässt sie die Bewegungsebene. Dies entspricht der Senke. Billardkugeln, die neu auf dem Tisch platziert werden, betreten im übertragenen Sinne die Bewegungsebene. Sie imitieren somit eine Quelle.

Die folgende Grafik veranschaulicht das Funktionsprinzip von Quelle und Senke anhand des Vergleichs mit einer Batterie, wo Elektronen anstelle von Ameisen agieren. Pluspol und Minuspol wirken dann wie Quelle und Senke.

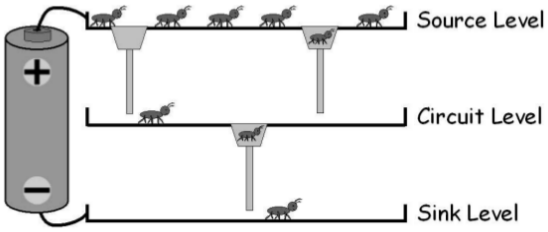
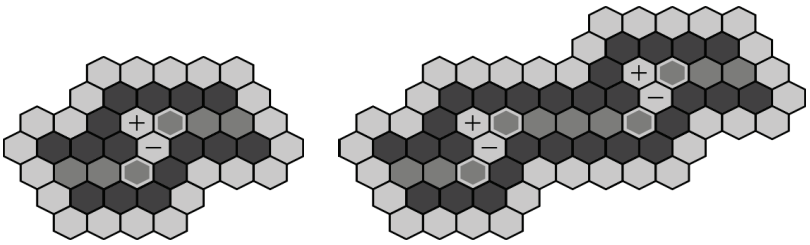


Abbildung 3: Ameisen Batterie

4 Logische Gatter im ameisenbasierten Computer

Auf Basis der Grundelemente lassen sich nun digitale logische Gatter konstruieren, mit denen man logische Operationen ausführen kann. Die Rolle der Elektronen übernehmen dabei die Ameisen, so dass man digitale Schaltungen mithilfe eines Ameisensystems nachzubauen vermag. Um das Funktionsprinzip der ameisenbasierten logischen Gatter nachvollziehen zu können, ist es sinnvoll, zunächst ein Grundverständnis für Logikgatter zu erlangen. Diese Gatter basieren auf dem Prinzip der zweielementigen booleschen Algebra, das heißt, sie arbeiten mit den Zuständen 0 und 1, wobei die 1 bei den digitalen Schaltungen für das Anliegen einer (hinreichend großen) Spannung steht und die 0 signalisiert, dass keine oder nur eine sehr geringe Spannung vorhanden ist. In Analogie dazu steht beim Arbeiten mit Ameisen der Zustand 1 für das Vorhandensein mindestens einer Ameise an einem Eingang bzw. Ausgang, während der Zustand 0 symbolisiert, dass sich dort keine Ameise befindet. Im Laufe der Zeit, also einhergehend mit der Bewegung der Ameisen durch die Gatterschaltung, können sich die einzelnen Zustände an den einzelnen Gatterein- und -ausgängen innerhalb der Schaltung verändern, was letztlich der Ausführung der entsprechenden logischen Operation entspricht. Nach Verstreichen einer gewissen Zeit sollten sich die einzelnen Zustände auf ihre endgültigen Werte fixieren, womit die Ausführung der Logikoperation abgeschlossen ist.
Der Ausgang eines logischen Gatters kann wiederum als Eingang eines nachgeschalteten logischen Gatters dienen, so dass auch kompliziertere logische Berechnungen möglich sind. Wir betrachten nachfolgend mehrere Logikgatter mit jeweils einem oder zwei Eingängen sowie jeweils einem Ausgang. Diese Grundausstattung an Gattern reicht aus, um damit beliebige Logikschaltungen aufbauen zu können. Damit entstehen vielseitige Schaltungen für verschiedene Zwecke.

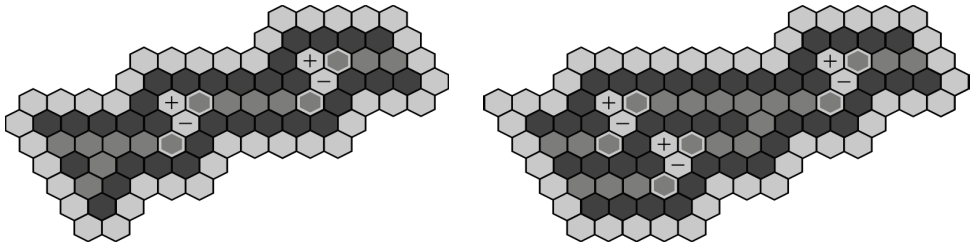
NOT Gatter: Negiert das am Eingang anliegende Signal. Das heißt, eine 0 am Eingang führt zu einer 1 am Ausgang. Eine 1 am Eingang erzeugt ausgangsseitig eine 0. Zur Realisierung dieses Verhaltens wird eine Anordnung von Wegelementen, Pheromonpumpen, einer Quelle und einer Senke wie in nachstehender Abbildung genutzt. Solange Ameisen den Gattereingang erreichen, werden sie von der nächstgelegenen Pheromonpumpe



NOT Gatter ID Gatter

innerhalb des Gatters angelockt in das Gatterinnere. Direkt neben der Pheromonpumpe ist eine Senke (−) platziert, in die die Ameisen gesteuert hineinlaufen. Gekoppelt mit der Senke ist eine Ameisenquelle (+), deren Hervorbringen von Ameisen davon abhängt, ob Ameisen in die Senke laufen oder nicht. Ist die Senke mit Ameisen belegt, bleibt die Quelle passiv. Das heißt, es gibt dann keine Ameisen, die zum Gatterausgang gelangen könnten. Folglich hat dann der Gatterausgang den Zustandswert 0. Anders ist die Situation, wenn keine Ameisen das Gatter betreten (0 am Eingang). In diesem Fall erreichen auch keine Ameisen die Senke, was die Quelle dazu veranlasst, fortwährend neue Ameisen hervorzubringen, die durch eine benachbarte Pheromonpumpe in Richtung Gatterausgang geführt werden (1 am Ausgang). Insgesamt wird damit das Verhalten eines Negators nachgebildet: Es erscheinen solange Ameisen am Ausgang wie keine Ameise am Eingang hineinkommt und umgekehrt.

ID Gatter: Dieses Element realisiert eine sogenannte identische Abbildung, das heißt, eine 0 am Eingang erzeugt auch eine 0 am Ausgang, eine 1 am Eingang führt ausgangsseitig ebenfalls zu einer 1. Dieses Verhalten wird erreicht durch Hintereinanderschaltung von zwei NOT Gattern oder anders formuliert, die doppelte Negation eines Logiksignals ergibt wieder seinen ursprünglichen Wert. Durch den Einsatz von Ameisenquellen in den beiden verknüpften NOT Gattern ist sichergestellt, dass am Ausgang des ID Gatters ein permanenter konstanter „Strom“ von Ameisen vorhanden ist und somit mögliche Schwankungen des Ameisenflusses am Gattereingang ausgeglichen werden.



OR Gatter AND Gatter

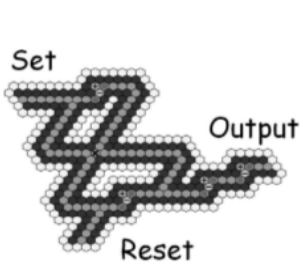
OR Gatter: Am Ausgang erscheinen Ameisen, sobald an einem der Eingänge eine Ameise sich einfindet. Die Laufwege der Ameisen von beiden Gattereingängen werden zusammengeführt und verlaufen dann als gemeinsamer

Weg durch das Gatter. Hierdurch erreicht man das Verhalten eines logischen ODERs: Es reicht, wenn an einem der beiden Eingänge Ameisen das Gatter betreten (Zustandswert 1), damit auf dem gemeinsamen Weg ein Ameisenstrom unterwegs ist. Nachgeschaltet ist dann ein ID Gatter, welches den Ameisenstrom am Ausgang stabilisiert und konstant hält. Nur dann, wenn keiner der beiden OR Gatter-Eingänge von Ameisen angelaufen wird, wird auch auf dem gemeinsamen Wegabschnitt innerhalb des Gatters kein Ameisenstrom beobachtbar sein und am Gatterausgang keine Ameisen auftreten.

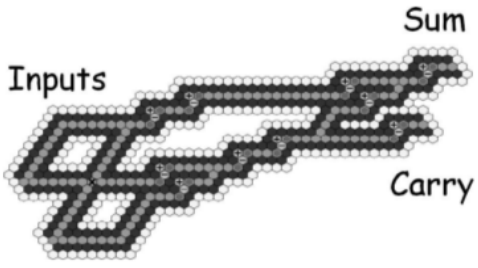
AND Gatter: Dieses Gatter realisiert eine logische UND Verknüpfung. Das heißt, nur dann, wenn an beiden Gattereingängen Ameisen einlaufen, wird auch ausgangsseitig ein Ameisenstrom erzeugt. Sind lediglich an einem der beiden Eingänge oder an keinem Eingang Ameisen vorhanden, so gibt es auch keinen Ameisenstrom am Ausgang. Die Umsetzung dieses Gatters nutzt einen kleinen Trick, denn das UND-Verhalten lässt sich nicht direkt erzeugen, sondern nur indirekt durch geschickte Kombination aus Negationen und OR Gatter. Zunächst wird jedes Eingangssignal für sich genommen negiert (NOT Gatter), danach beide Signalwege zusammengeführt, was einer ODER Verknüpfung entspricht und das daraus entstandene Signal erneut negiert. Man kann durch Ausfüllen der resultierenden Schaltbelegungstabellen (oder durch mathematische Umformungen) zeigen, dass das gewünschte Verhalten eines AND Gatters auf diese Weise entsteht.

5 Gatterschaltungen

Mithilfe der gerade vorgestellten ID-, NOT-, AND- und OR-Gatter ist es nun möglich, komplexe Schaltungen zu erzeugen, die die Grundlage heutiger Computer bilden. So können aus diesen Schaltungen zum Beispiel eine Arithmetisch-logische Einheit (ALU) – Bestandteil eines Prozessors – oder ein Random-Access Memory – der Arbeitsspeicher eines Rechners – aufgebaut werden. Mit dem Arbeitsspeicher lassen sich Werte über einen längeren Zeitraum speichern, was wiederum die Grundlage bildet für die Möglichkeit, mithilfe der ALU zwei gespeicherte Werte miteinander zu vergleichen und somit kompliziertere Berechnungen durchzuführen.



1Bit-Speicher



Halb-Addierer

1Bit-Speicher: Kann die zwei Zustände 1 und 0 am Ausgang halten bis entweder am „Set"-Eingang ein neuer Zustand mithilfe einer eintreffenden Ameise gesetzt wird oder der aktuelle Zustand über das Betreten einer Ameise des „Reset"-Eingangs gelöscht wird.

Halb-Addierer: Dient dem Addieren von zwei Zuständen (jeweils der Wert 0 oder 1), die an den beiden Eingängen in Form von einlaufenden Ameisen anliegen. Gibt das Ergebnis am „Sum"-Ausgang als Ameisenstrom aus. Wenn das Ergebnis größer als 1 wird, so wird am „Carry"-Ausgang der Zustand 1 für den Übertrag durch einen auslaufenden Ameisenstrom ausgegeben.

Setzt man nun mehrere Halb-Addierer hintereinander, so entsteht ein Mehr-Bit-Addierer, der es ermöglicht, mehr als nur zwei Zustände (Werte) zu addieren, so dass man damit die Möglichkeit hat, größere und mehrstellige Werte zu erzeugen.

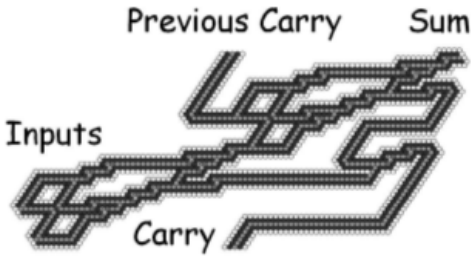


Abbildung 4: Mehr-Bit-Addierer

Mehr-Bit-Addierer: Addiert die Werte aus den „Inputs"-Eingängen mit dem „Previous"-Eingang zu einem Wert, der wiederum am nachgeschalteten „Sum"-Ausgang ausgegeben wird. Bei Vorliegen eines Übertrags wird auch am nachgeschalteten „Carry"-Ausgang der Zustand 1 ausgegeben.

6 Fazit

Wie im Fachartikel [1] aufgezeigt, wäre es theoretisch möglich, durch das Erstellen von beliebigen ameisenbetriebenen Schaltungen einen funktionstüchtigen Computer zu erzeugen, denn Logikgatter sowie 1Bit-Speicher sind die Grundbausteine, aus denen frei programmierbare Computer zusammengesetzt sind. Man benötigt zusätzlich lediglich ein Taktsignal. In der Praxis kann sich aber der Aufbau eines ameisenbasierten Computers durch Zusammenschaltung zahlreicher Grundelemente als schwierig erweisen, da nicht garantiert werden kann, dass alle Ameisen wirklich ausschließlich den Pheromonen folgen, das müsste erst in der Praxis noch überprüft werden.

Im Vergleich zu einem heutigen kompakten elektronischen Rechner wäre ein solcher Ameisencomputer recht groß, da idealerweise ein ganzer „Ameisenstaat" für den Betrieb der einzelnen benötigten Schaltungen im Rechner untergebracht werden müsste. Zudem wäre ein sehr großes Areal aus Wegeelementen notwendig, innerhalb dessen sich die einzelnen Ameisen bewegen können. Ein weiterer Aspekt, der die Nutzung eines solchen Rechners einschränken könnte, sind

die Umgebungsbedingungen, die vorherrschen müssen, damit die Ameisen ihren Dienst versehen. So ist dieser Rechner nicht überall einsetzbar, zum Beispiel nicht ohne weiteres unter Wasser. Darüber hinaus lässt sich der Rechner nicht rund um die Uhr betreiben, da auch die Ameisen einen Tag- und Nachtzyklus haben bzw. nach einer gewissen Zeit der Aktivität einfach ermüden. Folglich müsste man für ausreichend Redundanz sorgen und Ersatzameisen bereithalten. Es wäre also ein ziemlich großer Ameisenstaat vonnöten, damit der Ameisenrechner durchgehend arbeiten kann.

