Teil III

Anhang

Anhang A

Die Experimentalumgebung

In diesem Teil des Anhangs wird die Experimentalumgebung dargestellt. Diese Experimentalumgebung besteht hauptsächlich aus einem Programm und der hinzugehörigen Hardwareumgebung. Diese beiden Komponenten sollen nun dargestellt werden.

A.1 Hardwareumgebung

Das Programm sollte unabhängig von einer speziellen Hardwareumgebung lauffähig sein. Um jedoch eine problemlose Implementierung auf einem anderen Rechner zu gewährleisten, muß die Umgebung die folgenden Regeln erfüllen.

Das angewandte Betriebssystem muß ein Unix System V sein, das den Normen der X/Open Gruppe, insbesondere der Bände 1 - 4 und 6 - 7 erfüllen[XOp89]. Als graphische Oberfläche wurde für das System X11 gewählt und benutzt.

Die Experimente wurden auf einem Rechner mit einem 80386SX Prozessor mit 16 MHz Taktfrequenz durchgeführt.

A.2 Die Software

Die Software besteht aus 13 Moduln. Diese Moduln werden im allgemeinen über eine festgelegte Schnittstelle angesprochen. Die Software wurde in der Programmiersprache `C` entwickelt und hält sich ohne Ausnahme an die Sprachdefinition von KERNIGHAM und RITCHIE[Ker88]. Diese Programmiersprache bietet sich bei der gewählten Umgebung an, da hier die meisten Resourcen zur Verfügung stehen.

Das Makefile

Das Makefile zeigt die Abhängigkeiten zwischen den einzelnen Moduln. Es wird von dem Utility "**make**" gelesen und ausgewertet. Sollte ein Modul nach der letzten Übersetzung verändert worden sein, so werden alle Moduln, die von diesem Modul abhängig sind, neu übersetzt.

Durch die Benutzung dieser Datei wird ein Programm mit dem Namen "**genetics**" erzeugt, das unter X11 gestartet werden kann.

```
LIB=-lXm -lXt -lX11 -lPW -lnsl_s -lnet -lc -lbsd -lcurses -lm
OBJ=Hauptprogram.o Zeige_Baum.o Zufall.o Initial.o Kontrolle.o Bewert.o \
    Fpflanz.o Basicwin.o Benutz.o Ordnung.o domenu.o
genetic: ${OBJ}
        cc -O ${OBJ} -o genetics ${LIB}
Hauptprogram.o: Hauptprogram.c Zeige_Baum.c Benutz.c genetics.h
Benutz.o:       Benutz.c  domenu.c domenu.h genetics.h
Zeige_Baum.o:   Zeige_Baum.c genetics.h
Initial.o:      Initial.c Zeige_Baum.c Zufall.c genetics.h
Kontrolle.o:    Kontrolle.c genetics.h
Bewert.o:       Bewert.c genetics.h
Fpflanz.o:      Fpflanz.c Zufall.c genetics.h
Basicwin.o:     Basicwin.c genetics.h
Zufall.o:       Zufall.c genetics.h
domenu.o:       domenu.c domenu.h
Ordnung.o:      Ordnung.c genetics.h
```

Globale Variablen

Dieser Abschnitt definiert Makros und globale Variablen für das gesamte Programm. Die Definitionen sind nach Anwendungsgebieten sortiert.

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Modul: genetics.h                          Version 1.0            */
/*                                                                   */
/* Globale Variablendefinitionen fuer GENETICS                       */
/*                                                                   */
/* Datum : 31.August 1991                                            */
/*                                                                   */
/*-------------------------------------------------------------------*/

/* Standardincludes */
#include <X11/Xlib.h>
#include <X11/Xutil.h>
```

```
#include <X11/Xos.h>
#include <X11/Xatom.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/MessageB.h>
#include <Xm/PushB.h>

#include <stdio.h>
#include <curses.h>

/* Globale Festlegungen */
#define BITMAPDEPTH     1
#define TOO_SMALL       0
#define BIG_ENOUGH      1

#define XKOORDINATE     0
#define YKOORDINATE     1

#define WAHR            1
#define FALSCH          0
#define BESTER          0
#define MITTEL          1
#define SCHLECHT        2

#define FIRST_TIME      0
#define NORMAL_USE      1
#define LAST_TIME       2
#define FEHLER          3

#define BREITENSUCHE    0
#define TIEFENSUCHE     1
#define ARIADNE         2

#define MaxNachfolger           2
#define MaxIndividuen           50
#define MaxIndividuenHalbe      25
int             NoCurses;
int             NoX11;
int             MaxGeneration;
int             MaxKnoten;
int             MaxWesen;
int             MaxWesenHalbe;
long float      M_Wahrschein;
```

```
float           Fit_Verteil;

/* Definitionen fuer den Zufallsgenerator */
long            Zufall_r[97];
short           Zufall_iff;

/* Definitionen fuer X11 */
Display         *display;
GC              FB_Grafik, FS_Grafik, FbB_Grafik;
XEvent          Aktion;
XSizeHints      Baum_Size, Statistik_Size;
XIconSize       *size_list;
XFontStruct     *font_info;
unsigned int    icon_width, icon_height;
Window          F_Statistik, F_Baum, Fb_Baum;

/* Daten fuer das Statistik- Fenster */
unsigned int    FS_Hoehe, FS_Breite, Org_FS_Hoehe, Org_FS_Breite;

/* Daten fuer das Baum-Fenster */
unsigned int    FB_Hoehe, FB_Breite, Org_FB_Hoehe, Org_FB_Breite;
int             FB_aktiv;

/* Daten fuer den besten Baum */
unsigned int    FbB_Hoehe, FbB_Breite, Org_FbB_Hoehe, Org_FbB_Breite;
unsigned int    border_width, display_width, display_height;
int             window_size;

short           Generation, Programmende;
int             GesamtFitness, Beste_Fitness, Best_Wesen;
int             Korrektur_Ordnung, Korrektur_Topologie;

int Individuum[100][100][MaxIndividuen];
int Root[MaxIndividuen];
int Tiefe[100][MaxIndividuen], Beste_Tiefe[100];
int Fitness[MaxIndividuen];
int Statistik[100][3];
int Best_Individuum[100][100];
int Ordnungszahl[MaxIndividuen];
```

A.2.1 Die Hauptroutinen

Die Hauptroutinen stellen das Kernstück des programmierten Systems dar. Dieses Kernstück umfasst das Hauptprogramm sowie die Moduln zur Bewertung

und Fortpflanzung.

Das Hauptprogramm

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Modul: Hauptprogram.c                          Version 1.0              */
/*                                                                         */
/* Hauptprogramm fuer GENETICS                                             */
/*                                                                         */
/* Datum :                                                                 */
/*                                                                         */
/*-------------------------------------------------------------------------*/

#include "genetics.h"

void GA_Haupt_Run_Genetics()
  {
  unsigned int Alt_FS_Hoehe, Alt_FS_Breite;
  while(Programmende == FALSCH)
      {
       if (Generation < MaxGeneration)
         {
         GA_Kontrolle_main();
         GA_Ordnung_main();
         GA_Bewert_main();
         GA_Fpflanz_main();
         GA_Zeige_Baum_Zeichne_Statistik();
         Generation++;
         }
       XCheckMaskEvent(display, ExposureMask | ButtonPressMask |
                       StructureNotifyMask, &Aktion);
       switch (Aktion.type) {
       case Expose:
                    if (Aktion.xexpose.count != 0)
                        break;
                    GA_Basicwin_Initialisierung_Grafik();
                    GA_Zeige_Baum_Zeichne_Statistik_neu();
                    GA_Zeige_Baum_Zeichne_Baum();
                    break;
       case ConfigureNotify:
                    FS_Breite = Aktion.xconfigure.width;
                    FS_Hoehe = Aktion.xconfigure.height;
```

```
                    if ((FS_Breite != Alt_FS_Breite) ||
                        (FS_Hoehe  != Alt_FS_Hoehe))
                      {
                      Alt_FS_Breite = FS_Breite;
                      Alt_FS_Hoehe = FS_Hoehe;
                      XClearWindow (display,F_Statistik);
                      }
                    break;
        case ButtonPress:
                    GA_Benutzerschnittstelle (NORMAL_USE, 0, NULL);
                    break;
        default:
                    break;
        }
    }
  }

main(argc,argv)
unsigned int    argc;
char            **argv;
  {
/* Initialisierung des Servers */
  Programmende = FALSCH;
  NoCurses = FALSCH;
  GA_Benutzerschnittstelle (FIRST_TIME, argc, argv);
  GA_Initial_Wesen();
  GA_Basicwin_Initialisierung_Fenster_Baum();
  GA_Basicwin_Initialisierung_Fenster_bester_Baum();
  GA_Basicwin_Initialisierung_Fenster_Statistik();
  GA_Haupt_Run_Genetics();
  }
```

Die folgenden Moduln stellen die beiden Phasen im Ablauf der *Genetischen Algorithmen* dar.

