Einsetzbares Programm zur Lösung von Optimierungsproblemen mit Hilfe eines genetischen Algorithmus. Anzupassen ist im Wesentlichen nur die Fitnessfunktion F. (VC, 20.-25.5.2009)

```
-- Laufzeit mit den Standardeinstellungen: rund 500 Sekunden auf einem 1-GHz-Rechner.
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_Io; use Ada.Integer_Text_Io;
with Ada.Float_Text_Io; use Ada.Float_Text_Io;     -- Float wird hier allerdings nicht benutzt.
with Text_Io; use Text_Io;
with Ada.Numerics.Float_Random; use Ada.Numerics.Float_Random;
with Ada.Numerics.Generic_Elementary_Functions;

-- Gesucht wird das Maximum der Summe A(I)*I, deJong-Funktion. Methode: Genetischer Algorithmus über {0,1}^N.
-- Das Verfahren orientiert sich an den Algorithmen der Vorlesung, Abschnitt 5.2.1; hierfür sind manchmal Seitenzahlen
-- angegeben (Version 2009). Im Zweifel zunächst die Prozedur "Ein_Versuch" studieren.

procedure GA_Version_090523_Standard is

-- Wir legen zunächst die Genauigkeit der reellwertigen Berechnungen (hier: 18-stellig) fest:
   Genauigkeit: constant Integer := 18;
   type Real is digits Genauigkeit;
   package Fneu is new Float_Io (Num => Real);
   use Fneu;
   package My_Elementary_Functions is new Ada.Numerics.Generic_Elementary_Functions(Real);
   use My_Elementary_Functions;

-- Parameter des GA (hier: fest eingestellt. Sie müssen aber in Experimenten problemgerecht justiert werden.)
   N: constant Natural := 200;                  -- Zahl der Komponenten in jedem Individuum
   Indmax: constant Natural := 500;             -- obere Grenze für Zahl der Individuen in irgendeiner Population
   Mue: constant Natural := 30;                 -- konkrete Anzahl in der Population, Standard: 30
   Lambda: constant Natural := Mue*5;           -- Standard: Mue*5
   AnzElt: constant Natural := Mue;             -- Standard: Mue; AnzElt muss >= 2 sein!
                                                -- Beachte: Es dürfen Mue, Lambda, AnzElt nicht größer als Indmax sein!
   Q: constant Natural := 3;                    -- für Turnierselektion, Standard: 3
   P_C: constant Real := 0.9;                   -- Standard: 0.9
   P_Mut: constant Real := 1.0/Real(N);         -- Standard: 1/N
   KPCO: constant Natural := 2;                 -- Wähle den K-Punkt-Crossover. Standard: K = 2
   WfN: constant Real := 0.5;                   -- Wahrscheinlichkeit für "0" beim Initialisieren von Indivduen
   Plus: constant Boolean := false;             -- Plus-Strategie (oder Komma-Strategie bei Plus=false)
```

```
-- Datentypen
   type Drei is (Nu,Eins,Stern);                -- Nu = 0, Eins = 1, Stern = *
   subtype Bit is Drei range Nu..Eins;          -- beachte: wir nutzen hier den "Stern" nicht; vgl. Prozedur Mutation.
   type Schema is array(1..N) of Drei;          -- beachte: wir nutzen hier den Typ Schema nicht
   type Genotyp is array (1..N) of Bit;
   type Individuum is record
     Inhalt: Genotyp;
     Fit: Real;
     Alter: Natural;                            -- Alter := 0 wird beim Erzeugen gesetzt. (Diese Komponente wird hier nicht verwendet.)
     Erzeugt: Natural;                          -- in welcher Generation erzeugt? (Diese Komponente wird hier nicht verwendet.)
   end record;
   type Population is array(1..Indmax) of Individuum;

-- Zentrale Variablen des Verfahrens und Steuerparameter
   GZ: Natural := 0;                            -- Generationenzähler t;
   ZFB: Natural;                                -- Zählt die Fitnessfunktionsberechnungen
   ZEI: Natural;                                -- Zählt die erzeugten Individuen
   Pop, Kinder, Eltern: Population;             -- Pop hat Mue, Kinder hat Lambda, Eltern hat AnzElt viele Individuen
                                                -- Die Anzahl der Individuen muss bei Prozeduren stets mit übergeben werden!
   BestIndividuum: Individuum;
   G: Generator;                                -- zur Erzeugung von Zufallszahlen
   GZmax: constant Natural := 200;              -- Maximale Zahl an Generationen, Standard: 200
   AnzahlVersuche: constant Natural := 100;     -- Standard: 100
   Alles: Boolean := false;                     -- Alles ausdrucken ? (ziemlich viele Zwischenausgaben bei Alles = true)

-- Wähle zufällig eine natürliche Zahl.
   function Zufaellig(Unten, Oben: Natural) return Natural is
     R: Natural;                                -- Unten <= natürliche Zufallszahl <= Oben. "Unten <= Oben" wird nicht abgeprüft!
   begin
     R := Unten + 1 + Integer(Real(Random(G))*Real(Oben-Unten));
     if R < Unten then R := Unten; elsif R > Oben then R := Oben; end if;
                                                -- Beachte: Der Ada-Zufallszahlengenerator erzeugt manchmal den Wert 1.000...
     return R;
   end;
```

```
-- Fitnessfunktion analog zu deJong. Gesuchtes Maximum ist der 1-Vektor.
  function F (X: Genotyp) return Real is
    Wert: Integer := 0;
  begin
    ZFB := ZFB+1;                          -- jeder Aufruf der Fitnessfunktion wird gezählt
    for I in 1..N loop
      if X(I) = Eins then Wert := Wert + I; end if;
    end loop;
    return Real(Wert);
  end;

-- Erzeuge zufällig ein Individuum; erzeuge hierbei eine Null mit Wahrscheinlichkeit x, eine 1 mit 1-x.
  function Erzeuge_Ind_Zufaellig (x: Real) return Individuum is
    A: Individuum;
  begin
    ZEI := ZEI + 1;                        -- jeder Funktionsaufruf wird gezählt
    for I in 1..N loop
      if Real(Random(G)) < x then A.Inhalt(I) := Nu; else A.Inhalt(I) := Eins; end if;
    end loop;
    A.Fit := F(A.Inhalt);
    A.Alter := 0;
    A.Erzeugt := GZ;
    return A;
  end;

-- Initialisierung des Verfahrens und der Population Pop.
-- deJong-Funktion ist ein Maximierungsproblem. Wir berechnen entsprechend das bisher beste Individuum.
  procedure Init is
    BF: Real; Jbest: Natural;
  begin
    Reset(G);
    GZ := 0;
    ZFB := 0; ZEI := 0;
    for J in 1..Mue loop Pop(J) := Erzeuge_Ind_Zufaellig (WfN); end loop;
    Bf := Pop(1).Fit; Jbest := 1;
    for J in 1..Mue loop
      if Pop(J).Fit > Bf then Bf := Pop(J).Fit; Jbest := J; end if;
    end loop;
    BestIndividuum := Pop(Jbest);
  end;
```

```
-- Fitnessproportionale Selektion: Anz viele Individuen aus der Population P werden nach E kopiert. Siehe Algorithmus 1 auf Seite 42.
  procedure Selekt_Fit (P: in Population; LP: in Natural; E: out Population; Anz: in Natural) is      -- LP = Anzahl Elemente in P
    Gesamt_Fit, Z: Real; Links, Rechts, Mitte: Natural;
    D: array(0..LP) of Real;
    IndexEltern: array (1..Anz) of Natural;     -- Da E = P sein kann, werden die Indizes der für E bestimmten Individuen in "IndexEltern" gespeichert.
  begin                                         -- Dass Anz <= Indmax ist, wird zu Beginn des Hauptprogramms geprüft. Trotzdem testen:
    if Anz > Indmax then New_Line; Put("  **    Fehler Fit_prop Anzahl > Indmax    ** "); return;
    else
      D(0) := 0.0; Gesamt_Fit := 0.0;
      for I in 1..LP loop Gesamt_Fit := Gesamt_Fit + P(I).Fit; end loop;
      for I in 1..LP loop D(I) := D(I-1) + P(I).Fit/Gesamt_Fit; end loop;
      for R in 1..Anz loop
        Z := Real(Random(G));                   -- Suche das j mit D(j-1) <= Z < D(j). Dies geschieht durch binäre Suche.
        Links := 0; Rechts := LP-1;
        while Links + 1 < Rechts loop
          Mitte := (Links+Rechts)/2;
          if D(Mitte) <= Z then Links := Mitte; else Rechts := Mitte-1; end if;
        end loop;
        if Z < D(Rechts) then IndexEltern(R) := Rechts; else IndexEltern(R) := Rechts+1; end if;
      end loop;
      for I in 1..Anz loop E(I) := P(IndexEltern(I)); end loop;              -- Hier wird E mit Hilfe von IndexEltern zusammengestellt.
