Vortrag Proseminar Genetische Algorithmen: GENOCOP and GAFOC

Daniel Schiffner

7. August 2006

1 Ein evolutionäres Programm: GAFOC

Wir kehren zu den Dynamischen Kontroll Problemen zurück. Zur weiteren Notation: Sei $x = (x_0, \ldots, x_N)$ ein Vektor von Zuständen und $u = (u_0, \ldots, u_{N-1})$ ein Vektor von Kontrollen eines Systems.
Wir betrachten die Klasse $\mathcal{Z}$ von Optimierungsproblemen mit einer linearen Zustandsgleichen, die wir wie folgt formulieren können:

Optimimiere die Funktion $J(x, u)$

- Mit den Bedingungen:
  $x_{k+1} = a * x_k + b * u_k$ mit $k = 0, 1, \ldots, N-1$
- Und einige von diesen (oder alle) erfüllen folgende Bedingungen:
  1. Der letzte Zustand ist $x_N = C$, wobei C eine Konstante ist
  2. Die Grenzen aller Zustände: $\alpha \leq x_k \leq \beta,\ k = 1, \ldots, N$
  3. Die Grenzen aller Kontrollen: $\gamma \leq u_k \leq \delta,\ k = 0, 1, \ldots, N-1$
  4. Es gilt folgende Beziehung zwischen Zuständen und Kontrollen: $\tau x_k \leq u_k$ oder $\tau x_k \geq u_k,\ k = 0, 1, \ldots, N-1$

Für ein gegebenes Problem sind die Parameter a und b vorgeben, genauso wie der Ausgangszustand x_0 des Systems und die betreffende Funktion J.

Diese Formulierung ist allgemein genug und eine große Menge von Kontrollproblemen abzudecken.

Bei der „Numerical Optimization: Fine Local Tuning“ haben wir schon 2 Probleme getroffen, die in die Klasse $\mathcal{Z}$ gehören (linear-quadratic und harvest problem).

Das GAFOC (= **G**enetic **A**lgorithm **F**or **O**ptimal **C**ontrol problems) System wurde für die Lösung der Probleme der Klasse $\mathcal{Z}$ entworfen.

2 GENOCOP and GAFOC

Da jeder Zustand x_{k+1} eine (lineare) Funktion der vorhergehenden Zustände x_k und der Kontrollen u_k ist, hängt die Funktion $J(x, u)$ nur von u ab. Desweiteren können wir feststellen, dass jeder Zustand x_k nur von x_0 und $u_0, \ldots, u_{k-1}$ abhängt, z.B.:

$$x_k = a^k x_0 + b \sum_{i=0}^{k-1} a^i u_{k-1-i},\ (k = 1, 2, \ldots, N)$$

Somit können wir jedes Problem aus $\mathcal{Z}$ in ein Optimierungsproblem mit linearen Bedingungen transformieren.

Die Menge der von GAFOC lösbaren Probleme $\subset$ Die Menge der von GENOCOP lösbaren Probleme

Aber warum ein neues System, wenn das alte die Probleme auch lösen kann?

Im Buch sieht man den Kompromiss zwischen schwachen und starken Methoden zur Problemlösung. GA sind im allgemeinen zu schwach um nützlich bei den meisten Anwendungen zu sein. Eine stärkere Methode ist z.B. GENOCOP, der nur mit Funktionen arbeiten kann, die lineare Bedingungen haben. Dort haben wir gesehen, das GENOCOP wesentlich besser ist, als allgemeine GAs. Da GAFOC wieder spezifischer ist, kann davon ausgegangen werden, dass GAFOC GENOCOP in der Klasse $\mathcal{Z}$ schlagen wird. (Siehe auch Kapitel 9 mit GENETIC-2)

Ein weitere Vorteil ist, dass das System auch soweit verallgemeinert werden kann, dass es auch mit nicht-linearen Zustandsgleichungen funktioniert. Es ist also denkbar,

$$x_{k+1} = g(x_k, u_k),\ k = 0, 1, \ldots, N-1$$

zu definieren, wobei g nicht linear sein muss. Selbst wenn die obere Gleichung den Wert für u_k als eine Funktion von x_k und x_{k+1} ausgibt, z.B.:

$$u_k = h(x_k, x_{k+1}),\ k = 0, 1, \ldots, N-1$$

würde GAFOC nach einigen Modifikationen immer noch funktionieren.

Ein weiterer Grund ist, dass GAFOC zeigt, wie effektiv ein evulutionäres Programm, für ein gewisses Problem entwickelt, sein kann.

Kommentar:
Ein klassischer GA würde eine Population von möglichen Lösungen (Vektor u) haben. Während der Evaluierung würde er eine Folge von Zuständen (x) generieren, und würde Penalties auf die Zustände verteilen, die ausserhalb der Grenzen $[\alpha, \beta]$ liegen, oder andere Bedinungen verletzen. Diese Vorgehensweise würde wenig Chancen auf Erfolg haben!

3 Das GAFOC System

3.1 Repräsentation

Wie beim GENOCOP enthält GAFOC eine Population von Float Vektoren. Jeder Vektor $u = (u_0, u_1, \ldots, u_{N-1})$ ist ein Kontrollvektor für jedes Problem in $\mathcal{Z}$.

