An Immunological Approach to Combinatorial Optimization Problems & Exploring the Capability of Immune Algorithms : A Characterization of Hypermutation Operators

Ausarbeitung im Rahmen des Seminars
„Anwendungen und Algorithmen basierend
auf Prinzipien eines künstlichen Immunsystems“
Sommersemester 2005

Betreut von Dipl.-Inf. Thomas Stibor
Fachgebiet Sicherheit in der Informationstechnik
Technische Universität Darmstadt

von
Peter Halassek (peter@halassek.de)

Kurzdarstellung:
Dieses Referat befasst sich mit den zwei Publikationen von Vincenzo Cutello und Giuseppe Nicosia, die im Titel benannte sind und deren Referenzen. Es stellt die dort beschriebenen Ergebnisse vor und setzt sich kritisch mit ihnen auseinander.

Inhaltsverzeichnis

1 Einleitung

Dieses Seminar befasst sich mit den Publikationen [1] und [2] von Vincenzo Cutello und Giuseppe Nicosia und beleuchtet das Thema der auf Funktionsweisen des Immunsystems basierenden Algorithmen anhand dieses Beispiels genauer.

Es wird auf die Funktionen des Immunsystems, die für den Algorithmus wichtig sind, näher eingegangen (Kapitel 2). Anschließend wird der Basisalgorithmus vorgestellt (Kapitel 3) und anhand von zwei Testanwendung die Funktionsweise weiter verdeutlicht (Kapitel 4 und Kapitel 5). Es folgt ein kurzer Zwischenstand der bisherigen Ergebnisse (Kapitel 6). Kapitel 7 befasst sich mit den Erweiterungen des Immun-Algorithmus die in [2] besprochen werden. Diese werden in zwei weiteren Testanwendungen ebenfalls detaillierter vorgestellt (Kapitel 8 und 9). Den Abschluss bildet ein Fazit über diese Art der Suchalgorithmen (Kapitel 10).

In diesem Seminar werden die Vor- und Nachteile des vorgestellten Algorithmus besprochen, und es wird auf konkrete Anwendungen eingegangen. Ein Vergleich mit anderen Arbeiten, auch wenn dieser leider schwierig ist, wird auch versucht.

2 Das Immunsystem

2.1 Aufgabe des Immunsystems

Die Aufgabe des Immunsystems ist das Erkennen und Beseitigen von gefährliche Substanzen oder Moleküle. Diese Substanzen und Moleküle nennt man Antigene.

Das Immunsystem benötigt dazu spezielle Antikörper, die auf die jeweiligen Antigene angepasst sind.

Doch wie funktioniert diese Anpassung, und wie kann man sie für Problemlösungsverfahren adaptieren? Dies wird in den folgenden Kapiteln erläutert werden.

2.2 Klonale Selektion

Bei der klonalen Selektion werden nur Antikörper mit passenden oder möglichst ähnlichen Rezeptoren geklont (kopiert), um auf die Bedrohungen durch die Antigene zu reagieren.

Einige Zellen mit passenden Rezeptoren werden mit langer Lebenserwartung geklont, um ein Gedächnis für bereits erkannte Antigene aufzubauen.

Beim Klonen entsteht durch Hypermutation eine größere Vielfalt, die es ermöglicht, neue Erreger besser erkennen zu können. Das Immunsystem kann so auch Antigene erkennen, mit denen es vorher noch nie konfrontiert war.

2.3 Immunreaktion in zwei Phasen

Die Immunreaktion läuft im Körper in zwei Phasen ab:

1. Phase:
Erkennen und bekämpfen eines neuen Erregers und entwickeln des Gedächnisses

2. Phase:
Bereits bekannte Antigene werden durch spezialisierte Gedächniszellen erkannt und eine schnellere und stärkere Reaktion kann erfolgen.

2.4 Anwendung auf Problemlösungsverfahren

Um die Systematik des Immunsystems auf Problemlösungsverfahren anzuwenden setzt man die Antigen gleich den Parametern einer Problemestellung. Die zu den Parametern passenden Lösungen bilden dann die dazugehörigen Antikörper. Die beiden Phasen werde wie folgt adaptiert:

1. Phase (Trainingsphase):
Man beginnt mit zufälligen möglichen Teil-Lösungen, die nach und nach zu endgültigen optimierten Lösungen heranreifen. Man erhält einen oder mehrere Antikörper (wenn mehrere Lösungen möglich sind) mit korrekten Lösungen.

2. Phase (Testphase):
Trainierte Antikörper können verwendet werden, um ähnliche Probleme schneller zu lösen.

3 Der Immun-Algorithmus

3.1 Variablen

Die Grundlage des hier vorgestellten Immun-Algorithmuses stellt immer eine Random Search dar. Als Antigen verwendet man ein Menge an Variablen des kombinatorischen Optimierungsproblems. Als Antikörper verwendet man mögliche Lösungen als Bitstrings der Länge l.

Eine weitere wichtige Variable ist die Populationsgröße d und die Anzahl der Klone pro Zelle dub.

3.2 Vorgehen

Wie in folgendem Schema ersichtlich funktioniert der Algorithmus mit einer einfachen Schleife. Zum Zeitpunkt t = 0 wird die Population $P^{(t)}$ mit einer Menge d an zufälligen Lösungen initialisiert. Anschließend wird die Fitnessfunktion für jede Lösung ausgewertet. Mit dieser Startpopulation kann jetzt in die Schleife gestartet werden.

Bei jedem Schleifendurchlauf wird mit Hilfe der Terminierungsfunktion T festgestellt ob eine ausreichende Lösung gefunden wurde und der Algorithmus dann passend beendet. Meistens wird hier zusätzlich noch eine Zählvariable eingebaut die eine gewisse Maximalzahl an Generationen vorgibt. Dadurch werden Deadlocks vermieden, und die Ausführungszeit ist vorher abschätzbar.