    end if;
  end;

-- Turnierselektion mit jeweils QQ Teilnehmern; Population P mit LP Individuen
  procedure Selekt_Turnier (P: in Population; LP: in Natural; QQ: in Positive; E: out Population; Anz: in Natural) is
    Fitbest: Real;  Z, Zbest: Natural := 0;
    IndexEltern: array (1..Anz) of Natural;     -- Da E = P sein kann, werden die Indizes der für E bestimmten Individuen in "IndexEltern" gespeichert.
  begin                                         -- Dass Anz <= Indmax ist, wird zu Beginn des Hauptprogramms geprüft. Trotzdem testen:
    if Anz > Indmax then New_Line; Put("  **    Fehler Turnier Anzahl > Indmax    ** "); return;
    else
      for R in 1..Anz loop
        Z := Zufaellig(1,LP);                   -- Wähle das erste Individuum zufällig
        Zbest := Z;  Fitbest := P(Z).Fit;
        for K in 2..QQ loop
          Z := Zufaellig(1,LP);                 -- Ziehe weitere QQ-1 Individuen, notiere das Beste
          if Fitbest < P(Z).Fit then Zbest := Z; Fitbest := P(Z).Fit; end if;
        end loop;
        IndexEltern(R) := Zbest;                -- Speichere den Index des Individuums mit der besten Fitness.
      end loop;
      for I in 1..Anz loop E(I) := P(IndexEltern(I)); end loop;              -- Nun wird E mit Hilfe von IndexEltern zusammengestellt.
    end if;
  end;
```

```
-- Simpel programmierte Mutation: Biflipping mit Wahrscheinlichkeit P_Mut mit anschließender Fitnessberechnung.
-- Siehe Algorithmus (*) Seite 75. In dieser Prozedur wird Rechenzeit verschwendet, vgl. Vorlesung.
  procedure Mutation (Kinder: in out Population; LP: in Natural) is
  begin
    for J in 1..LP loop
      for I in 1..N loop
        if Real(Random(G)) <= P_Mut then
          if Kinder(J).Inhalt(I) = Nu then Kinder(J).Inhalt(I) := Eins;
          else Kinder(J).Inhalt(I) := Nu;
          end if;
        end if;
      end loop;
      Kinder(J).Fit := F(Kinder(J).Inhalt);
    end loop;
  end;

-- Einpunkt-Crossover, aber nur mit Wahrscheinlichkeit P_C (mit 1-P_C wird E1 zurückgegeben), siehe Seite 54.
  function Einpunktcrossover (E1, E2: in Individuum) return Individuum is
    J: Natural; Kind: Individuum;
  begin
    if Real(Random(G)) < P_C then
      J := Zufaellig(1,N-1);
      for I in 1..J loop Kind.Inhalt(I) := E1.Inhalt(I); end loop;
      for I in J+1..N loop Kind.Inhalt(I) := E2.Inhalt(I); end loop;
      -- Kind.Fit := F(Kind.Inhalt);              -- Fitness hier berechnen, falls es nicht anschließend in der Prozedur Mutation geschieht!
      Kind.Alter := 0;
      Kind.Erzeugt := GZ;
      ZEI := ZEI + 1;                             -- Die erzeugten Individuen werden gezählt.
      return Kind;
    else
      return E1;
    end if;
  end;
```

```
-- NP-Punkt-Crossover mit Wahrscheinlichkeit P_C (mit Wahrscheinlichkeit 1-P_C wird E1 zurückgegeben).
-- Algorithmus von Seite 55 mit der Variante von Seite 57; hierfür muss D sortiert werden.
  function NPunktcrossover (E1, E2: in Individuum; NP: in Positive) return Individuum is
    K: Natural;  Kind: Individuum;  Elter1: Boolean := true;
    D: array (0..NP+1) of Natural;                  -- Feld der gleichverteilten NP Indizes
    procedure Sortiere_D is                         -- mit Bubble-Sort programmiert, da NP meist klein ist
      Hilf: Natural;
    begin
      for I in 1..NP-1 loop
        for J in 1..I loop
          if D(J) > D(J+1) then
            Hilf := D(J); D(J) := D(J+1); D(J+1) := Hilf;
          end if;
        end loop;
      end loop;
    end;
  begin
    if Real(Random(G)) < P_C then
      D(0) := 0; D(NP+1) := N; K := N-NP;        -- (Kleiner Fehler im Skript S. 57)
      for I in 1..NP loop D(I) := Zufaellig(1,K); end loop;
      Sortiere_D;
      for I in 2..NP loop D(I) := D(I) + I-1; end loop;
      -- Hier gilt 0 = D(0) < D(1) < D(2) < ... < D(NP) < D(NP+1) = N. Jetzt abwechselnd die Gene von E1 und E2 in das Kind übertragen:
      for J in 1..NP loop
        if Elter1 then
          for I in D(J-1)+1..D(J+1) loop Kind.Inhalt(I) := E1.Inhalt(I); end loop;
        else
          for I in D(J-1)+1..D(J+1) loop Kind.Inhalt(I) := E2.Inhalt(I); end loop;
        end if;
        Elter1 := not Elter1;
      end loop;
      -- Kind.Fit := F(Kind.Inhalt);  --Fitness hier berechnen, falls es nicht danach in der Prozedur Mutation geschieht!
      Kind.Alter := 0;
      Kind.Erzeugt := GZ;
      ZEI := ZEI + 1;                               -- Die Zahl der erzeugten Individuen wird gezählt.
      return Kind;
    else
      return E1;
    end if;
  end;
```

```
-- Hier benutzen wir nur den Generationenzähler. Erweiterungsmöglichkeiten: Man kann auch die Varianz der Population abfragen
-- oder die Differenz der Fitnesswerte des besten und schlechtesten Individuums usw.
  function Terminierung return Boolean is
  begin  return (GZ > GZmax); end;

  procedure Erhoehe_Generationenzaehler_und_Alter is
  begin
    GZ := GZ + 1;
    for J in 1..Mue loop Pop(J).Alter := Pop(J).Alter + 1; end loop;
  end;

-- Für die Plusstrategie muss man Pop und Kinder vereinigen (zur neuen Population Pop).
  procedure Vereinige(P: in out Population; LpP: in Natural; K: in Population; LpK: in Natural) is
  begin
    if LpP+LpK > Indmax then New_Line; Put("  **   Fehler bei Vereinigung  ** "); return;
    else
      for I in 1..LpK loop P(I+LpP) := K(I); end loop;
    end if;
  end;

-- Das beste Individuum der Population P in BestIndividuum ablegen, sofern es besser als das bisher notierte Individuum ist.
  procedure Bisherbestes (P: in Population; LP: in Natural) is              -- P hat Lp Elemente.
    BF: Real  := BestIndividuum.Fit;
  begin
    for J in 1..LP loop
      if BF < P(J).Fit then Bestindividuum := P(J); BF := Bestindividuum.Fit; end if;
    end loop;
  end;

-- Berechne die Varianz der Fitnesswerte, also die Schwankung um den Mittelwert der Population P. (P hat LP Elemente.)
  function Varfit (P: Population; LP: Natural) return Real is
    V, S, H: Real := 0.0;                                -- S = Gesamt-Fitness von P
  begin
    for J in 1..LP loop S := S + P(J).Fit; end loop;
    S := S/Real(LP);                                     -- S ist nun der Mittelwert aller Fitnesswerte in P
    for J in 1..LP loop
      H := P(J).Fit-S;
      V := V + H*H;
    end loop;
    return sqrt(V/Real(LP));
  end;
```

```
-- Für die Ausgabe: Anzahl der Dezimalziffern einer natürlichen Zahl.
  function Laenge(Z: Integer) return Natural is
    L, K: Integer;
  begin
    if Z < 0 then  K := -Z; L := 2; else K := Z; L := 1; end if;
    while K > 9 loop  K := K/10; L := L+1; end loop;
    return L;
  end;

-- Für die Ausgabe von reellwertigen Zahlen:
  function Stellen_Vor_Komma (Z: Real) return Natural is
    L: Natural; X: Real;
  begin
    if Z < 0.0 then  X := -Z; L := 2; else X := Z; L := 1; end if;
    while X >= 10.0 loop X := X/10.0; L := L+1; end loop;
    return L;
  end;

-- Zwischenausgabe (wird nur aufgerufen, wenn Alles auf true gesetzt ist)
  procedure Drucken is
    Zahlnullen: Integer := 0;  Posletztenull: Integer := 0;  Hilf: Real;
  begin
    New_Line; New_Line; Put("Aktuelle Generation: "); Put(GZ,Laenge(GZ));
    Put(",  bisher bestes Individuum:"); New_Line;
    for I in 1..N loop
      if Bestindividuum.Inhalt(I) = Nu then Put("0 "); Zahlnullen := Zahlnullen+1; PosLetzteNull := I;
      else Put("1 "); end if;
    end loop;
    New_Line; Put("Nullen: "); Put(ZahlNullen,Laenge(ZahlNullen));
    Put(",  Einsen: "); Put(N-Zahlnullen,Laenge(N-Zahlnullen));
    Put(",  Position der letzten Null: "); Put(PosLetzteNull,Laenge(PosLetzteNull));
    New_Line; Put("Fitnesswert: "); Put(Bestindividuum.Fit,Stellen_Vor_Komma(Bestindividuum.Fit),9,0);