3.2 Anfangspopulation

Es können (wie bereits erwähnt) folgende Beschränkungen auftreten:

1. Der letzte Zustand ist $x_N = C$, wobei C eine Konstante ist
2. Die Grenzen aller Zustände: $\alpha \leq x_k \leq \beta,\ k = 1, \ldots, N$
3. Die Grenzen aller Kontrollen: $\gamma \leq u_k \leq \delta,\ k = 0, 1, \ldots, N-1$
4. Es gilt folgende Beziehung zwischen Zuständen und Kontrollen: $\tau x_k \leq u_k$ oder $\tau x_k \geq u_k,\ k = 0, 1, \ldots, N-1$

GAFOC initialisiert seine Population so, dass alle Bedingungen erfüllt werden. Der Algorithmus sieht wie folgt aus:

```
procedure initialization
begin
  success = FALSE
  status = TRUE
  if(τ x_k ≤ u_k) then left = TRUE else left = FALSE
  if(τ x_k ≥ u_k) then right = TRUE else right = FALSE
  if(x_N = C) then final = TRUE else final = FALSE
  k = 0
  repeat
    if left then lhs = τ x_k else lhs = −∞
    if right then rhs = τ x_k else rhs = ∞
    if(b > 0) then
      left-range = max{(α − a ∗ x_k)/b, γ, lhs}
      right-range = min{(β − a ∗ x_k)/b, δ, rhs}
    else
      left-range = max{(β − a ∗ x_k)/b, γ, lhs}
      right-range = min{(α − a ∗ x_k)/b, δ, rhs}
    if right-range − left-range< 0 then status = FALSE
    if status then generiere die k.te Komponente des Vektors u, so das gilt:
      left-range ≤ u_k ≤ right-range
    berechne x_{k+1} = a ∗ x_k + b ∗ u_k
    k = k + 1
  until k = N
  if final then u_{N−1} = (C − a ∗ x_{N−1})/b
  if (max{γ, lhs} ≤ u_{N−1} ≤ min{δ, rhs} AND status) then success =TRUE
end
```

Der Algorithmus ist in bestimmten Fällen nicht sehr gut, und hat eventuell eine schlechte Erfolgschance. Mit den folgenden Veränderungen wurden allerdings gute Ergebnisse erzielt: Die Berechnung der Komponenten des Vektors u wurden verändert:
Wähle beliebigen Bruchteil μ im Bereich $[0, 1]$ und zufälliges Bit b Wenn $b = 0$ dann wähle Komponente aus Bereich [left-range, r], sonst [r,right-range], wobei r =left-range+(right-range−left-range) $* \mu$.

Durch diese Veränderungen wurde eine Erfolgsrate für das linear-quadratic Problem von 100% erreicht, beim harvest Problem und anderen ist sie etwas niedriger.

3.3 Evaluierung

Zur Evaluierung von Problemen aus der Klasse $\mathcal{Z}$ benutzen wir die gegebene Funktion J. Deswegen ist $eval(u) = J(u, x)$

3.4 Genetische Operatoren

Die benutzten Operatoren wurden bereits erklärt (Beim „Fine Local Tuning“) Im folgenden werden nur der *arithmetical crossover* und der *simple crossover* nochmals erläutert.

- *arithmetical crossover*
  Der *arithmetical crossover* ist eine linear Kombination zweier (gültiger) Kontrollvektoren u^1 und u^2:
  $$u = a * u^1 + (1 - a) * u^2, \text{ mit } a \in [0, 1].$$
  Dieser produziert immer gültige Nachfahren.

- *simple crossover*
  Dieser würde nicht immer gültige Nachfahren produzieren, z.B:
  $$\begin{aligned} u'^1 = \quad & (u'^1_0, \ldots, u'^1_{N-1}) \\ & = (u^1_0, \ldots, u^1_k, u^2_{k+1}, \ldots, u^2_{N-1}) \text{ und} \\ u'^2 = \quad & (u'^2_0, \ldots, u'^2_{N-1}) \\ & = (u^2_0, \ldots, u^2_k, u^1_{k+1}, \ldots, u^1_{N-1}) \end{aligned}$$
  für ein $0 \leq k \leq N - 2$. Die letzte Komponente dieser Vektoren wird in
  $$u'^1_{N-1} = (C - a * x'^1_{N-1})/b \text{ und } u'^2_{N-1} = (C - a * x'^2_{N-1})/b$$
  geändert, wobei x'^1 und x'^2 die neuen Zustandsvektoren sind. Die Nachfahren u'^1 und u'^2 müssen die Bedingungen nicht erfüllen.
  Jedoch steigt die Erfolgsrate, je weiter das Programm kommt, da die Population gegen das Optimum konvergiert.

- *non-uniform mutation*
  Diese ist wiederum für das Feintuning verantwortlich

3.5 Parameter

Für die Experimente wurden folgende Einstellungen verwendet, im folgenden wird ein Beispiel gezeigt.

- $pop\_size = 70$
- Die Durchläufe wurden für jeweils für 40.000 Generationen gemacht
- $a = 0,25$ (für arithmetischen Crossover)
- Wahrscheinlichkeiten für Crossover: $p_{ac} = 0,15$ und $p_{sc} = 0,05$.