Neben den zahlreichen Aspekten, die gegen eine praktische Umsetzung mit lebenden Ameisen sprechen, sind die Ressourcen, die für die Herstellung eines Ameisencomputers benötigt werden, bei weitem nicht so kostenintensiv, wie es bei einem handelsüblichen elektronischen Rechner der Fall wäre. Darüber hinaus ist der Energiebedarf, der für den Betrieb gebraucht würde, vermutlich auch sehr gering, da nur Nahrung und Pheromone bereitgestellt werden müssten. Ein weiterer und zugleich schöner Aspekt, der für die Umsetzung spricht, ist der Anblick eines solchen Rechners. Zuzusehen, wie sich die Ameisen durch die Gänge bewegen, erscheint einfach faszinierend und ermöglicht es, den Computer quasi im Detail beim Rechnen zu beobachten.

Literatur

[1] L. Michael. Ant-Based Computing. In M. Capcarrere et al. (Eds.). ECAL 2005. Series *Lecture Notes in Artificial Intelligence* **3630**:572-583, Springer Verlag, 2005

[2]

Daniel Noack

Membrane Computing

Reflexion der Fachartikel

Vincenzo Manca, Rosario Lombardo.
Computing with multi-membranes.
In M. Gheorghe, G. Păun, G. Rozenberg, A. Salomaa, S. Verlan (eds.). Membrane Computing. Series *Lecture Notes in Computer Science 7184*:282-299, Springer Verlag, 2012

Luca Marchetti, Vincenzo Manca.
A methodology based on mp theory for gene expression analysis.
In M. Gheorghe, G. Păun, G. Rozenberg, A. Salomaa, S. Verlan (eds.). Membrane Computing. Series *Lecture Notes in Computer Science* **7184**:300-313, Springer Verlag, 2012

1 Introduction to P Systems

"Every living organism is a powerful computer!" This statement seems to be a bit contradicting, since computers are normally build of electrically, non-living elements. But on second thought the meaning is clear. Information processing is a very important feature in life. Living creatures are much better in processing complex information than conventional computers. We are able to filter important information from our environment, just by looking on things. There is no current computer, which is capable of such fast and extensive information processing. In living cells DNA (deoxyribonucleic acid), RNA (ribonucleic acid), and proteins are used to store information. These are known to have a much higher storage density than current caches or the RAM (random access memory) in our computer systems. Moreover they have a higher degree of compactness and are regenerative. This superiority of the living computers motivated the research area of *Molecular Computing*. It is about learning how to solve problems and new algorithmic ideas from nature. Therefore, it can be seen as the connector between computer science and the life sciences, containing many subcategories like:

DNA-Computing Computing by combination and recombination.

Neural Computing Computing by learning, comparing, and evaluating.

Evolutionary Computing A universal heuristics.

Amorphous Computing Computing by self-organizing swarm behavior.

Membrane Computing Microflow computer with distributed and selectively permeable reaction spaces.

There are some more subcategories, but this work will focus on *Membrane Computing*, as it was introduced by Gheorghe Păun [1]. Therefore, this chapter gives a brief introduction to basic P Systems and concludes with an example system. These P Systems can be modeled in many different ways, depending on the users needs. Chapter 2 explains how to compute with *Multi-Membranes*, an advanced P System model, as it was proposed by Vincenzo Manca and Rosario Lombardo [2]. Here, general computing examples like limited subtraction,

multiplication, etc., will be given. The chapter is concluded by the proof of computational universality, meaning that multi-membrane systems have the same computational power of register machines. Chapter 3 shows a use case for membrane computing. As proposed by Luca Marchetti and Vincenzo Manca, metabolic P Systems can be used for gene expression analysis [3]. In this chapter gene expression in general is explained and, furthermore, how metabolic P Systems can be applied to improve this process.

1.1 Membrane Structure

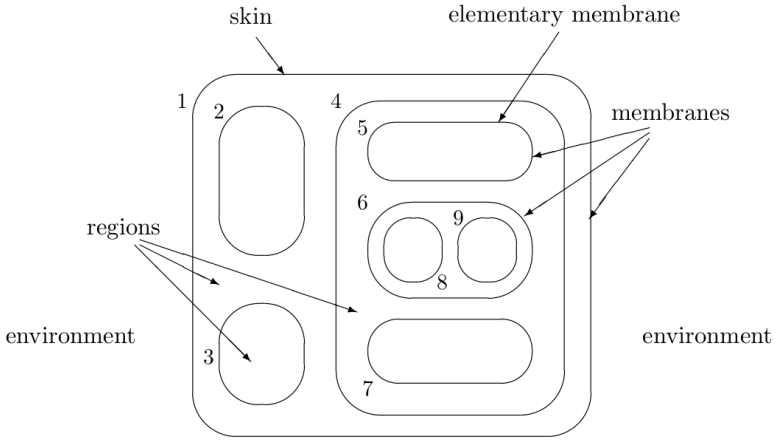


Figure 1.1: The basic membrane structure (taken from [1]).

Figure 1.1 shows the basic structure of a membrane system. The inner part of a membrane is called its region. A region can contain further membranes. If the region does not contain further membranes, the corresponding membrane is called an elementary membrane. The outermost membrane, which is not included by any other membrane is called the skin membrane. The skin membranes outer region is called environment. Every membrane is labeled with a unique identifier (standard case is, to use integers beginning at the skin membrane with 1).

This hierarchical membrane structure builds the basis for P Systems. It can be seen like a cell, which itself is also hierarchically structured. The graphical representation above is good readable for human beings, but not for a computer. It is easy to see that also a tree based representation is possible. Thus, known algorithms of graph theory can be applied. Another

possible representation is the text based representation. For the above membrane structure, a textual representation could be:

$$[_1[_2]_2[_3]_3[_4[_5]_5[_6[_8]_8[_9]_9]_6[_7]_7]_4]_1$$

The textual representation is not unique, meaning that several text strings can represent the same membrane structure.

1.2 Evolution Rules

The membrane structure is not enough to do computations. To compute something, also some kind of objects inside the membranes and some rules, to transform these objects in some way, are needed. In the basic P System model every region contains a multiset of symbol-objects. These objects can be seen as different chemicals in a cell compartment. The objects can evolve by so called *evolution rules*. Every membrane has its own set of *evolution rules*, which can contain rules of three types. The rule types are: *multiset rewriting rules*, *communication rules*, *membrane handling rules*.

Multiset rewriting rules are used to transform multisets of objects, just like in a chemical reaction. Thus, these rules are of the form $u \rightarrow v$ (u and v are multisets of objects). Furthermore the rules can be extended with target indications on the right side of the rule. These extended rules are called *communication rules*. Target indicators are *here*, *in*, and *out*. The *here* indicator is optional and means that the object remains in the current region. Using the *out* indicator results in releasing the object to the outer region. With *in*, one of the inner membranes is non-deterministically chosen and the object moves into this region. An example could be the rule $aab \rightarrow (a, here)(b, out)(c, here)(c, in)$. Here, two objects of type a and one of type b are consumed to produce the objects a, b and two times c. One a and one c remain in the membrane. The b leaves the membrane to the outer region and the other c choses non-deterministically one of the inner membranes to move to. Rules of the form $u \rightarrow v\delta$ are called *membrane handling rules*. Such a rule dissolves the membrane with its rules and releases the contained symbol-objects to the outer region. The skin membrane cannot be dissolved and objects which reached the environment cannot be retrieved from there.

A rule is only applicable if all of the symbol-objects of the left side of the rule are present. The rules are used in a maximally parallel and non-deterministic manner. Therefore, as much as possible rules are applied at one time such that no more rule can be applied.

1.3 Formal Definition of a Transition P System

A *Transition P System* (of degree m) is a construct of the form:

$$\Pi = (O, C, \mu, w_1, w_2, ..., w_m, R_1, R_2, ..., R_m, i_o),$$

where:

1. O: Alphabet of objects (finite and non-empty)
2. $C \subset O$: Set of catalysts
3. μ: Membrane structure (with m membranes)
4. $w_1, w_2, ..., w_m$: Multiset of objects (strings) in regions 1,2,...,m
5. $R_1, R_2, ..., R_m$: Sets of evolution rules for regions 1,2,...,m
6. i_o: Label of output region (0 for environment)

The rules can be of the same form as described in section 1.2. Catalysts are only aid to evolve other objects. For example rules of the form $cu \to cv$ or $cu \to cv\delta$ use c as catalyst for evolving u to v. A *Transition P System* only stops when no rule is applicable anymore. After stopping, the content of the output region is taken as the result of the computation. For a better understanding of how such a *Transition P System* works, an example is given in the following figure.

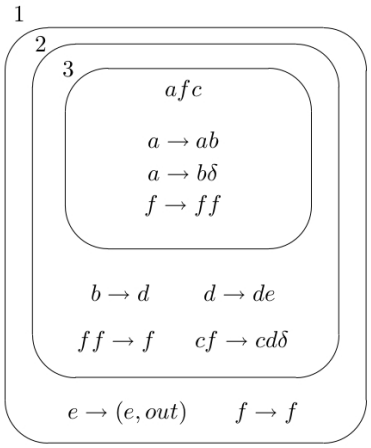


Figure 1.2: Example Transition P System with its initial configuration (taken from [1]).

Figure 1.2 shows a *Transition P System* consisting of three, hierarchically, nested membranes. The innermost membrane, the only elementary membrane, contains the symbol-objects a, f, and c. It is not obvious what the system does. To get a better understanding of this P System Fig. 1.3 depicts the beginning of the evolution tree, of the example system.

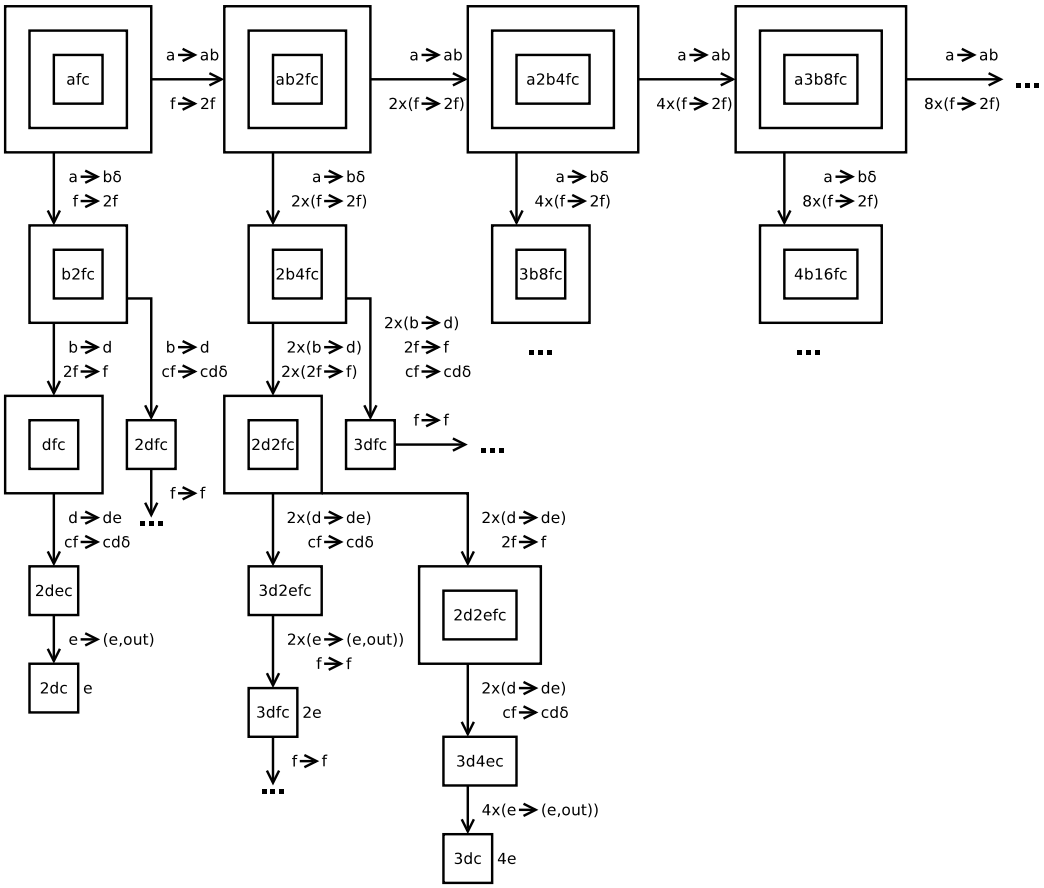


Figure 1.3: Evolution tree of the example Transition P System.

As first step, objects of type f and b can be produced till the elementary membrane is dissolved. In every step the number of f is doubled and one b is added. After dissolving the elementary membrane, there is only one branch of the tree, which leads to a halting state. Other branches are infinitely, meaning the system does not stop by returning a result. This behavior is caused by the rule $f \to f$, which is applicable if not all objects of type f were dissolved, before dissolving the inner membrane. By looking on the valid results, it is easy to see that the system computes the square numbers (in dependence of the number of b).

This is only a basic definition of a P System. Such systems can be modified in dependency of the users needs. For example one could say that infinite branches are unwanted, because a system never stopping is useless. In that case rule prioritization could be introduced to eliminate infinite branches. In the shown example it is enough to say that rule $cf \rightarrow cd\delta$ is only applicable if rule $ff \rightarrow f$ is not applicable. This means, $ff \rightarrow f$ has a higher priority. Another problem can be the non-determinism. It is not really useful for real computation. Therefore, *Multi-Membranes* are presented in chapter 2, as a deterministic membrane system variant.

2 Multi-Membranes

Multi-Membranes, as proposed in [2], are a computation model inspired by Metabolic P Systems [4]. Advantages of this model are that it is deterministic, distributed, and computationally universal (proof in section 2.3). Here, we have objects of only one type, transfered by fluxes specified by the membrane contents. With this model it is easy to give a natural description of arithmetical functions. Textual information is not needed, meaning that the model can be described by pure graphs.

2.1 Introduction

The basic P System model only used *inclusion* of membranes to build a membrane structure. For the definition of multi-membranes a *junction* relation is needed, too. Two membranes are joined if they share parts of their borders. Thus, an *elementary multi-membrane* is the junction of elementary membranes. Furthermore, a *general multi-membrane* is obtained by iterative inclusion and junction on an elementary multi-membrane. To define computation start and stop, multi-membranes with *central structure* are of importance. An elementary multi-membrane has a central structure, if a clearly defined central membrane (it is clear which one is the central membrane) to which are joined two or more membranes. The joined membranes are called *frontier membranes*. Therefore, a general multi-membrane with central structure is a multi-membrane with central structure to which membranes or multi-membranes with central structure have been hierarchically included into the central membranes. To get a better understanding of these definitions Fig. 2.1 depicts general multi-membranes with central structure, whereby Fig. 2.2 shows only general multi-membranes (without central structure).