    New_Line; Put("Erstmals erzeugt in Generation: ");
    Put(Bestindividuum.Erzeugt,Laenge(Bestindividuum.Erzeugt));
    Put("Alter damals: "); Put(Bestindividuum.Alter,Laenge(Bestindividuum.Alter));
    New_Line; Put("Gesamtzahl der Fitnessberechnungen: "); Put(ZFB, Laenge(ZFB));
    New_Line; Put("Gesamtzahl der erzeugten Individuen: "); Put(ZEI,Laenge(ZEI));
    New_Line; Hilf := Varfit(Pop,Mue); Put("Aktuelle Varianz"); Put(Hilf,Stellen_Vor_Komma(Hilf),9,0);
    New_Line;
  end;
```

```
-- Erste Ausgabe mit allen Parametern
  procedure Startausgabe is
  begin New_Line;
    New_Line; Put("Datum: 23.5.2009, Autor: VC, Version1: Summe A(i)*i maximieren, Standard-Werte.");
    New_Line; Put("Eingabe: Nur die Fitnessfunktion. Ausgabe: Bestes Individuum.");
    New_Line; Put("Initialisierung: Zufaellig werden "); Put(Mue, Laenge(Mue));
    Put(" Individuen erzeugt mit Wahrscheinlichkeit fuer Null = "); Put(WfN,Stellen_vor_Komma(WfN)+1,9,0);
    New_Line; Put("Laenge der Individuen N = "); Put(N,Laenge(N));
    New_Line; Put("Groesse der Population Mue = "); Put(Mue,Laenge(Mue));
    New_Line; Put("Anzahl der Kinder Lambda = "); Put(Lambda,Laenge(Lambda));
    New_Line; Put("Anzahl der ausgewaehlten Eltern AnzElt = "); Put(AnzElt,Laenge(AnzElt));
    New_Line; Put("Teilnehmerzahl bei der Turnierselektion Q = "); Put(Q,Laenge(Q));
    New_Line; Put("Crossoverwahrscheinlichkeit P_C = "); Put(P_C,Stellen_Vor_Komma(P_C),9,0);
    New_Line; Put("Mutationswahrscheinlichkeit pro Bit P_Mut = "); Put(P_Mut,Stellen_Vor_Komma(P_Mut),9,0);
    New_Line; Put("Maximale Zahl an Generationen GZmax = "); Put(GZmax,Laenge(GZmax));
    New_Line; Put("Erwarteter maximaler Fitnesswert: "); Put(N*(N+1)/2,Laenge(N*(N+1)/2));
    New_Line; Put("K-Punkt-Crossover mit K = "); Put(KPCO,Laenge(KPCO)); New_Line;
    Put("Verwendet wird die "); if Plus then Put("Plusstrategie"); else Put("Kommastrategie"); end if; New_Line;
    Put("Jeweils werden "); Put(AnzahlVersuche,Laenge(AnzahlVersuche)); Put(" Versuche durchgefuehrt.");
  end;

-- Basiszyklus der Evolutionären Algorithmen speziell für den binären Fall des GA:
  procedure Ein_Versuch is
    R1, R2: Natural;                              -- Hilfsindizes für die Auswahl zweier Eltern
  begin  Init;                                    -- Nun Evolutionszyklus iterieren:
    while not Terminierung loop
      Selekt_Fit (Pop, Mue, Eltern, AnzElt);      -- Fitnessproportionale Elternselektion
      for I in 1.. Lambda loop                    -- Rekombination. Waehle zwei Eltern mit verschiedenen (!) Indizes aus:
        R1 := Zufaellig (1,AnzElt); R2 := Zufaellig (1,AnzElt);
        while R2 /= R1 loop R2 := Zufaellig (1,AnzElt); end loop;
        if KPCO = 1 then Kinder(I) := Einpunktcrossover (Eltern(R1), Eltern(R2));
        else Kinder(I) := Npunktcrossover (Eltern(R1), Eltern(R2), KPCO);
        end if;
      end loop;
      Mutation (Kinder, Lambda);                  -- Mutation der Kinder; hier wird auch die Fitness für jedes Kind berechnet.
      BisherBestes (Kinder, Lambda);              -- Ermittelt das bisher beste Individuum.
      if Plus then
        Vereinige(Pop, Mue, Kinder, Lambda);                -- Pop := " Pop konkateniert mit Kinder "
        Selekt_Turnier(Pop, Mue+Lambda, Q, Pop, Mue);       -- nächste Population; Plusstrategie
      else Selekt_Turnier(Kinder, Lambda, Q, Pop, Mue);     -- nächste Population; Kommastrategie
      end if;
      Erhoehe_Generationenzaehler_und_Alter;
    end loop;
  end;
```

```
-- Hauptprogramm: Ein Experiment = AnzahlVersuche oft "Ein_Versuch" durchführen und statistische Daten hierzu ermitteln.
begin
if (Mue <= Indmax) and (Lambda <= Indmax) and (AnzElt <= Indmax)
  and (AnzElt > 1) and (Mue > 1) and (Lambda > 1) and ((not Plus) or ((Mue+Lambda) <= Indmax)) then
  -- for KExp in 1..AnzExp loop       -- Dies aktivieren, um AnzExp viele Experimente mit mehreren Varianten durchführen zu können.
  Startausgabe;                       -- Nun: Die Arbeitsumgebung für ein Experiment festlegen.
  declare
    Fit_Mittel, GZopt_Mittel,ZFB_Mittel, ZEI_Mittel, VarianzFit_Mittel: Real := 0.0;
    Gesamtbest: Individuum;
    procedure Schlussausdruck is
      ZahlNullen: Natural := 0;  LetzteNull: Natural := 0;
    begin
      if Alles then New_Line; Put("--------------------------------------------------------------------------------");
        New_Line; Put("Ausgangssituation:"); New_Line; Startausgabe;
      end if;
      New_Line; Put("Statistik nach ");
      Put(AnzahlVersuche, Laenge(AnzahlVersuche)); Put(" Versuchen: "); New_Line;
      Put("Mittlere Zahl der Fitnessberechnungen in jedem Versuch: "); Put(ZFB_Mittel,Stellen_Vor_Komma(ZFB_Mittel),9,0);
      New_Line; Put("Mittlere Zahl der erzeugten Individuen in jedem Versuch: "); Put(ZEI_Mittel,Stellen_Vor_Komma(ZEI_Mittel),9,0);
      New_Line; Put("Durchschnittliche Fitness am Ende jedes Versuchs: "); Put(Fit_Mittel,Stellen_Vor_Komma(Fit_Mittel),9,0);
      New_Line; Put("Durchschnittliche Varianz der Fitness jeweils am Versuchsende: ");
      Put(Varianzfit_Mittel,Stellen_Vor_Komma(Varianzfit_Mittel),9,0);
      New_Line; Put("Mittlere Zahl an Generationen in jedem Versuch bis zum lokalen Optimum: ");
      Put(GZopt_Mittel,Stellen_Vor_Komma(GZopt_Mittel),9,0);
      New_Line; Put("Insgesamt bestes Individuum aus allen Versuchen:"); New_Line;
      for I in 1..N loop
        if Gesamtbest.Inhalt(I) = Nu then Put("0 "); Zahlnullen := Zahlnullen+1; LetzteNull := I; else Put("1 "); end if;
      end loop;
      New_Line; Put("Positionen der Nullen im besten Individuum:"); New_Line;
      for I in 1..N loop
        if Gesamtbest.Inhalt(I) = Nu then Put(I,4); end if;
      end loop;
      New_Line; Put("Anzahl Nullen: "); Put(ZahlNullen,Laenge(ZahlNullen));
      Put(",  Anzahl Einsen: "); Put(N-Zahlnullen,Laenge(N-Zahlnullen));
      Put(",  Position der letzten Null: "); Put(LetzteNull,Laenge(LetzteNull));
      New_Line; Put("Fitnesswert des besten Individuums: ");
      Put(Gesamtbest.Fit,Stellen_Vor_Komma(Gesamtbest.Fit),9,0); New_Line;
    end;  -- Ende des Schlussausdrucks
```

```
  begin
    -- Ein_Versuch wird AnzahlVersuche oft durchgeführt; die entsprechenden Mittelwerte werden berechnet und ausgegeben.
    Ein_Versuch;
    Gesamtbest := Bestindividuum;
    Fit_Mittel := Bestindividuum.Fit;
    GZopt_Mittel := Real(Bestindividuum.Erzeugt);
    Varianzfit_Mittel := Varfit(Pop,Mue);                -- Varianzberechnung in der letzten Population des ersten Versuchs
    if Alles then New_Line; New_Line; Put("Situation nach dem ersten Versuch :"); Drucken; end if;
    for Zaehlen in 2..AnzahlVersuche loop
      Ein_Versuch;
      if Alles then New_Line; Put("Situation nach dem "); Put(Zaehlen, Laenge(Zaehlen)); Put(". Versuch :"); Drucken; end if;
      if Bestindividuum.Fit > Gesamtbest.Fit then Gesamtbest := Bestindividuum; end if;
      Fit_Mittel := Fit_Mittel + Bestindividuum.Fit;
      GZopt_Mittel := GZopt_Mittel + Real(Bestindividuum.Erzeugt);
      Varianzfit_Mittel := Varianzfit_Mittel + Varfit(Pop,Mue);    -- siehe oben
      ZFB_Mittel := ZFB_Mittel + Real(ZFB);
      ZEI_Mittel := ZEI_Mittel + Real(ZEI);
    end loop;
    Fit_Mittel := Fit_Mittel/Real(AnzahlVersuche);
    GZopt_Mittel := GZopt_Mittel / Real(AnzahlVersuche);
    Varianzfit_Mittel := Varianzfit_Mittel/Real(AnzahlVersuche);
    ZFB_Mittel := ZFB_Mittel/Real(AnzahlVersuche);
    ZEI_MIttel := ZEI_Mittel/Real(AnzahlVersuche);
    Schlussausdruck;
  end;
  -- end loop;   -- Ende KExp = Ende der Schleife für AnzExp viele Experimente.
else Put("Fehlerhafter Parameter Mue, Lambda oder AnzElt. => Abbruch.");
end if;
end;
```