- Wahrscheinlichkeit für Mutation: $p_{nm} = 0,10$
- Funktion:

$$\max \sum_{k=0}^{N-1} q^k * u_k^{1-v} + q^N * s * x_N^{1-v}$$

- In Abhängigkeit von:

$$x_{k+1} = a(x_k - u_k),\ \text{für } k = 0, 1, \ldots, N-1$$

- Und den Bedingungen:

$$x_k \geq 0,\ \text{für } k = 0, 1, \ldots, N,$$
$$u_k \geq 0,\ \text{für } k = 0, 1, \ldots, N-1 \text{ und}$$
$$u_k \leq x_k,\ \text{für } k = 0, 1, \ldots, N-1 (right = \text{TRUE}, left = \text{FALSE}, \tau = 1)$$

- Der beste Kontrollvektor lässt sich wie folgt berechnen:

$$\hat{u}_k = x_k / A_k^{1/v},\ \text{wobei } A_k^{1/v} \text{ folgende Gleichung erfüllt:}$$

$$A_k^{1/v} = 1 + \gamma * A_{k+1}^{1/v} \text{ und es gilt :}$$
$$A_N = s \text{ und } \gamma = (q * a^{1-v})^{1-v}$$

- Der optimale Wert der Funktion ist dann

$$J^* = A_0 x_0^{1-v}$$

  Man kann dann zeigen, dass für $N \to \infty$ gilt:

$$\hat{u}^k \to (1-\gamma) x_k, \quad \text{und}$$

$$J^* \to x_0^{1-v} / (1-\gamma)^v$$

Die Ergebnisse wurden mit folgenden Parameterwerten erzielt:

- $a = 1,02$
- $q = 0,95$
- $s = 0,5$
- $v = 0,5$
- $x_0 = 1$
- $N = 5, 10, 20, 45$

Die Initialisierung hatte eine Erfolgsrate von 100% für das obige Problem. Nach 10.000 Generationen lieferte das GAFOC System ein exaktes Ergebnis (auf 6 Stellen genau).

N	Berechnete Lösung	GAFOC	Fehler
5	2,105094	2,105094	0,000%
10	2,882455	2,882455	0,000%
20	3,198550	3,198550	0,000%
45	3,505587	3,505587	0,000%

4 Implementation von GENOCOP - Erste Versuche und Ergebnisse

Die Version 1.0 von GENOCOP wurde im Sommer 1992 implemententiert. Um diesen zu testen wurden einige Testprobleme zusammengetragen (dazu gehören quadratische, nicht-lineare, und unstetige Funktionen mit mehreren linearen Bedingungen). Alle Versuche wurden auf einer SUN SPARC station 2 durchgeführt. Für alle Versuche wurden folgende Parameter verwendet:

- $pop\_size = 70$
- $k = 28$ (Anzahl der Eltern pro Generation)
- $b = 2$ (Koeffizient der non-uniform Mutation)
- $a = 0,25$ (Parameter des arithmetischen Crossover)

Alle Versuche wurden 10mal hintereinander durchgeführt. Für die meisten Probleme wurden 500 oder 1000 Generationen verwendet (Gewisse Probleme benötigten eine größere Anzahl von Iterationen). Einige ausgesuchte Ergebnisse werden im folgenden präsentiert:

4.1 Test #1

- Problem

$$\text{minimiere } f(\bar{X}, y) = -10,5x_1 - 7,5x_2 - 3,5x_3 - 2,5x_4 - 1,5x_5 - 10y - 0,5\sum_{i=1}^{5} x_i^2$$

- In Abhängigkeit von

$$\begin{aligned} 6x_1 + 3x_2 + 3x_3 + 2x_4 + x_5 \leq 6,5, &\quad 10x_1 + 10x_3 + y \leq 20, \\ 0 \leq x_i \leq 1, &\quad 0 \leq y. \end{aligned}$$

- Die globale Lösung ist $(\bar{X}^*, y^*) = (0, 1, 0, 1, 1, 20)$, und $f(\bar{X}^*, y^*) = -213$
- GENOCOP fand in allen 10 Läufen eine Lösung nah der Optimalen, eine typische war:

$$(0.000055, 0.999998, 0.000041, 0.999989, 1.000000, 19.999033)$$

$$f(\bar{X}, y) = -212.990997$$

- Eine Berechnung mit 1000 Iterationen benötigte 29 Sekunden CPU-Zeit

4.2 Test #2

- Problem

$$\text{minimiere } f(\bar{X}) = \sum_{j=1}^{10} x_j (c_j + \ln \frac{x_j}{x_1 + \ldots + x_{10}})$$

- In Abhängigkeit von

$$\begin{aligned} x_1 + 2x_2 + 2x_3 + x_6 + x_{10} = 2, &\quad x_4 + 2x_5 + x_6 + x_7 = 1, \\ x_3 + x_7 + x_8 + 2x_9 + x_{10} = 1, &\quad x_i \geq 0.000001, (i = 1, \ldots, 10), \end{aligned}$$

  und

$$c_1 = -6.089;\ c_2 = -17.164;\ c_3 = -34.054;\ c_4 = -5.914;$$

$$c_5 = -24.721;\ c_6 = -14.986;\ c_7 = -24.100;\ c_8 = -10.708;$$

$$c_9 = -26.662;\ c_{10} = -22.179;$$

- Die bisher beste Lösung war:

$$(\bar{X}^*) = (.01773548, .08200180, .8825646, .0007233256, .4907851, .0004335469,\\ .01727298, .007765639, .01984929, .05269826)$$

