Proseminar Genetische Algorithmen: GENOCOP and GAFOC

Daniel Schiffner

7. August 2006

Optimierungsprobleme - Definition

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 linearen Zustandsgleichungen.

Optimierungsprobleme - Definition 2

- 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 $\tau x_k \geq u_k,\ k = 0, 1, \ldots, N-1$

GAFOC

- 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.

- 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.
- Jeder Zustand hängt x_k nur von x_0 und $u_0, \ldots, u_{k-1}$ ab, z.B.:

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

- Jedes Problem aus $\mathcal{Z}$ kann in ein Optimierungsproblem mit linearen Bedingungen transformiert werden.

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

Warum neues System?

- GAFOC stärkere Methode somit sind bessere Ergebnisse zu erwarten.
- Ebenso mit GENETIC-2 (Kapitel 9)
- Spezifische Implementation hat höhere Effektivität
- Kann weiter verallgemeinert werden, so dass es auch mit nicht-linearen Gleichungen funktioniert

Das GAFOC System

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 $\mathcal{Z}$.

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:

Anfangspopulation - Algorithmus

```
1  procedure initialization
2  begin
3    success = FALSE
4    status = TRUE
5    if(τ x_k ≤ u_k) then left = TRUE else left = FALSE
6    if(τ x_k ≥ u_k) then right = TRUE else right = FALSE
7    if(x_N = C) then final = TRUE else final = FALSE
8    k = 0
9    repeat
10     if left then lhs = τ x_k else lhs = −∞
11     if right then rhs = τ x_k else rhs = ∞
12     if(b > 0) then
13       left-range = max{(α − a ∗ x_k)/b, γ, lhs}
14       right-range = min{(β − a ∗ x_k)/b, δ, rhs}
15     else
16       left-range = max{(β − a ∗ x_k)/b, γ, lhs}
17       right-range = min{(α − a ∗ x_k)/b, δ, rhs}
18     if right-range − left-range< 0 then status = FALSE
19     if status then generiere die k.te Komponente des Vektors u, so das gilt:
20       left-range ≤ u_k ≤ right-range
21     berechne x_{k+1} = a ∗ x_k + b ∗ u_k
22     k = k + 1
23   until k = N
24   if final then u_{N−1} = (C − a ∗ x_{N−1})/b
25   if (max{γ, lhs} ≤ u_{N−1} ≤ min{δ, rhs} AND status) then success =TRUE
26 end
```

Probleme mit diesem Algorithmus

- Eventuell schlechte Erfolgschance

Daher folgende Veränderungen:

- Die Berechnung der Komponenten des Vektors u wurden verändert
- Wähle beliebigen Bruchteil μ im Bereich $[0, 1]$
- Wähle zufälliges Bit b
- Falls $b = 0$, dann wähle Komponente aus Bereich [left-range, r]
- sonst [r,right-range],
- mit r =left-range+(right-range−left-range) $* \mu$.

Evaluierung

Zur Evaluierung von Problemen aus der Klasse $\mathcal{Z}$ benutzen wir die gegebene Funktion J.

$$eval(u) = J(u, x)$$

Genetische Operatoren

- Die benutzten Operatoren wurden bereits erklärt (*Fine Local Tuning*).
- Im folgenden werden nur *arithmetical crossover* und *simple rossover* nochmals dargestellt.

Genetische Operatoren - arithmetical crossover

- *arithmetical crossover*:
  Linear Kombination zweier (*gültiger*) Kontrollvektoren u^1 und u^2:

$$u = a * u^1 + (1 - a) * u^2, \text{ mit } a \in [0, 1].$$

- Produziert immer gültige Nachfahren.

Genetische Operatoren - simple crossover

- *simple crossover*
- Nicht immer gültige Nachfahren, z.B:

$$\begin{aligned} u'^1 = \quad & (u_0'^1, \ldots, u_{N-1}'^1) \\ & = (u_0^1, \ldots, u_k^1, u_{k+1}^2, \ldots, u_{N-1}^2) \text{ und} \\ u'^2 = \quad & (u_0'^2, \ldots, u_{N-1}'^2) \\ & = (u_0^2, \ldots, u_k^2, u_{k+1}^1, \ldots, u_{N-1}^1) \end{aligned}$$

für ein $0 \leq k \leq N-2$.
Die letzte Komponente dieser Vektoren wird in

$$u_{N-1}'^1 = (C - a * x_{N-1}'^1)/b \text{ und } u_{N-1}'^2 = (C - a * x_{N-1}'^2)/b$$

geändert, wobei x'^1 und x'^2 die neuen Zustandsvektoren sind.