Es folgen Experimente und der Versuch, die Parameter optimal für die deJong-Funktion einzustellen.

Beispiel: Ausgabe zweier Läufe der Standardversion: Bei beiden Läufen erscheint natürlich der gleiche "Startausdruck"

Datum: 23.5.2009, Autor: VC, Version1: Summe A(i)*i maximieren, Standard-Werte.
Eingabe: Nur die Fitnessfunktion. Ausgabe: Bestes Individuum.
Initialisierung: Zufaellig werden 30 Individuen erzeugt mit Wahrscheinlichkeit fuer Null = 0.500000000
Laenge der Individuen N = 200
Groesse der Population Mue = 30
Anzahl der Kinder Lambda = 150
Anzahl der ausgewaehlten Eltern AnzElt = 30
Teilnehmerzahl bei der Turnierselektion Q = 3
Crossoverwahrscheinlichkeit P_C = 0.900000000
Mutationswahrscheinlichkeit pro Bit P_Mut = 0.005000000
Maximale Zahl an Generationen GZmax = 200
Erwarteter maximaler Fitnesswert: 20100
K-Punkt-Crossover mit K = 2
Verwendet wird die Kommastrategie
Jeweils werden 100 Versuche durchgefuehrt.

Die beiden Läufe ergaben folgende Ergebnisse (Laufzeit: jeweils knapp 500 Sekunden auf einem 1-Ghz-Rechner)

Statistik nach 100 Versuchen:
Mittlere Zahl der Fitnessberechnungen in jedem Versuch: 29878.200000000
Mittlere Zahl der erzeugten Individuen in jedem Versuch: 26886.090000000
Durchschnittliche Fitness am Ende jedes Versuchs: 19715.590000000
Durchschnittliche Varianz der Fitness jeweils am Versuchsende: 61.484981294
Mittlere Zahl an Generationen in jedem Versuch bis zum lokalen Optimum: 195.610000000
Insgesamt bestes Individuum aus allen Versuchen:
0 1 1 0 1 1 0 0 0 1 1 0 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Positionen der Nullen im besten Individuum: 1 4 7 8 9 12 17 20 24 31 45
Anzahl Nullen: 11, Anzahl Einsen: 189, Position der letzten Null: 45
Fitnesswert des besten Individuums: 19922.000000000

Statistik nach 100 Versuchen:
Mittlere Zahl der Fitnessberechnungen in jedem Versuch: 29878.200000000
Mittlere Zahl der erzeugten Individuen in jedem Versuch: 26882.860000000
Durchschnittliche Fitness am Ende jedes Versuchs: 19719.520000000
Durchschnittliche Varianz der Fitness jeweils am Versuchsende: 64.167502704
Mittlere Zahl an Generationen in jedem Versuch bis zum lokalen Optimum: 194.770000000
Insgesamt bestes Individuum aus allen Versuchen:
1 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Positionen der Nullen im besten Individuum: 4 11 14 20 22 30
Anzahl Nullen: 6, Anzahl Einsen: 194, Position der letzten Null: 30
Fitnesswert des besten Individuums: 19999.000000000

Nun muss man Experimente durchführen, um die Parameter an das Problem anzupassen. Da die deJong-Funktion nur genau ein Optimum besitzt, das von jedem Individuum aus durch Hill Climbing erreichbar ist, wird sich vermutlich eine elitäre Strategie bei der Selektion als besonders gut herausstellen. Hierfür muss man die Mutationswahrscheinlichkeit P_Mut gering halten und zugleich das Q in der Turnierselektion erhöhen.

Wir aktivieren also die Schleife KExp und ersetzen den Platzhalter " -- for KExp in 1..AnzExp loop" durch

```
for KE1 in 3..7 loop
  Q := KE1;
  for KE2 in 1..4 loop
    P_Mut := 1.0/Real(KE2*N);
```

fügen am Ende zweimal "end loop" hinzu und streichen "constant" bei den Deklarationen von Q und P_Mut.