und $f(\bar{X}^*) = -47.707579$

- GENOCOP fand einen besseren Wert als den obigen (in allen 10 Durchläufen):

$$(\bar{X}^*) = (.04034785, .15386976, .77497089, .00167479, .48468539,\\ .00068965, .02826479, .01849179, .03849563, .10128126)$$

und $f(\bar{X}^*) = -47.750765$

- Eine Berechnung mit 500 Iterationen benötigte 11 Sekunden CPU-Zeit

4.3 Test #3

- Problem:

$$\text{minimiere } f(\bar{X}) = \begin{cases} f_1 = x_2 + 10^{-1}(x_2 - x_1)^2 - 1.0 & \text{, für} 0 \leq x_1 \leq 2 \\ f_2 = \frac{1}{27\sqrt{3}}((x_1 - 3^2 - 9)x_2^3 & \text{, für} 2 \leq x_1 \leq 4 \\ f_3 = \frac{1}{3}(x_1 - 2)^3 + x_2 - \frac{11}{3} & \text{, für} 4 \leq x_1 \leq 6 \end{cases}$$

- In Abhängigkeit von:

$$x_1/\sqrt{3} - x_2 \geq 0,\\ -x_1 - \sqrt{3}x_2 + 6 \geq 6,\\ 0 \leq x_1 \leq 6, \text{ und } x_2 \geq 0.$$

- Die Funktion besitzt 3 Lösungen:

$$\bar{X}_1^* = (0,0),\ \bar{X}_2^* = (3, \sqrt{3}),\ \text{und} \bar{X}_3^* = (4,0)$$

in allen Fällen ist: $f(\bar{X}_i^*) = -1$ $(i = 1, 2, 3)$

- Um hier eine Lösung zu finden wurde wie folgt vorgegangen:
Es wurden 3 unterschiedliche Experimente durchgeführt. Im Experiment k $(k = 1, 2, 3)$ wurden alle f_i um 0.5 erhöht, ausser f_k. Somit ist für das erste Experiment $\bar{X}_1^* = (0, 0)$, für das zweite $\bar{X}_2^* = (3, \sqrt{3})$ und $\bar{X}_3^* = (4, 0)$ für das dritte die optimale Lösung. So konnte GENOCOP alle Optima in jedem Durchlauf finden.

- Eine Berechnung mit 500 Iterationen benötigte 9 Sekunden CPU-Zeit.

4.4 Zusammenfassung

Die Ergebnisse zeigen einige Eigenschaften der GENOCOP Methode. Zum einen hat GENOCOP immer das Optimum erreicht, im Test #2 sogar ein noch besseres Ergebnis berrechnet. Diese GA-basierte Methode arbeitet direkt mit den linearen Bedingungen, man muss also keine *penalty* Parameter für jedes Problem zuweisen. Das System wird erweitert um Integer und Bool'sche Werte zu verarbeiten. Auch werden die festen Wahrscheinlichkeiten für alle Reproduktions-Operatoren durch adaptive ersetzt. Auch ist GENOCOP die Grundlage des GENOCOP II Systems, welches *jedes* Set von nicht-linearen Bedingungen (Gleichungen und Ungleichungen) lösen kann.

5 Modifikationen von GENOCOP

Für die Version 2.0 wurden einige Änderungen an den genetischen Operatoren gemacht. Im folgenden werden nochmal alle Operatoren besprochen:

- *uniforme mutation* und *boundary mutation* wurden nicht verändert.
  Um den Nutzen der boundary mutation zu zeigen, wurde folgendes Testscenario durchgeführt:
  - Problem:
    $$\text{maximiere } f(x_1, x_2) = 4x_1 + 3x_2$$
  - In Abhängigkeit von:
    $$\begin{aligned} &2x_1 + 3x_2 \leq 6; \\ &-3x_1 + 2x_2 \leq 3; \\ &2x_1 + x_2 \leq 4 \text{ und} \\ &0 \leq x_i \leq 2,\ i = 1, 2 \end{aligned}$$
  - Das bekannte Optimum ist $(x_1, x_2) = (1.5, 1.0)$ und $f(1.5, 1.0) = 9.0$.
  - Es wurden jeweils 10 Durchläufe durchgeführt, 10 mit *boundary mutation*, 10 ohne. Mit *boundary mutation* wurde jedes mal das Optimum entdeckt, und zwar innerhalb von durchschnittlich 32 Generationen. Ohne war nach 100 Generationen der *beste* Punkt $x = (1.501, 0.997)$ und $f(x) = 8.996$ (Der schlechteste war $x = (1.576, 0.847)$ mit $f(x) = 8.803$).
- *non-uniform mutation* wurde durch Ändern der Funktion Δ modifiziert. Die neue Version ist
  $$\Delta(t, y) = y * r * (1 - \frac{t}{T})^b,$$
  wobei r eine beliebige Zahl aus $[0, 1]$ ist, T die maximale Anzahl von Generationen, und b ein Parameter. GENOCOP hatte die *non-uniform Mutation* von „links nach rechts“ angewandt, der neue generiert eine randomisierte Anordnung der Vektorkomponenten. Der Operator hat sich in den meisten Anwendungen als sehr nützlich erwiesen,z.B.:
  - Problem:
    $$\text{maximiere } f(x_1, x_2, x_3, x_4, x_5) = \begin{array}{l} (x_1 - 1.0)^2 + (x_1 - x_2^2)^2 + (x_2 - 1.0)^2 + \\ +(x_1 - x_3^2)^2 + (x_3 - 1.0)^2 + (x_1 - x_4^2)^2 + \\ +(x_4 - 1.0)^2 + (x_1 - x_5^2)^2 + (x_5 - 1.0)^2, \end{array}$$
  - In Abhängigkeit von
    $$0 \leq x_i \leq 10,\ i = 1, \ldots, 5$$
  - Das globale Optimum ist $(x_1, x_2, x_3, x_4, x_5) = (1, 1, 1, 1, 1)$ mit $f(1, 1, 1, 1, 1) = 0$.
  - Mit verschiedenen Durchläufen wurde folgendes Ergebnis geliefert:

Generation Number	Beste Evaluation mit	Beste Evaluation ohne
50	1.85312903	7.45429516
100	0.80920660	0.44312307
500	0.00698480	0.06966695
1000	0.00000000	0.01593169
5000	0.00000000	0.00155015

  Mit Operator wurde das globale Optimum gefunden, ohne war der beste Punkt
  $$(1.02938247, 1.01172984, 1.01243937, 1.01172805, 1.01172495).$$
  Auch ist die neue Version von Δ wesentlich effektiver als die bisherige.

- *simple crossover* wurde fast unverändert gelassen. Anstelle den größten Wert a mit einer binären Suche zu finden, wurde jetzt ausgehend vom Wert $a = 1$, dieser Wert um den Faktor $\frac{1}{q}$ verringert, falls mindestens ein Nachkomme nicht gültig war. Nach q Versuchen muss es ein gültigen Nachfahren geben, da diese identisch mit den Eltern sind. Der Wert a wird jedes mal neu berechnet, allerdings ist ein maximales Absteigen nicht oft notwendig und die Wahrscheinlichkeit dafür sinkt auch stark, je weiter sich die Population entwickelt.

- *single-arithmetical crossover* wurde enfernt, da er keinen Nutzen für das System hatte.

- *arithmetical crossover* ist der selbe wie „whole arithmetical crossover" mit einer Veränderung. Der originale Operator arbeitete mit einem festen Wert $a = 0.25$, der neue generiert einen zufälligen Wert aus $[0, 1]$ wenn er aufgerufen wird. Die Wichtigkeit dieses Operators wird am folgenden Beispiel deutlich:

  – Problem:

  $$\begin{aligned}\text{minimiere } f(x_1, x_2, x_3, x_4, x_5) = \quad & -5\sin(x_1)\sin(x_2)\sin(x_3)\sin(x_4)\sin(x_5)+ \\ & -\sin(5x_1)\sin(5x_2)\sin(5x_3)\sin(5x_4)\sin(5x_5),\end{aligned}$$

  – In Abhänigkeit von:

  $$0 \le x_i \le \pi, \text{ für } 1 \le i \le 5.$$

  – Die bekannte Lösung ist $(x_1, x_2, x_3, x_4, x_5) = (\pi/2, \pi/2, \pi/2, \pi/2, \pi/2)$ und $f(\pi/2, \pi/2, \pi/2, \pi/2, \pi/2) = -6$.

  – Es scheint, dass das System ohne *arithmetical crossover* eine langsamere Konvergenz hat, wie man anhand der Tabelle leicht erkennt:

Generationen	Ohne AC	Mit AC
50	-5.9814	-5.9930
100	-5.9966	-5.9996

  Auch ist das neue System stabiler (Abweichung vom besten Ergebnis in 10 Läufen), und schneller als die ursprüngliche Implementation.

- *heuristic crossover* wurde eingeführt. Dieser Operator ist recht einzigartig, denn

  1. er benutzt die Werte der Zielfunktion um die Richtung der Suche zu bestimmen
  2. er produziert nur einen Nachfahren, oder
  3. produziert überhaupt keinen Nachfahren.

  Dazu benutzt er folgende Produktionsregel:

  $$x_3 = r * (x_2 - x_1) + x_2,$$

  wobei x_1 und x_2 die Eltern sind, r ein zufälliger Wert aus $[0, 1]$ und x_2 ist besser als x_1, also $f(x_2) \ge f(x_1)$ für Maximierungsprobleme. Es ist möglich, das der Operator einen ungülten Nachfahren produziert. Er generiert dann einen neuen Wert für r und versucht es erneut. Nach w Fehlschlägen bricht er ab. In einem Beispiel wird der Nutzen des Operators deutlich:

  – Problem:

  $$\begin{aligned}\text{minimiere } f(\bar{X}) = \quad & 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2+ \\ & +(1 - x_3)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2))+ \\ & +19.8(x_2 - 1)(x_4 - 1),\end{aligned}$$

  – In Abhängigkeit von:

  $$-10.0 \le x_i \le 10.0, \; i = 1, 2, 3, 4.$$

  – Die bekannte Lösung ist $\bar{X}^* = (1, 1, 1, 1)$ und $f(\bar{X}^*) = 0$.