Innerhalb der Schleife wird jeder Antikörper aus $P^{(t)}$ dup mal geklont (kopiert) und bildet die Menge P^{clo}. Diese Menge der Klone P^{clo} wird dann mutiert. Es sind verschiedene Mutationen möglich (siehe 3.3). Nachdem die Fitness der mutierten Antikörper P^{hyp} ermittelt wurde, wird aus den mutierten Antikörpern P^{hyp} und der alten Population $P^{(t)}$ die neue Population $P^{(t+1)}$ ausgewählt, indem die d fittesten Antikörper verwendet werden und die restlichen verworfen werden. Alle hier angesprochene Mengen sind Multisets, d.h. sie können das gleiche Element mehrmals enthalten. Dadurch wird sichergestellt, das besonders fitte Antikörper auch mehrmals geklont und mutiert werden können.

```
t = 0
initiate P(0) = zufällige mögliche Lösungen
evaluate P(0)
while ( T (P(t) ) = 0 )
   Pclo = cloning (P(t),dup)
   Phyp = hypermutation (Pclo)
   evaluate (Phyp)
   P(t+1) = select ((Phyp ∪ P(t)), d)        (multiset!)
   t = t+1
wend
```

Abbildung 1 Der Algorithmus in Psydocode

3.3 Mutation

Die Art der Mutation repräsentiert unterschiedliche Suchfunktionen im Lösungsraum. Als Mutation ist jede Map-Funktion f: {0,1} → {0,1} möglich und denkbar. Hier wird allerdings

nur ein einfaches Bit flip an beliebiger, zufälliger Stelle genutzt, da wir keine zu großen Veränderungen bewirken wollen.

3.4 Wichtige Größe: Population

Die Populationsgröße d stellt eine kritische Variable dar. Ist sie zu klein gewählt, kann es zu einer vorzeitige Konvergenz auf eine falsche Lösung kommen, und die tatsächliche Lösung wird dann nicht mehr gefunden.

Ist d jedoch zu groß gewählt, sind viele überflüssige Berechnungen zu machen und die Laufzeit des Algorithmus wird unrentable. Eine Fitness-Verbesserung dauert dann zu lange.

Um diesen beiden Effekten entgegen zu wirken, wird hier nur ein einfacher Algorithmus verwendet: ohne dynamische Populationsgröße können durch Beeinflussung von d und dub auch nichtdominante Lösungen gefunden werden und die Dynamik der Berechnungen lassen sich besser vorhersagen.

3.5 Nichtdominante Lösungen

Nichtdominante Lösungen werden durch einen besonderen Effekt auch ohne dynamische Populationsgröße gefunden. Dabei wird in der „Expansions Phase“ die Population vergrößert. Dies geschieht durch das Cloning und Hypermutation. Die Fitness Landscape wird dabei nach möglichen Lösungen untersucht. Anschließend wird in der „Reduktions Phase“ wird die Population dann wieder kleiner, da eine Selektion der besten Lösungen stattfindet.

4 Testanwendung 1

4.1 Problemstellung

Die erste Testanwendung stellt das Minimum Hitting Set Problem dar. Bei diesem Problem ist eine Menge U und eine Menge S von Untermengen von U gegeben. Außerdem ist eine positive Zahl $k \leq |U|$ gegeben. Gesucht wird eine eine Untermenge U' von U, sodass $|U'| \leq k$ und U' enthält aus jeder Menge in S mindestens ein Element.

Dieses Problem ist NP-vollständig. Das es in NP liegt lässt sich leicht feststellen, da eine geratene Lösung U# in polynomieller Zeit überprüft werden kann. NP-Härte lässt sich durch polynomielle Reduktion auf Vertex Cover [4] zeigen.

Als Antikörper wird ein Bitstring der Länge |U| gewählt, der jeweils angibt ob ein bestimmtes Element aus U in U' enthalten ist oder nicht. Dadurch können alle möglichen Lösungen durchsucht werden.

4.2 Fitness-Funktion

Die Fitness-Funktion spielt eine entscheidende Rolle im Immun-Algorithmus. Sie entscheidet ob eine korrekte Lösung gefunden wird und wie schnell dies geschieht. Wir verwenden als Fitness-Funktion:
$f_{hs}(x)= |x| + (|S| - Hits(x))$

Die möglichen Lösungen müssen beide Terme minimieren

Für $|S| - Hits(x) = 0$ hat man eine gültige Lösung gefunden.

4.3 Welche Lösung ist besser?

Die Chance einen Term zu minimieren sind bei beiden Termen gleich gut.

Beispiel:
$|S| = 50000$
$|x_1| = 40 \quad Hits(x_1) = 49997 \quad f_{hs}(x_1)=43$
$|x_2| = 42 \quad Hits(x_2) = 49999 \quad f_{hs}(x_2)=43$

Welche Lösung ist besser? Dies lässt sich im Moment noch nicht feststellen, deshalb sollten beide Lösungen gleichwertig weiterverfolgt werden.

4.4 Ergebnisse

Der Algorithmus wird mit folgenden Parametern getestet:
$|U| = 100$
$|S| = 1.000 \;\; 10.000 \;\; 50.000$
kurz geschrieben als hs100-1000 etc.

In Tabelle 1 kann man die Ergebnisse des Tests ablesen. Jeder Versuch wurde 100 Mal durchgeführt und die Ergebnisse dann gemittelt. Best gibt an, wie groß die beste, gefundene Lösung ist. #min gibt an, wie viele Minimallösungen gefunden wurden. AES ist die Abkürzung für average number of evaluations to solutions, gibt also an wieviele Berechnungsschritte bis zu einer gültigen Lösung nötig waren.

	hs100-1000		hs100-10000		hs100-50000	
d	25	100	50	200	50	200
dup	15	30	15	15	15	10
$best$	6	6	9	9	39	39
$\#min$	1	3	3	5	3	2
AES	2275	18000	6800	27200	45050	98200

Tabelle 1 Minimum Hitting Set Instanzen

Man kann erkennen das durch einen höheren Wert für d mehr nichtdominierende Lösungen gefunden werden. Ein höherer Wert für dup ergibt eine schnellere Konvergenz.