Another important aspect are *channels*. A channel connects two membranes, connected to the same region. One is the source membrane and the other the target membrane. Figure 2.3 depicts a multi-membrane with channels, for clarification. The channels are labeled r_1 to r_5.

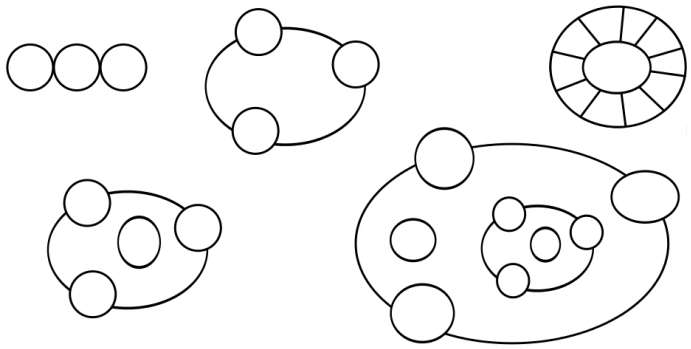


Figure 2.1: Multi-membranes with central structure (taken from [2]).

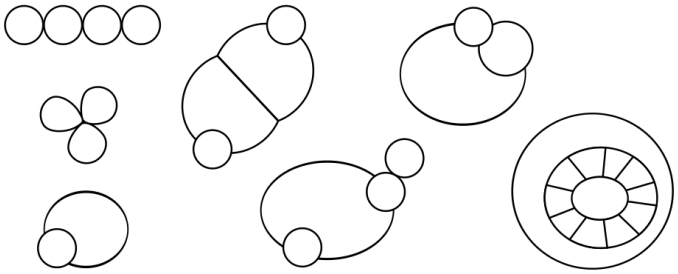


Figure 2.2: Multi-membranes without central structure (taken from [2]).

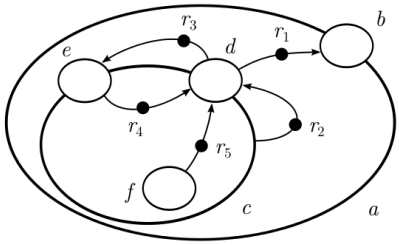


Figure 2.3: Multi-membranes with channels (taken from [2]).

In multi-membranes the rules are channels, which are denoted as follows:

$$a \rightarrow b \ \# \ \phi$$

Here, a and b are membrane labels, and ϕ is the *flux*. ϕ can be a natural number or a membrane label. In case of a natural number, ϕ denotes how many objects can move from

membrane a to membrane b in one time step. Otherwise, the membrane denoted by ϕ defines the number of objects to be moved. This number is determined by the number of objects contained in membrane ϕ. Such rules are only triggered if the source membrane contains at least as many objects, denoted by the flux. There are also rules of the form $\emptyset \to b \# \phi$ or $a \to \emptyset \# \phi$. They are used for production of elements or for deletion. With these prerequisites it is possible to define a *Multi-Membrane Computing System*.

Multi-Membrane Computation System A Multi-Membrane System with central structure is a construct of the form:

$$M = (L, R, \mu)$$

L: Set of membrane labels
R: Set of multi-membrane rules ($a, b \in L$ and $\phi \in \mathbb{N}$ or $\phi \in L$)

$$a \to b \# \phi \text{ (transition)}$$
$$\emptyset \to b \# \phi \text{ (extra-in)}$$
$$a \to \emptyset \# \phi \text{ (extra-out)}$$

μ: Initial multi-membrane configuration
Frontier membranes can be marked by: START, HALT, IN and OUT

There is only one START membrane and only one HALT membrane. These two membranes are used to start and stop the computation process. Before putting an object into the start membrane, which starts the computation, the input is done via the membranes marked by IN (objects are put in these membranes). If there is an object in the HALT membrane, the computation stops and the objects contained in membranes labeled OUT are taken as result. Furthermore, it is important that all rules taking objects from the same membrane are triggered together. If not all fluxes are defined or the source membrane does not have enough objects, to fire all rules together, none of the rules is fired. A flux denoted by a frontier membrane is only defined if the corresponding multi-membrane has an object in its HALT membrane. An undefined flux forbids the rule to fire. Moreover, if a multi-membrane system never gets an object in its HALT membrane (it never stops), the result for the given input is undefined (like a function which is undefined for several inputs).

To get a better understanding of how such multi-membrane systems work, some examples for arithmetical functions are given in section 2.2.

2.2 Examples of Multi-Membrane Systems

First of all it is necessary to understand the graphical symbols. Therefore, a table with descriptions is given in Fig. 2.4.

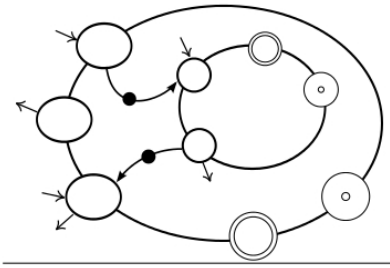
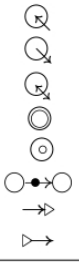
Multi-membrane system	Graphical symbol	Description
		In Out In-Out Start Halt Rule $a \to b \ \#\varphi$ Expulsion rule $a \to \emptyset \ \#\varphi$ Introduction rule $\emptyset \to b \ \#\varphi$

Figure 2.4: Graphical elements of multi-membranes and their description (taken from [2]).

Multi-membrane systems built of these symbols combined with the rules, lead to annotated graphs as shown in Fig. 2.5. Objects are represented as dots inside the membranes. The

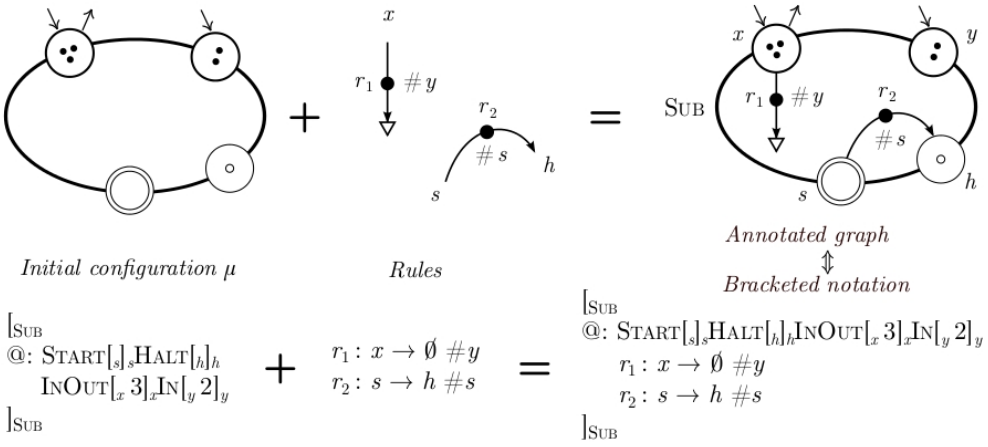


Figure 2.5: Annotated graph and bracketed notation for limited subtraction, with given input (taken from [2]).

example demonstrates processing of input data, what is unnecessary for just denoting the computation system. But for the first example of the limited subtraction, we will rely on this example input of 3 and 2. For larger numbers it makes sense to write the number instead

of dots inside the membranes. Figure 2.5 also discloses the bracketed notation as textual equivalent. In the textual notation membranes with its contents and the rules are denoted as given in the figure. Frontier membranes are marked by the @ symbol at the beginning of the line.

As mentioned before, Fig. 2.5 depicts the limited subtraction with values 3 and 2. The limited subtraction is defined like the normal subtraction if the subtrahend can be subtracted from the subtracter (without getting less then zero). Otherwise, the subtracter is taken as result. In the example, x and y are the input membranes, whereby x is also an output membrane. When taking a closer look, it is easy to see that x is the subtracter and y the subtrahend. At the first time step (i.e., an object is put to s), r_1 tries to delete as many objects as located in y, from x. At the same time step also r_2 fires and halts the system (i.e. one object reaches h), which defines the remaining objects in x as result. The given system computes the limited subtraction in one time step.

Another example is the multiplication. Multiplying two numbers a and b can be seen as repeated addition, whereby the b is added a times. This behavior is implemented within the multi-membrane depicted in Fig. 2.6.

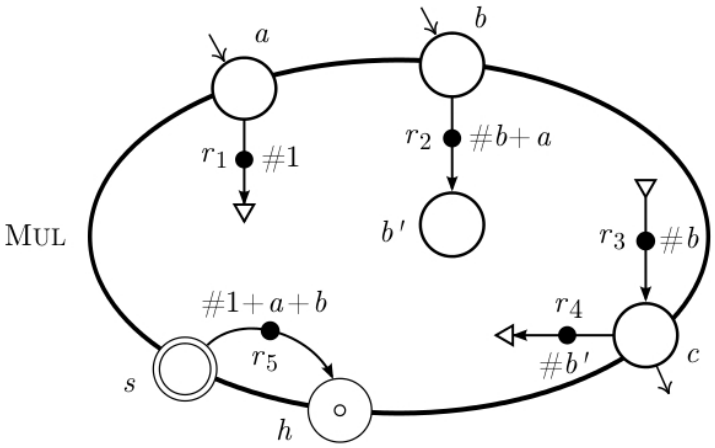


Figure 2.6: Multi-membrane for multiplication of two numbers a and b (taken from [2]).

Here, we have five rules. r_5 is for halting the system to finish the computation. This rule only fires, when a and b are empty. r_3 is for the addition of b to the output and r_1 decreases a. r_2 clears b to stop the addition, when a gets cleared (i.e., r_2 only fires if a is empty). The problem is that at the same time once more b is added to the output (i.e., $(a+1) * b$ objects

are in the output membrane). At the last step (r_5 is triggered), b objects are removed from the output membrane by r_4. This is done by using the objects in b', which has been saved by r_2. The valid output now is $a * b$.

As mentioned before, it is possible to annotate a *pure graph* version of a multi-membrane. This annotation for the multiplication can be seen in Fig. 2.7, as an example. The used membranes for the fluxes are marked by dotted arrow-lines, leading to the channel (rule). Constants in fluxes are modelled via isolated membranes containing the corresponding amount of objects.

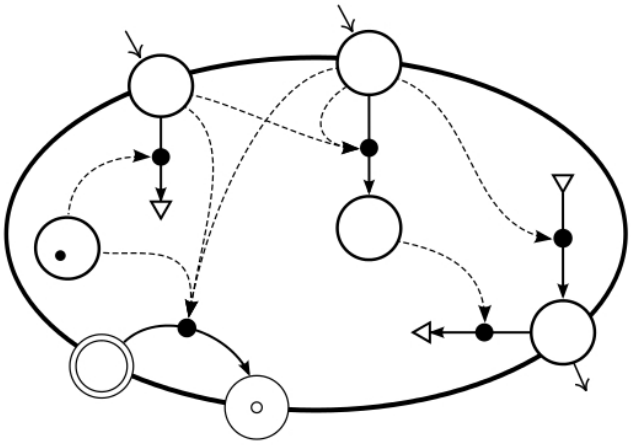


Figure 2.7: *Pure graph* annotation for a multi-membrane modeling the multiplication of two numbers a and b (taken from [2]).

To complete the examples, a 3-digit abacus with carry is given in Fig. 2.8. This rather complex example again has already given input numbers $N = 728$ and $M = 322$ (the abacus is in base 10).

First step is to sum up the single digits into the corresponding output membranes (i.e., $R_1 = N_1 + M_1$, the others analogous). R is the output membrane for the potentially occurring digit of thousands. The system runs in four steps, what can be seen, when having a look on the START and HALT membrane as well as on k and t. The switching to HALT is dependent on the contents of k. Thus, it is only possible if k contains only one object, what happens after three steps making the fourth step the last one. The second, third, and fourth step are necessary to push the occurring carry through all digits if needed. The carries are taken from an output membrane and pushed towards the next one, only if the source membrane contains more than nine objects. This behavior is ensured by the rule that several

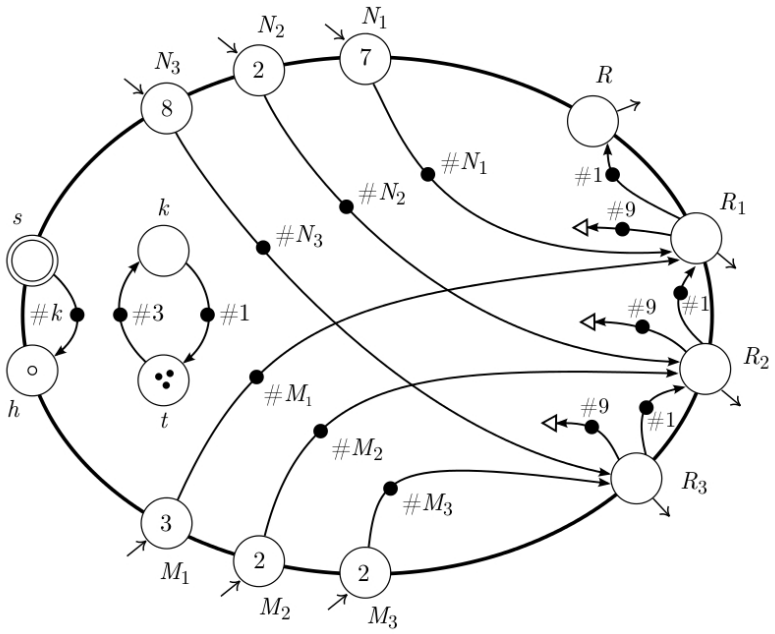


Figure 2.8: 3-digit abacus with carry in base 10 (taken from [2]).

rules taking objects from the same source membrane must fire together or none of them can fire. Thus, the rule taking nine objects from the output membrane can only fire if the carry rule also can fire, and vice versa.

2.3 Computational Universality

Multi-membrane systems are a universal computation model, meaning they have the same computational power of register machines. To proof this, first of all we need a definition of the used register machine model. Here, a variant of the Minsky's model [5] with the following set of instructions is used:

INC(R) Increment of register R

DEC(R) Decrement of register R

JNZ(R, I) Go-to instruction I if the content of register R is not zero

HALT Halt the computation

The register machine, as its name says, comprises a set of registers $R_1, ..., R_n$ and a sequence of instructions $I_1, ..., I_m$, of the above described types. A computation is done by putting some natural numbers in the registers and beginning to process the instructions sequentially (except for the jump instructions). By executing a HALT instruction, the machine stops and the content of the registers is taken as result.