Der Testlauf liefert die erwarteten Ergebnisse dieser 20 Experimente. In die Tabelle ist eingetragen, für welches Q und welchen Wert von P_Mut sich welcher Fitnesswert mit wie viele Nullen für das beste Individuum nach 100 Versuchen ergab:

Q = P_Mut	3	4	5	6	7
0.005000	200158	201000	201000	201000	201000
0.002500	200255	201000	201000	201000	201000
0.001667	199409	200832	201000	201000	201000
0.001250	1985213	200635	200981	201000	201000

Vermutung: Der Wert P_Mut ≈ 1/N scheint recht gut zu sein, und Q = 5 ebenfalls.
Die Verschlechterung in der ersten Spalte ist leicht zu erklären: Anfangs werden 30 Individuen mit je 200 Komponenten erzeugt. Mehrere der Komponenten werden daher in allen Individuen 0 sein. Diese Nullen kann nicht die Rekombination, sondern nur die Mutation beseitigen. Je kleiner die Mutationsrate ist, um so länger dauert dieser Prozess. Da jedes Experiment aber bereits nach 200 Generationen abgebrochen wird, liefern die besonders kleinen Mutationswahrscheinlichkeiten daher schlechtere Ergebnisse.

Andererseits muss eine Verschlechterung auch auftreten, wenn die Mutationswahrscheinlichkeit zu groß wird, weil dann die Wahrscheinlichkeit, einige der vielen Einsen in eine Null umzuwandeln, steigt. Führt man entsprechende Experimente mit 10-mal kleineren und mit 10-mal größeren Werten als in der obigen Tabelle durch, so bestätigt sich diese Vermutung (die Bedeutung der Zahlen ist die gleiche wie in der vorherigen Tabelle):

Q = P_Mut	3	4	5	6	7
0.05000	16092 53	16642 49	16830 51	16883 43	17064 40
0.02500	17530 42	17856 40	18272 35	18553 33	18715 26
0.01667	18475 26	18663 31	19013 21	19269 21	19516 19
0.01250	18977 24	19230 20	19652 13	19829 10	19867 10
0.0005000	18952 27	19358 22	19873 11	19929 7	20046 6
0.0002500	17781 34	18387 24	19374 19	19349 13	19509 19
0.0001667	17281 40	17739 35	18310 35	18609 27	18800 29
0.0001250	16840 49	17457 40	17852 33	18181 28	18674 26

Feststellung: Das im Mittel optimale P_Mut ist eine Funktion von Q. Man kann diesen Zusammenhang P_Mut = f(Q) über Experimente tabellieren und hierdurch für Probleme, die eine ähnliche Struktur besitzen, den aufwendigen Vorgang, die Parameter einzeln zu justieren, stark beschleunigen, indem man einen der beiden Parameter in der Tabelle nachschlägt. Man beachte aber, dass der funktionale Zusammenhang für jede andere Parameterkombination von N, P_C, Mue, Lambda usw. anders sein kann und auch sein wird!

Nun verändern wir Lambda und AnzElt: Mue = 30, Lambda = 30, AnzElt = 15, P_C = 0.9, N = 200, GZmax = 200, 2-Punkt-Crossover und wollen für die Plus- und die Kommastrategie herausfinden, welche Werte für Q und P_Mut günstig sind. Die beiden Einträge in jedes Feld der Tabelle bedeuten: Fitnesswert für das beste Individuum nach 100 Versuchen, wobei die obere Zahl sich auf die Plus- und die untere auf die Kommastrategie bezieht.

Q = P_Mut	3	4	5	6	7
0.020000	19480 17605	19905 17915	19996 18300	20053 18834	20071 18691
0.005000	19966 19682	20100 19997	20100 20090	20100 20099	20100 20100
0.002222	19708 19748	19943 19997	20080 20100	20100 20083	20100 20100
0.001250	19059 19162	19671 19703	19824 19772	20009 19952	20036 20065
0.0008000	18562 18907	19146 19234	19391 19546	19514 19763	19676 19924
0.0005556	17931 18032	18848 18411	18826 18996	19300 19315	19235 19212
0.00040816	17232 17413	18286 18317	18465 18733	18576 18795	18860 19027
0.0003125	16966 17059	17522 17680	17910 17942	18273 18189	18305 18219

Dieses Experiment scheint darauf hinzudeuten, dass bei hohen Mutationsraten, also für P_Mut > 2/N, die Plusstrategie, bei kleineren Mutationsraten, also für P_Mut < 0.16/N, die Kommastrategie bessere Ergebnisse liefert. Erneut erweist sich die Größenordnung P_Mut ≈ 1/N als sehr günstig, mit leichten Vorteilen für die Plusstrategie, sofern Q < 7 ist.

Nun müsste man viele weitere Experimente durchführen, um noch mehr Einblicke in die Zusammenhänge zwischen den Parametern zu bekommen. Zum Beispiel:

Welchen Einfluss hat der Wert K beim K-Punkte-Crossover (also der Wert von KPCO)? Ausprobieren mit den Werten [N = 200, Mue = 30, Lambda = 30, AnzElt = 15, Q = 3, P_C = 0.9, P_Mut = 1/N, WfN = 0.5, Plusstrategie, GZmax = 200, jeder Wert wurde als bestes Individuum aus 100 Versuchen ermittelt] ergab z.B.:

KPCO:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Beste Fitness:	20013	20018	19969	19957	19948	19988	19995	19966	19963	20016	19983	20006	19947	19988	19978
Zahl der Nullen:	7	6	6	6	7	6	9	7	10	8	9	7	5	8	6

Eine zweite Messreihe mit den gleichen Werten ergab:

KPCO:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Beste Fitness:	20032	20027	19970	20024	20020	19961	19994	19958	19944	20002	19993	19952	20001	19965	19972
Zahl der Nullen:	6	7	10	7	7	11	7	8	10	5	5	7	7	7	9

Dies sieht weitgehend nach einer zufälligen Verteilung aus. Man kann bestenfalls die Aussage vermuten, dass $1 \leq KPCO \leq 2$ keine schlechte Wahl ist. Aber: Dass das Verfahren kaum von KPCO abhängt, liegt wohl daran, dass die Eltern "Eltern(R1) und Eltern(R2)" im Basiszyklus "Ein_Versuch" nicht angeordnet sind, wodurch der systematische Fehler, den KPCO = 2 bewirkt (vgl. Vorlesung S. 58 und 59), nicht zum Tragen kommt. Wir fügen daher in der Prozedur "Ein_Versuch" hinter der Anweisung

```
while R2 /= R1 loop R2 := Zufaellig (1,AnzElt); end loop;
```

die Zeile:

```
if Eltern(R2).Fit > Eltern(R1).Fit then H := R1; R1 := R2; R2 := H; end if;
```

(mit der neuen Natural-Variablen H)

ein, d.h., Eltern(R2) besitzt stets die kleinere Fitness der beiden Eltern. Dann erhält man z.B. folgende Messreihe:

KPCO:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Beste Fitness:	20040	20012	20003	19972	19939	20006	19990	19967	19941	19995	19982	20004	19948	19944	20026
Zahl der Nullen:	7	7	5	7	9	6	7	10	6	9	7	7	9	8	5

Der erwartete Trend, dass sich KPCO = 2 nun dem KPCO = 1 überlegen erweist, wird hierdurch nicht bestätigt, wohl aber könnte man erneut die Werte 1 und 2 für KPCO im Mittel für besser als die größeren Werte halten. Doch um statistisch gesicherte Aussagen zu erhalten, ist "AnzahlVersuche = 100" ein viel zu kleiner Wert.

Als Ergebnis dieser Messreihen halten wir also nur fest, dass KPCO = 1 und KPCO = 2 für die deJong-Funktion gute Werte sind.

(Beachte aber die Grundsatzbetrachtung zu GZmax einige Folien später.)

Die Zeile "`if Eltern(R2).Fit > Eltern(R1).Fit then ...`" lassen wir im Folgenden wieder weg.

Ein weiteres Experiment muss sich der Crossover-Wahrscheinlichkeit P_C widmen. Wir halten die Werte [N = 200, Mue = 30, Lambda = 30, AnzElt = 15, Q = 3, P_Mut =1/N, KPCO = 2, WfN = 0.5, Plusstrategie, GZmax = 200, jeder Wert wurde als bestes Individuum aus 100 Versuchen ermittelt] fest und testen 21 Werte für P_C = 1.0, 0.98, 0.96, ..., 0.60. Auch hierbei konnte kein Zusammenhang aufgespürt werden, vielmehr streuten die Werte ziemlich zufällig zwischen 19937 und 20070.