Untersucht man den Zusammenhang von d und dup gegenüber AES, so erhält man für hs100-1000 folgende Ergebnisse:

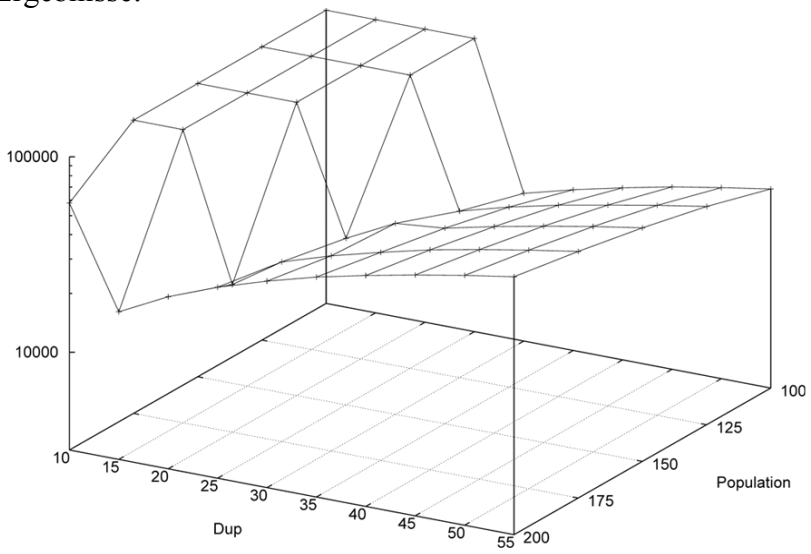


Abbildung 2 3D-Repräsentation der Versuchsergebnisse, mit den Dimensionen: d,dup und AES

Aus dieser Grafik lässt sich ablesen, das für kleine d (kleiner 175) und kleine dup (kleiner 25) keine Lösungen gefunden werden, da dort der Algorithmus nach 100.000 AES abgebrochen wurde. Dies liegt daran das die Gesamtpopulation zu gering ist, um die Fitness Landscape ausreichend weit zu untersuchen und daher keine Lösungen gefunden werden.

Werden die Werte für d und dup größer kann man erkennen, dass die Grafik dort einen Knick nach unten hat. Dort werden am effektivsten Lösungen berechnet. Werden die Werte noch größer steigt der Berechnungsaufwand wieder langsam an.

Ziel ist also, Parameterwerte möglichst dicht an diesem Knick zu finden um effizient Lösungen berechnen zu können.

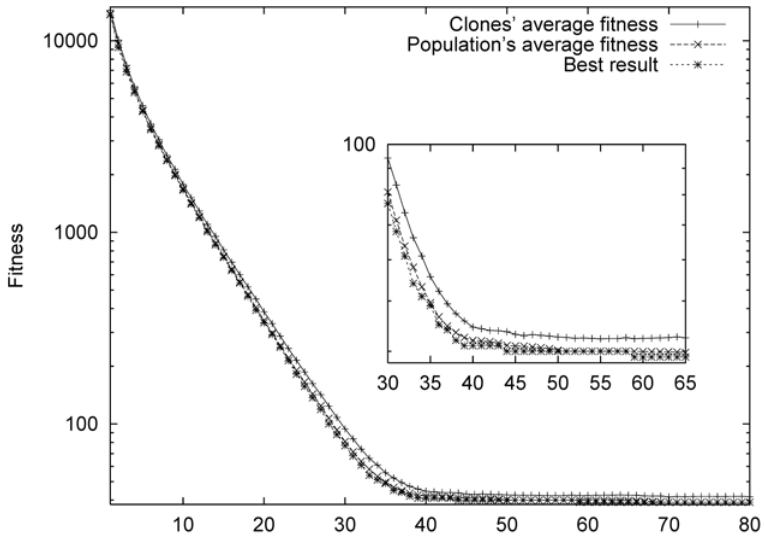


Abbildung 3 Fitness-Funktion aufgetragen gegen die Anzahl Generationen (Y-Achse logarithmisch, mit unterem Wert 39)

Sieht man sich den Verlauf der Fitness Kurve aller Antikörper an, kann man erkennen das Fitness exponentiell besser wird, bis dann ab Generation 60 ein Minimal Set gefunden wird. Danach werden nur noch nichtdominierende Lösungen mit gleicher Kardinalität gefunden.

5 Testanwendung 2

5.1 Problemstellung

Die zweite Testanwendung für den Immun-Algorithmus stellt das 3-Sat Problem dar. Gegeben sind dabei eine Menge von Variablen V und eine Menge C mit boolschen Formeln von jeweils 3 Variablen.

Gesucht wird eine Besetzung der Variablen mit Boolschen Werten, sodass alle Formeln erfüllt sind.

Das Problem ist NP-vollständig[8].

Als Antikörper wird ein Bitstring der Länge |V| gewählt, der die Werte die einzelnen Variablen enthält. Dadurch können alle möglichen Lösungen durchsucht werden.

5.2 Fitness-Funktion

Wir verwenden als Fitness-Funktion:
$f_{sat}(x)$ = #ErfüllteFormeln (c,x)

Die Fitness Funktion muss maximiert werden. Eine gültige Lösung ergibt sich für $f_{sat}(x) = |C|$

5.3 Problem-Erstellung

Um 3-SAT Instanzen zu generieren wurde A. van Gelders 3-SAT problem instance generator MKCNF.C[1] genutzt.

Wir verwenden Instanzen, die in der „transition phase“ des 3-SAT Problems liegen und besonders schwer zu lösen sind [5]. Dabei ist |C| = 4,3 |V|. Die Startwerte des Zufallsgenerators (rs:) sind mit angegeben, damit man die Instanzen selbst durchführen kann.