The increment and decrement instructions can be modeled easily as depicted in Fig. 2.9. Here,

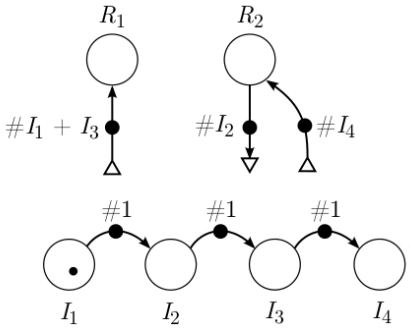


Figure 2.9: Increment and decrement in a multi-membrane system (taken from [2]).

we have instructions $I_1, ..., I_4$, which are executed sequentially. Register R_1 is incremented by I_1 and I_3 and register R_2 is incremented by I_4 and decremented by I_2. Furthermore, a conditional construct is necessary to model the go-to instruction. The output of the

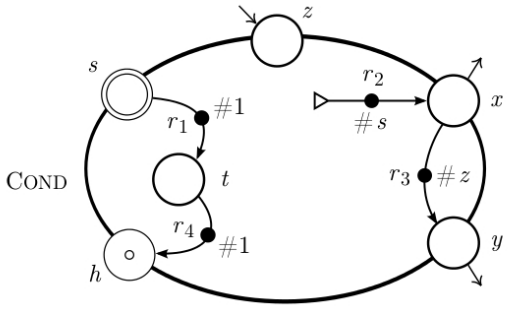


Figure 2.10: Conditional construct for jumps (taken from [2]).

conditional construct, shown in Fig. 2.10, is dependent on the input membrane z. The output tuple (x, y) will be $(1, 0)$ if $z = 0$, otherwise $(Z = 1)$ it is $(0, 1)$. Thus, making this conditional construct a good helper for the go-to statement.

Figure 2.11 depicts the general multi-membrane system modeling a register machine. Like

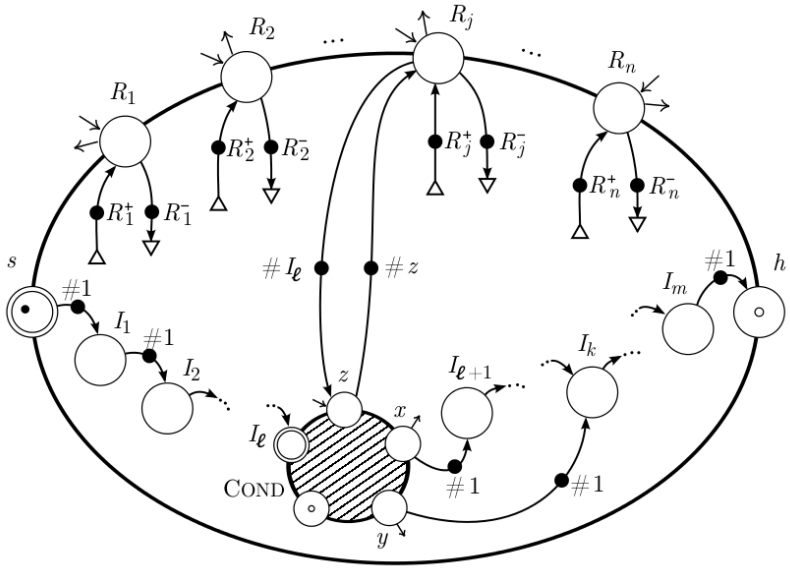


Figure 2.11: Multi-membrane System modeling a register machine (taken from [2]).

in Fig. 2.9, increment and decrement instructions are done in one step. The conditional multi-membrane system is used to implement the go-to statement. This multi-membrane sub-system is activated when I_l gets one object. First step is, to extract one object from R_j to z (i.e., R_j is non-zero). In dependence of z, one object is added either to x or to y. The output membranes x and y get valid, when the conditional multi-membrane system got an object in its HALT membrane. The jump is performed if $y = 1$, meaning $R_j \neq 0$. These operations are sufficient to model the above described register machine model. □

3 Use Case for Gene Expression Analysis

In this chapter a methodology using metabolic P Systems for gene expression analysis will be described. This methodology has been developed by Luca Marchetti and Vincenzo Manca [3].

All genetic information of an individual is saved in DNA. Transcriber proteins can read parts of the DNA and build RNA construction plans. These RNA construction plans are used to build proteins, which are the fundamental components in our body. The RNA can be pulled out to analyze processes inside an organism. This analysis process, called gene expression analysis, is shown in Fig. 3.1. Gene expression analysis is used to analyze the behavior of

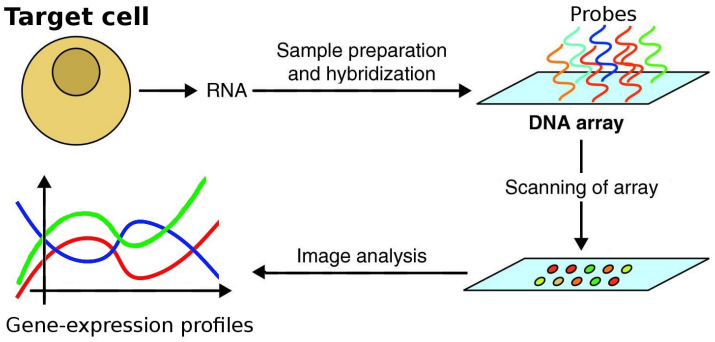


Figure 3.1: Gene expression analysis (taken from [3]).

organisms on certain stimuli (e.g., light), meaning the interaction of the genes. Therefore, a DNA microarray of probes arranged in groups is given. The probes represent the genes under study. Now, a certain stimulus has to be applied (e.g., turn light on). After that, RNA probes are extracted from the target cell and pulled over the microarray. The probes will attach to the corresponding group on the microarray. The observation of the probes over time leads to so called gene expression profiles. The dependencies of the genes can be read from the curve shapes and directly mapped to a metabolic P System. Metabolic P Systems are an advantageous way to model gene networks, used to describe dependencies between genes. These MP Systems are a better choice than differential equation system. MP Systems

are defined on the level of whole molecules and the number of molecules of same type is very limited. Furthermore, MP Systems, as already mentioned, can be directly derived from gene expression profiles. A corresponding gene network could look like in Fig. 3.2.

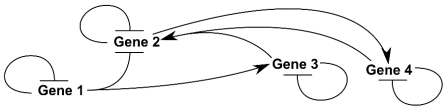


Figure 3.2: Example gene network (taken from [3]).

The table depicted in Fig. 3.3 shows the basic MP reactions and its corresponding representation in gene networks. By using these translations a MP grammar for the above given gene

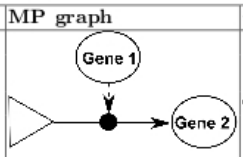
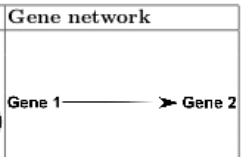
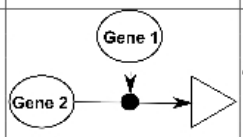
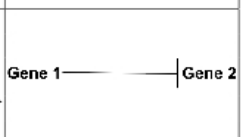
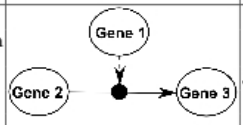
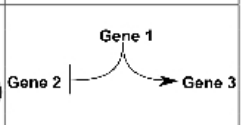
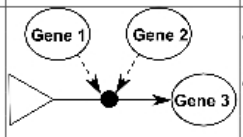
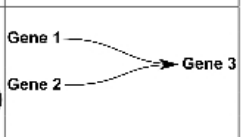
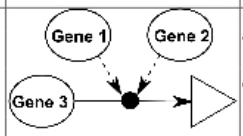
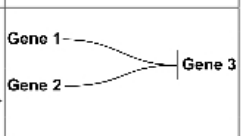
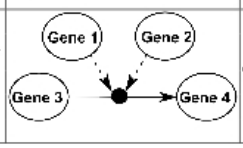
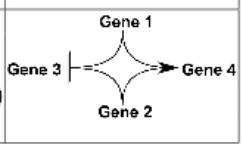
MP grammar	MP graph	Gene network
Simple promotion $r : \emptyset \to Gene\ 2$ $\varphi : k_1 \cdot Gene\ 1$		
Simple inhibition $r : Gene\ 2 \to \emptyset$ $\varphi : k_1 \cdot Gene\ 1$		
Simple promotion/inhibition $r : Gene\ 2 \to Gene\ 3$ $\varphi : k_1 \cdot Gene\ 1$		
Combined promotion $r : \emptyset \to Gene\ 3$ $\varphi : k_1 \cdot Gene\ 1 + k_2 \cdot Gene\ 2$		
Combined inhibition $r : Gene\ 3 \to \emptyset$ $\varphi : k_1 \cdot Gene\ 1 + k_2 \cdot Gene\ 2$		
Comb. promotion/inhibition $r : Gene\ 3 \to Gene\ 4$ $\varphi : k_1 \cdot Gene\ 1 + k_2 \cdot Gene\ 2$		

Figure 3.3: Table with MP reactions and corresponding gene network elements (taken from [3]).

network could look like in Fig. 3.4.

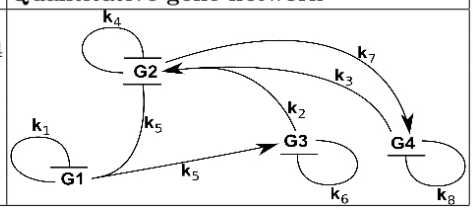
MP grammar		Quantitative gene network
$r_1 : G1 \to \emptyset$	$\varphi_1 = k_1 \cdot G1$	
$r_2 : \emptyset \to G2$	$\varphi_2 = k_2 \cdot G3 + k_3 \cdot G4$	
$r_3 : G2 \to \emptyset$	$\varphi_3 = k_4 \cdot G2$	
$r_4 : G2 \to G3$	$\varphi_4 = k_5 \cdot G1$	
$r_5 : G3 \to \emptyset$	$\varphi_5 = k_6 \cdot G3$	
$r_6 : \emptyset \to G4$	$\varphi_6 = k_7 \cdot G2$	
$r_7 : G4 \to \emptyset$	$\varphi_7 = k_8 \cdot G4$	

Figure 3.4: grammar of example gene network (taken from [3]).

The problem is that the measured data, stored in the gene expression profiles, is inaccurate (i.e., there are outliers because of temperature dependency, etc.). Thus, the measured data has to be normalised. The authors suggested solution consists of three steps. First of all, the data has to be compressed (e.g., by a log_2 scale). After that, the quantiles need to be removed (i.e., remove the outliers). The last step is to use regression to find the corresponding function. Some examples of inaccurate measurements and its normalized functions can be seen in Fig. 3.5. Based on the normalized data, the corresponding MP grammar can be obtained.

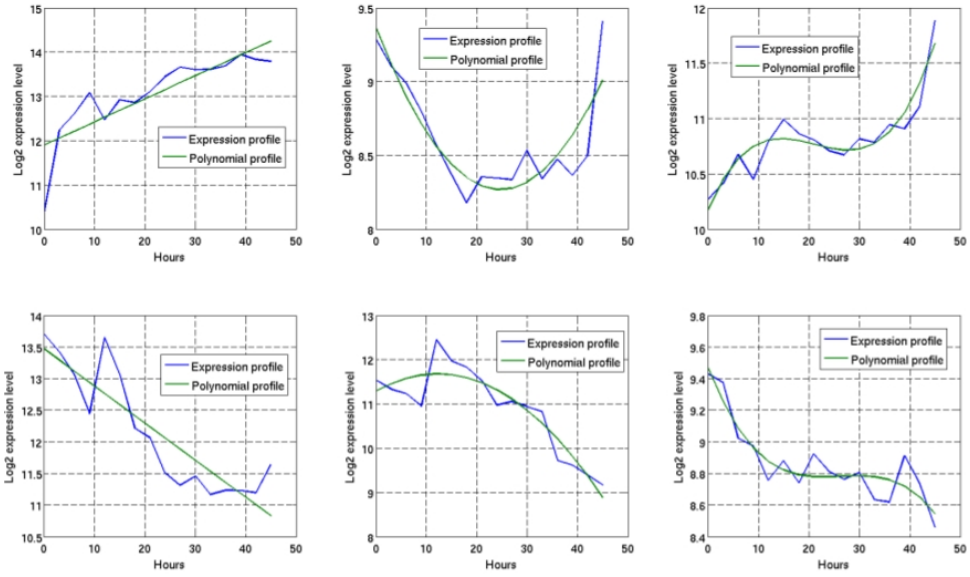


Figure 3.5: grammar of example gene network (taken from [3]).

Bibliography

[1] Gheorghe Păun. Introduction to membrane computing. In Gabriel Ciobanu, Gheorghe Păun, and MarioJ. Pérez-Jiménez, editors, *Applications of Membrane Computing*, Natural Computing Series, pages 1–42. Springer Berlin Heidelberg, 2006.

[2] Vincenzo Manca and Rosario Lombardo. Computing with multi-membranes. In Marian Gheorghe, Gheorghe Păun, Grzegorz Rozenberg, Arto Salomaa, and Sergey Verlan, editors, *Membrane Computing*, volume 7184 of *Lecture Notes in Computer Science*, pages 282–299. Springer Berlin Heidelberg, 2012.

[3] Luca Marchetti and Vincenzo Manca. A methodology based on mp theory for gene expression analysis. In Marian Gheorghe, Gheorghe Păun, Grzegorz Rozenberg, Arto Salomaa, and Sergey Verlan, editors, *Membrane Computing*, volume 7184 of *Lecture Notes in Computer Science*, pages 300–313. Springer Berlin Heidelberg, 2012.

[4] Vincenzo Manca. Fundamentals of metabolic P systems. *Handbook of membrane computing*, 19, 2009.