Das Bewertungsmodul

In diesem Modul werden die Funktionen der Bewertungsphase dargestellt.

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Modul: bewert.c                          Version 1.0              */
/*                                                                   */
/* Bewertungsabschnitt fuer GENETICS                                 */
```

```
/*                                                                    */
/* Datum : 31.August 1991                                             */
/*                                                                    */
/*--------------------------------------------------------------------*/

#include <stdio.h>
#include "genetics.h"

int GA_Bewert_addiere_Tiefe(Wesen)
int Wesen;
  {
  int x, sum = 0;
  for (x = 0; x < MaxKnoten; x++)
    sum += Tiefe[x][Wesen];
  return(sum);
  }

int GA_Bewert_bewertete_Tiefe(Wesen)
int Wesen;
  {
  int x, sum = 0;
  for (x = 0; x < MaxKnoten; x++)
    sum += MaxKnoten + 1 - Tiefe[x][Wesen];
  return(sum);
  }

int GA_Bewert_MaxOrdnung()
  {
  int x = 0, sum = 0, y = 0;
  for (y = 0; y < MaxKnoten; y++)
    for (x = y; x < MaxKnoten; x++)
      sum++;
  return(sum);
  }

void GA_Bewert_main()
  {
/* OGF - optimale Gesamtfitness
   OEF - optimale Einzelfitness
   PGF - Prozent der Gesamtfitness*/
  int x, y, z, OGF = 0, OEF = 400, MaxWert = 0;
  int BaumFoerderung = 0, Bester_Baum = 0;
  int MinWert;
  double Prozent, PGF, q = 1, p = 0, PMaxWert, PMinWert;
```

```
  float TopologischeFitness, OrdnungsFitness, OEF1, OEF2;
  GesamtFitness = 0;
  OEF1 = (float)((MaxKnoten * MaxKnoten) + BaumFoerderung);
  OEF2 = (float)(GA_Bewert_MaxOrdnung());
  OGF = OEF * MaxWesen;
  MinWert = OEF;
  for (x = 0; x < MaxWesen; x++)
    {
/* Fitnessfunktion Mittlere Weglaenge */
    if (Root[x] < MaxWesen + 1)
      TopologischeFitness = TopologischeFitness + BaumFoerderung;
    if (Root[x] != MaxWesen + 1)
    TopologischeFitness = (int)((OEF1 - GA_Bewert_addiere_Tiefe(x)) *q +
                          (OEF1 - GA_Bewert_bewertete_Tiefe(x)) *p);
    else
      TopologischeFitness = 0;
/* Fitnessfunktion Ordnung */
    OrdnungsFitness = Ordnungszahl[x];
    if ((OrdnungsFitness < 0) || (OrdnungsFitness > OEF2))
      OrdnungsFitness = 0;
/* Skalieren und addieren der Fitnesswerte */
    TopologischeFitness = (TopologischeFitness / OEF1) * 400;
    OrdnungsFitness = (OrdnungsFitness / OEF2) * 400;
    Fitness[x] = (int)(Fit_Verteil * TopologischeFitness + (1 - Fit_Verteil) *
                       OrdnungsFitness);
    GesamtFitness += Fitness[x];
    if (Fitness[x] > Beste_Fitness)
      {
      Best_Wesen = x;
      Beste_Fitness = Fitness[x];
      for (y = 0; y < MaxKnoten; y++)
        for (z = 0; z < MaxKnoten; z++)
          Best_Individuum[y][z] = Individuum[y][z][x];
      for (y = 0; y < MaxKnoten; y++)
        Beste_Tiefe[y] = Tiefe[y][x];
      GA_Zeige_Baum_Zeichne_Baum();
      }
    if (Fitness[x] > MaxWert)
       {
       MaxWert = Fitness[x];
       Bester_Baum = x;
       }
    if (Fitness[x] < MinWert)
       MinWert = Fitness[x];
```

```
    }
  GA_Zeige_Baum_Zeichne_bester_Baum(Bester_Baum);
  Statistik[Generation][BESTER]  = MaxWert;
  Statistik[Generation][MITTEL]  = GesamtFitness / MaxWesen;
  Statistik[Generation][SCHLECHT]= MinWert;
  PGF = (float)GesamtFitness / (float)OGF * 100;
  PMinWert = ((float)MinWert / (float)GesamtFitness) * 100;
  PMaxWert = ((float)MaxWert / (float)GesamtFitness) * 100;
  GA_Benutz_Statusausgabe(Generation, OGF, GesamtFitness, PGF,
                          Beste_Fitness, MaxWert, MinWert, PMaxWert,
                          PMinWert);
/* Ausgabe ohne Curses */
  if (NoCurses == WAHR)
    {
    system("tput clear");
    printf("Generation : %i \n",Generation);
    printf("Auswertung der Baeume :\n");
    printf(" Optimale Werte :\n");
    printf(" Gesamtfitness (topologisch): %i, Einzelfitness: %i \n",OGF,OEF);
    printf(" erreichte Fitness:           %i\n",Beste_Fitness);
    printf(" Gesamtfitness (topologisch): %i -- %2.2f v.H.\n",GesamtFitness,PGF);
    printf("Einzelwerte :\n");
    for (x = 0; x < MaxWesen; x++)
      {
      Prozent = ((float)Fitness[x] / (float)GesamtFitness) * 100;
      printf(" Wesen %2i : Fitness = %i -- %2.2f v.H.\n",x,Fitness[x],Prozent);
      }
    }
/*  print_best_tree(); */
  }
```

Modul zur Kontrolle der Ordnung

In diesem Modul wird die Ordnung in dem Baum kontrolliert und protokolliert.
Das Ergebnis wird im Bewertungsmodul beurteilt.

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Modul: Ordnung.c                          Version 1.0             */
/*                                                                   */
/* Feststellung der Ordnung des Graphen                              */
/*                                                                   */
/* Datum :                                                           */
```

```
/*                                                                    */
/*--------------------------------------------------------------------*/

#include "genetics.h"

void GA_Ordnung_Initial()
  {
  int x;
  for (x = 0; x < MaxWesen; x++)
    Ordnungszahl[x] = 0;
  }

void GA_Ordnung_Feststellen(Wesen)
int Wesen;
  {
  int x, y;
  for (x = 0; x < MaxKnoten; x++)
    for (y = 0; y < MaxKnoten; y++)
      {
      if ((Individuum[x][y][Wesen] == 1) && (x < y))
        Ordnungszahl[Wesen]++;
      if ((Individuum[x][y][Wesen] == 1) && (x > y))
        {
        Ordnungszahl[Wesen]--;
        if (Korrektur_Ordnung == WAHR)
          Individuum[x][y][Wesen] = 0;
        }
      }
  if (Ordnungszahl[Wesen] < 0)
    Ordnungszahl[Wesen] = 0;
  }

void GA_Ordnung_main()
  {
  int x;
  GA_Ordnung_Initial();
  for (x = 0; x < MaxWesen; x++)
    GA_Ordnung_Feststellen(x);
  }
```

Das Kontrollmodul

In diesem Abschnitt werden die verschiedenen internen Kontrollen und Prüfungen durchgeführt. Außerdem werden mehrere interne Tabellen, wie z.B. die Tie-

fentabelle, besetzt. Dieses Modul ist für die Bewertung der Individuen notwendig.