Die Instanzen sind:

- (i1): sat30-129 (rs: 83791)
- (i2): sat30-129 (rs: 83792)
- (i3): sat30-129 (rs: 83792)
- (i4): sat40-172 (rs: 62222)
- (i5): sat50-215 (rs: 82635)

5.4 Ergebnisse

	(i1)	(i2)	(i3)	(i4)	(i5)
d	50	50	50	50	75
dup	20	30	20	30	15
AES	14632	292481	11468	24276	50269

Tabelle 2 3-SAT Instanzen

Die Rechenzeit hängt proportional von der Anzahl und Schwierigkeit der Formeln ab

1 Verfügbar per anonymous ftp von ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/UCSS/instances

5.5 Erweiterung der Fitness-Funktion

Mit SAW (stepwise adaption of weights)

Case	V	C	Randseed	SuccessR	AES	SuccessR	AES
1	30	129	83791	1	2708	1	6063
2	30	129	83892	0.94	22804	0.96	78985
3	30	129	83792	1	12142	1	31526
4	40	172	83792	1	9078	1	13328
5	40	172	72581	0.82	37913	1	2899
6	40	172	62222	1	37264	0.94	82031
7	50	215	87112	0.58	17342	1	28026
8	50	215	82635	1	42137	1	60160
9	50	215	81619	0.26	67217	0.32	147718
10	100	430	87654	0.32	99804	0.06	192403
11	100	430	78654	0.04	78816	0.44	136152
12	100	430	77665	0.32	97173	0.58	109091

Tabelle 3 3-SAT Instanzen

In dieser Tabelle kann man die Ergebnisse einer erweiterte Version des Algorithmus begutachten. Die Erweiterung wird SAW (stepwise adaptation of weights) genannt[6]. Jede zu erfüllende Formel bekommt ein Gewicht zugeordnet. Das Gewicht aller Formeln, die nach T_p Generationen immer noch nicht gelöst sind wird erhöht ($\delta w = 1$). Wenn eine Formel mit einem höheren Gewicht gelöst wird, gibt es auch einen größeren Fitnesszuwachs und damit mehr Druck auf den Algorithmus, diese „schwierigen“ Formeln zu lösen.

In Tabelle 3 steht der Paramter SuccessR für die Erfolgsquote, die bei den verschiedenen Instanzen erreicht wurde. Jede Instanz wurde 100 Mal durchgeführt. Die letzten beiden Spalten beziehen sich auf die Ergebnisse aus [7], die auch einen evolutionären Algorithmus mit SAW Erweiterung benutzt. Der hier vorgestellte Algorithmus übertrifft die Ergebnisse aus [7] in den meisten Fällen, sowohl in der Erfolgsquote, als auch in der benötigten Rechenzeit.

6 Zwischenstand

Die Vorteile des vorgestellten Algorithmus sind, dass er relativ einfach ist, und sich seine Dynamik durch nur zwei veränderbare Parameter gut voraussagen lässt. Die benötigte Populationsgröße ist gering (Größenordnung 1000 und weniger).

Nachteile sind, das ein lokales Minimum bei zu kleinem d oder dup nicht gefunden werden kann, und bei zu hohe Werte wird unnötiger Overhead produziert.

Die Selektion benutzt einen elitären Auswahloperator, der zwar die Suche beschleunigt, aber zu einem lokalen Minimum führen kann, und damit die Vielfalt der Lösungen einschränkt

7 Variation des Algorithmus

7.1 Variation der Mutation

Um eine schnellere Konvergenz zu erzielen, kann man die Art der Mutation variieren.

(H1) Static Hypermutation

Bei der Static Hypermutation wird eine feste Anzahl an Mutationen ausgeführt. Diese Variante entspricht der bisher vorgestellten Methode.

(H2) Proportional Hypermutation

Bei der Proportional Hypermutation wird die Anzahl Mutationen proportional zur Güte der Fitness-Funktion gewählt. Je besser die Fitness eines Antikörper, umso mehr wird er mutiert.

(H3) Inversely Propotional Hypermutation

Hier wird die Anzahl Mutationen antiproportional zur Fitness-Funktion gewählt. Je besser die Fitness eines Antikörpers ist, umso weniger wird er mutiert.

(M) Hypermacromutation

Anzahl der Mutationen wird aus einem vorher gewählten Intervall zufällig gewählt und ausgeführt.

7.2 Aging Operator

Der hier verwendete Aging Operator wird „static pure aging“ genannt. Dabei bekommt jeder Antikörper τ. Wenn Antikörper zu alt sind ($\tau > \tau_B$), dann werden sie entfernt, egal welche Fitness-Funktion sie haben.

Beim Klonen bekommen geklonte Antikörper das gleiche Alter wie ihre Ursprungsantikörper. Wenn bei der Mutation eine Verbesserung der Fitness stattfindet, wird das Alter auf 0 gesetzt, damit der Antikörper wieder eine volle Lebensspanne hat, um seine Fitness weiter zu verbessern. Wird τ_B gößer als die maximale Generationszahl gesetzt so wird kein Aging benutzt.

7.3 Selection

Die variierte Selektion wird „$(\mu + \lambda)$-Selection“ genannt. Dabei wird keine Redundanz mehr genutzt. Die Parameter werden wie folgt gewählt:

$\mu = d$
$\lambda = Nc$ oder $2Nc$

Es werden dabei aus λ Antikörpern immer μ Antikörper ausgewählt.

Sollten nicht genug Antikörper den Alterungsprozess überleben, so werde so viele neue erzeugt, das mindestens wieder μ Antikörper existieren.

7.4 Ablauf

Immune Algorithm$(\ell, d, dup, \tau_B, c, H, HM)$
$Nc := d * dup$;
$t := 0$;
$P^{(t)}$:= Initial_Pop();
Evaluate($P^{(0)}$);
while (¬ Termination_Condition()) **do**
    $P^{(clo)}$:= Cloning ($P^{(t)}, Nc$);
    if (H) **then** $P^{(hyp)}$:= Hypermutation ($P^{(clo)}, c, \ell$);
        Evaluate($P^{(hyp)}$);
    if (HM) **then** $P^{(macro)}$:= Hypermacromutation (P^{clo});
        Evaluate ($P^{(macro)}$);
    $(P_a^{(t)}, P_a^{(hyp)}, P_a^{(macro)})$:= Aging($P^{(t)}, P^{(hyp)}, P^{(macro)}, \tau_B$);
    $P^{(t+1)}$:= $(\mu + \lambda)$-Selection ($P_a^{(t)}, P_a^{(hyp)}, P_a^{(macro)}$);
    $t := t + 1$;
end_while

Abbildung 4 Der erweiterte Algorithmus in Psydocode

Wie in Abbildung 4 zu erkennen ist, wurde der ursprüngliche Algorthmus durch Hypermacromutation ergänzt, und die Auswahl der neuen Population erfolgt nun über Aging und anschließender (μ + λ)-Selection. Die boolschen Parameter H und HM entscheiden darüber, ob eine Hypermutation bzw. eine Hypermacromutation durchgeführt wird. Der Parameter c gibt an wie stark sich die jeweils verwendete Variante der Mutation auswirkt.