– Das System ohne diesen Operator generierte eine Lösung in 10.000 Generationen, in der jede Variable innerhalb von 10% der optimalen Lösung war. Eine typische Lösung war:

$$(x_1, x_2, x_3, x_4) = (0.95534354, 0.91261518, 1.04239142, 1.08687556),$$

mit $f(x_1, x_2, x_3, x_4) = 0.00683880$. Mit *heuristic crossover* war das System wesentlich besser, ein typisches Ergebnis war:

$$(x_1, x_2, x_3, x_4) = (1.000581, 1.001166, 0.999441, 0.998879),$$

mit $f(x_1, x_2, x_3, x_4) = 0.0000012$.

Es scheint, dass der *heuristic crossover* die Genauigkeit der Lösung erhöht. Seine Hauptaufgaben sind:

1. Fine Local Tuning
2. Suche in die vielversprechende Richtung

Alle bisher genannten Operatoren wurden in die neue Version übertragen. Zum Vergleich ein letztes Beispiel auf Basis der letzten genannten Funktion: GENOCOP Ver(1.0) fand nach 1.000.000 (!) Generationen die Lösung:

$$(x_1, x_2, x_3, x_4) = (0.983055, 0.966272, 1.016511, 1.033368)$$

mit $f(x_1, x_2, x_3, x_4) = 0.001013$. Die neue Version mit den neuen Operatoren fand nach 10.000 Generationen:

$$(x_1, x_2, x_3, x_4) = (1.000581, 1.001166, 0.999441, 0.998879)$$

mit $f(x_1, x_2, x_3, x_4) = 0.0000012$.

Weitere Möglichkeit wäre auch die Anpassung der Wahrscheinlichkeiten / Häufigkeiten der Operatoren. Am besten wäre ein adaptives System, wie es die „Evolution Strategies“ verfolgen.

6 Nichtlineare Optimierung: GENOCOP II

6.1 Grundproblem

Das Problem der Nichtlinearen Optimierung $\mathcal{NPP}$ kann wie folgt definiert werden:

- Finde $\bar{X}$ so dass:
- $$\text{optimiere } f(\bar{X}), \bar{X} = (x_1, \ldots, x_q) \in \mathrm{R}^q$$
- In Abhängigkeit von $p \geq 0$ Gleichungen der Form:
$$c_i(\bar{X}) = 0,\ i = 0, \ldots, p$$
- und $m - p \geq 0$ Ungleichungen der Form:
$$c_i(\bar{X}) \leq 0,\ i = p + 1, \ldots, m.$$

Im folgenden werden wir ein System (GENOCOP II) präsentieren, dass diese Gleichungen lösen kann. Um Verwechslungen zu vermeiden, werden wir den Ur-GENOCOP als GENOCOP I bezeichnen. Die analytische Grundlage ist, dass die Funktion und die Bedingungen zweimal stetig ableitbar sind, so dass dieses Problem aus $\mathcal{NPP}$ in eine Folge von lösbaren Subproblemen wird.

Zur Lösung der Probleme wurden in den letzten Jahren einige Fortschritte gemacht und einige Vorgehensweisen entwickelt, unter denen folgende zu finden sind:

- sequential quadratic penalty function
- recursive quadratic programming method
- penalty trajectory methods
- SOLVER method

Für GENOCOP II wurde erste („sequential quadratic penalty function“) Methode als Grundlage verwendet. Diese Methode ersetzt das Ur-Problem durch das Problem $\mathcal{NPP}'$:

$$\text{optimiere } F(\bar{X}, r) = f(\bar{X}) + \frac{1}{2r}\bar{C}^T\bar{C},$$

wobei $r > 0$ und $\bar{C}$ der Vektor aller aktiven Bedingungen $c_1, \ldots, c_l$. Es wurde gezeigt, dass die Lösungen aus $\mathcal{NPP}$ gleich denen von $\mathcal{NPP}'$ sind, wenn $r \to 0$ geht. Ein erster Lösungsansatz war die Minimierung von $F(\bar{X}, r)$ für eine abnehmende Folge von r. Dies funktionierte allerdings nicht, da die Probleme für $r \to 0$ die Probleme zu unhandlich wurden. Es wurde eine andere Lösung gefunden, mit dem Vorteil, das sie im Gegensatz zu anderen Lösungsmöglichkeiten bei der Berechnung der Suchrichtung nicht Abhängig von den Bedingungen ist und eine Methode zur Verfügung stellt, die Werte von r zu berechnen. Mit genannter Methode und GENOCOP I ist es nun möglich, das neue System zu implementieren:

```
procedure GENOCOP II
begin
  t ← 0
  split the set of constraints C into
   C = L ∪ N_e ∪ N_i
  select a starting point X̄_s
  set the set of active constraints, A to
   A ← N_e ∪ V
  set penalty τ ← τ_0
  while( not termination condition) do
  begin
    t ← t+1
    execute GENOCOP I for the function
      F(X̄,r) = f(X̄) + 1 (1/2r) ĀᵀĀ
      with linear constraints L
      and starting point X̄_s
    save best individual X̄*:
      X̄_s ← X̄*
    update A:
      A ← A − S ∪ V,
    decrease penalty τ
      τ ← g(τ,t)
  end
end
```

- Die Bedingungen C werden in drei Teilmengen zerlegt: lineare Bedingungen L, nicht-lineare Gleichungen N_e, und nicht-lineare Ungleichungen N_i
- Startpunkt $\bar{X}_s$ wird ausgewählt (muss nicht zulässig sein) oder vom Benutzer eingegeben.
- A besteht anfänglich aus Elementen von N_e und $V \subset N_i$ der verletzten Bedingungen aus N_i. Eine Bedingung zählt als verletzt bei dem Punkt $\bar{X}$ genau dann, wenn $c_j(\bar{X}) > \delta$ ($j = p{+}1, \ldots, m$), wobei δ ein Parameter der Funktion ist. Zuletzt wird τ auf τ_0 gesetzt (Parameter der Funktion).