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Modul: Kontrolle                              Version 1.0               */
/*                                                                         */
/* Kontrollfunktionen im Ablauf                                            */
/*                                                                         */
/* Datum :                                                                 */
/*                                                                         */
/*-------------------------------------------------------------------------*/
#include "genetics.h"
int Neu_Individuum[100][100][MaxIndividuen];

int GA_Kontrolle_ist_Wurzel(Nachfolger,Wesen)
int Nachfolger, Wesen;
  {
  int x, sum = 0;
  for (x = 0; x < MaxKnoten; x++)
    {
    sum = sum + Neu_Individuum[x][Nachfolger][Wesen];
    }
  if (sum == 0)
    return (WAHR);
  else
    return (FALSCH);
  }


void GA_Kontrolle_ist_Baum (Wesen)
int Wesen;
  {
  int x, y, AnzWurzel = 0, AnzKnoten, Wurzel, lokale_Tiefe;
/* Initialisierung der Tiefentabelle */
  for (x = 0; x < MaxKnoten; x++)
    Tiefe[x][Wesen] = MaxKnoten + 1;
/* Finden der Wurzel */
  for (x = 0; x < MaxKnoten; x++)
    if (GA_Kontrolle_ist_Wurzel (x, Wesen) == WAHR)
       {
       AnzWurzel++;
       Tiefe[x][Wesen] = 0;
```

```
       Root[Wesen] = x;
       }
/* Wenn weniger als eine Wurzel vorhanden ist, dann ist es ein Graph,
   wenn genau eine Wurzel vorhanden ist, dann ist es ein Baum
   ansonsten ist es ein Wald */
  if (AnzWurzel < 1)
    Root[Wesen] = MaxWesen + 1;
  else
    {
    if (AnzWurzel == 1)
      {
      Wurzel= Root[Wesen];
      lokale_Tiefe = 1;
      for (x = 0; x < MaxKnoten; x++)
        if (Neu_Individuum[Wurzel][x][Wesen] ==  1)
          Tiefe[x][Wesen] = lokale_Tiefe;
      }
    else
      {
      Root[Wesen] = MaxWesen + 2;
      for (x = 0; x < MaxKnoten; x++)
        if (Tiefe[x][Wesen] == 0)
          Tiefe[x][Wesen] = 1;
      }
    }
/* naechste Schicht bearbeiten */
  AnzKnoten = 0;
  lokale_Tiefe = 1;
  do {
    AnzKnoten = 0;
    for (x = 0; x < MaxKnoten; x++)
      if (Tiefe[x][Wesen] == lokale_Tiefe)
        {
        AnzKnoten++;
        for (y = 0; y < MaxKnoten; y++)
          if (Neu_Individuum[x][y][Wesen] == 1)
            if  (Tiefe[y][Wesen] == MaxKnoten + 1)
              Tiefe[y][Wesen] = lokale_Tiefe + 1;
            else
              Neu_Individuum[x][y][Wesen] = 0;
/* Wenn ein Knoten von mehr als einer Wurzel angesprochen wird, dann wird
die entsprechende tiefere Verbindung gekappt, da sie eine Rueckfuehrung
darstellt. */
       }
```

```
    lokale_Tiefe++;
    } while (AnzKnoten > 0);
  }

int GA_Kontrolle_Rest_Graph (Wesen)
  int Wesen;
  {
  int x, y;
  for (x = 0; x < MaxKnoten; x++)
    {
    if (Tiefe[x][Wesen] == MaxWesen + 1)
      {
      for (y = 0; y < MaxKnoten; y++)
        Neu_Individuum[y][x][Wesen] = 0;
      return (WAHR);
      }
    }
  return (FALSCH);
  }

void GA_Kontrolle_main()
  {
  int Wesen, x, y, z;
  for (x = 0; x < MaxKnoten; x++)
    for (y = 0; y < MaxKnoten; y++)
      for (z = 0;  z < MaxWesen; z++)
        Neu_Individuum[x][y][z] = Individuum[x][y][z];
  for (Wesen = 0; Wesen < MaxWesen; Wesen++)
    {
    GA_Kontrolle_ist_Baum (Wesen);
    while (GA_Kontrolle_Rest_Graph (Wesen) == WAHR)
      GA_Kontrolle_ist_Baum (Wesen);
/*      printf("Graph: %i Nicht vollstaendig\n",Wesen); */
    }
/*  for (x = 0; x < MaxKnoten; x++) printf(" %2i ",x); printf("\n");
  for (Wesen = 0; Wesen < MaxWesen; Wesen++)
    {
    for (x = 0; x < MaxKnoten; x++)
      printf(" %2i ",Tiefe[x][Wesen]);
    printf("\n");
    }
  printf("\n");*/
  }
```

Das Fortpflanzungsmodul

Dieses Modul beinhaltet die Prozeduren zur Fortpflanzung der Individuen.

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Modul: Fpflanz                                  Version 1.0             */
/*                                                                         */
/* Fortpflanzungsphase                                                     */
/*                                                                         */
/* Datum : 12.11.1991                                                      */
/*                                                                         */
/*-------------------------------------------------------------------------*/
#include "genetics.h"

void GA_Fpflanz_Roulette()
  {
  int Wheel[100], Prozent, t, u, x, y = 0, z;
  int Neu_Individuum[100][100][MaxIndividuen];
/* Vorbereiten des Roulette- Rades */
  for (x = 0; x < MaxWesen; x++)
    {
    Prozent = (int)(((float)Fitness[x] / (float)GesamtFitness) * 100);
    for (z = 0; z < Prozent; z++)
      Wheel[y++] = x;
    }
/* Werfen des Roulette- Rades */
  for (x = 0; x < MaxWesen; x++)
    {
    z = GA_Zufall_main() % y;
    for (t = 0; t < MaxKnoten; t ++)
      for (u = 0; u < MaxKnoten; u++)
        Neu_Individuum[t][u][x] = Individuum[t][u][Wheel[z]];
    }
/* Uebertragen der Ergebnisse */
/* ohne Generation Gapping
  for (x = 0; x < MaxWesen; x++) */
/* mit Generation Gapping */
  for (t = 0; t < MaxKnoten; t++)
    for (u = 0; u < MaxKnoten; u++)
      Individuum[t][u][0] = Individuum[t][u][Best_Wesen];
  for (x = 1; x < MaxWesen; x++)
    for (t = 0; t < MaxKnoten; t++)
      for (u = 0; u < MaxKnoten; u++)
```

```
        Individuum[t][u][x] = Neu_Individuum[t][u][x];
  }

void GA_Fpflanz_Paarung()
  {
  int Paare[MaxIndividuenHalbe], t, u, x, y = 0, z, Besetzt;
  int Neu_Individuum[100][100][MaxIndividuen];
  for (x = 0; x < MaxWesenHalbe; x++)
    Paare[x] = MaxWesen;
  for (x = 0; x < MaxWesenHalbe; x++)
    {
    t = (GA_Zufall_main() % MaxWesenHalbe) + MaxWesenHalbe;
    do {
      Besetzt = FALSCH;
      for (y = 0; y < x; y++)
        if (Paare[y] == t)
          {
          Besetzt = WAHR;
          t = (GA_Zufall_main() % MaxWesenHalbe) + MaxWesenHalbe;
          }
      } while (Besetzt == WAHR);
    Paare[x] = t;
    }
  for (x = 0; x < MaxWesenHalbe; x++)
    {
    for (y = 0; y < MaxKnoten; y++)
      for (z = 0; z < MaxKnoten; z++)
        {
        Neu_Individuum[y][z][x] = Individuum[y][z][x];
        Neu_Individuum[y][z][Paare[x]] = Individuum[y][z][Paare[x]];
        }
    t = GA_Zufall_main() % MaxKnoten;
    u = GA_Zufall_main() % MaxKnoten;
    for (y = 0; y < t; y++)
      for (z = 0; z < u; z++)
        {
        Neu_Individuum[y][z][x] = Individuum[y][z][Paare[x]];
        Neu_Individuum[y][z][Paare[x]] = Individuum[y][z][x];
        }
    for (y = t; y < MaxKnoten; y++)
      for (z = 0; z < MaxKnoten; z++)
        {
        Neu_Individuum[y][z][x] = Individuum[y][z][Paare[x]];
        Neu_Individuum[y][z][Paare[x]] = Individuum[y][z][x];
```

```
        }
    }
/* Uebertragen der Ergebnisse */
  for (x = 0; x < MaxWesen; x++)
    for (y = 0; y < MaxKnoten; y++)
      for (z = 0; z < MaxKnoten; z++)
        Individuum[y][z][x] = Neu_Individuum[y][z][x];
  }

void GA_Fpflanz_Mutation()
  {
  long float Elemente;
  int Nr, x, y, z;
  Elemente = MaxKnoten * MaxKnoten * MaxWesen;
  Elemente *= M_Wahrschein;
  if (Elemente > 0)
    for (Nr = 0; Nr < Elemente; Nr++)
      {
      x = GA_Zufall_main() % MaxKnoten;
      y = GA_Zufall_main() % MaxKnoten;
      z = GA_Zufall_main() % MaxWesen;
      if (Individuum[x][y][z] == 0)
        Individuum[x][y][z] = 1;
      else
        Individuum[x][y][z] = 0;
      }
  }

void GA_Fpflanz_main()
  {
  GA_Fpflanz_Roulette();
  GA_Fpflanz_Paarung();
  GA_Fpflanz_Mutation();
  }
```

Das Initialisierungsmodul

In diesem Abschnitt werden die verschiedenen Individuen initialisiert. Bei dieser Initialisierung werden Bäume erzeugt.