[5] Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[3]

Christoph Schwalbe

Chemisches Rechnen: Künstliche Evolution masseerhaltender chemischer Reaktionsnetzwerke zur Berechnung der Quadratwurzelfunktion

Reflexion der Fachartikel

Anthony M.L. Liekens, Huub M.M. Ten Eikelder,
Marvin N. Steijaert, Peter A.J. Hilbers.
Simulated Evolution of Mass Conserving Reaction Networks.
In S. Bullock, J. Noble, R.A. Watson, M.A. Bedau (eds.).
Proceedings ALIFE 2008. MIT Press, 2010

Anastasia Deckard, Herbert M. Sauro.
Preliminary Studies on the
In Silico Evolution of Biochemical Networks.
ChemBioChem **5(10)**:1423-31, 2004

Einleitung

In dem Paper „Simulated Evolution of Mass Conserving Reaction Networks“ von den Autoren Anthony Liekens, Huub Ten Eikelder, Marvin N. Steijaert und Peter Hilbers wird beschrieben, wie künstliche chemische Reaktionsnetzwerke gewonnen werden können, die einerseits in der Lage sind, mathematische Berechnungen auszuführen und andererseits das Naturgesetz der Masseerhaltung erfüllen. Bisherige Ansätze der künstlichen Evolution haben zwar chemische Reaktionsnetzwerke hervorgebracht, die sich für Berechnungsaufgaben eignen, aber sie verletzen die Forderung der Masseerhaltung, da in den entsprechenden Modellen manche Stoffe „ins Nichts“ zerfallen oder zusätzliche Reaktionsprodukte erzeugt werden, deren Bestandteile nicht durch Ausgangsstoffe bereitgestellt werden. Insofern wird durch die Erkenntnisse des Papers ein deutlicher Beitrag auf dem Weg zu einer praktischen Umsetzung chemischer Algorithmen geleistet. Die wichtigsten Ideen daraus sollen nachfolgend vorgestellt werden.
Zu Beginn des Papers wird eine kurze Einleitung gegeben, was künstliche chemische Algorithmen sind und für welchen Zweck sie eingesetzt werden können. Dabei werden diese Algorithmen mit chemischen Reaktionsnetzwerken dargestellt und ihr Verhalten mittels Differentialgleichungen beschrieben.

Künstliche Chemische Algorithmen

In biologischen Organismen gibt es Netzwerke von chemischen Reaktionen. Diese kontrollieren und regeln das Verarbeiten von Informationen in einer Zelle. Ein Ziel der künstlichen Chemie ist es, diese Netzwerke zu studieren und zu analysieren, um sie zielgerichtet für bestimmte Aufgaben erzeugen zu können. Solche Netzwerke sind dann in der Lage, Berechnungen und Vergleiche durchzuführen. Bei der künstlichen Chemie sollen die Algorithmen gezielt Aufgaben der Informationsverarbeitung lösen und Berechnungen durchführen können. In dem zugrunde liegenden Paper [2] wird beschrieben, wie künstlich erzeugte Netzwerke einfache mathematische Berechnungen durch chemische Reaktionen absolvieren können. Ein Beispiel dieser Netzwerke veranschaulicht folgende Grafik:

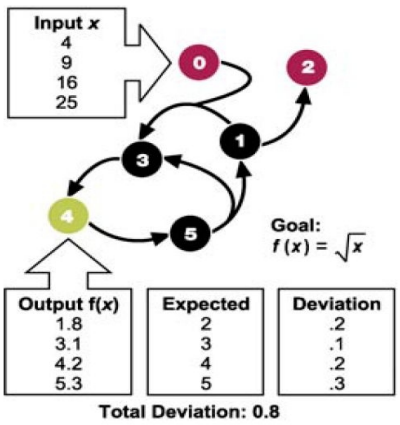


Figure 5. *Calculating deviation for a network. The network is simulated for several different inputs, and the readouts measured. All the deviation scores are summed for each of the nodes. Each node is tested, and the node with the lowest total deviation score is the output for the network.*

Mit Hilfe dieses Netzwerks soll die Quadratwurzel für x bestimmt werden. Dafür wird eine bestimmte Anfangskonzentration des Stoffes *Input x* als Eingabe verwendet. Durch Reaktionen innerhalb des Netzwerkes entsteht als Ausgabewert eine entsprechende Stoffkonzentration *Output f(x)*. Dieser Wert wird unmittelbar als Ergebnis der Quadratwurzelberechnung interpretiert.

Quelle: [2]

Methoden, Moleküle und Reaktionsnetzwerke

Eine allgemeinere und weiter vereinfachte Darstellung eines Reaktionsnetzwerkes folgt hier:

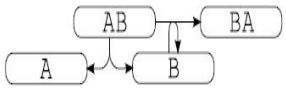


Figure 1: A small reaction network with two reactions.

Eine Menge von Reaktanden AB (Input Moleküle) wird in A und B aufspalten, außerdem reagiert eine Menge Reaktanden zu BA und B, wobei auch B wiederum aus der Reaktion hervorgeht.

Quelle: [1]

Allgemeiner lassen sich chemische Reaktionen abstrakt beschreiben: Es wird eine Menge von Grundbausteinen, in unserem Beispiel A und B, angenommen, aus denen komplexere Moleküle durch Verkettung zusammengesetzt werden können. AB und BA wären solche komplexen Moleküle. Im Modell gibt es für die Länge von Molekül-Zeichenketten keine Begrenzung, sie dürfen beliebig, aber endlich lang werden. Man kann sich jeden Grundbaustein vereinfacht als ein Atom oder als eine chemische Gruppe bzw. Substruktur vorstellen. Die Anzahl der Grundbausteine und die Reihenfolge der Grundbausteine innerhalb der Zeichenkette beschreibt jeweils eine einzigartige *Spezies* (Molekülart). Jeder Spezies wird dann seine Masse m zugeordnet.

Spezies können sowohl Ausgangsstoffe (Substrate) als auch Reaktionsprodukte sein. Eine Reaktion setzt dann eine bestimmte Anzahl Substrate zu entsprechenden Reaktionsprodukten um. Dies lässt sich durch aufgespreizte Pfeile veranschaulichen, so dass ein Geflecht (Netzwerk) von abstrakten chemischen Reaktionen entsteht. Ein konkretes Netzwerk, zum Beispiel das aus Abbildung 1, bezeichnet man als ein Individuum (Repräsentation) eines Reaktionsnetzwerks. Es enthält eine Menge von Reaktionen sowie eine bestimmte Anzahl an eingebundenen Spezies. Eine Spezies wird als Ausgabespezies festgelegt. Ferner wird vereinbart, dass jede Reaktion höchstens zwei Ausgangsstoffe und höchstens zwei Reaktionsprodukte beinhaltet. Diese Begrenzung ist von der Natur inspiriert worden, denn dort sind enzymatische Reaktionen mit mehr als zwei Substraten sehr selten. Damit das Massenerhaltungsgesetz in chemischen Reaktionsnetzwerken erfüllt wird, muss für jede einzelne Reaktion Gesamtmasse der Ausgangsstoffe der gesamten Masse der Reaktionsprodukte entsprechen [3].

Damit der Verlauf der Reaktionen und mithin das Verhalten des gesamten chemischen Reaktionsnetzwerkes modelliert werden kann, wird das Massenwirkungsgesetz zu Hand genommen [4]. Dadurch kann bestimmt werden, wie schnell oder wie langsam die einzelnen Reaktionen ablaufen sollen, so dass ein „Feintuning“ der Berechnung erfolgt, um das gewünschte Ergebnis so genau wie möglich zu erhalten. Ein Beispiel: das Grundbaustein-Alphabet {A,B} mit dem Reaktionsnetzwerk aus dem obigen Bild (Figure 1) symbolisiert die Reaktionen r_1 : AB → A + B und r_2 : AB + B → BA + B mit den jeweiligen Reaktionsraten k_1 und k_2 (beide > 0). Diese beiden Parameter bestimmen die Geschwindigkeit der Reaktion und sind für die Genauigkeit des Ergebnisses maßgeblich.

Netzwerkevaluation und Fitness

Ein individuelles Netzwerk wird überprüft mit seinen Eingangs- und Ausgangsspezies, und es wird betrachtet, wie die Menge der beteiligten Moleküle sich über der Zeit verhalten. Dafür kann ein Netzwerk mit mathematischen Formeln beschrieben und analysiert werden. Mit gewöhnlichen Differentialgleichungen wird der Systemzustand und das eingeschwungene Verhalten beschrieben. Erfüllt ein Netzwerk die gewünschte Funktion, wird das Individuum behalten, erfüllt es diese nur ungenügend, wird es verworfen. Inwiefern ein Netzwerk die Funktion genügend oder ungenügend erfüllt, wird anhand des Ausgangsstoff-Konzentrationsverlaufs bestimmt. Je näher die nach einer

gewissen Reaktionszeit erreichte Ausgangsstoff-Konzentration am gewünschten Ergebnis liegt, umso höher ist die „Fitness“, also die Eignung des Reaktionsnetzwerks für die gestellte Aufgabe.

Als Beispiel werden die Reaktionen aus Abbildung 1 verwendet, wobei die Schreibweise [*s*] der Konzentration der Spezies *s* entspricht.:

$$d[AB]/dt = -k1[AB] - k2[AB][B]$$
$$d[A]/dt = d[B]/dt = k1[AB]$$
$$d[BA]/dt = k2[AB][B]$$

Zur numerischen Lösung solcher Differentialgleichungssysteme (englisch: ordinary differential equations, abgekürzt ODE) und damit zur Ermittlung der Stoffkonzentrationsverläufe existiert bereits eine standardisierte Beschreibungssprache, nämlich die *Systems Biology Markup Language* (SBML). Außerdem gibt es eine Bibliothek, die Netzwerkbeschreibungen in diesem Format auswerten kann. Sie heißt *SBML ODE Solver Library*.

Um das Verhalten des Netzwerkes zu simulieren, sind einige Initialisierungen notwendig. Zum Zeitpunkt 0 besitzt das Netzwerk eine gewisse Stoffkonzentration der Eingangsspezies (Eingabewerte). Alle anderen Stoffkonzentrationen innerhalb des Netzwerkes werden üblicherweise auf 0 gesetzt. Danach können schrittweise die Stoffkonzentrationen im nächsten betrachteten Zeitpunkt bestimmt werden und dann zum darauffolgenden Zeitpunkt usw., bis das System den gewünschten Endzustand erreicht hat, sich also die Konzentrationswerte kaum noch oder gar nicht mehr verändern.

Genetischer Algorithmus

Genetische Algorithmen verwenden zufällig generierte Reaktionsnetzwerke mit einer bestimmten Menge an Reaktionen und Spezies. In den erwähnten Versuchen des Papers wurden die Reaktionen mit 2 bis 7 zufälligen Atomen initialisiert. Die Reaktionsraten k wurden zwischen 0 und 10 gewählt. Ein neues Individuum in der nächsten Generation wird erschaffen, indem zwei Elternteile (Reaktionsnetzwerke) ähnlicher Fittness der vorangegangenen Generation ausgewählt werden. Diese werden über Kreuz miteinander kombiniert, um neue Nachkommen zu erzeugen. Diese Nachkommen enthalten erwünschte wie auch unerwünschte Veränderungen ihrer Reaktionsnetzwerkstruktur, sogenannte Mutationen, und müssen deshalb wieder den Evaluations- und Fitnesstest durchlaufen. Es gibt einen Mutationsoperator μ, der bei der Kreuzung der Individuen die Häufigkeit (Anzahl Veränderungsschritte) angibt. Dabei unterscheidet man zwei Stufen. In der ersten Stufe wird allein die Netzwerkstruktur verändert. In der zweiten Stufe erfolgen lediglich Anpassungen der Reaktionsraten (Parameterwerte), mit denen die Reaktionen ablaufen. Darüber hinaus existiert ein weiterer evolutionärer Operator, der ein zufälliges Atom in das Alphabet einfügt und/oder ein zufälliges Atom daraus entfernt. Hin und wieder lässt man diesen Operator aktiv werden. Der Pool an Reaktionsnetzwerken, die sogenannte Population, evolviert durch aufeinanderfolgendes Einwirken von Mutationen. Hierbei entstehen strukturell verschiedene Reaktionsnetzwerke. Es werden weitere Schritte vorgenommen, um die Häufigkeit der einzelnen Mutationen aufeinander abzustimmen, so dass ein möglichst ein breites Spektrum unterschiedlicher Reaktionsnetzwerke entsteht. Ein typischer genetischer Algorithmus startet mit 100 Individuen und mit nur wenigen Reaktionen pro Individuum.

Ein wichtiger Grund für die festgelegten Mutationsarten ist die Minimierung der Netzwerkkomplexität. Das Ziel ist, das kleinstmögliche Netzwerk zur Lösung der gewünschten Berechnungsaufgabe zu finden, also dasjenige mit sehr wenigen Reaktionen, um so elementare mathematischen Operationen abzubilden. Obwohl mit einer beschränkten Menge an Platz und wenigen Reaktionen es nicht immer möglich ist, beliebig komplizierte numerische Berechnungen darzustellen, kann das gewünschte Ergebnis zumindest angenähert bzw. approximiert werden.

Parameteroptimierung

Es werden die im vorangegangenen Abschnitt erwähnten genetischen Algorithmen eingesetzt, um neue „bessere“ und effizientere Netzwerke zu erhalten. Für ein Feintuning wurde mit 100 Individuen über 100 Generationen getestet mit dem Ergebnis, dass einfache mathematische Formeln mit einfachen chemischen Reaktionen nachgebildet werden können. Als Beispiel: Eine Multiplikation $[A]*[B]$ mit den Eingangsspezies A und B kann als Reaktion A + B → AB mit der Reaktionsrate k = 1 dargestellt werden. Da die Reaktionsrate k mit einem zufällig gewählten Wert zu Beginn der künstlichen Evolution initialisiert wird, bedarf es mehrerer Korrekturzyklen durch entsprechende evolutionäre Operatoren, bis der Zielwert, der zum gewünschten Berechnungsergebnis führt, entweder erreicht oder hinreichend gut angenähert ist.

Netzwerke zur Berechnung der Quadratwurzel

Der genetische Algorithmus wurde verwendet, um ein Netzwerk zu konstruieren, welches für eine anfängliche Konzentration der Eingangsspezies ABC (zusammengesetzt aus Grundbausteinen des Alphabets {A,B,C}) die Quadratwurzel berechnet und als Konzentration einer Ausgabespezies bereitstellt. Mit den verschiedenen Zahlen X = 1, 4, 9, 16, …, 100 wurden jeweils unabhängig voneinander verschiedene Testeingaben vorgegeben. Die ermittelten Ausgangskonzentrationen (Berechnungsergebnisse) der evolvierten Netzwerke wurde anschließend mit den erwarteten Werten verglichen. Welches Netzwerk der Wurzelberechnung am nächsten kam, hat als Ausgangskonzentration, in seinem eingeschwungenen Zustand das gewünschte Ergebnis.
Es folgen ein paar Beispielnetzwerke für die Quadratwurzelberechnung:

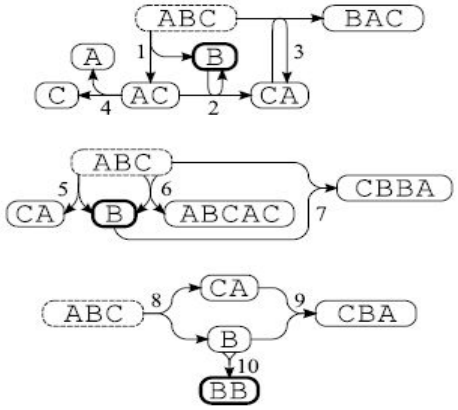


Figure 2: Three evolved networks that compute the square root function. The input molecule is denoted by the dashed outline, where the output is in bold.