8 Testanwendung 3

8.1 Problemstellung

Um den erweiterten Algorithmus zu testen werden sog. „Trap Functions“ verwendet. Sie verwenden als Eingabe die Anzahl der 1er in einem Bitstring der Länge l:

$$f(x) = \widehat{f}(u(x)) = \widehat{f}\left(\sum_{k=1}^{l} x_k\right)$$

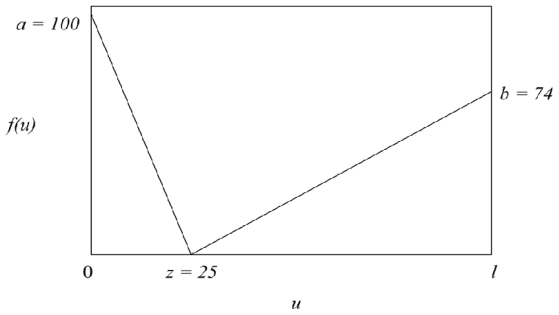
Abbildung 5 Formale Beschreibung der Summe der 1er im Bitstring

Wir unterscheiden in dieser Testanwendung zwischen einer „basic trap function“ und eine „complex trap function“. Diese werden im Folgenden vorgestellt.

basic trap function

Die Formel für die „basic trap function“ sieht wie folgt aus:

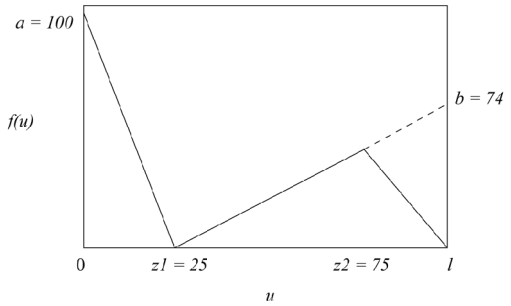
$$\hat{f}(u) = \begin{cases} \frac{a}{z}(z-u), & \text{if } u \leq z \quad ; \\ \frac{b}{l-z}(u-z), & \text{otherwise} \quad . \end{cases}$$



Abhängig von den verwendeten Parametern a,b und z und der Bitstringlänge l ergeben sich unterschiedliche Funktionsgraphen.

complex trap function

$$\hat{f}(u) = \begin{cases} \frac{a}{z}(z_1 - u), & \text{if } u \leq z_1 \quad ; \\ \frac{b}{l-z_1}(u - z_1), & \text{if } z_1 < u \leq z_2 \quad ; \\ \frac{b(z_2-z_1)}{l-z_1}(1 - \frac{1}{l-z_2}(u - z_2)), & \text{otherwise} \quad . \end{cases}$$



Die verwendete Formel wurde um einen weiteren Fall ergänzt. Der Parameter z wurde durch die beiden Parameter z_1 und z_2 ersetzt. Die „complex trap function“ ist der der „basic trap function“ sehr ähnlich.Dies ist beabsichtigt, da man durch gleichsetzten des Parameters z_2 mit l und z_1 mit z jede „complex trap function“ in eine „basic trap function“ umwandeln kann und somit den Algorithmus nur einmal programmieren muss.

8.2 Ergebnisse

Folgende Parametersets wurden für den Test verwendet:

I (l=10, z=3, a=12, b=6)

II (l=20, z=5, a=20, b=14)

III (l=50, z=10, a=80, b=39)

IV (l=75, z=20, a=80, b=54)

V (l=100, z=25, a=100, b=74)

Für die „complex trap“ wurde: $z_1 = z$, $z_2 = l - z_1$ gesetzt.

In den Tabellen 4 und 5 lassen sich die erzielten Ergebnisse ablesen. S steht dabei für eine „simple trap function“ und C für eine „complex trap function“. In den einzelnen Zellen steht als erstes Fett gedruckt die Erfolgsquote, gefolgt von der durchschnittlichen AES in kursiv. Außerdem wurden die Parameter τ_B und c angegeben, die das beste Ergebnis lieferten. Es wurden getestet für:

$\tau_B \in \{0{,}1, 0{,}2, 0{,}3, \ldots, 1{,}0\}$

$c \in \{1, 5, 10, 15, 20, 25, 50, 100, 150, 200, \infty\}$

In Tabelle 5 sind die Ergebnisse mit einer weiteren Erweiterung des Algorithmus angegeben. FCM steht dabei für „stop at first constructive mutation“. Dies bedeutet, das bei der Hypermacromutation keine weiteren Mutationen gemacht werden, sobald eine Verbesserung in der Fitnessfunktion aufgetreten ist. Dies verbessert die Ergebnisse deutlich.

Die erzielten Ergebnisse sind vergleichbar mit denen aus [8], wobei dort nur die Traps C(I) bis C(III) verwendet wurden.

Trap	*Static*	*Proportional*	*Inversely*	*Hypermacro M-mut*
S(I)	**100**, *576.81*	**100**, *604.33*	**100**, *504.76*	**100**, *4334.94*
	$\tau_B = 25$, $c = 0.9$	$\tau_B = 100$, $c = 0.2$	$\tau_B = 5$, $c = 0.3$	$\tau_B = 1$
S(II)	**34**, *82645.85*	**100**, *50266.61*	**97**, *58092.70*	**5**, *5626.8*
	$\tau_B = 20$, $c = 1.0$	$\tau_B = 25$, $c = 0.8$	$\tau_B = 20$, $c = 0.2$	$\tau_B = 50$
S(III)	**0**	**0**	**0**	**5**, *174458.6*
				$\tau_B = 50$
S(IV)	**0**	**0**	**0**	**0**
S(V)	**0**	**0**	**0**	**0**
C(I)	**100**, *389.67*	**100**, *404.94*	**100**, *371.15*	**100**, *1862.09*
	$\tau_B = 100$, $c = 0.6$	$\tau_B = 50$, $c = 0.1$	$\tau_B = 10$, $c = 0.2$	$\tau_B = 5$
C(II)	**100**, *36607.15*	**100**, *22253.70*	**100**, *44079.57*	**56**, *84001.29*
	$\tau_B = 15$, $c = 0.1$	$\tau_B = 25$, $c = 0.3$	$\tau_B = 10$, $c = 0.2$	$\tau_B = 50$
C(III)	**1**, *15337.00*	**0**	**2**, *236484.50*	**1**, *185318*
	$\tau_B = 50$, $c = 0.7$		$\tau_B = 25$, $c = 0.1$	$\tau_B = 50$
C(IV)	**0**	**0**	**0**	**0**
C(V)	**0**	**0**	**0**	**0**