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Modul: Initial                          Version 1.0                     */
```

```
/*                                                                    */
/* Initialisierung der Adjazenzmatrix                                 */
/*                                                                    */
/* Datum : 20.Juli 1991                                               */
/*                                                                    */
/*--------------------------------------------------------------------*/
#include <stdio.h>
#include "genetics.h"

/* ein Element ist genau dann Blatt, wenn es nur Vorgaenger hat */
int GA_Initial_ist_Blatt(Wurzel,Wesen)
int Wurzel, Wesen;
  {
  int x, sum = 0;
  for (x = 0; x < MaxKnoten; x++)
     sum = sum + Individuum[Wurzel][x][Wesen];
  if (sum == 0)
     {
     for (x = 0; x < MaxKnoten; x++)
        sum = sum + Individuum [x][Wurzel][Wesen];
     if (sum != 0)
        return (WAHR);
     }
  return (FALSCH);
  }

/* ein Element ist genau dann Wurzel, wenn es keine Nachfolger hat */
int GA_Initial_ist_Wurzel(Nachfolger,Wesen)
int Nachfolger, Wesen;
  {
  int x, sum = 0;
  for (x = 0; x < MaxKnoten; x++)
     {
     sum = sum + Individuum[x][Nachfolger][Wesen];
     }
  if (sum == 0)
     return (WAHR);
  else
     return (FALSCH);
  }

/* ein Element ist genau dann frei, wenn es weder Wurzel noch Knoten ist */
int GA_Initial_ist_freies_Element(Wesen)
int Wesen;
```

```
{
  int x, y, sum = 0;
  for (x = 0; x < MaxKnoten; x++)
    {
    for (y = 0; y < MaxKnoten; y++)
      sum = sum + Individuum[x][y][Wesen] + Individuum[y][x][Wesen];
    if (sum == 0)
      return(WAHR);
    else
      sum = 0;
    }
  return (FALSCH);
}

void GA_Initial_Baue_Baum(Wesen)
int Wesen;
{
  int Wurzel, Nachfolger, x;
/* Suche eine Wurzel */
  Wurzel = GA_Zufall_main() % MaxKnoten;
  Root[Wesen] = Wurzel;
/* TESTLAUF
  printf("Wurzel des Baumes: %i\n",Wurzel);
*/
  do {
/* Suche Nachfolger entsprechend eingestellter Rang */
    Nachfolger = Wurzel;
    for (x = 0; x < MaxNachfolger; x++)
      {
       while ((Nachfolger == Wurzel) ||
              (GA_Initial_ist_Wurzel(Nachfolger,Wesen) == FALSCH))
         Nachfolger = GA_Zufall_main() % MaxKnoten;
       if (Root[Wesen] != Nachfolger)
         {
/* TESTLAUF
          printf("Nachfolger : %i\n",Nachfolger);
*/
          Individuum[Wurzel][Nachfolger][Wesen]=1;
         }
       Nachfolger = Wurzel;
      }
/* Neue Wurzel suchen */
    while (GA_Initial_ist_Blatt(Wurzel,Wesen) == FALSCH)
        Wurzel = GA_Zufall_main() % MaxKnoten;
```

```
/* TESTLAUF
    printf("Neue Wurzel: %i \n",Wurzel);
*/
  } while (GA_Initial_ist_freies_Element(Wesen) == WAHR);
}