Quelle: [1]

Die drei Netzwerke können jeweils mit Differentialgleichungen dargestellt werden. Nun werden für die Netzwerke die verschiedenen Reaktionsgleichungen genannt. Jeweils wird ein k mit angegeben, welches für die Geschwindigkeit der Reaktion steht. Für das erste Netzwerk:
ABC → B + AC mit k_1 = 1,159
AC + B → CA + B mit k_2 = 3,716
ABC + CA → BAC + CA mit k_3 = 5,590
AC → A + C mit k_4 = 7,881

Die Reaktionsgleichungen für das zweite Netzwerk:
ABC → B + CA mit k_5 = 7,359
ABC + ABC → B + ABCAC mit k_6 = 1,502
ABC + B → CBBA mit k_7 = 1,468

Es folgen die Reaktionsgleichungen für das dritte Netzwerk:
ABC → B + CA mit k_8 = 5,56937
CA + B → CBA mit k_9 = 6,62825
B + B → k10 BB mit k_{10} = 8,34484

Bei allen drei Netzwerken entsteht ein gewisser Berechnungsfehler, also eine leichte Abweichung vom richtigen Ergebnis. Damit das Problem gelöst werden kann, gibt es eine sogenannte Müllspezies (Waste). Beispielsweise gibt es in der zweiten Reaktionsgleichung Waste-Moleküle CBBA, diese werden in kleinere Waste-Moleküle BB und CA zerlegt. Dabei verändert diese Auffang- oder Ausgleichsreaktion nicht die Produktion der eigentlichen Ausgabe, nämlich der Moleküle B aus den Eingangs-Molekülen ABC. Mithilfe der Müllspezies lässt sich stets die gewünschte Masseerhaltung erreichen. Zudem kann dank der Müllspezies das erwartete Ergebnis stärker angenähert werden. Es folgt eine Darstellung, wie mit der Müllspezies gearbeitet werden kann.

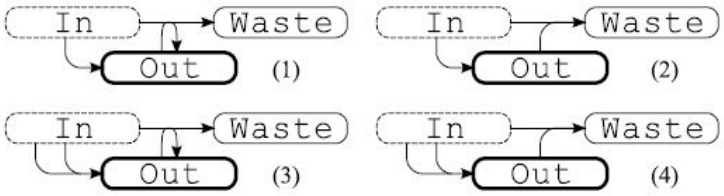


Figure 3: Four network motifs act as the kernels of evolved square root networks.

Quelle: [1]

Bei dem Einsatz der Müllspezies wurden auch Untersuchungen vorgenommen, denn die vier unterschiedlichen Arten, die Müllspezies in die Reaktionsnetzwerke einzubinden, haben auch unterschiedliche Folgen für das jeweilige Gesamtverhalten. Die Beschreibungen (3) und (4) sind am schlechtesten für den Einsatz bei der Wurzelberechnung, wobei die Beschreibung (1) ein sehr gut approximiertes Ergebnis erzielt hat. Die Beschreibung (2) wird als mittelmäßig bewertet. Der relative Wurzelberechnungsfehler von (1) liegt bei 0,049 mit k_{out}/k_{waste} = 0,579, damit sind die Abweichungen unterhalb der 5%-Grenze und mithin gering. Alle betrachteten Reaktionsnetzwerke erfüllen das Massenerhaltungsgesetz.

Analyse der Quadratwurzelberechnung

Zur Wurzelberechnung wurde (1) aus Abbildung 3 näher untersucht. Das Verhalten dieser Reaktion lässt sich durch ein Differentialgleichungssystem beschreiben. Bezeichnet man der Einfachheit halber die Eingabespezies In als *x* sowie die Ausgabespezies Out als *y* und die Müllspezies Waste als *w*, so entsteht:

$$dx(t)/dt = -k1\,x(t) - k2\,x(t)\,y(t)$$
$$dy(t)/dt = k1\,x(t)$$
$$dw(t)/dt = k2\,x(t)\,y(t)$$

Für die Berechnung der Quadratwurzel wird eine initiale Stoffkonzentration *x* vorgegeben (Zeitpunkt 0), so dass $x(0) = x$ und $y(0) = 0$ sowie $w(0) = 0$ wobei der Wert von $y(t)$ für große t den Wert $\sqrt{(x)}$ erreichen soll. Als nächstes wird das Verhalten der Reaktion untersucht. Die Eingangsspezies *x* wird allmählich abgebaut (aufgebraucht): $\lim_{t \to \infty} x(t) = 0$

Der Zielwert der Ausgabespezies sei $\hat{y}=\lim_{t\to\infty} y(t)$ und die begleitend entstehende Konzentration der Müllspezies sei $\hat{w}=\lim_{t\to\infty} w(t)$

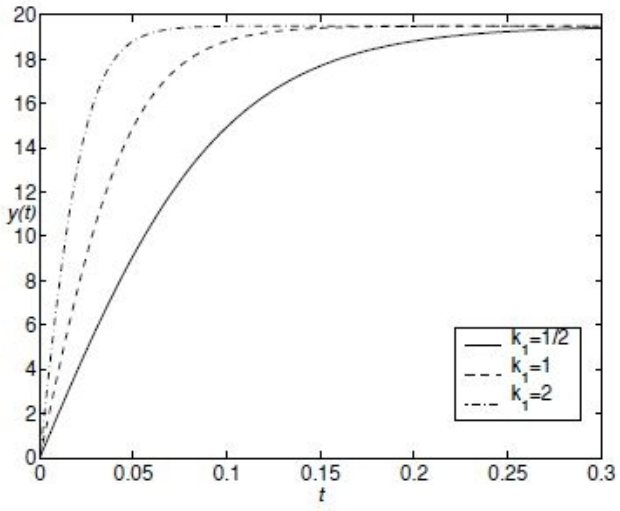
Als nächstes wurde die Summe der Konzentrationen berechnet mit $t\geq 0$

$x(t)+y(t)+w(t)=X$ Nun soll *y*(*t*) die Wurzel aus *x*(0) berechnen, also wird $y^2(t)$ betrachtet.

Mit den Differentialgleichungen $d\,y^2\frac{(t)}{dt}=2k1\,y(t)\,x(t)=(\frac{2k1}{k2})(dw(t)/dt)$ wird die Wurzelberechnung näher beschrieben. Durch Umstellen wird die Gleichung zu: $y^2(t)-\frac{2k1}{k2}w(t)=0$ Damit folgt: $\hat{y}+\hat{w}=x, \widehat{y^2}-\frac{2k1}{k2}\hat{w}=0$ Daraus ergibt sich nach weiteren Umstellungen: $\hat{y}=\frac{-k1}{k2}+\sqrt{(\frac{2k1}{k2}x+\frac{k^21}{k^22})}$ Somit ist *x* nur noch von *k1* und *k2* abhängig. Zur weiteren Vereinfachung wird nun *k2* = 2 * *k1* gesetzt. Damit ist eine bekannte quadratische Gleichung entstanden. $\hat{y}=\frac{-1}{2}+\sqrt{(x+\frac{1}{4})}$ Eine gute Annäherung an Wurzel *x* kann dadurch erreicht werden.

Einschwingverhalten

Als weitere Tatsache wird im Paper erwähnt, das die chemischen Quadratwurzelberechnungen Zeit brauchen. Wie wir gesehen haben, ist erst nach einer gewissen Zeit *t* das gewünschte Ergebnis in guter Genauigkeit erzielt worden. Im folgenden Bild ist es für das Beispiel *x* = 400 zu sehen. Nach einer gewissen Zeit *t* = 0,25 bzw. 0,3 liegt für *y* ein eingeschwungener Wert vor, die Konzentration der Ausgabespezies hat den Zielwert 20 nahezu erreicht:



Quelle: [1]

Figure 4: exact solution for $X = 400$ and $k_1 = 1/2, 1$ and 2

Zusammenfassung und Diskussion der Ergebnisse

Die Autoren Liekens, Eikelder, Steijaert und Hilbers haben einen genetischen Algorithmus entwickelt, der sowohl den Masseerhaltungssatz für chemische Reaktionen einhält, als auch gezeigt, dass mit Hilfe von chemischen Reaktionen und künstlicher Evolution Netzwerke für mathematische Berechnungen gebaut werden können. Aus anfänglichen Stoffkonzentrationen als Eingabedaten wird durch den Verlauf der chemischen Reaktionen ein Berechnungsergebnis als finale Stoffkonzentration geliefert.
Ist die Reaktion im eingeschwungenen Zustand, so hat sie etwas „berechnet“. Die chemischen Reaktionen sind stabil dank der Einhaltung des Masseerhaltungsgesetzes und das erzielte Ergebnis kann wiederum als Operand auf nachgeschaltete mathematische Terme übertragen werden. Mit Hilfe einer Müllspezies kann das gewünschte Ergebnis ziemlich genau erhalten werden.
Eine Beschränkung hat der Ansatz dann leider doch: Ist ein Netzwerk in sein eingeschwungenes Verhalten eingetreten, die chemischen Reaktionen also im Gleichgewicht angelangt, so muss für eine neue Berechnung das Netzwerk zurückgesetzt werden. Die Müllspezies muss wieder aus dem System zurück geholt oder entfernt werden und alle anderen Hilfsspezieskonzentrationen auf 0 initialisiert, damit von vorn gerechnet werden kann. Die Autoren nennen daher ihre Netzwerke „Single shot networks“ - nur für einen Berechnungs-Schuss geeignet, danach muss „nachgeladen“ werden.

Quellen

[1] Simulated Evolution of Mass Conserving Reaction Networks
Autoren: Anthony M. L. Liekens, Huub M. M. Ten Eikelder, Marvin N. Steijaert, Peter A. J. Hilbers
Link: http://www.informatik.uni-trier.de/~ley/pers/hd/l/Liekens:Anthony_M=_L=

[2] Preliminary Studies on the In Silico Evolution of Biochemical Networks
Autoren: A. Deckard und H. M. Sauro
Link: http://onlinelibrary.wiley.com/doi/10.1002/cbic.200400178/pdf , ChemBioChem, 5(10):1423-31.

[3] Masseerhaltungsgesetz, http://de.wikipedia.org/wiki/Massenerhaltung

[4] Massenwirkungsgesetz, http://de.wikipedia.org/wiki/Massenwirkungsgesetz

[4]

Elisabeth Vogel

DNA-Computing: Lösung einer Instanz des 3-SAT-Problems

Reflexion des Fachartikels

Ravinderjit S. Braich, Paul W.K. Rothemund, Cliff Johnson,
Nickolas Chelyapov, Leonard Adleman.
Solution of a 20-Variable 3-SAT Problem on a DNA Computer.
Science **296(5567)**:499-502, 2002

Kapitel 1

Einführung

Das Erfüllbarkeitsproblem der Aussagenlogik (SAT) ist von großer Bedeutung, vor allem für die Komplexitätstheorie und die Optimierung boolescher Formeln. Darüber hinaus ist SAT ein *NP*-vollständiges Problem und in den späteren Kapiteln dieser Ausarbeitung wird deutlich werden, dass es in der Klasse *NP* eine wichtige, vielleicht sogar die wichtigste, Rolle übernimmt.
Grundlage der hier aufgeführten Gedankengänge, Algorithmen und technischen Komponenten ist folgender wissenschaftliche Artikel aus dem *Sciencexpress*:

„Solution of a 20-Variable 3-SAT Problem on a DNA Computer“ [1]

Dabei wird vorausgesetzt, dass elementares Wissen im Bereich boolesche Algebra (insbesondere Aussagenlogik) vorhanden ist. Entsprechende Informationen können zum Beispiel aus dem Buch *Logik für Informatiker* [2] bezogen werden.
Einleitend soll eine weiterführende Definition des *(Boolean) satisfiability problem* oder kurz SAT gegeben werden. Darauf aufbauend wird auch 3-SAT kurz beleuchtet werden, um eine solide Basis für den nötigen Berechnungsablauf zu schaffen.
Danach werden die Voraussetzungen definiert und im Weiteren der DNA-Computer mit all seinen Besonderheiten sowie der eigentliche Berechnungsablauf vorgestellt. Abschließend wird auf die Bestimmung der Antwort sowie die Grenzen des Verfahrens eingegangen werden.
In einem Zusatzkapitel wird ein kleiner Exkurs zum Aufbau von DNA und ihren Eigenschaften zu finden sein. Dieses Kapitel stellt keinen essentiellen Bestandteil dieser Ausarbeitung dar, sondern soll lediglich zum schnelleren Verständnis des Verhaltens eines DNA-Computers dienen.

1.1 SAT - Boolean satisfiability problem

Im diesem Abschnitt wird es jeweils um eine minimale Definition von SAT und vertiefend von 3-SAT gehen. Beide Definitionen werden dabei von Beispielen begleitet, um sie zu verdeutlichen.

1.1.1 Definition

Formal wird das Erfüllbarkeitsproblem der Aussagenlogik wie folgt definiert.

Definition 1 (SAT - (Boolean) satisfiability problem) *Das Erfüllbarkeitsproblem der Aussagenlogik ist ein Entscheidungsproblem. SAT ist NP-vollständig. Es stellt sich die Frage, ob eine aussagenlogische Formel erfüllbar ist.*

Dieses Problem ist in exponentieller Zeit entscheidbar und zwar in der Anzahl der Variablen. Ein Ansatz wäre zum Beispiel das Aufstellen einer Wahrheitswertetabelle. Bisher ist nicht bekannt, ob es auch einen Algorithmus gibt, der SAT in Polynomzeit lösen kann. Ein möglicher Berechnungsansatz soll aber in dieser Ausarbeitung beleuchtet werden.
Es ist noch wichtig zu wissen, dass SAT in der Klasse der *NP*-Probleme liegt. Das bedeutet, das SAT in polynomieller Zeit mit einer nichtdeterministischen Turingmaschine lösbar ist. Ob es auch mit einer deterministischen Turingmaschinen in polynomieller Zeit lösbar ist, ist nicht bekannt, die Frage kann aber folgendermaßen formuliert werden: $SAT \in P$?
Außerdem ist SAT laut Definition auch *NP*-vollständig. Das heißt also, das jedes Problem aus *NP* in polynomieller Zeit auf SAT reduziert werden kann. (Reduktion [3])
Diese Eigenschaft wurde von Stephen A. Cook und Leonid Levin Anfang der 1970er Jahre unabhängig voneinander bewiesen. [4]
Sollte also die Eigenschaft $SAT \in P$ gelten, dann wäre jedes Problem aus *NP* auch in *P* und daraus kann gefolgert werden, dass $P = NP$ [5] ist.