Tabelle 4 Ergebnisse der verschiedenen Mutationsarten einzeln getestet

Trap	*Hypermacro FCM*	*Static+Macro*	*Proportional+Macro*	*Inversely+Macro*
S(I)	**100**, *1495.90*	**100**, *452.94*	**100**, *834.01*	**100**, *477.04*
	$\tau_B = 1$	$\tau_B = 25$, $c = 0.2$	$\tau_B = 50$, $c = 0.2$	$\tau_B = 15$, $c = 0.2$
S(II)	**28**, *64760.25*	**99**, *37915.17*	**100**, *51643.11*	**100**, *35312.29*
	$\tau_B = 1$	$\tau_B = 25$, $c = 0.1$	$\tau_B = 20$, $c = 0.7$	$\tau_B = 100$, $c = 0.2$
S(III)	**15**, *15677.53*	**100**, *18869.84*	**1**, *243038.00*	**100**, *20045.81*
	$\tau_B = 100$	$\tau_B = 50$, $c = 0.1$	$\tau_B = 20$, $c = 0.2$	$\tau_B = \infty$, $c = 0.1$
S(IV)	**24**, *40184.83*	**100**, *37871.71*	**0**	**100**, *42082.00*
	$\tau_B = 200$	$\tau_B = 25$, $c = 0.1$		$\tau_B = 25$, $c = 0.2$
S(V)	**27**, *139824.44*	**100**, *78941.79*	**0**	**100**, *80789.94*
	$\tau_B = 1$	$\tau_B = 100$, $c = 0.1$		$\tau_B = 50$, $c = 0.2$
C(I)	**100**, *826.78*	**100**, *367.67*	**100**, *644.67*	**100**, *388.42*
	$\tau_B = 50$	$\tau_B = 10$, $c = 0.1$	$\tau_B = 10$, $c = 0.1$	$\tau_B = 10$, $c = 0.2$
C(II)	**96**, *54783.25*	**100**, *24483.76*	**100**, *23690.82*	**100**, *29271.68*
	$\tau_B = 15$	$\tau_B = 10$, $c = 0.2$	$\tau_B = 50$, $c = 0.3$	$\tau_B = 5$, $c = 0.2$
C(III)	**39**, *112533.18*	**27**, *172102.85*	**1**, *147114.00*	**24**, *149006.5*
	$\tau_B = 15$	$\tau_B = 20$, $c = 0.1$	$\tau_B = 50$, $c = 0.1$	$\tau_B = 20$, $c = 0.1$
C(IV)	**5**, *227135.80*	**2**, *99259.00*	**0**	**2**, *154925.00*
	$\tau_B = 15$	$\tau_B = 25$, $c = 0.5$		$\tau_B = 15$, $c = 0.4$
C(V)	**2**, *353579.00*	**0**	**0**	**0**
	$\tau_B = 15$			

Tabelle 5 Ergebnisse der verschiedenen Mutationsarten in Kombination mit Hypermacromutation

9 Testanwendung 4

9.1 Problemstellung

Als zweiter Testfall für den erweiterten Algorithmus wir das 2D-HP Modell verwendet. Hierbei handelt es sich um eine Modell aus der Bio-Chemie, bei dem die zweidimensionale Struktur einer bestimmten Aminosäurenkette vorhergesagt werden soll. Gegeben ist die Reihenfolge der Aminosäuren. Man unterscheidet diese in 2 verschiedene Arten:

- Hydrophobe / Non Polar (H)
- Hydrophile / Polar (P)

Der räumliche Aufbau geht so vonstatten: Nachdem eine Aminosäure im Raster platziert wurde kann die nächste Aminosäure entweder in Bewegungsrichtung oder um 90° nach links oder nach rechts gedreht platziert werden. Als mögliche Lösungen ergeben sich also eine Kette der Länge (l-1) von Bewegungsrichtungen, die sich aus {F, L, R} zusammensetzten. Werden bei dieser Platzierung zwei H-Aminosäuren nebeneinander platziert, so ergibt sich daraus eine Energiebilanz von -1. Alle anderen Verbindungen sind neutral und werden mit 0 bewertet. Es gilt also die Zielsetzung: Maximieren der H-H Kontakte für eine gegebene Sequenz.

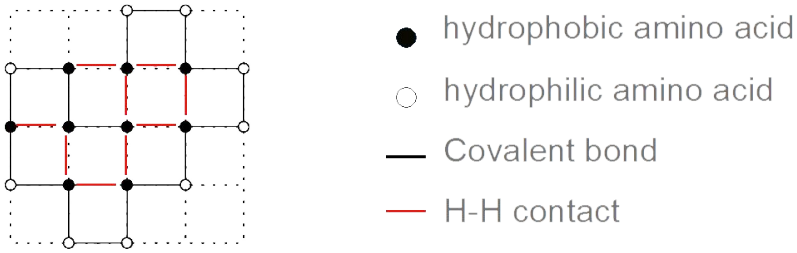


Abbildung 6 Beispiellösung für eine gegebene Sequenz aus H- und P- Säuren

Um die Möglichkeiten des Immun-Algorithmus zu testen wurden die ersten neun Instanzen des Tortilla 2D HP Benchmarks[2] benutzt.