- Kann die Schranke $[(\alpha - a * x_p^1)/b, (\beta - a * x_p^1)/b]$ verletzten $(p > k)$

Jedoch steigt die Erfolgsrate, je weiter das Programm kommt, da die Population gegen das Optimum konvergiert.

Experiment: Parameter

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

- $pop\_size = 70$
- Generationen: 40.000
- $a = 0.25$ (für *arithmetical crossover*)
- Wahrscheinlichkeiten für Crossover: $p_{ac} = 0.15$ und $p_{sc} = 0.05$.
- Wahrscheinlichkeit für Mutation: $p_{nm} = 0.10$

Experiment: Funktion

- 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:

$$x_k \geq 0, \text{ für } k = 0, 1, \ldots, N,$$

$$u_k \geq 0, \text{ für } k = 0, 1, \ldots, N-1 \text{ und}$$

$$u_k \leq x_k, \text{ für } k = 0, 1, \ldots, N-1$$

$$(\mathit{right} = \text{TRUE}, \mathit{left} = \text{FALSE}, \tau = 1)$$

Funktionsparameter

Die Ergebnisse wurden mit folgenden Parameterwerten erzielt:

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

Experiment: Ergebnisse

- Initialisierung hatte Erfolgsrate von 100%.
- Nach 10.000 Generationen lieferte GAFOC exaktes Ergebnis (auf 6 Stellen).

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%

Implementation GENOCOP: Fakten

- Version 1.0 Sommer 1992 implementiert
- Getestet wurden quadratische, nicht-lineare, und unstetige Funktionen mit linearen Bedingungen
- Alle Versuche wurden auf einer SUN SPARC station 2 durchgeführt
- Es wird nur ein Teil aller Tests aufgeführt

Parameter

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 *arithmetical crossover*)
- Alle Versuche 10-mal durchgeführt
- Für die meisten Probleme 500 oder 1000 Generationen verwendet (manche benötigen größere Anzahl von Iterationen)

Test #1

- Problem

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

- 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 dem Optimum, 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

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

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

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

Test #3 - Seite 1

- 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:

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

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

Test #3 - Seite 2

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

Zusammenfassung

Eigenschaften der GENOCOP Methode.

- GENOCOP hat immer das Optimum gefunden
- In Test # 2 sogar ein besseres (!)
- Diese Methode arbeitet direkt mit den linearen Bedingungen, daher keine *penalty*-Werte nötig

Weiterentwicklung von GENOCOP

- Integer und Bool'sche Werte sollen verarbeitet werden
- Die Wahrscheinlichkeiten der Operatoren soll adaptiv werden

Zudem ist GENOCOP die Grundlage des GENOCOP II Systems.

Modifikationen von GENOCOP

Für die Version 2.0 wurden einige Änderungen an den genetischen Operatoren gemacht.

- *uniform mutation* und *boundary mutation* wurden nicht verändert. Um den Nutzen der *boundary mutation* zu zeigen, wurde folgendes Testszenario durchgeführt:

boundary mutation

- 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 $f(1.5, 1.0) = 9.0$.
- Jeweils 10 Durchläufe, 10 mit *boundary mutation*, 10 ohne.
- Mit *boundary mutation* jedes mal das Optimum gefunden (durchschnittlich in 32 Generationen).
- Ohne *boundary mutation* 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

- *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 beliebige Zahl aus $[0, 1]$, T maximale Anzahl der Generationen, und b Systemparameter.
- GENOCOP hatte die *non-uniform mutation* von „links nach rechts“ angewandt, neue Version generiert eine randomisierte Anordnung der Vektorkomponenten.

non-uniform mutation - Seite 2

Der Operator hat sich in den meisten Anwendungen als sehr nützlich erwiesen,z.B.:

- Problem:

$$\text{minimiere } f(x_1, x_2, x_3, x_4, x_5) = \begin{aligned}&(x_1 - 1.0)^2 + (x_1 - x_2^2)^2 + (x_2 - 1.0)^2 + \\ &+(x_1 - x_3^2)^2 + (x_3 - 1.0)^2 + (x_1 - x_4^2)^2 + \\ &+(x_4 - 1.0)^2 + (x_1 - x_5^2)^2 + (x_5 - 1.0)^2,\end{aligned}$$

- 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 $f(1, 1, 1, 1, 1) = 0$.

non-uniform mutation - Seite 3

- In 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

$$(1.02938247, 1.01172984, 1.01243937, 1.01172805, 1.01172495)$$

- Neue Version von Δ wesentlich effektiver als die bisherige

simple crossover

- *simple crossover* wurde fast unverändert gelassen:
  - Anstelle größten Wert a mit binären Suche zu finden, wird ausgehend vom Wert $a = 1$, der Wert um $\frac{1}{q}$ verringert, falls mindestens ein Nachkomme nicht gültig ist.
  - Nach q Versuchen muss es gültigen Nachfahren geben
  - Der Wert a wird jedes mal neu berechnet
  - Beachte: Ein maximales Absteigen nicht oft notwendig, und Wahrscheinlichkeit sinkt stark, je weiter die Population konvergiert.

arithmetical crossover

- *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.
  - Original Operator arbeitete mit festem $a = 0.25$
  - Neuer generiert zufälligen Wert aus $[0, 1]$ wenn aufgerufen

arithmetical crossover - Seite 2

Die Wichtigkeit dieses Operators wird am folgenden Beispiel deutlich:

- Problem:

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

- In Abhänigkeit von:

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

- 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 $f(\pi/2, \pi/2, \pi/2, \pi/2, \pi/2) = -6$.

arithmetical crossover - Seite 3

- 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

- Neues System stabiler (Abweichung vom besten Ergebnis in 10 Läufen)
- Schneller als die ursprüngliche Implementation

heuristic crossover

- *heuristic crossover* wurde eingeführt.
  1. Benutzt die Werte der Zielfunktion, um die Richtung der Suche zu bestimmen
  2. Produziert nur einen Nachfahren, oder
  3. Produziert keinen Nachfahren.
- Benutzt 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 (z.B. $f(x_2) \geq f(x_1)$).
- Es ist möglich das Operator ungülten Nachfahren produziert.
  - Generiert neuen Wert für r und versucht es erneut.
  - Nach w Fehlschlägen: Abbruch

heuristic crossover - Seite 2

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:

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

- Die bekannte Lösung ist $\bar{X}^* = (1, 1, 1, 1)$ und $f(\bar{X}^*) = 0$.
- Ohne Operator (Lösung in 10.000 Generationen) jede Variable innerhalb von 10% der optimalen Lösung. Typische Lösung:

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

mit $f(x_1, x_2, x_3, x_4) = 0.00683880$.

- Mit *heuristic crossover* wesentlich besser, typisches Ergebnis:

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

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

heuristic crossover - Seite 3

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.

Ergebnisse

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

Ausblick

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

Nichtlineare Optimierung - Definition

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

- Finde $\bar{X}$ so dass:
- 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:

$$c_i(\bar{X}) \leq 0, \ i = p + 1, \ldots, m.$$

Lösungsmethoden

Für die Lösung der Probleme wurden einige vorgehensweisen entwickelt, unter denen folgende zufinden sind:

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

Für GENOCOP II wurde „sequential quadratic penalty function“-Methode als Grundlage verwendet.

Vereinbarung:

Um Verwechslungen zu vermeiden, werden wir den Ur-GENOCOP als GENOCOP I bezeichnen.

sequential quadratic penalty function - Definition

Ersetzten des Ur-Problems durch $\mathcal{NPP}'$:

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

wobei $r > 0$ und $\bar{C}$ Vektor aller aktiven Bedingungen $c_1, \ldots, c_l$. Die Lösungen aus $\mathcal{NPP}$ sind gleich $\mathcal{NPP}'$, wenn $r \to 0$.