void GA_Initial_Wesen()
  {
   int x,y,z;
/* Vorinitialisierung der Individuen */
   Beste_Fitness = 0;
   for (z = 0; z < MaxWesen; z++)
     {
     for (x = 0; x < MaxKnoten; x++)
       for (y = 0; y < MaxKnoten; y++)
         Individuum[x][y][z] = 0;
     }
/* Aufbau der Baeume */
   for (z = 0; z < MaxWesen; z++)
     {
     GA_Initial_Baue_Baum(z);
/* Testausgabe der erzeugten Baeume
     printf("\n");
     print_tree(z); */
     }
  }
```

A.2.2 Moduln für die graphische Schnittstelle

Diese Moduln sind gesondert gehalten, um eine Anpassung an andere Umgebungen zu ermöglichen.

Das Grundmodul für die Initialisierung der graphischen Ausgabe

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Modul: Basicwin                          Version 1.0              */
/*                                                                   */
/* X11- Schnittstelle fuer GENETICS                                  */
/*                                                                   */
/* Datum : 10.Juli 1991                                              */
/*                                                                   */
/*-------------------------------------------------------------------*/
```

```
/* Standardincludes */
#include "genetics.h"
#include <stdio.h>

int screen_num;
char *progname = "genetic";

void GA_Basicwin_Initialisierung_Fenster_Statistik()
{
     int x, y;
/* Text fuer das Icon */
     char *FS_icon = "Statistik";
     char *FS_name = "Genetische Algorithmen: Statistik";
     Pixmap icon_pixmap;
     int count;

     window_size = 0;
     border_width = 4;

/* Uebernahme der Groesse des Bildschirmes aus den Systemdaten */
     screen_num = DefaultScreen(display);
     display_width = DisplayWidth(display, screen_num);
     display_height= DisplayHeight(display,screen_num);

     x = 0;
     y = 400;

     FS_Hoehe = (int)(1.25 * (MaxKnoten * MaxKnoten - MaxKnoten));
     FS_Breite = MaxGeneration + 50;
     Org_FS_Hoehe = FS_Hoehe;
     Org_FS_Breite= FS_Breite;

/* Aufbau eines einzelnen Fensters entsprechend den Voreinstellungen */
     F_Statistik = XCreateSimpleWindow(display, RootWindow(display, screen_num),
             x, y, FS_Hoehe, FS_Breite, border_width, BlackPixel (display,
             screen_num), WhitePixel (display, screen_num));

     XGetIconSizes (display, RootWindow (display, screen_num),&size_list,
                    &count);

     Statistik_Size.flags = PPosition | PSize | PMinSize;
     Statistik_Size.x = x;
     Statistik_Size.y = y;
```

```
    Statistik_Size.width = FS_Breite;
    Statistik_Size.height= FS_Hoehe;
    Statistik_Size.min_width = 300;
    Statistik_Size.min_height = 200;

    XSetStandardProperties (display, F_Statistik, FS_name, FS_icon,
        icon_pixmap,progname,0, &Statistik_Size);
    XSelectInput (display, F_Statistik, ExposureMask |
                ButtonPressMask | StructureNotifyMask);

    GA_Basicwin_erzeuge_Grafik_Umgebung(F_Statistik, &FS_Grafik, font_info);
    XMapWindow(display, F_Statistik);
}

void GA_Basicwin_Initialisierung_Fenster_Baum()
{
    int x, y;
/* Text fuer das Icon */
    char *FB_icon = "bester Graph";
    char *FB_name = "Genetische Algorithmen: Graph";
    Pixmap icon_pixmap;
    int count;

    window_size = 0;
    border_width = 4;

/* Uebernahme der Groesse des Bildschirmes aus den Systemdaten */
    screen_num = DefaultScreen(display);
    display_width = DisplayWidth(display, screen_num);
    display_height= DisplayHeight(display,screen_num);

    x = y = 0;

    FB_Hoehe = MaxWesen * 10;
    FB_Breite = MaxWesen * 10;

/* Aufbau eines einzelnen Fensters entsprechend den Voreinstellungen */
    F_Baum = XCreateSimpleWindow(display, RootWindow(display,screen_num),
            x, y, FB_Hoehe, FB_Breite, border_width, BlackPixel (display,
            screen_num), WhitePixel (display,screen_num));

    XGetIconSizes (display, RootWindow (display, screen_num),&size_list,
                &count);
```

```
     Baum_Size.flags = PPosition | PSize | PMinSize;
     Baum_Size.x = x;
     Baum_Size.y = y;
     Baum_Size.width = FB_Breite;
     Baum_Size.height= FB_Hoehe;
     Baum_Size.min_width = 300;
     Baum_Size.min_height = 200;

     XSetStandardProperties (display, F_Baum, FB_name, FB_icon,
          icon_pixmap,progname,0, &Baum_Size);
     XSelectInput (display, F_Baum, ExposureMask | ButtonPressMask
                    | StructureNotifyMask);

     GA_Basicwin_erzeuge_Grafik_Umgebung(F_Baum, &FB_Grafik, font_info);
     XMapWindow(display, F_Baum);
}

void GA_Basicwin_Initialisierung_Fenster_bester_Baum()
{
     int x, y;
/* Text fuer das Icon */
     char *FB_icon = "Baum";
     char *FB_name = "Genetische Algorithmen: Baum";
     Pixmap icon_pixmap;
     int count;

     window_size = 0;
     border_width = 4;

/* Uebernahme der Groesse des Bildschirmes aus den Systemdaten */
     screen_num = DefaultScreen(display);
     display_width = DisplayWidth(display, screen_num);
     display_height= DisplayHeight(display,screen_num);

     x = y = 0;

     FbB_Hoehe = MaxWesen * 10;
     FbB_Breite = MaxWesen * 10;

/* Aufbau eines einzelnen Fensters entsprechend den Voreinstellungen */
     Fb_Baum = XCreateSimpleWindow(display, RootWindow(display,screen_num),
             x, y, FbB_Hoehe, FbB_Breite, border_width, BlackPixel (display,
             screen_num), WhitePixel (display,screen_num));
```

```
    XGetIconSizes (display, RootWindow (display, screen_num),&size_list,
                   &count);

    Baum_Size.flags = PPosition | PSize | PMinSize;
    Baum_Size.x = x;
    Baum_Size.y = y;
    Baum_Size.width = FbB_Breite;
    Baum_Size.height= FbB_Hoehe;
    Baum_Size.min_width = 300;
    Baum_Size.min_height = 200;

    XSetStandardProperties (display, Fb_Baum, FB_name, FB_icon,
         icon_pixmap,progname,0, &Baum_Size);
    XSelectInput (display, Fb_Baum, ExposureMask | ButtonPressMask
                  | StructureNotifyMask);

    GA_Basicwin_erzeuge_Grafik_Umgebung(Fb_Baum, &FbB_Grafik, font_info);
    XMapWindow(display, Fb_Baum);
}

GA_Basicwin_erzeuge_Grafik_Umgebung(Fenster, Grafik_Umgebung, font_info)
Window Fenster;
GC *Grafik_Umgebung;
XFontStruct *font_info;
{
   unsigned long valuemask = 0;
   XGCValues values;
   unsigned int line_width = 0;
   int line_style = LineSolid;
   int cap_style = CapButt;
   int join_style = JoinRound;

   *Grafik_Umgebung = XCreateGC (display, Fenster, valuemask, &values);
   XSetFont (display, *Grafik_Umgebung, font_info->fid);
   XSetLineAttributes (display, *Grafik_Umgebung, line_width, line_style,
                       cap_style, join_style);
}

GA_Basicwin_load_font (font_info)
XFontStruct **font_info;
{
   char *fontname = "fixed";
   if ((*font_info = XLoadQueryFont (display,fontname)) == NULL)
      {
```

```
      GA_Benutzerschnittstelle(FEHLER,2,NULL);
      }
}

GA_Basicwin_Initialisierung_Grafik()
{
  int x, Dehnungsfaktor_X, Dehnungsfaktor_Y;
  int len10, len50, len100;
  char *Marke10 = "10";
  char *Marke50 = "50";
  char *Marke100= "100";
  len10 = strlen(Marke10);
  len50 = strlen(Marke50);
  len100= strlen(Marke100);
  Dehnungsfaktor_Y = FS_Hoehe / Org_FS_Hoehe;
  Dehnungsfaktor_X = FS_Breite/ Org_FS_Breite;
  if (Dehnungsfaktor_X < 1)
    Dehnungsfaktor_X = 1;
  if (Dehnungsfaktor_Y < 1)
    Dehnungsfaktor_Y = 1;

  XDrawLine(display,F_Statistik,FS_Grafik, 30, 0, 30, FS_Hoehe);
  XDrawLine(display,F_Statistik,FS_Grafik, 0,(FS_Hoehe - 30),
            FS_Breite, (FS_Hoehe - 30));
  for (x = 30; x < FS_Breite; x += (5 * Dehnungsfaktor_X))
    XDrawLine(display,F_Statistik,FS_Grafik, x, (FS_Hoehe - 30),
            x, (FS_Hoehe - 25));
  for (x = 30; x < FS_Breite; x += (50 * Dehnungsfaktor_X))
    XDrawLine(display,F_Statistik,FS_Grafik, x, (FS_Hoehe - 30),
            x, (FS_Hoehe - 20));
  for (x = (FS_Hoehe - 30); x > 0; x -= (5 * Dehnungsfaktor_Y))
    XDrawLine(display,F_Statistik,FS_Grafik, 25, x,
            30, x);
  for (x = (FS_Hoehe - 30); x > 0; x -= (50 * Dehnungsfaktor_Y))
    XDrawLine(display,F_Statistik,FS_Grafik, 20, x,
            30, x);
  XDrawString (display, F_Statistik,FS_Grafik,
               30 + (10 * Dehnungsfaktor_X), FS_Hoehe - 15,
               Marke10, len10);
  XDrawString (display, F_Statistik,FS_Grafik,
               30 + (50 * Dehnungsfaktor_X), FS_Hoehe - 15,
               Marke50, len50);
  XDrawString (display, F_Statistik,FS_Grafik,
               30 + (100 * Dehnungsfaktor_X), FS_Hoehe - 15,
```

```
                Marke100, len100);
  XDrawString (display, F_Statistik,FS_Grafik,
                1, (FS_Hoehe - 30 - (10 * Dehnungsfaktor_Y)),
                Marke10, len10);
  XDrawString (display, F_Statistik,FS_Grafik,
                1, (FS_Hoehe - 30 - (50 * Dehnungsfaktor_Y)),
                Marke50, len50);
  XDrawString (display, F_Statistik,FS_Grafik,
                1, (FS_Hoehe - 30 - ( 100 * Dehnungsfaktor_Y)),
                Marke100, len100);
}

GA_Basicwin_TooSmall (win, gc, font_info)
Window win;
GC gc;
XFontStruct *font_info;
{
   char *string1 = "Zu klein !";
   int y_offset, x_offset;

   y_offset = font_info->ascent + 2;
   x_offset = 2;

   XDrawString (display, win, gc, x_offset, y_offset, string1,
                strlen(string1));
}
```

Graphische Ausgabe

In diesem Modul wird die laufende Ausgabe der Zustände realisiert.

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Modul: Zeige_Baum                          Version 1.0                  */
/*                                                                         */
/* Graphische Ausgabe fuer GENETICS                                        */
/*                                                                         */
/* Datum : 10.Juli 1991                                                    */
/*                                                                         */
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include "genetics.h"
```

```
void GA_Zeige_Baum_print_tree(Wesen)
int Wesen;
  {
  int x, y;
  for (x = 0; x < MaxKnoten; x++)
    {
    for (y = 0; y < MaxKnoten; y++)
      printf(" %i ",Individuum[x][y][Wesen]);
    printf("\n");
    }
  }

GA_Zeige_Baum_print_best_tree()
  {
  int x, y;
  for (x = 0; x < MaxKnoten; x++)
    {
    for (y = 0; y < MaxKnoten; y++)
      printf(" %i ",Best_Individuum[x][y]);
    printf("\n");
    }
  }

void GA_Zeige_Baum_Zeichne_Statistik_neu()
  {
  int Dehnungsfaktor_X, Dehnungsfaktor_Y, x, y, z;
  Dehnungsfaktor_Y = FS_Hoehe / Org_FS_Hoehe;
  Dehnungsfaktor_X = FS_Breite/ Org_FS_Breite;
  if (Dehnungsfaktor_X < 1)
    Dehnungsfaktor_X = 1;
  if (Dehnungsfaktor_Y < 1)
    Dehnungsfaktor_Y = 1;
  for (z = 0; z < (Generation - 2); z++)
    {
    x = 30 + z * Dehnungsfaktor_X;
    y = (FS_Hoehe - 30) - (Statistik[z][BESTER] * Dehnungsfaktor_Y);
    XDrawPoint(display,F_Statistik,FS_Grafik, x, y);
    y = (FS_Hoehe - 30) - (Statistik[z][MITTEL] * Dehnungsfaktor_Y);
    XDrawPoint(display,F_Statistik,FS_Grafik, x, y);
    y = (FS_Hoehe - 30) - (Statistik[z][SCHLECHT] * Dehnungsfaktor_Y);
    XDrawPoint(display,F_Statistik,FS_Grafik, x, y);
    }
  }
```