- In der Hauptschleife wird GENOCOP I aufgefrufen, der die Funktion optimieren soll. Weiterhin werden die linearen Bedingungen L übergeben. Zu beachten ist, das GENOCOP I mit *pop_size gleichen* Elementen startet.

- Danach wird das beste Ergebnis gespeichert und A aktualisiert, wobei S (Satisfied) und V Teilmengen von N_i sind.

Der Ablauf kann an folgendem Beispiel illustriert werden:

- Problem:
$$\text{minimiere } f(\bar{X}) = x_1 * x_2^2,$$

- In Abhängigkeit von einer nichtlinearen Bedingung:
$$c_1 : 2 - x_1^2 - x_2^2 \geq 0.$$

- Die bekannte Lösung ist: $\bar{X}^* = (-0.816497, -1.154701)$, und $f(\bar{X}^*) = -1.088662$

- GENOCOP II läuft nun wie folgt:

  1. Startpunkt ist $\bar{X}_0 = (-0.99, -0.99)$
  2. Nach der ersten Iteration ist A leer und das System konvergierte zu
     $\bar{X}_1 = (-1.5, -1.5),\ f(\bar{X}_1) = -3.375$
  3. Dieser Punkt verletzt c_1 und diese wird aktiv. $\bar{X}_1$ wird als Startpunkt für die 2.te Iteration benutzt.
  4. Die zweite Iteration mit $\tau = 10^{-1}, A = \{c_1\}$ ergab
     $\bar{X}_2 = (-0.831595, -1.179690),\ f(\bar{X}_2) = -1.122678$
  5. Mit $\bar{X}_2$ als Startpunkt liefert die 3.te Iteration mit $\tau = 10^{-2}, A = \{c_1\}$
     $\bar{X}_3 = (-0.815862, -1.158801),\ f(\bar{X}_3) = -1.09985$
  6. Die weiteren Punkte $\bar{X}_i$ mit $i = 4, 5, \ldots$ näherten sich dem Optimum weiter an

Um GENOCOP II weiter zu testen, wurden einige ausgewählte Probleme betrachtet. Die Fälle schließen quadratische, nicht-lineare und unstetige Funktionen mit mehreren nicht-linearen Bedingungen mit ein. Hier werden nur ein paar ausgesuchte Probleme beschrieben.

Alle Tests wurden auf einer SUN SPARC station 2 durchgeführt. Dabei wurden folgende Parameter verwendet:

- $pop\_size = 70$
- $k = 28$ (Anzahl der Eltern pro Generation)
- $b = 6$ (Koeffizient der non-uniform mutation)
- $\delta = 0.01$ (Parameter für Entscheidung, ob Bedingung aktiv oder nicht)
- In den meisten Fällen gilt für den penalty-Koeffizient: $\tau_0 = 1$, also $(g(\tau, 0) = 1)$ und $g(\tau, t) = 10^{-1} * g(\tau, t-1)$
- GENOCOP I wurde mit 1000 Generationen aufgerufen (für schwere Probleme wurden evtl. mehr benötigt)

6.2 Test #1

- Problem:

$$\text{minimiere } f(\bar{X}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2,$$

- In Abhängigkeit von:

$$\begin{array}{lcl} c_1 & : & x_1 + x_2^2 \geq 0, \\ c_2 & : & x_1^2 + x_2 \geq 0, \end{array}$$

  Und den Grenzen:

$$-0.5 \leq x_1 \leq 0.5, \text{ und } x_2 \leq 1.0.$$

- Die bekannte Lösung ist $\bar{X}^* = (0.5, 0.25)$ und $f(\bar{X}^*) = 0.25$
- Ausgangspunkt ist $\bar{X}_0 = (0, 0)$
- GENOCOP II fand das Optimum nach 1(!) Iteration, da keine Bedingung beim Optimum aktiv war

6.3 Test #2

- Problem:

$$\text{minimiere } f(x, y) = -x - y,$$

- In Abhängigkeit von:

$$\begin{array}{l} c_1 : y \leq 2x^4 - 8x^3 + 8x^2 + 2, \\ c_2 : y \leq 4x^4 - 32x^3 + 88x^2 - 96x + 36, \end{array}$$

  Und den Grenzen:

$$0 \leq x \leq 3 \text{ und } 0 \leq y \leq 4.$$

- Die bekannte Lösung ist $\bar{X}^* = (2.3295, 3.1783)$ und $f(\bar{X}^*) = -5.5079$.
- Ausgangspunkt ist $\bar{X}_0 = (0, 0)$
- GENOCOP II kam nach der 4.ten Iteration dem Optimum sehr nahe.