1.1.2 Beispiel

Zur Verdeutlichung der Definition soll ein kleines Beispiel dienen. Als Eingabe liegt eine aussagenlogische Formel F in konjunktiver Normalform vor:

$$F_{SAT} = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2})$$

Wie oben beschrieben lässt sich diese, doch sehr kurze und einfache Formel schnell mit Hilfe einer Wahrheitswertetabelle lösen:

x_1	x_2	x_3	$\overline{x_1} \vee x_2 \vee x_3$	$x_2 \vee \overline{x_3}$	$x_1 \vee \overline{x_2}$	**F**
0	0	0	1	1	1	1
0	0	1	1	0	1	0
0	1	0	1	1	0	0
0	1	1	1	1	0	0
1	0	0	0	1	1	0
1	0	1	1	0	1	0
1	1	0	1	1	1	1
1	1	1	1	1	1	1

Folgende korrekte Belegungen für die Variablen x_1, x_2 und x_3 gibt es also:

$$\begin{array}{ccccc} x_1 = 0 & & x_1 = 1 & & x_1 = 1 \\ x_2 = 0 & \text{oder} & x_2 = 1 & \text{oder} & x_2 = 1 \\ x_3 = 0 & & x_3 = 0 & & x_3 = 1 \end{array}$$

1.2 3-SAT

SAT lässt sich auf viele verschiedene Arten einschränken oder auch erweitern. 3-SAT stellt dabei eine Einschränkung von SAT dar. Diese Variante fordert, dass jede Klausel drei Literale enthält. Obwohl es sich um eine Einschränkung des ursprünglichen Problems handelt, ist auch 3-SAT *NP*-vollständig, denn SAT lässt sich in polynomieller Zeit auf 3-SAT reduzieren. [6]
Jede Formel nach dem Muster von 3-SAT und p Variablen, sowie q Klauseln lässt sich auch als Graph darstellen. In Kapitel 2 ist dazu ein entsprechendes Beispiel zu finden. Eine Spezialform dieses Graphen führt zu einer weiteren Variante von SAT bzw. hier auch von 3-SAT. Ist eine Formel in 3-SAT und ist ihr Graph planar, so handelt es sich um P3SAT. Auch P3SAT ist *NP*-vollständig. Der Graph im Abschnitt 2.2 ist planar. Diese weitere spezielle Form von SAT soll aber hier nicht weiter Thema sein.

1.2.1 Definition

Definition 2 (3-SAT) *3-SAT ist das Erfüllbarkeitsproblem der Aussagenlogik mit 3 Literalen je Klausel. 3-SAT ist NP-vollständig. Es stellt sich die Frage, ob eine aussagenlogische Formel erfüllbar ist.*

Als Eingabe gibt es also eine Formel in konjunktiver Normalform mit 3 Literalen pro Klausel. Abkürzend wird diese Struktur auch als 3CNF bezeichnet. Formal kann eine Formel in 3-SAT mit n booleschen Variablen also folgendermaßen ausgedrückt werden.

$$\begin{array}{lcl} F_{3SAT} & = & C_1 \wedge C_2 \wedge ... \wedge C_m \text{ mit} \\ C_i & = & (L_{i,1} \vee L_{i,2} \vee L_{i,3}), i = 1,...,m \text{ wobei} \\ & & L_{i,j} \in \{x_k, \overline{x_k} \mid k \in \{1,...,n\}\},\ j \in \{1,2,3\},\ i = 1,...,m \end{array}$$

Alle weiteren Eigenschaften erbt 3-SAT von SAT. Lediglich die Struktur einer Formel in 3-SAT unterliegt den eben genannten speziellen Regeln.

1.2.2 Beispiel

Als Beispiel soll folgende kurze Formel dienen. Im Anschluss befindet sich die dazugehörige Wahrheitswertetabelle. Analog zu SAT sind alle Belegungen der Variablen x_1, x_2 und x_3 korrekt, welche die Formel mit booleschem Ergebnis *true* (F=1) erfüllen.

$$F_{3SAT} = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3})$$

x_1	x_2	x_3	$\overline{x_1} \vee x_2 \vee x_3$	$x_1 \vee x_2 \vee \overline{x_3}$	**F**
0	0	0	1	1	1
0	0	1	1	0	0
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0	0	1	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Kapitel 2

Voraussetzungen

In diesem Kapitel soll es um die aussagenlogische Formel gehen, welche für den hier verwendeten DNA-Computer gewählt wurde. Außerdem soll die Kodierung der Formel mit Hilfe von DNA erklärt werden.

2.1 Die Probleminstanz

2.1.1 Die Formel als Grundlage

Folgende Formel nach dem Muster von 3-SAT soll mit Hilfe eines DNA-Computers auf ihre Lösbarkeit hin getestet werden. Sie besteht aus 20 booleschen Variablen in 24 Klauseln. Das bedeutet, dass es für diese Formel 2^{20} (= 1'048'576) mögliche Belegungen der booleschen Variablen gibt.

$$\begin{aligned} F_G \quad = \quad & (\overline{x_3} \vee \overline{x_{16}} \vee x_{18}) \wedge (x_5 \vee x_{12} \vee \overline{x_9}) \wedge (\overline{x_{13}} \vee \overline{x_2} \vee x_{20}) \wedge (x_{12} \vee \overline{x_9} \vee \overline{x_5}) \wedge \\ & (x_{19} \vee \overline{x_4} \vee x_6) \wedge (x_9 \vee x_{12} \vee \overline{x_5}) \wedge (\overline{x_1} \vee x_4 \vee \overline{x_{11}}) \wedge (x_{13} \vee \overline{x_2} \vee \overline{x_{19}}) \wedge \\ & (x_5 \vee x_{17} \vee x_9) \wedge (x_{15} \vee x_9 \vee \overline{x_{17}}) \wedge (\overline{x_5} \vee \overline{x_9} \vee \overline{x_{12}}) \wedge (x_6 \vee x_{11} \vee x_4) \wedge \\ & (\overline{x_{15}} \vee \overline{x_{17}} \vee x_7) \wedge (\overline{x_6} \vee x_{19} \vee x_{13}) \wedge (\overline{x_{12}} \vee \overline{x_9} \vee x_5) \wedge (x_12 \vee x_1 \vee x_{14}) \wedge \\ & (x_{20} \vee x_3 \vee x_2) \wedge (x_{10} \vee \overline{x_7} \vee \overline{x_8}) \wedge (\overline{x_5} \vee x_9 \vee \overline{x_{12}}) \wedge (x_{18} \vee \overline{x_{20}} \vee x_3) \wedge \\ & (\overline{x_{10}} \vee \overline{x_{18}} \vee \overline{x_{16}}) \wedge (x_1 \vee \overline{x_{11}} \vee \overline{x_{14}}) \wedge (x_8 \vee \overline{x_7} \vee \overline{x_{15}}) \wedge (\overline{x_8} \vee x_{16} \vee \overline{x_{10}}) \end{aligned}$$

Speziell für diese Formel gibt es genau eine erfüllende Belegung, die aber mit einem DNA-Computer verifiziert werden wird. Man kann mit einem DNA-Computer exakt diese Belegung herausfinden aber das soll in einem späteren Kapitel betrachtet werden.
Für diese gegebene Formel ist folgende Belegung korrekt:

$$\begin{array}{llllll} x_1 = 0 & x_2 = 1 & x_3 = 0 & x_4 = 0 & x_5 = 0 & x_6 = 0 \\ x_7 = 1 & x_8 = 1 & x_9 = 0 & x_{10} = 1 & x_{11} = 1 & x_{12} = 1 \\ x_{13} = 0 & x_{14} = 0 & x_{15} = 1 & x_{16} = 1 & x_{17} = 1 & x_{18} = 0 \\ x_{19} = 0 & x_{20} = 0 & & & & \end{array}$$

2.2 Der DNA-Pool

2.2.1 Ausgangspunkt

Die Komponenten der oben definierten Formel müssen nun konkret an das DNA-Profil angepasst werden. Das bedeutet, dass zunächst eine Einigung darüber getroffen werden muss, wie die einzelnen Sektionen dargestellt werden sollen. Da es sich um einen Brute-Force-Ansatz handelt, muss als Grundlage die Kodierung aller möglichen Belegungen der booleschen Variablen zur Verfügung stehen. Die Grafik in Abbildung [2.1] soll diesen Ansatz sinngemäß darstellen.

Jeder Pfad durch diesen Graphen symbolisiert dabei eine mögliche Belegung. Bei 20 Variablen wären das 2^{20} verschiedene Pfade.
Um diese Sequenzen kodieren zu können müssen aber vorher noch alle Teile des gewählten Graphen einzeln in DNA „umgeformt“ werden. Das heißt, dass jeder Knoten x_i und jede Kante als spezifischer, halbseitig antiparallel-komplementärer DNA-Einzelstrang dargestellt werden muss. Außerdem werden noch die komplementären Stränge zu den x_i benötigt, sowie als Abschluss noch die Verschmelzung der Stränge, sodass am Ende jeder mögliche Pfad als DNA-Doppelstrang existiert.

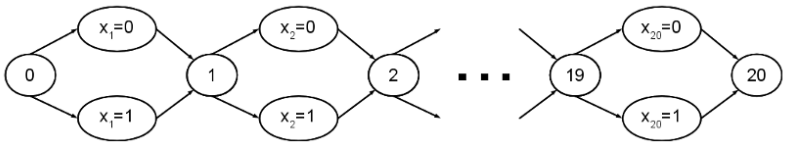
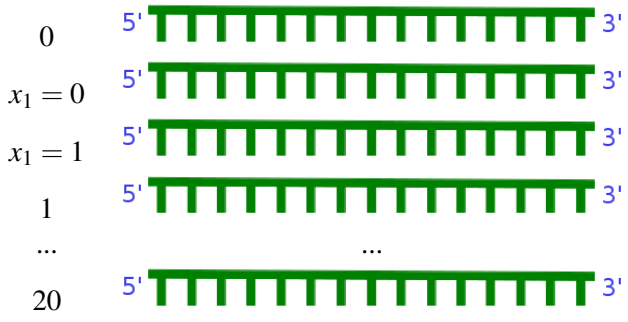


Abbildung 2.1: Grafische Darstellungsweise

2.2.2 Notwendige Komponenten

Im diesem Abschnitt wird kurz die grafische Darstellung der notwendigen Komponenten gezeigt. Eine kurze Einführung zum Thema DNA findet sich im Zusatzkapitel.

1. **Kodierung aller Knoten**
   Jeder Knoten wird als G-freie, 15 Basen umfassende Nucleotidsequenz dargestellt. Es handelt sich um G-freie Sequenzen, um mögliche Seiteneffekte weitgehend zu vermeiden. Folgende Darstellung wurde in dieser Ausarbeitung gewählt.



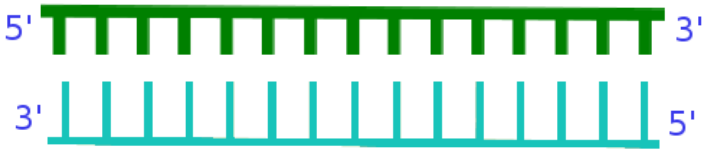
   Ein Beispiel wäre also die Kodierung von $x_1 = 0$:

   $x_1 = 0 \rightarrow$ 5’-CTCCTACAATTCCTA-3’

2. **Kodierung aller Kanten**
   Die Kanten werden ebenfalls als spezifische, halbseitig antiparallel-komplementäre DNA-Einzelstränge kodiert. Die Kanten sind dabei „spiegelverkehrt“ zu den x_i, denn mit ihrer Hilfe sollen die einzelnen Knoten später miteinander verbunden werden.

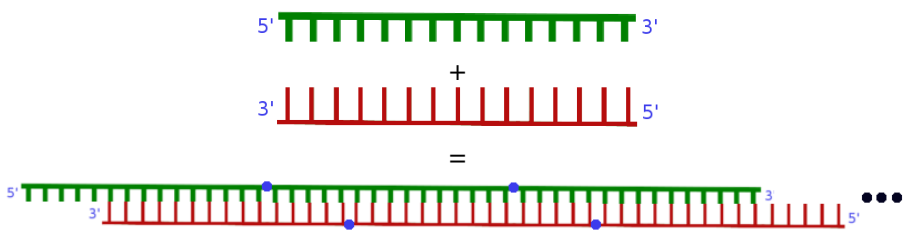
3. **Komplementäre Stränge zu den Kodierungen der x_i**
   Bei den komplementären Strängen zu den x_i handelt es sich nicht um die jeweils entgegengesetzte Belegung von TRUE und FALSE sondern um die Komplemente der Basen, also T zu A und C zu G. Diese Einzelstränge werden später eine Art Filter sein, mit dessen Hilfe die korrekte Belegung der Variablen herausgefunden werde kann.

   Als Vergleich der entgegengesetzten Kodierung soll wieder das Beispiel x_1 dienen:



$x_1 = 0 \rightarrow$ 5'-CTCCTACAATTCCTA-3'
$x_1 = 0 \rightarrow$ 3'-GAGGATGTTAAGGAT-5'

4. **Kodierung aller möglichen Pfade durch den Graphen**
   Abschließend müssen noch alle möglichen Pfade durch den Graphen kodiert werden. Dazu werden die Einzelstränge der Knoten und Kanten entsprechend ihrer Komplemente verbunden. Die dabei entstehenden Doppelstränge, die in mehrfacher Ausführung existieren, werden zusammenfassend als *Library* bezeichnet. Sie stellen den DNA-Pool dar, der, falls vorhanden, unter anderem auch die korrekten Lösungen der Formel enthält. Diese Lösungen müssen im Nächsten Schritt herausgefiltert werden.



Kapitel 3

DNA-Computer & Berechnungsablauf

Nachdem im vorherigen Kapitel der biologische Teil der Recheneinheit erklärt wurde, wird nun auf die technische Komponente eingegangen werden. Sie definiert im weitesten Sinne die Datenbahn, auf der die DNA-Doppelstränge gefiltert und in gewissem Maße auch sortiert werden.
Zuerst soll dabei die Elektrophorese-Box als Grundkomponente betrachtet werden. Diese Box wurde aber um einige zusätzliche Baueinheiten erweitert, auf deren Wichtigkeit auch in diesem Kapitel eingegangen werden wird.