```
void GA_Zeige_Baum_Zeichne_Statistik()
  {
  int Dehnungsfaktor_X, Dehnungsfaktor_Y, x, y;
  Dehnungsfaktor_Y = FS_Hoehe / Org_FS_Hoehe;
  Dehnungsfaktor_X = FS_Breite/ Org_FS_Breite;
  if (Dehnungsfaktor_X < 1)
    Dehnungsfaktor_X = 1;
  if (Dehnungsfaktor_Y < 1)
    Dehnungsfaktor_Y = 1;
  x = 30 + Generation * Dehnungsfaktor_X;
  y = (FS_Hoehe - 30) - (Statistik[Generation][BESTER] * Dehnungsfaktor_Y);
  XDrawPoint(display,F_Statistik,FS_Grafik, x, y);
  y = (FS_Hoehe - 30) - (Statistik[Generation][MITTEL] * Dehnungsfaktor_Y);
  XDrawPoint(display,F_Statistik,FS_Grafik, x, y);
  y = (FS_Hoehe - 30) - (Statistik[Generation][SCHLECHT] * Dehnungsfaktor_Y);
  XDrawPoint(display,F_Statistik,FS_Grafik, x, y);
  }

void GA_Zeige_Baum_Zeichne_Baum()
  {
  int Koordinate[MaxIndividuen][2], x, y, z, Zeichenlaenge;
  char *Zeichen[4];
  for (x = 0; x < MaxWesen; x++)
    {
    Koordinate[x][XKOORDINATE] = 0;
    Koordinate[x][YKOORDINATE] = 0;
    }
  XClearWindow (display, F_Baum);
  y = x = 0;
  for (y = 0; y < MaxKnoten; y++)
    {
    for (z = 0; z < MaxKnoten; z++)
      {
      if (Beste_Tiefe[z] == y)
        {
        Koordinate[z][XKOORDINATE] = (x++ * 30 + 15);
        Koordinate[z][YKOORDINATE] = (y * 50 + 15);
        }
      }
    x = 0;
    }
  for (y = 0; y < MaxKnoten; y++)
    {
```

```
    (void) sprintf(Zeichen,"%d",y);
    Zeichenlaenge = strlen(Zeichen);
    XDrawString(display, F_Baum, FB_Grafik, Koordinate[y][XKOORDINATE],
                                            Koordinate[y][YKOORDINATE],
                                            Zeichen, Zeichenlaenge);
    }
  for (x = 0; x < MaxKnoten; x++)
    for (y = 0; y < MaxKnoten; y++)
      if (Best_Individuum[x][y] == 1)
        {
        if (Koordinate[x][YKOORDINATE] < Koordinate[y][YKOORDINATE])
          XDrawLine(display,F_Baum,FB_Grafik,Koordinate[x][XKOORDINATE],
                                            Koordinate[x][YKOORDINATE],
                                            Koordinate[y][XKOORDINATE],
                                            Koordinate[y][YKOORDINATE] -10);
        if (Koordinate[x][YKOORDINATE] > Koordinate[y][YKOORDINATE])
          XDrawLine(display,F_Baum,FB_Grafik,Koordinate[x][XKOORDINATE],
                                            Koordinate[x][YKOORDINATE] -10,
                                            Koordinate[y][XKOORDINATE],
                                            Koordinate[y][YKOORDINATE]);
        if (Koordinate[x][YKOORDINATE] == Koordinate[y][YKOORDINATE])
          XDrawLine(display,F_Baum,FB_Grafik,Koordinate[x][XKOORDINATE],
                                            Koordinate[x][YKOORDINATE] +3,
                                            Koordinate[y][XKOORDINATE],
                                            Koordinate[y][YKOORDINATE] +3);
        }
  }

void GA_Zeige_Baum_Zeichne_bester_Baum(bester_Baum)
int bester_Baum;
  {
  int Koordinate[MaxIndividuen][2], x, y, z, Zeichenlaenge;
  char *Zeichen[4];
  for (x = 0; x < MaxWesen; x++)
    {
    Koordinate[x][XKOORDINATE] = 0;
    Koordinate[x][YKOORDINATE] = 0;
    }
  XClearWindow (display, Fb_Baum);
  y = x = 0;
  for (y = 0; y < MaxKnoten; y++)
    {
    for (z = 0; z < MaxKnoten; z++)
      {
```

```
      if (Tiefe[z][bester_Baum] == y)
        {
        Koordinate[z][XKOORDINATE] = (x++ * 30 + 15);
        Koordinate[z][YKOORDINATE] = (y * 50 + 15);
        }
      }
    x = 0;
    }
  for (y = 0; y < MaxKnoten; y++)
    {
    (void) sprintf(Zeichen,"%d",y);
    Zeichenlaenge = strlen(Zeichen);
    XDrawString(display, Fb_Baum, FbB_Grafik, Koordinate[y][XKOORDINATE],
                                              Koordinate[y][YKOORDINATE],
                                              Zeichen, Zeichenlaenge);
    }
  for (x = 0; x < MaxKnoten; x++)
    for (y = 0; y < MaxKnoten; y++)
      if (Individuum[x][y][bester_Baum] == 1)
        {
        if (Koordinate[x][YKOORDINATE] < Koordinate[y][YKOORDINATE])
          XDrawLine(display,Fb_Baum,FbB_Grafik,Koordinate[x][XKOORDINATE],
                                               Koordinate[x][YKOORDINATE],
                                               Koordinate[y][XKOORDINATE],
                                               Koordinate[y][YKOORDINATE] -10);
        if (Koordinate[x][YKOORDINATE] > Koordinate[y][YKOORDINATE])
          XDrawLine(display,Fb_Baum,FbB_Grafik,Koordinate[x][XKOORDINATE],
                                               Koordinate[x][YKOORDINATE] -10,
                                               Koordinate[y][XKOORDINATE],
                                               Koordinate[y][YKOORDINATE]);
        if (Koordinate[x][YKOORDINATE] == Koordinate[y][YKOORDINATE])
          XDrawLine(display,Fb_Baum,FbB_Grafik,Koordinate[x][XKOORDINATE],
                                               Koordinate[x][YKOORDINATE] +3,
                                               Koordinate[y][XKOORDINATE],
                                               Koordinate[y][YKOORDINATE] +3);
        }
  }
```

A.2.3 Moduln für die Benutzerschnittstelle

Benutzerschnittstelle

```
/*-------------------------------------------------------------------*/
```

```
/*                                                                    */
/* Modul: Benutz.c                              Version 1.0           */
/*                                                                    */
/* Benutzerschnittstelle fuer GENETICS                                */
/*                                                                    */
/* Datum : 20.August 1991                                             */
/*                                                                    */
/*--------------------------------------------------------------------*/

#include "genetics.h"
#include "domenu.h"

char *menutab[5] = {
    "Starten des Systems",
    "Aendern der Parameter",
    "",
    "Abbrechen",
    };

struct menu Menu = {
    4, 25,
    "Genetische Algorithmen: Auswahlmenu",
    &menutab[0]
    };

static XmStringCharSet OSF_CharSet = (XmStringCharSet) XmSTRING_DEFAULT_CHARSET;

void GA_Benutz_error(Fehlerart)
int Fehlerart;
  {
  attron(A_BLINK|A_REVERSE);
  switch(Fehlerart) {
    case 1:
      mvaddstr(12,10,"Genetics: Keine Verbindung mit dem X-Server moeglich\0");
      break;
    case 2:
      mvaddstr(12,10,"Genetics: Definierter X11-Font nicht auffindbar\0");
    default:
      mvaddstr(12,10,"Genetics: Nicht klassifizierbarer Fehler\0");
      break;
    }
  attroff(A_BLINK|A_REVERSE);
  refresh();
  endwin();
```