Iteration #	Bester Punkt	Aktive Bedingungen
0	(0,0)	keine
1	(3,4)	c_2
2	(2.06,3.98)	c_1, c_2
3	(2.3298,3.1839)	c_1, c_2
4	(2.3295,3.1790)	c_1, c_2

6.4 Test #3

- Problem:

$$\text{minimiere } f(\bar{X}) = (x_1 - 2)^2 + (x_2 - 1)^2$$

- In Abhängigkeit von

$$c_1 : -x_1^2 + x_2 \geq 0$$

  und

$$x_1 + x_2 \leq 2$$

- Die Lösung ist $\bar{X}^* = (1, 1)$ und $f(\bar{X}^*) = 1$

- Der gültige Startpunkt ist $\bar{X}_0 = (0, 0)$
- GENOCOP II kam der Lösung nach der 6.ten Iteration sehr nahe

Iteration #	Bester Punkt	Aktive Bedingungen
0	(0,0)	c_1
1	(1.496072,0.503928)	c_1
2	(1.020873,0.979127)	c_1
3	(1.013524,0.986476)	c_1
4	(1.002243,0.997757)	c_1
5	(1.000217,0.999442)	c_1
6	(1.000029,0.999971)	c_1

6.5 Folgerungen

Vorteile der Methode:

- Keine Berechnung des Gradienten oder der Hesseschen Matrix
- Es kann jeder GA anstelle von GENOCOP I verwendet werden, allerdings sollten alle Bedingungen für die aktiven Bedingungen in Betracht gezogen werden, weshalb GENOCOP I darin schneller ist, da die linearen Bedingungen extra betrachtet werden.

Auch geht die Suche nach weiteren Paradigmen bezüglich der Bedingungen weiter. Es gibt zwei nennenswerte Methoden:

- Die erste handelt in zwei Phasen:
  1. Die Population entwickelt sich mit einem Standard-GA, in der die Fitness-Funktion von der Bedingungserfüllung abhängt
  2. Die Endpopulation der vorherigen Phase (gesehen als Speicher mit Informationen über die Bedingungen) wird als Anfangspopulation des GAs benutzt mit der Zielfunktion als Fitness-Funktion, welche 0 Fitness vergibt, wenn eine Bedingung verletzt wird.
- Die zweite weist allen gültigen Punkten einen höheren Fitnesswert zu als ungültigen, womit diese natürlich bevorzugt werden.

Auch wird untersucht, ob sich GAs für „integer programming" Probleme benutzen lassen. In der vorgeschlagenen Methode werden die Bedingungen durch eine nichtlineare penalty Funktion entschärft. Das betrachtete Problem ist:

$$\text{minimiere } \mathbf{cx}$$

In Abhängigkeit von

$$A\mathbf{x} - \mathbf{b} \geq 0, \tag{1}$$

$$\sum_{j=1}^{n_i} x_{ij} = 1 \text{ für} i = 1, \ldots, m, \tag{2}$$

$$x_{ij} \in \{0, 1\}. \tag{3}$$

Für jede Menge, $i = 1, \ldots, m$ fordern die Gleichungen (2) und (3), dass genau eine Variable in $\{x_{ij}\}_{j=1}^{n_i}$ gleich 1 ist. Matrix A ist $k \times n$ ($n = \sum i = 1^m n_i$) und $\mathbf{b}$ ist ein konstante k-dimensionaler Vektor. Die meisten Methoden für die Lösung solcher Probleme benutzen entweder lineare Programmierung, LaGrange Relaxation oder Varianten dieser. LaGrange Relaxation verwirft ein paar Bedingungen indem eine gewichtete lineare Penalty für Bedingungsverletzung eingeführt wird. Die

„korrekten" Gewichte können in guten Grenzen oder sogar optimalen Lösungen resultieren. Dabei wird das Originalproblem (1) durch die Formulierung

$$\text{minimiere } \mathbf{cx} - \lambda(Ax - \mathbf{b})$$

in Abhängigkeit von

$$Ex = e_m, x_{ij} \in \{0,1\}$$

ersetzt.($Ex = e_m$ sind die Bedingungen (multiple choice) und e_m ist ein Vektor von Einsen). Die vorgeschlagene Methode ersetzt nun das Problem durch:

$$\text{minimiere } \mathbf{cx} + p_\lambda(x)$$

in Abhängigkeit von

$$Ex = e_m, x_{ij} \in \{0,1\}$$

wobei $p_\lambda(x) = \sum_{i=1}^{k} \lambda_i[\min\{0, A_i x - b_i\}]^2$. Diese Funktion wird durch einen GA optimiert. Es gibt einige interessante Ideen in diesem GA. Zuerst zeigten die Versuche, dass ausgehend von großen Werten für λ kein effizienter Algorithmus zu finden war. Deshalb passt der Algorithmus den Wert für λ während des Laufes mit einer Sequenz an. Ähnlich wie bei GENOCOP II ist eine langsame Rate gut für die Genauigkeit auf Kosten der Geschwindigkeit und eine schnelle Rate kann zu ungenauen Ergebnissen führen. Zusätzlich benutzt der erwähnte Algorithmus Zufallsschlüssel, wobei die Lösung ein Vektor von Zufallszahlen ist. Ihre Reihenfolge decodiert die Lösung (Vergleiche dazu die „evolution strategies" und das TSP). Eine Lösung wird als String der Länge gleich der Anzahl der „multiple choice" Mengen. Jede Position i kann jeden Wert aus $\{1, \ldots, n_i\}$ annehmen, wobei n_i die Anzahl der Variablen der Menge ist.