3.1 Aufbau Elektrophorese-Box

3.1.1 Technischer Aufbau

Standardmäßig besteht eine Elektrophorese-Box (Abbildung [3.1]) aus einem negativen und einem positiven Pol. Der negative Pol definiert dabei den Start und der positive Pol das Ziel. Aufgrund der entgegengesetzten Ladung entsteht ein elektrisches Feld.
DNA ist negativ geladen, das bedeutet, dass DNA in den negativen Pol eingesetzt wird und über ein Gelbett (vgl. Agarose-Gelelektrophorese [7]) zum positiven Pol wandert. Die Driftgeschwindigkeit hängt dabei von der Länge der DNA-Kette ab: Je länger eine DNA-Kette ist, desto langsamer wandert der Strang.

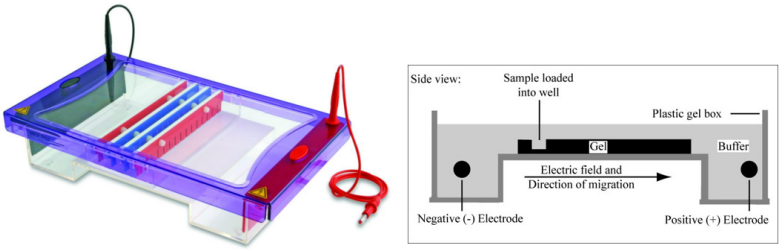


Abbildung 3.1: Elektrophorese-Box, links: [8] ,rechts: [9]

3.1.2 Besonderheiten

Folgender Aufbau der Elektrophorese-Box (Abbildung [3.2]) wird für den DNA-Computer verwendet. Es handelt sich hierbei um den standardmäßigen Grundaufbau inklusive einiger Veränderungen, die vor allem dazu dienen sollen, die DNA-Doppelstränge aufzubrechen und auch wieder zusammenzufügen.

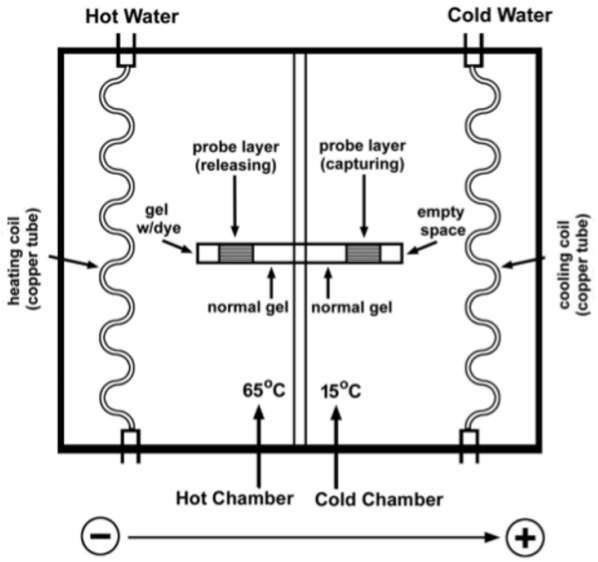


Abbildung 3.2: Elektrophorese-Box mit Modifikationen [1]

Der Box wurden zwei Kammern hinzugefügt, die jeweils einmal durch heißes und einmal durch kaltes Wasser spezifische Temperaturen erhalten. Die heiße Kammer (*Hot Chamber*) erhält dabei die Funktion des Aufbrechens, das bedeutet, dass durch die hohe Temperatur (65°C) die Bindungen zwischen den Basen aufgelöst werden. Die Doppelstränge werden also wieder zu Einzelsträngen aufgetrennt, deren Elemente nur noch durch eine Ligation verbunden sind. Dieser Prozess wird auch als *Melting* bezeichnet.
Im Gegensatz dazu führt die kalte Kammer (*Cold Chamber*) dazu, dass sich die Einzelstränge wieder zu Doppelsträngen verbinden. In dieser Kammer wird die Umgebung auf ca. 15°C herunter gekühlt, was den Vorgang des *Annealing*, also dem Zusammenfügen von Strängen, begünstigt.
Zusätzlich befindet sich innerhalb des Gelbettes eine Art Filter, der die oben beschriebenen komplementären Stränge zu den Kodierungen der x_i enthält. Diese Stränge müssen komplementär sein, damit die *Knotenstränge* des DNA-Pools in der Lage sind an ihnen andocken zu können. Der Filter dient in diesem Sinne also als eine Art Selektionseinheit, die Stränge herausfiltern soll, die definitiv nicht die Formel erfüllen werden. Alle anderen Stränge durchwandern ungehindert das Gelbett. Um die Prozesse des Meltung und des Annealing verstehen zu können, ist Basiswissen im Bereich Desoxyribonukleinsäure notwendig. In dieser Ausarbeitung befindet sich als abschließendes Kapitel eine kurze Einführung in diesen Themenkomplex. Ich verweise an dieser Stelle darauf, weil Grundlagenwissen im Bereich Aufbau und Verhalten von DNA sehr zum Verständnis der Funktionsweise dieses DNA-Computers beiträgt.

3.2 Berechnungsablauf

Im Allgemeinen lässt sich der Berechnungsablauf nun wie in Abbildung [3.3] darstellen. Der nachfolgende Prozess wird für jede Klausel der booleschen Formel einmal durchgeführt, beginnend mit der ersten: $(\overline{x_3} \vee \overline{x_{16}} \vee x_{18})$. Schlussendlich hat es also 20 Durchläufe gegeben, wobei nach jedem Turnus die Box komplett gereinigt werden und ein neuer Filter eingesetzt werden muss. Es handelt sich hier also um ein semi-automatisches Verhalten.
Ein Durchlauf besteht dabei aus folgenden Schritten, mit dem Ziel die korrekte/n Belegung/en herauszufiltern:
Die Pool-Stränge (*Library*) werden in die negativ geladenen Seite der Elektrophorese-Box geladen. Sie befinden sich damit gleichzeitig in der warmen Kammer. Sie bewirkt, dass die Doppelstränge zu Einzelsträngen denaturieren (vgl. Melting). Während die Stränge also durch das Gelbett zum positiven Pol wandern, brechen sie auf. (vgl. aufgespaltene Pool-Stränge).
Auf der anderen Seite des Gelbettes, also im positiv geladenen Teil der Elektrophorese-Box befindet sich eine Art Filter, der die komplementären Stränge der gerade auszuwertenden Klausel der Formel enthält. Sobald die Pool-Stränge in die kalte Kammer kommen werden sie zur Hybridisierung (vgl. Annealing) animiert und zwar mit den Strängen aus dem Filter. Es bleiben also alle Pool-Stränge an dem Filter „hängen“, die die gerade zu analysierende Klausel erfüllen. Alle anderen Stränge wandern zum Ende der Box und sind für die weitere Lösung der Formel nicht mehr von Bedeutung. An dem Punkt der Hybridisierung kommt auch die Wichtigkeit der G-freien Basen zum tragen. Wären die Basen nicht G-frei, so würde es zur massiven Verschmelzung der Einzelstränge unter anderem mit sich selbst kommen. Dieser Seiteneffekt ist aber so weit wie möglich zu vermeiden, denn jeder Strang, der mit sich selbst verschmilzt bedeutet unweigerlich einen Verlust, denn er ist für den restlichen DNA-Pool unbrauchbar. Abschließend werden die gefilterten Stränge durch erneute Erwärmung vom Separator gelöst und bilden damit den neuen DNA-Pool.
Der oben beschriebene Durchlauf wird für jede weitere Klausel wiederholt. Anschließend findet die Bestimmung der Antwort statt.

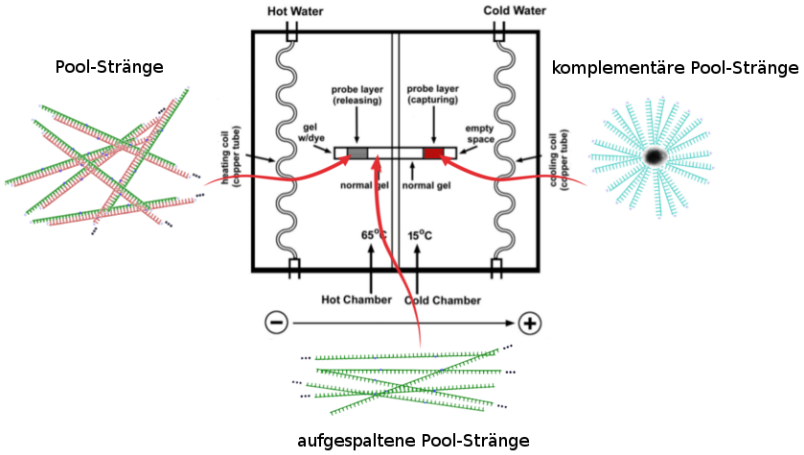


Abbildung 3.3: Schematischer Aufbau des Berechnungsablaufs,
unter Verwendung von [1]

Kapitel 4

Bestimmung der Antwort

Nach dem letzten Durchlauf werden die übrig gebliebenen Stränge PCR amplifiziert [10] und im Gel als sogenannte Bande sichtbar gemacht. Die Existenz dieser Bande entspricht der Problemlösung „Ja“, oder anders formuliert: Es gibt eine oder mehrere erfüllende Belegungen für die vorgegebene Formel. In Abbildung [4.1] ist diese Bande für die gegebene Formel aus Kapitel 2 durch rote Kreise markiert.
Wichtig ist aber zu wissen, dass es sich bei dem Ergebnis bis zu diesem Punkt nicht um die exakte Belegung der booleschen Variablen handelt, sondern lediglich um die Antwort auf die Frage: Existiert so eine Lösung überhaupt? In diesem Fall ist es eine eindeutige positive Antwort.
An dieser Stelle findet das Paper, welches dieser Ausarbeitung zu Grunde liegt sein Ende. Es wird nicht weiter darauf eingegangen, in welcher Weise und ob es möglich ist jetzt noch aus den Antwortsträngen die exakte Lösung zu extrahieren. Es müsste aber lediglich eine Sequenzierung der DNA-Einzelstränge aus dem Antwortpool unternommen werden. Mit Hilfe dieser Methode sollte es am Ende möglich sein auf die genaue Belegung der Wahrheitswerte zu kommen, sofern denn eine Bande existiert.

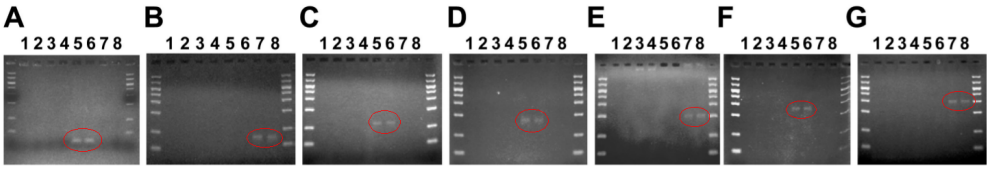


Abbildung 4.1: Visualisierung der Bande [1]

Kapitel 5

Zusatzkapitel: Exkurs DNA

Dieses Zusatzkapitel soll dazu dienen die Eigenschaften und das Verhalten von DNA so kurz wie möglich zu erklären, um den DNA-Computer, der in dieser Ausarbeitung vorgestellt wird verständlicher zu machen. DNA steht für Desoxyribonukleinsäure und kommt in allen Lebewesen sowie auch in bestimmten Virentypen vor. Das Biomolekül ist dabei der Träger der Erbinformationen.

Der strukturelle Grundaufbau eines DNA-Doppelstrangs entspricht dem in Abbildung [5.1] gezeigten Modell. Folgende Komponenten sind dabei zu sehen:

- A → Adenin
- T → Thymin
- C → Guanin
- G → Cytosin

Außerdem werden diese Basen durch ein Rückgrat aus alternierenden Desoxyribose- und Phosphorsäure-Untereinheiten („Phosphatrückgrat“) fixiert.

Abbildung 5.1: DNA [11]

Die Basen sind dabei in Paare aufgeteilt, die jeweils in sich selbst komplementär sind. Ein Paar bilden dabei Adenin ⇔ Thymin und das zweite Paar ist Guanin ⇔ Cytosin. Da es sich um komplementäre Paare handelt sind sie innerhalb ihrer Gruppe in der Lage zu hybridisieren. Wie auch schon in Abbildung [5.1] gezeigt, können sich Adenin und Thymin miteinander verbinden, analog gilt das auch für Guanin und Cytosin. Die Art der Verbindung entspricht dabei einer Wasserstoffbrückenbindung, die allerdings unterschiedlich stark ist, je nachdem, zwischen welchen Basen sie sich befindet:

- Adenin ⇔ Thymin: 2 Wasserstoffbrücken
- Guanin ⇔ Cytosin: 3 Wasserstoffbrücken

Genau auf diesem Verhalten, nämlich dem Aufbrechen und Verschmelzen von Basenpaaren, baut der in dieser Ausarbeitung vorgestellte DNA-Computer auf.

Literaturverzeichnis

[1] Ravinderjit S. Braich, Paul W. K. Rothemund, Cliff Johnson, Nickolas Chelyapov, Leonard Adleman. *Solution of a 20-Variable 3-SAT Problem on a DNA Computer*. Sciencexpress, März 2002.

[2] Jürgen Dassow, Frithjof Staiß. *Logik für Informatiker*. Vieweg+Teubner Verlag, April 2005.

[3] Definition: Polynomzeitreduktion. http://www-lehre.informatik.uni-osnabrueck.de/~graph/skript/12_Begriff_der.html. letzter Zugriff: Sept. 2014.

[4] The Complexity of Theorem-Proving Procedures. http://www.cs.toronto.edu/~sacook/homepage/1971.pdf. letzter Zugriff: Sept. 2014.

[5] Millennium-problem (4): P=NP. http://de.wikipedia.org/wiki/Millennium-Probleme. letzter Zugriff: Sept. 2014.

[6] P-completeness: Some Reductions. http://www-verimag.imag.fr/~duclos/teaching/inf242/SAT-3SAT-and-other-red.pdf. letzter Zugriff: Sept. 2014.

[7] Agarose-Gelelektrophorese. http://de.wikipedia.org/wiki/Agarose-Gelelektrophorese. letzter Zugriff: Sept. 2014.

[8] OncoGenomics LifeSciences. http://www.oncogenomicsls.com/instruments.aspx. letzter Zugriff: Sept. 2014.

[9] Agarose Gel Electrophoresis. goo.gl/MYQrNg. letzter Zugriff: Sept. 2014.

[10] DNA-Amplifikation (PCR). http://www.lfu.bayern.de/analytik_stoffe/biol_analytik_dna_untersuchungen/dna_amplifikation/index.htm. letzter Zugriff: Sept. 2014.

[11] Nucleobases and Their Production during the Photolysis of Astrophysically-relevant Ices. http://www.astrochem.org/sci/Nucleobases.php. letzter Zugriff: Sept. 2014.