```
  exit (1);
  }

void GA_Benutz_ClearScreen()
  {
  clear();
  mvaddstr(0,5,"Genetische Algorithmen: Selbstorganisierende Datenstrukturen\0");
  refresh();
  }

void GA_Benutz_Vorinitial()
  {
  Korrektur_Ordnung = FALSCH;
  Korrektur_Topologie = FALSCH;
  MaxWesen = 20;
  MaxWesenHalbe = 10;
  MaxGeneration = 300;
  MaxKnoten = 20;
  M_Wahrschein = 0.0005;
  }

void GA_Benutz_Einstellung()
  {
  int Eingabe, weiter = FALSCH;
  float FEingabe, Rechfeld;
  long float LfEingabe;
  while (weiter == FALSCH)
    {
    GA_Benutz_ClearScreen();
    mvaddstr(1,50,"Eingabe : 0 = Nein, 1 = Ja\0");
    mvaddstr(3,5,"Systemparameter :\0");
    mvaddstr(4,5,"Maximale Generationen :\0");
    mvaddstr(4,45,"Anzahl der Knoten :\0");
    mvaddstr(5,5,"Mutationswahrscheinlichkeit :\0");
    mvaddstr(5,45,"Anzahl der Individuen :\0");
    mvaddstr(7,5,"Fitnessfunktionen:\0");
    mvaddstr(8,5,"Verteilung Fitness:\0");
    mvaddstr(21,5,"Korrektur :\0");
    mvaddstr(22,5,"Ordnung   :\0");
    mvaddstr(22,45,"Topologie :\0");
    mvaddstr(23,5,"O.K. ?\0");
    refresh();
/* Eingabe Anzahl der Generationen */
    do {
```

```
      move(4,34); scanw("%i",&Eingabe);
      printw("%i",Eingabe);
      } while ((Eingabe < 1) || (Eingabe > 999));
    MaxGeneration = Eingabe;
    refresh();
/* Eingabe Anzahl der Knoten */
    do {
      move(4,65); scanw("%i",&Eingabe);
      printw("%i",Eingabe);
      } while ((Eingabe < 10) || (Eingabe > 99));
    MaxKnoten = Eingabe;
    refresh();
/* Eingabe der Mutationswahrscheinlichkeit */
    do {
      move(5,34); scanw("%lf",&LfEingabe);
      printw("%lf",LfEingabe);
      } while ((LfEingabe >= 1.0) || (LfEingabe < 0.000005));
    M_Wahrschein = LfEingabe;
    refresh();
/* Eingabe der Populationsgroesse */
    do {
      move(5,69); scanw("%i",&Eingabe);
      printw("%i",Eingabe);
      } while ((Eingabe < 2) || (Eingabe > 50));
    MaxWesen = Eingabe;
    MaxWesenHalbe = MaxWesen / 2;
    refresh();
/* Verteilung der Fitnessfunktionen */
    do {
      move(8,34); scanw("%f",&FEingabe);
      Rechfeld = 1 - FEingabe;
      printw("%1.2f:%1.2f",FEingabe,Rechfeld);
      } while ((FEingabe > 1.0) || (FEingabe < 0.01));
    Fit_Verteil = FEingabe;
    refresh();
/* Eingabe Schalter fuer Ordnungs-Korrektur */
    do {
      move(22,17); scanw("%i",&Eingabe);
      printw("%i",Eingabe);
      } while ((Eingabe != 0) && (Eingabe != 1));
    if (Eingabe == 1)
      Korrektur_Ordnung = WAHR;
    else
      Korrektur_Ordnung = FALSCH;
```

```
    refresh();
/* Eingabe Schalter fuer Topologie-Korrektur */
    do {
      move(22,57); scanw("%i",&Eingabe);
      printw("%i",Eingabe);
      } while ((Eingabe != 0) && (Eingabe != 1));
    if (Eingabe == 1)
      Korrektur_Topologie = WAHR;
    else
      Korrektur_Topologie = FALSCH;
    refresh();
    do {
      move(23,12); scanw("%i",&Eingabe);
      printw("%i",Eingabe);
      } while ((Eingabe != 0) && (Eingabe != 1));
    if (Eingabe == 1)
      weiter = WAHR;
    refresh();
    }
  }

void GA_Benutz_Startmenu()
  {
  int Ergebnis, weiter;
  GA_Benutz_Vorinitial();
  weiter = FALSCH;
  while (weiter == FALSCH)
    {
    GA_Benutz_ClearScreen();
    Ergebnis = GA_domenu_main(&Menu);
    switch(Ergebnis)
      {
      case 0:  weiter = WAHR;
               break;
      case 1:  GA_Benutz_Einstellung();
               break;
      default: resetty();
               endwin();
               exit (0);
      }
    }
  }

void GA_Benutz_Statusausgabe(Generation,OGF,GF,PGF,BF, MaxWert,
```

```
                              MinWert, PMaxWert, PMinWert)
int Generation, OGF, GF, BF, MaxWert, MinWert;
double PGF, PMaxWert, PMinWert;
  {
  GA_Benutz_ClearScreen();
  mvaddstr(5,5,"Generation :\0");
  mvaddstr(7,5,"Erreichbare Fitness:\0");
  mvaddstr(8,5,"Erreichte   Fitness:\0");
  mvaddstr(8,45,"in Prozent :\0");
  mvaddstr(10,5,"Beste Einzelwerte :\0");
  mvaddstr(11,5,"im Ablauf            :\0");
  mvaddstr(12,5,"in dieser Generation :\0");
  mvaddstr(12,45,"prozentualer Anteil :\0");
  mvaddstr(14,5,"Schlechteste Einzelwerte :\0");
  mvaddstr(15,5,"in dieser Generation :\0");
  mvaddstr(15,45,"prozentualer Anteil :\0");
  move(5,20); printw("%i",Generation);
  move(7,26); printw("%i",OGF);
  move(8,26); printw("%i",GesamtFitness);
  move(8,60); printw("%2.2f",PGF);
  move(11,29); printw("%i",BF);
  move(12,29); printw("%i",MaxWert);
  move(12,67); printw("%2.2f",PMaxWert);
  move(15,29); printw("%i",MinWert);
  move(15,67); printw("%2.2f",PMinWert);
  refresh();
  }

void GA_Benutzerschnittstelle (Art,argc,argv)
int Art, argc;
char **argv;
  {
  char          *progname = "Genetic";
  char          *display_name = NULL;
  if (Art == FIRST_TIME)
    {
    initscr(); savetty(); cbreak(); nonl(); echo(); standend();
    GA_Benutz_Startmenu();
    GA_Benutz_ClearScreen();
    mvaddstr(10,12,"Initialisierung X-Windows:\0");
    refresh();
    XtToolkitInitialize();
    if ((display =XtOpenDisplay (NULL, NULL, argv[0], "Mwm", NULL,
                                 0, &argc, argv)) == NULL)
```

```
        GA_Benutz_error(1);
    GA_Basicwin_load_font (&font_info);
    mvaddstr(10,39,"O.K.\0");
    refresh();
    }
  if (Art == NORMAL_USE)
    {
    XUnloadFont (display, font_info->fid);
    XFreeGC (display, FS_Grafik);
    XCloseDisplay (display);
    Programmende = WAHR;
    resetty();
    endwin();
    }
  if (Art == FEHLER)
    {
    GA_Benutz_error(argc);
    }
  }
```

Zusatzmodul zum Einsatz von Curses

Dieses Programmstück ist [Hav87] angelehnt. Es dient zur Benutzung der Unix-Bibliothek CURSES zur Ausnutzung der Full-Screen Kapazitäten.

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Modul: domenu                                  Version 1.0              */
/*                                                                         */
/* Menusystem unter CURSES                                                 */
/*                                                                         */
/* Datum :                                                                 */
/*                                                                         */
/*-------------------------------------------------------------------------*/

#include <curses.h>
#include "domenu.h"

GA_domenu_isblank(s)
char *s;
{
  while(*s == ' ')
    s++;
```

```
  return *s == '\0' ? 1: 0;
}

GA_domenu_main(m)
struct menu *m;
{

  int option, lastoption, j, c, y, x;
  char *p;

  /* save current terminal state, then set required modes */
  savetty();

  cbreak(); nonl(); noecho(); standend();

  /* empty screen */
  clear();

  /* initialize keypad, TRUE is defined in curses.h */
  keypad(stdscr,TRUE);

  /*print centred title on line one */
  move(0, (COLS - strlen(m->m_title))/2);
  addstr(m->m_title);

  /* work out position for top left corner of menu */
  y = (LINES - m->m_height)/2 + 1;
  x = (COLS  - m->m_width)/2;

  /* display menu */
  for(j = 0; j < m->m_height; j++)
    mvaddstr( y+j, x, m->m_data[j]);

  /* initial values for cursor pos. and option setting */
  move( y, x);

  /* this assumes first line in menu isn't blank */
  lastoption = option = 0;

  for (;;) {

    /* remove highlight bar from last option */
    if(lastoption != option)
```

```
    mvaddstr( lastoption+y, x, m->m_data[lastoption]);

  /* put highlight bar on current option */
  standout();
  mvaddstr( option +y, x, m->m_data[option]);
  standend();
  move(option +y, x);

  /* save current option */
  lastoption = option;

  refresh();

  /* process input */
  switch( (c = getch()) ){

    case '\r':                  /* option selected, so return */
    case '\n':
      if(option < 0) {
        beep();
        break;
      }

      /* restore initial state and return */
      resetty();
      return option;

    case KEY_DOWN:              /* move current down if possible */
    case KEY_RIGHT:             /* or wrap around       */
      do {
         option = (++option < m->m_height) ? option: 0;
      }while( GA_domenu_isblank(m->m_data[option]));
      break;

    case KEY_UP:                /* move current line up or wrap */
    case KEY_LEFT:              /* around */
      do{
        option = (--option >= 0) ? option : m->m_height -1;
      }while( GA_domenu_isblank(m->m_data[option]));
    default:
      for(j = 0; j < m->m_height; j++){

        for(p = m->m_data[j]; *p == ' '; p++)
          ;
```

```
          if( *p == '\0')       /* blank line */
            continue;

          if( *p == c){
            option = j;
            break;
          }
        }
        if(j >= m->m_height)
          beep();
        break;

      }
    }
}
```