Lösungsansätze:

1. Minimierung von $F(\bar{X}, r)$ für eine abnehmende Folge von r.
   - Problem wurde für kleine r zu unhandlich
2. Andere Lösung mit Unabhängigkeit der Bedingungen gefunden

GENOCOP II

Mit Methode 2 und GENOCOP I Implementierung möglich

GENOCOP II

```
1  procedure GENOCOP II
2  begin
3    t ← 0
4    split the set of constraints C into
5     C = L ∪ N_e ∪ N_i
6    select a starting point X̄_s
7    set the set of active constraints, A to
8     A ← N_e ∪ V
9    set penalty τ ← τ_0
10   while( not termination condition) do
11   begin
12     t ← t + 1
13     execute GENOCOP I for the function
14       F(X̄, r) = f(X̄) + 1 (1/2r) Ā^T Ā
15       with linear constraints L
16       and starting point X̄_s
17     save best individual X̄*:
18       X̄_s ← X̄*
19     update A:
20       A ← A − S ∪ V,
21     decrease penalty τ
22       τ ← g(τ, t)
23   end
24 end
```

Erklärung des Algorithmus

- 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. Bedingung zählt als verletzt im Punkt $\bar{X}$ genau dann, wenn $c_j(\bar{X}) > \delta$ $(j = p+1, \ldots, m)$, wobei δ ein Parameter der Funktion ist. τ wird auf τ_0 gesetzt (Parameter der Funktion).
- In Hauptschleife: Aufruf von GENOCOP I. Übergabe der linearen Bedingungen L. Zu beachten ist, das GENOCOP I mit *pop_size* gleichen Elementen startet.
- Bestes Ergebnis wird gespeichert und A aktualisiert, wobei S und V Teilmengen von N_i sind.

Beispiel

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 $f(\bar{X}^*) = -1.088662$

Beispiel - Seite 2

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

Tests

Um GENOCOP II zu testen, wurden einige ausgewählte Probleme betrachtet. Die Fälle schließen

- quadratische,
- nicht-lineare und
- unstetige Funktionen

mit mehreren nicht-linearen Bedingungen mit ein. Alle Tests wurden auf einer SUN SPARC station 2 durchgeführt.

Paremeter

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)

Test #1

- Problem:

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

- In Abhängigkeit von:

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

  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

Test #2

- Problem:

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

- In Abhängigkeit von:

$$c_1 : y \leq 2x^4 - 8x^3 + 8x^2 + 2,$$
$$c_2 : y \leq 4x^4 - 32x^3 + 88x^2 - 96x + 36,$$

  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.

Test #2 - Ergebnisse

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

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

Test #3 - Ergebnisse

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

Folgerungen

Vorteile von GENOCOP II:

- Keine Berechnung des Gradienten oder der Hesseschen Matrix
- Es kann jeder GA anstelle von GENOCOP I verwendet werden

Auch geht die Suche nach weiteren Paradigmen bezüglich der Bedingungen weiter.

Paradigmen

Es gibt zwei nennenswerte Methoden:

- Erste arbeitet in zwei Phasen:
  1. Population entwickelt sich mit Standard-GA, in der die Fitness-Funktion von der Bedingungserfüllung abhängt
  2. Endpopulation der vorherigen Phase als Anfangspopulation des GAs mit Zielfunktion als Fitness-Funktion, welche 0 Fitness vergibt, falls eine Bedingung verletzt wird.

- Zweite weist allen gültigen Punkten höheren Fitnesswert zu als ungültigen, womit diese bevorzugt werden.

Die nächste Version des GENOCOP II wird diese Methoden optional beinhalten. 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.

integer programming - Beispiel

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.

Lösungsideen

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 guter Näherung oder sogar in optimalen Lösungen resultieren.

Lösungsideen - Seite 2

Das original Problem (1) wird durch folgendes ersetzt:

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

in Abhängigkeit von

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

($Ex = e_m$ 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.

Interessante Aspekte des erwähnten GA´s

Es gibt einige interessante Ideen in diesem GA.

- Ausgehend von großen Werten für λ ineffizient
- Anpassung des Wert für λ während des Laufzeit mit Sequenz
- Ähnlich wie bei GENOCOP II
  - langsame Rate gut für die Genauigkeit auf kosten der Geschwindigkeit
  - schnelle Rate kann zu ungenauen Ergebnissen führen