Wichtig sind bei allen evolutionären Verfahren die Abhängigkeit von Mue und das Verhältnis Lambda/Mue. Die Werte Mue = 30 und Lambda von 105 bis 150 sowie Lambda/Mue zwischen 3 und 7 (bei der Plusstrategie kleiner als bei der Kommastrategie) gelten gemeinhin als keine schlechte Wahl, jedoch muss dies für jedes Problem neu justiert werden. Wir halten nun also die Werte [N = 200, Lambda = 3*Mue, AnzElt = Mue, Q = 3, P_C = 0.9, P_Mut = 1/N, KPCO = 2, WfN = 0.5, Plusstrategie, GZmax = 200, jeder Wert wurde als bestes Individuum aus 100 Versuchen ermittelt] fest und testen die Zahlen 5, 10, 15, ..., 75 für Mue. Man erhält dann z.B. (falls Sie dies nachprüfen wollen - und dies ist stark erwünscht! -, so beachten Sie, dass die Rechenzeit für diese Messreihe bereits 3 Stunden auf einem 1-GHz-Rechner beträgt; dass die folgenden Werte besser sind als vergleichbare Werte in früheren Messreihen, liegt wohl an der hier verwendeten Plusstrategie, die für die deJong-Funktion vor allem bei kleineren Werten von Mue etwas bessere Ergebnisse zu liefern scheint):

Mue:	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75
Beste Fitness:	19277	19626	19909	19999	20015	20054	20065	20075	20097	20075	20070	20082	20083	20085	20093
Zahl der Nullen:	20	13	13	8	6	4	4	3	1	4	4	3	2	1	2

Hinweis: Ab Mue = 85 wurde bei unseren Messungen stets das Optimum gefunden.
Der Verlauf ist wie erwartet: Die Gesamtzahl der in 200 Generationen erzeugten Individuen ist proportional zu Mue und daher wird für kleinere Mue mit wachsendem Mue auch ein mindestens so gutes bestes Individuum gefunden; zugleich steigt auch die Rechenzeit (ermitteln Sie aus dem Programm die Abhängigkeit der Rechenzeit von Mue, GZmax und AnzahlVersuche!). Bei mittleren Mue-Werten (hier also ab Mue = 40) befindet man sich für GZmax = 200 schon nahe beim Optimum und muss nun mit zunehmender Zufälligkeit recht hoher Fitnesswerte rechnen. Erst für deutlich größere Werte (also etwa ab Mue = 90) kann man hoffen, fast jedes Mal in 100 Versuchen das Optimum zu finden.

Interessant ist, dass für Mue von 5 bis 75 das Optimum kein Mal gefunden wurde. Das Verfahren tut sich also zum Ende hin schwer, die letzten Optimierungen aufzuspüren. Dies zeigen die durchschnittliche Varianz am Ende jedes Versuchs und vor allem die durchschnittliche Fitness über alle 100 Versuche (hier: gerundet), die die Verbesserungen im Mittel deutlich widerspiegelt:

Mue:	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75
durchschn. Fitness:	18574	19299	19596	19734	19812	19873	19894	19924	19950	19969	19968	19990	19995	20007	20018
Varianz am Ende:	36.6	46.9	46.1	44.8	46.0	44.3	44.9	43.5	46.2	42.5	42.0	41.5	41.9	43.1	42.9

Interpretieren Sie diese Messreihe! Was besagt die Varianz am Ende jedes Versuchs?

Weitere Messungen legen es nahe, AnzElt kleiner als Mue zu wählen, d.h., eine noch stärkere Auswahl der Eltern auf gute Individuen vorzunehmen. AnzElt = 3/4·Mue scheint z.B. ein brauchbarer Wert zu sein. Nun können wir die Standard-Parameter neu festlegen.

Folgerung aus unseren bisherigen Messungen: Parametereinstellungen "Standard_neu"
Vorgegeben sind die Werte [N = 200, WfN = 0.5, GZmax = 200, jeder Wert wurde als bestes Individuum aus 100 Versuchen ermittelt]. Dann scheinen sich folgende Parameterwerte für die deJong-Funktion gut zu eignen:
Mue = 40, Lambda = 120, AnzElt = 30, Q = 5, P_C = 0.9, P_Mut = 1/N, KPCO = 2, Plusstrategie.

Testlauf. Ein Experiment mit diesen Parameterwerten benötigt 460 Sekunden auf einem 1-GHz-Rechner. Man beachte den hervorragenden Durchschnittswert 20099.45 für die mittlere Fitness am Ende *jedes* der 100 Versuche sowie die geringere Zahl an Fitnessberechnungen gegenüber der Standardversion (bei sogar besserem Ergebnis).

Datum: 24.5.2009 - Standard_neu, Autor: VC, Version 1a: Summe A(i)*i maximieren, Standard_neu-Werte.
Eingabe: Nur die Fitnessfunktion. Ausgabe: Bestes Individuum.
Initialisierung: Zufaellig werden 40 Individuen erzeugt mit Wahrscheinlichkeit fuer Null = 0.500000000
Laenge der Individuen N = 200
Groesse der Population Mue = 40
Anzahl der Kinder Lambda = 120
Anzahl der ausgewaehlten Eltern AnzElt = 30
Teilnehmerzahl bei der Turnierselektion Q = 5
Crossoverwahrscheinlichkeit P_C = 0.900000000
Mutationswahrscheinlichkeit pro Bit P_Mut = 0.005000000
Maximale Zahl an Generationen GZmax = 200
Erwarteter maximaler Fitnesswert: 20100
K-Punkt-Crossover mit K = 2
Verwendet wird die Plusstrategie
Jeweils werden 100 Versuche durchgefuehrt.

Statistik nach 100 Versuchen:
Mittlere Zahl der Fitnessberechnungen in jedem Versuch: 23918.400000000
Mittlere Zahl der erzeugten Individuen in jedem Versuch: 21535.030000000
Durchschnittliche Fitness am Ende jedes Versuchs: 20099.450000000
Durchschnittliche Varianz der Fitness jeweils am Versuchsende: 9.035169303
Mittlere Zahl an Generationen in jedem Versuch bis zum lokalen Optimum: 180.120000000
Insgesamt bestes Individuum aus allen Versuchen:
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Positionen der Nullen im besten Individuum:
Anzahl Nullen: 0, Anzahl Einsen: 200, Position der letzten Null: 0
Fitnesswert des besten Individuums: 20100.000000000

Generelle Kritik am bisherigen Vorgehen:

Auch wenn die Experimente im Mittel bei Wiederholungen ähnliche Ergebnisse liefern werden, so sind wir doch ziemlich kritiklos bezüglich des zu lösenden Problems vorgegangen. Wir haben einige Annahmen gemacht, vor allem:
- 200 Generationen sind bis zur Konvergenz des Verfahrens ausreichend.
- 100 Versuche genügen für eine statistisch ausreichende Anzahl, um halbwegs stabile Resultate zu erhalten.

Als erstes muss man sich jedoch immer überzeugen, wie die Fitnesslandschaft ungefähr beschaffen ist, wie die Konvergenzgeschwindigkeit ist und wie sich die Diversität mit den Generationen entwickelt. Weiß man hierüber nichts, so kann sich im Laufe der Experimente herausstellen, dass alle Messungen unbrauchbar sind, da sie unter falschen Annahmen durchgeführt wurden. In der Praxis bedeutet dies, dass mehrere Monate Arbeit überflüssig gewesen sein können.