2 http://www.cs.sandia.gov/tech reports/compbio/tortilla-hp-benchmarks.html

9.2 Beispiele

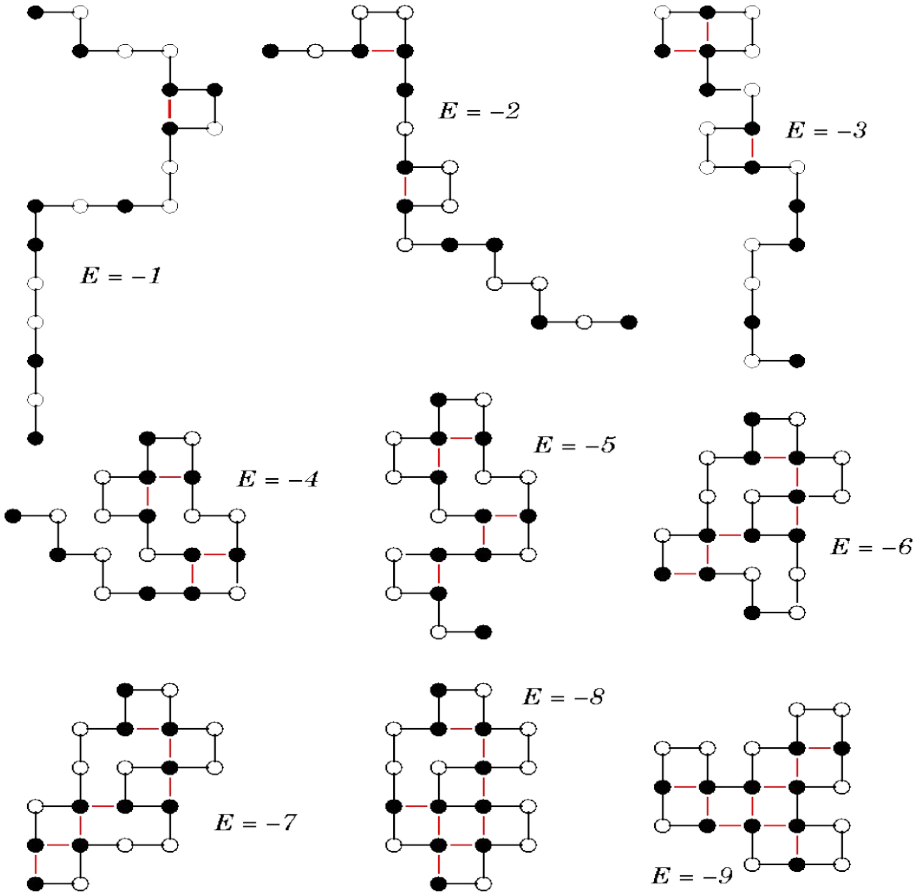


Abbildung 7 Beispiele verschiedener Konfigurationen der selben Sequenz

In Abbildung 7 kann man verschiedene Konfigurationen der selben Sequenz begutachten. Die sich daraus ergebenden Energiebilanzen sind jeweils angegeben.

9.3 Ergebnisse

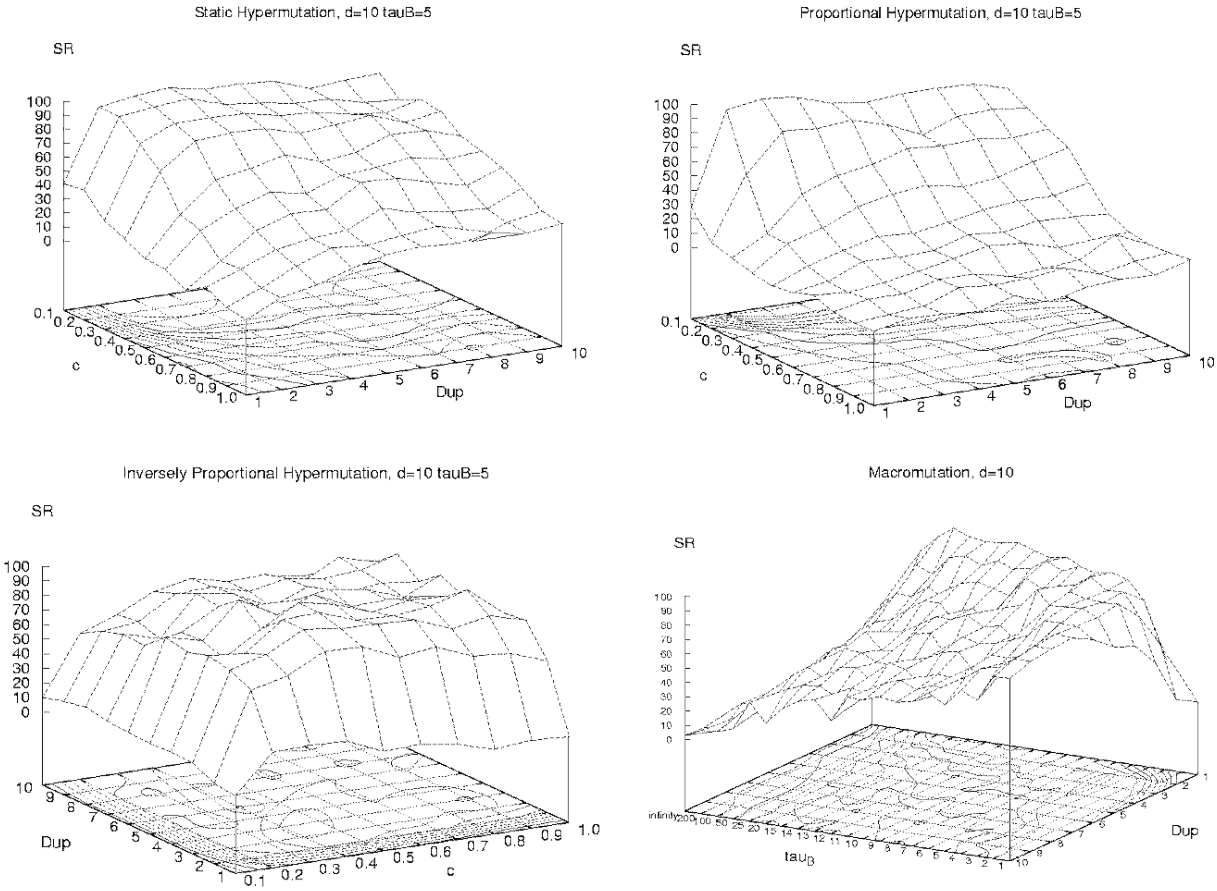


Abbildung 8 Graphische Darstellung der Erfolgsquote in Abhängigkeit der Parameter dup und c bzw. τ_B

In Abbildung 8 kann man erkennen das die statische Hypermutation und die proportionale Hypermutation einen ähnlichen Verlauf habe. Die proportionale Hypermutation hat aber eine stärker abfallende Kurve da gute Lösungen durch die stärkere Mutation leichter wieder kaputtgemacht werden.