Auch wir haben unsystematisch gearbeitet und hatten viel Glück, dass mit den neuen Standardwerten im Mittel 180 Generationen ausreichten, um das Optimum zu erreichen. Gleich zu Beginn unserer Messungen hätten wir feststellen müssen, dass die

Mittlere Zahl an Generationen in jedem Versuch bis zum lokalen Optimum

stets zwischen 194 und 197 liegt, also ganz dicht bei GZmax = 200. Dies deutet darauf hin, dass der Optimierungsprozess bei allen Versuchen am Ende noch nicht abgeschlossen war und man unverzüglich ein Experiment mit einem viel größeren GZmax hätte durchführen *müssen*. Auch hätten wir die durchschnittliche Varianz am Ende der Versuche beachten müssen, welche faktisch aussagte, dass die Individuen in der letzten Population bzgl. der Fitnesswerte sehr dicht beieinander lagen und dass es daher am Ende vermutlich viele Mehrfachexemplare in der Population gab (dies wäre zugleich die Erklärung, warum zum Ende hin die Konvergenz immer langsamer wurde, da die Rekombination zweier gleicher Exemplare nichts verändert und die Konvergenzgeschwindigkeit dann überwiegend von der Mutation abhängt). Denn schließlich geben die Messwerte keineswegs an, ob der Algorithmus das Optimum findet, sondern sie geben nur Hinweise darauf, mit welcher Geschwindigkeit sich der evolutionäre Algorithmus auf das Optimum zubewegt. Wir hätten also vor Beginn der Messreihen

- für die Konvergenzdauer mindestens ein Experiment mit den Standardwerten, aber mit GZmax = 500 durchführen müssen,

- eine Prozedur hinzufügen müssen, die feststellt, wie viele verschiedene Individuen am Ende in der Population vorhanden sind.

Was wäre hierbei eventuell anders abgelaufen? Wenn die Dauer bis zur Erreichung des Optimums deutlich größer als 200 ist, dann würden wir uns auf kleine Werte von Mue konzentrieren, um die Laufzeit gering zu halten; insbesondere hätten wir untersucht, ob Mue < 10 oder sogar Mue = 1 für den evolutionären Algorithmus bereits ausgereicht hätte. Wenn viele Individuen am Ende in der Population gleich sind, so würden wir doppelte Individuen entfernen oder beim Aufbau der nächsten Population doppelte Individuen von vornherein nicht zulassen. Wir untersuchen auf den nächsten 3 Folien, ob wir hier zu wenig gründlich vorgegangen sind (die Antwort wird "ja" lauten).

Testlauf der ursprünglichen Standardwerte, aber mit GZmax = 500:

Datum: 24.5.2009, Autor: VC, Version1: Summe A(i)*i maximieren, Standard-Werte, aber mit groesserem GZmax..
Eingabe: Nur die Fitnessfunktion. Ausgabe: Bestes Individuum.
Initialisierung: Zufaellig werden 30 Individuen erzeugt mit Wahrscheinlichkeit fuer Null = 0.500000000
Laenge der Individuen N = 200
Groesse der Population Mue = 30
Anzahl der Kinder Lambda = 150
Anzahl der ausgewaehlten Eltern AnzElt = 30
Teilnehmerzahl bei der Turnierselektion Q = 3
Crossoverwahrscheinlichkeit P_C = 0.900000000
Mutationswahrscheinlichkeit pro Bit P_Mut = 0.005000000
Maximale Zahl an Generationen GZmax = 500
Erwarteter maximaler Fitnesswert: 20100
K-Punkt-Crossover mit K = 2
Verwendet wird die Kommastrategie
Jeweils werden 100 Versuche durchgefuehrt.

Statistik nach 100 Versuchen:
Mittlere Zahl der Fitnessberechnungen in jedem Versuch: 74428.200000000
Mittlere Zahl der erzeugten Individuen in jedem Versuch: 66992.560000000
Durchschnittliche Fitness am Ende jedes Versuchs: 20055.890000000
Durchschnittliche Varianz der Fitness jeweils am Versuchsende: 59.490897321
Mittlere Zahl an Generationen in jedem Versuch bis zum lokalen Optimum: 422.700000000
Insgesamt bestes Individuum aus allen Versuchen:
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Positionen der Nullen im besten Individuum:
Anzahl Nullen: 0, Anzahl Einsen: 200, Position der letzten Null: 0
Fitnesswert des besten Individuums: 20100.000000000

Hier erkennt man, wie relativ schlecht die Standardwerte in Wahrheit sind. Zwar wird das Optimum gefunden, aber es werden im Mittel 423 Generationen bis zur Konvergenz benötigt, und der Wert 20055.89 zeigt, dass sehr oft das Optimum gar nicht erreicht wurde.

Die Prozedur Doppel auf der folgenden Folie berechnet die Zahl der verschiedenen Individuen in einer Population.

```
-- Berechne, wie viele verschiedene Elemente eine Population P mit LP Elementen besitzt.
  procedure Doppel (P: in Population; Lp: in Natural; -- P_ohne_Doppel: out Population;
                    L: out Natural) is
    AnzP: Natural := 0; RR: Real;
    Gleich: Boolean; K: Natural;
    Dopp: array(1..Lp) of Boolean := (others => False);
    -- Nach der ersten "for I ..."-Schleife gilt:  Dopp(J) ist true <=> es gibt ein I < J mit: I-tes Individuum in P = J-tes Individuum in P
    -- Die I mit Dopp(I) = false liefern alle nicht-doppelten Individuen. Es sind am Ende AnzP Stück.
  begin
    for I in 1..LP loop
      if not Dopp(I) then
        RR := P(I).Fit;
        for J in I+1 .. LP loop
          if (not Dopp(J)) and then (RR = P(J).Fit) then
            Gleich := True;  K := 1;
            while Gleich and then (K <= N) loop
              Gleich := P(I).Inhalt(K) = P(J).Inhalt(K);
              K := K+1;
            end loop;
            if Gleich then Dopp(J) := True; end if;
          end if;
        end loop;
      end if;
    end loop;
    for I in 1..LP loop
      if not Dopp(I) then
        AnzP := AnzP + 1;
        -- P_ohne_Doppel(AnzP) := P(I);  -- = P, aber ohne doppelte Individuen
      end if;
    end loop;
    L := AnzP;
  end;
```

Einzufügende Teile, damit die Zahl der verschiedenen Individuen ermittelt werden kann. (Man kann dies auch effizienter implementieren.)

Im Hauptprogramm, nach "declare":
```
Versch, Versch_Mittel: Natural;
```

Zu Beginn des Hauptprogramms:
```
Versch_Mittel := 0;
```

Im Hauptprogramm hinter *beiden* Aufrufen "Ein_Versuch":
```
Doppel(Pop,Mue,Versch);  -- Versch = Zahl der verschiedenen Individuen in Pop.
Versch_Mittel := Versch_Mittel + Versch;
```

In Schlussausdruck einfügen:
```
New_Line; Put("Mittlere Zahl der verschiedenen Individuen in der letzten Generation jedes Versuchs: ");
Put(Real(Versch_Mittel)/Real(AnzahlVersuche),Stellen_Vor_Komma(Real(Versch_Mittel)/Real(AnzahlVersuche)),9,0);
```

Testläufe Standard-Werte mit Zählung verschiedener Individuen (d.h. mit der Prozedur Doppel).
N = 200, Mue = 30, Lambda = 150, AnzElt = 30, Q = 3, P_C = 0.9, P_Mut = 1/N, KPCO = 2, WfN = 0.5, Kommastrategie, 2-Punkt-Crossover, jeder Wert wurde als bestes Individuum aus 100 Versuchen ermittelt

Lauf 1 mit GZmax = 200:
Statistik nach 100 Versuchen:
Mittlere Zahl der Fitnessberechnungen in jedem Versuch: 29878.200000000
Mittlere Zahl der erzeugten Individuen in jedem Versuch: 26898.170000000
Durchschnittliche Fitness am Ende jedes Versuchs: 19717.610000000
Durchschnittliche Varianz der Fitness jeweils am Versuchsende: 63.848730668
Mittlere Zahl an Generationen in jedem Versuch bis zum lokalen Optimum: 195.640000000
Mittlere Zahl der verschiedenen Individuen in der letzten Generation jedes Versuchs: 16.140000000
Insgesamt bestes Individuum aus allen Versuchen:
1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Positionen der Nullen im besten Individuum: 2 3 11 17 24 51
Anzahl Nullen: 6, Anzahl Einsen: 194, Position der letzten Null: 51
Fitnesswert des besten Individuums: 19992.000000000

Am Ende eines Versuchs sind im Mittel **16 von 30** Individuen der Population Doppel von anderen Individuen.

Lauf 2 mit GZmax = 500:
Statistik nach 100 Versuchen:
Mittlere Zahl der Fitnessberechnungen in jedem Versuch: 74428.200000000
Mittlere Zahl der erzeugten Individuen in jedem Versuch: 66994.540000000
Durchschnittliche Fitness am Ende jedes Versuchs: 20055.960000000
Durchschnittliche Varianz der Fitness jeweils am Versuchsende: 58.492596251
Mittlere Zahl an Generationen in jedem Versuch bis zum lokalen Optimum: 420.240000000
Mittlere Zahl der verschiedenen Individuen in der letzten Generation jedes Versuchs: 14.830000000
Insgesamt bestes Individuum aus allen Versuchen:
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Positionen der Nullen im besten Individuum:
Anzahl Nullen: 0, Anzahl Einsen: 200, Position der letzten Null: 0
Fitnesswert des besten Individuums: 20100.000000000

Am Ende eines Versuchs sind im Mittel **15 von 30** Individuen der Population Doppel von anderen Individuen.

Rund die Hälfte der Individuen in der letzten Generation sind also Doppelgänger. Dies ist ein zu hoher Prozentsatz, so dass man sich Maßnahmen überlegen muss, um diese Individuen zu entfernen oder gar nicht erst aufzunehmen. *Machen Sie sich hierzu Gedanken!* An welchen Stellen muss was verändert werden, welcher zusätzliche Aufwand ist dann erforderlich und wie stark kann man die Anzahl der Fitnessberechnungen verringern?

Fazit unserer Überlegungen: Hinweise zum Vorgehen beim Einsatz eines Evolutionären Algorithmus.

Wir haben Folgendes erkannt:

Bei *allen* evolutionären Verfahren ermittele man stets die Konvergenzgeschwindigkeit und die Diversität und Varianz in den Populationen, unterscheide hierbei zwischen dem Beginn des Verfahrens, einem Mittelbereich und dem Verhalten, wenn sich das Verfahren lokalen Optima bereits stark genähert hat. Erst nach diesen Untersuchungen beginne man, die Messreihen (= Experimente) zu planen und zu installieren. In der Regel müssen zu verschiedenen Phasen des Verfahrens auch verschiedene Strategien angewandt werden. Typischerweise liegt bei frühen Generationen die Betonung auf der Rekombination und bei den späteren Generationen, also gegen Ende des Verfahrens, auf der Mutation.

Legen Sie bei den Tests "leichte Problemfälle", "durchschnittliche Problemfälle" und "besonders hartnäckige Problemfälle" fest und speichern Sie dies als Testfälle zusammen mit den zwischenzeitlich gefundenen bisher besten Lösungen (Anlage einer Test-Datenbank; Benchmarks). *Merke:* Ohne gute Systematik und ohne gute Dokumentation sollte man gar nicht erst starten, sondern dann genügt es, auf Gut-Glück herumzuprobieren.

Wenn Sie eine systematische Vorgehensweise wählen, dann erarbeiten Sie sich eine eigene Vorgehensweise. Hierbei sollten Sie die folgenden Fragen einbeziehen. (Die Auswahl dieser 8 Fragen ist nicht "erschöpfend".)

Frage 1: Wie groß ist die "Hoffnungslosigkeit", ein gutes Optimum zufällig zu finden?
Um dies beantworten zu können, erzeuge man zufällig viele / einige Millionen Individuen und schätze die Qualität des besten Individuums ab. Wenn man die Größenordnung des Optimums kennt, dann kann man abschätzen, ob es sich vielleicht lohnt, statt eines heuristischen Verfahrens eine reine Zufallssuche zu implementieren. Meist sind allerdings die Individuenbereiche derart groß, dass man durch eine Zufallssuche faktisch nichts erreicht.

Frage 2: Gibt es "große breite Optima"?
In unserem Beispiel gab es nur ein großes breites Optimum. Um einem Problem ansehen zu können, ob es "gutmütig" ist, also nur breite Optima besitzt, implementiere man stets das Verfahren des (steilsten) Anstiegs, auch Hill Climbing genannt. Hier gibt es mehrere Varianten, von denen man - wenn möglich - das MaxHC (maximales Hill Climbing) programmieren sollte; man braucht es ohnehin nach Durchführung des evolutionären Verfahrens zur Nachbesserung der gefundenen Optima.

Frage 3: Wie hilfreich ist eine "echte Population"?
Viele Probleme lassen sich auch gut mit einer "einelementigen Population", also mit nur einem Individuum (und ggfls. mehreren Kindern, mindestens aber einem) behandeln.Im Anschluss an Experimente mit dem Hill Climbing prüfe man, ob ein Metropolis, Simulated Annealing, Threshold, Sintflut, ... bereits ausreichend wäre.

Frage 4: Kann man große zusammenhängende Gebiete im Individuenbereich identifizieren, in denen wahrscheinlich kein Optimum liegen kann?
Hierzu durchstöbere man den Individuenbereich in großen Schritten. Dort, wo schlechte Fitnesswerte sind, analysiere man das Gebiet etwas genauer. Stellen sich hierbei nur recht schlechte Werte heraus, so markiere man das Gebiet (z.B., indem man einen Vektor, der ungefähr in die Mitte des Gebiets zeigt, merkt). Die Menge solcher Vektoren bildet eine Tabu-Menge: Individuen, die ihnen zu nahe kommen, werden sofort verworfen. Im Laufe des Verfahrens kann man automatisch weitere Tabu-Bereiche entdecken und speichern.

Frage 5: Wie justiert man die Parameter?
In der Praxis geht man gerne sequenziell vor: Man versucht, einen "Standard"-Parametersatz zu finden, der einigermaßen brauchbar ist. Dann versucht man herauszufinden, von welchen Parametern die Güte der Lösungen am stärksten abhängt; dies liefert eine Bedeutungs-Reihenfolge der Parameter. Dann justiert man ausgehend von dem Standard-Parametersatz zuerst den wichtigsten Parameter und übernimmt dessen besten Wert in den Standard-Parametersatz. Hiermit justiert man anschließend den zweitwichtigsten Parameter usw.
(Wir sind anders vorgegangen: Wir haben alle Parameter festgehalten und immer genau einen variiert; für jeden Parameter ergab es jeweils ein recht guter Wert. Diese Werte bilden dann den neuen Standardparametersatz, mit dem man genauso verfährt usw. Dies dauert länger, ist aber flexibler und kann Fehlentscheidungen korrigieren.)
Schließlich überlege man stets, ob man die Justierung der Parameter nicht selbst wieder durch einen evolutionären Algorithmus vornehmen lässt. Jeder Parametersatz ist ein Individuum und die Fitness ist die Eignung dieses Parametersatzes zur Lösung des vorgegebenen Problems.

Frage 6: Wie kann man die Diversität aufrecht erhalten?
Man muss in regelmäßigen Abständen die Population überprüfen, damit sie nicht frühzeitig zu einem Optimum hin konvergiert. Droht dies, so reduziere man dynamisch die Fitness für alle Individuen, die ein "Cluster" bilden (die also zu nahe beieinander sind oder auf das gleiche Optimum zulaufen). Man kann auch Abänderungen vornehmen oder Mehrfachexemplare durch zufällig erzeugte neue Individuen ersetzen.

Frage 7: Wie kann man die Konvergenzgeschwindigkeit erhöhen?
Einen Hinweis gibt die "1/5-Regel". Generell sollte man die Mutationswahrscheinlichkeit flexibel halten und "Richtungen", in die sich Individuen mit wachsender Fitness bewegen, identifizieren und dann fördern.

Frage 8: Welchen Einfluss hat die Initialisierung (Population 0)?
Eine sehr großen! Meist versucht man, die Individuen der Startpopulation gleichmäßig über den Individuenbereich zu verteilen. Oft verwendet man andere Heuristiken, um eine Startpopulation zu erhalten, die bereits gute Individuen enthält. Allerdings sollte man erfahrungsgemäß Pop_0 nicht mit (allzu) deterministischen Verfahren erzeugen.