Einen gegenteiligen Effekt sieht man bei der invers proportionalen Hypermutation. Hier gibt es nur schlechte Ergebnisse für eine geringe dup-Zahl und für ein niedriges c, bei dem der invers proportionale Effekt sehr wenig greifen kann. Sonst bildet die Grafik fast eine Ebene und man kann überall gute ergebnisse erwarten.

Bei der Macromutation ist eine schiefe Ebene zu erkennen. Kleinere Werte für dup und τ_B sind hier empfehlenswerter.

9.4 Vergleich

Sequence *E**		1 - 9	2 - 9	3 - 8	4 - 14	5 - 23	6 - 21	7 - 36	8 - 42	9 - 10
New ACO	(Hoos et al., 03)	- 9	- 9	- 8	- 14	- 23	- 21	- 36	- 42	
CG	(Dill et al.,96)	- 9	- 9	- 8	- 14	- 23	- 21	- 35	- 42	
Tabu Search	(Dill et al.,03)	- 9	- 9	- 8	- 14	- 23	- 21		- 42	
IA		**- 9**	**- 9**	**- 8**	**- 14**	**- 23**	**- 21**	**- 35**	**- 39**	**- 10**
EMC	(Liang et al.,01))	- 9	- 9	- 8	- 14	- 23	- 21	- 35	- 39	
PERM	(Hsu et al.,03)	- 9	- 9	- 8	- 14	- 23	- 21	- 36	- 38	
MMA	(Krasnogor et al.,02)	- 9		- 8	- 14	- 22	- 21		- 39	- 10
CI	(Toma et al., 96)	- 9	- 9	- 8	- 14	- 23	- 21	- 35		
GA	(Unger et al.,93)	- 9	- 9	- 8	- 14	- 22	- 21	- 34	- 37	
Pioneer Search	(Konig et al.,99)	- 9			- 14	- 23			- 37	
ACO + LS	(Hoos et al., 02)	- 9	- 9	- 8	- 14	- 23	- 21	- 34	- 32	- 10
SC	(Konig et al.,99)	- 9			- 14	- 22			- 34	
Metropolis MC	(Unger et al.,93)	- 9	- 9	- 8	- 13	- 20	- 21	- 33	- 35	
only LS	(Hoos et al., 02)	- 9	- 9	- 8	- 14	- 21	- 20	- 33	- 33	- 10
Multiple MC	(Unger et al.,93)	- 9	- 9	- 7	- 13	- 19	- 20	- 32	- 32	
SGA	(Konig et al.,99)	- 9			- 14	- 20			- 32	

d= 500 -1500

d=10

The reported energy values are the lowest obtained by each method.

Tabelle 6 Vergleich der Ergebnisse verschiedener Algorithmen

Vergleicht man die Ergebnisse des erweiterten Immun-Algorithmus mit den Ergebnissen anderer Algorithmen, so zeigt sich ein durchweg gutes Ergebnis. Die ermittelten Minimalenergieen stimmen immer mit den bereits bekannten überein oder verbessern diese sogar noch. Der vorgestellte Algorithmus verwendet dabei wesentlich geringere Populationsgrößen als andere Algorithmen.

10 Fazit

Anhand der vorgestellten Ergebnisse kann man sehen, das es prinzipiell möglich ist mit Immun- bzw. Evolutionären Algorithmen brauchbare Ergebnisse zu erzielen. Diese lassen sich durch vielfältige Erweiterungen (z.B. SAW, FCM, ...) oder Kombination verschiedener Techniken (z.B. Antiproportionale Hypermutation+Hypermacromutation) weiter steigern. Es bedarf jedoch einer starken Optimierung der verwendeten Parameter, das diese Algorithmen im Sinne von Rechenzeit und Speicherbedarf mit herkömmlichen Algorithmen mithalten können.

Die verfolgten Ansätze zeigen weitere Möglichkeiten für die Zukunft auf. Ob diese jedoch für die Praxis relevante Verfahren entwerfen können, bleibt abzuwarten.

Der Vergleich mit anderen Verfahren und Algorithmen ist sehr schwer, da meistens keine absoluten Vergleichswerte abgegeben werden, sondern nur vage Angaben, wie etwa die Anzahl der Evaluationen der Fitness-Funktion, die aber nur einen kleinen Teil des Algorithmus ausmacht.

11 Literaturverzeichnis

1. An Immunological Approach to Combinatorial Optimization Problems
   Vincenzo Cutello and Giuseppe Nicosia
   F.J. Garijo, J.C. Riquelme, and M. Toro (Eds.): IBERAMIA 2002, LNAI 2527, pp. 361–370, 2002.
   (c) Springer-Verlag Berlin Heidelberg 2002
2. Exploring the Capability of Immune Algorithms:
   A Characterization of Hypermutation Operators
   G. Nicosia et al. (Eds.): ICARIS 2004, LNCS 3239, pp. 263–276, 2004.
   (c) Springer-Verlag Berlin Heidelberg 2004
3. An Analysis of the Behaviour of Simplified Evolutionary Algorithms on Trap Functions
   Siegfried Nijssen, Thomas Bäck
4. Garey, M. R., Johnson, D. S.: Computers and Intractability: a Guide to the Theory of NP-completeness. New York: Freeman (1979).
5. Mitchell, D., Selman, B., Levesque, H. J.: Hard and easy distributions of SAT problems. Proc. of the AAAI. San Jose, CA (1992) 459–465.
6. Eiben, A. E., van der Hauw J. K., van Hemert J. I.: Graph coloring with adaptive evolutionary algorithms. J. of Heuristics 4 (1998) 25–46.
7. Bäck, T., Eiben, A. E., Vink, M. E.: A superior evolutionary algorithm for 3-SAT. Proc. of the 7th Annual Conference on Evolutionary Programming. Lecture Notes in Computer Science 1477 (1998) 125–136.
8. Nijssen S., Back, T.: An analysis of the Behavior of Simplified Evolutionary Algorithms on Trap Functions. IEEE Trans. on Evol. Comp., 7(1), pp.11-22, (2003)