Die hinzugehörende Includedatei

```
/*menu.h -- contains definition of menu structure */

struct menu {
  int m_height;         /* menu height          */
  int m_width;          /* menu width           */
  char *m_title;        /* menu title           */
  char **m_data;        /* pointer to data      */
  };
```

Schnittstelle zum Zufallsgenerator

Wie im theoretischen Teil ausgeführt, bedeutet ein Zufallsgenerator mit zu kleiner Bandbreite eine Einschränkung in der Funktionalität. Für die Experimente wird ein Zufallsgenerator mit einer Bandbreite von 48 Bit benutzt.

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Modul: Zufall                                Version 1.0                */
/*                                                                         */
/* Schnittstelle zum Zufallsgenerator                                      */
/*                                                                         */
/* Datum :                                                                 */
/*                                                                         */
/*-------------------------------------------------------------------------*/
#include <sys/time.h>
```

```
#include "genetics.h"

long GA_Zufall_main()
  {
  return(lrand48());
  }
```

Literaturverzeichnis

[Arb81] M.A.Arbib, A.J.Kfoury, R.N.Moll: A Basis for Theoretical Computer Science
Springer-Verlag
1.Auflage, New York, 1981

[Bau84] F.L.Bauer, G.Goos: Informatik
Eine einführende Übersicht, Zweiter Teil
Springer-Verlag, Sammlung Informatik Band 91
3.Auflage, Berlin, 1984

[Bos86] K.Bosch: Elementare Einführung in die Wahrscheinlichkeitsrechnung
Serie vieweg studium / Basiswissen
Vieweg Verlag
5.Auflage, Braunschweig, 1986

[Bra91] T.Bratke, J.Gramatzki, T.Wagner, R.Welker: Entwicklung eines Programms zur Stundenplanbelegung mit Hilfe von genetischen Algorithmen
aus: [Joc91]

[Dav91] L.Davis: Handbook of Genetic Algorithms
Verlag Van Nostrand Reinhold
1.Auflage, New York, 1991

[Den77] E.Denert, R.Franck: Datenstrukturen
Wissenschaftsverlag, Bibliographisches Institut
1.Auflage, Berlin,München, 1977

[Dew86] A.K.Dewdney: Computer-Kurzweil
aus: Spektrum der Wissenschaft, Januar 1986
Orginal:
Exploring the field of genetic algorithms in a primordial computer sea full of flips
aus: Scientific American, Mai, 1985

[Dob84] P.Dobrinski, G.Krakau, A.Vogel: Physik für Ingenieure
B.G.Teubner Verlag
6.Auflage, Stuttgart, 1984

[Doe73] W.Dörfler, J.Mühlbacher: Graphentheorie für Informatiker
Verlag Walter de Gruyter
Berlin, 1973

[Doe77] W.Dörfler: Mathematik für Informatiker
Band 1: Finite Methoden und Algebra
Carl Hanser Verlag
1.Auflage, München, 1977

[Dor78] L.L.Dornhoff, F.E.Hohn: Applied Modern Algebra
Verlag MacMillan Publishing Co
1.Auflage, Urbana-Champaign, 1978

[Eng84] G.Engeln-Müllges, F.Reuther: Numerische Mathmatik für Ingenieure
Wissenschaftsverlag, Bibliographisches Institut
5.Auflage, Aachen, 1987

[Eve83] S.Evens: Graph Algorithms
Computer Science Press
2.Auflage, Haifa, 1983

[Gol89] D.E.Goldberg: Genetic Algorithms in Search, Optimization and Machine Learning
Verlag Addison-Wesley
1.Auflage, Ann Arbor, 1989

[Gri90] A.Grimm: Genetische Algorithmen
aus: [Joc91]

[Gri91] A.Grimm: Untersuchungen über den Einsatz von Genetischen Algorithmen bei topologischen Optimierungen
aus: [Joc91]

[Gue91] C.Günthner: Modifiziertes Tourenproblem am Beispiel einer Telefonkette
aus: [Joc91]

[Hav87] K.Haviland, B.Salama: Unix System Programming
Verlag Addison-Wesley
1.Auflage, Wokingham, 1987

[Hec76] J.Heck: Zufallsgraphen
aus: Graphen-Sprachen und Algorithmen auf Graphen
Hrsg. U.Pape

Serie Applied Computer Science,
Berichte zur praktischen Informatik Band 1
Hanser Verlag
1. Auflage, Berlin, 1976

[Hof85] D.R.Hofstadter: Gödel, Escher, Bach: ein endlos geflochtenes Band
Verlag Klett-Cotta
5.Auflage, Stuttgart, 1985

[Hol68] J.H.Holland: Hierarchical description of universal spaces and adaptive systems
aus: Technical Report ORA Projects 01252 and 08226
University of Michigan
Department of Computer and Communication Sciences
Ann Arbor, 1968

[Hop88] J.E.Hopcroft, J.D.Ullman: Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie
Verlag Addison-Wesley
1.Auflage, Bonn, 1988

[Joc88] F.Jochum: Datenbanksysteme, Folien zur Vorlesung
Gummersbach, 1988

[Joc91] F.Jochum (Hrsg.): Genetische Algorithmen - Prinzipien, Anwendungen, Experimente
Fachbereich Informatik
Fachhochschule Köln, Abt. Gummersbach
1.Auflage, Gummersbach, 1991

[Ker88] B.W.Kernigham, D.M.Ritchie: The C Programming Language
Verlag Prentice Hall
2.Auflage, New Jersey, 1988

[Kie90] B.Kießwetter: Experimentelle Untersuchung und Anwendung Genetischer Algorithmen in den Bereichen maschinelles Lernen und Optimierung
Diplomarbeit an der FH Köln, Abt. Gummersbach
Gummersbach, 1990

[Kla83] R.Klar: Digitale Rechenautomaten
Verlag de Gruyter, Sammlung Göschen
3.Auflage, Berlin, 1983

[Kno83] H.Knodel (Hrsg.): Linder Biologie
J.B.Metzlersche Verlagsbuchhandlung
19.Auflage, Stuttgart, 1983

[Knu68] D.E.Knuth: The Art of Computer Programming
Vol.1: Fundamental Algorithms
Verlag Addison-Wesley
1.Auflage, Reading, 1968

[Knu81] D.E.Knuth: The Art of Computer Programming
Vol.2: Seminumerical Algorithms
Verlag Addison-Wesley
2.Auflage, Reading, 1981

[Knu73] D.E.Knuth: The Art of Computer Programming
Vol.3: Sorting and Searching
Verlag Addison-Wesley
1.Auflage, Reading, 1973

[Kuh79] R.Kuhlen: Datenbasen, Datenbanken, Netzwerke
Praxis des Information Retrieval
Band 2: Konzepte von Datenbanken
Verlag K.G.Saur
1.Auflage, München, 1979

[Lip81] J.D.Lipson: Elements of Algebra and Algebraic Computing
Verlag Addison-Wesley
1.Auflage, Reading, 1981

[Luc91] C.B.Lucansius, M.J.J.Blommers, L.M.C.Buydens, G.Kateman: A Genetic Algorithm for Conformational Analysis of DNA
aus: [Dav91], Kap.18, S.251 ff.

[Maj76] M.E.Majster: Datenstrukturen und Operationen
aus: Graphen, Algorithmen, Datenstrukturen
Hrsg. H.Noltemeier
Serie Applied Computer Science,
Berichte zur praktischen Informatik Band 4
Hanser Verlag
1. Auflage, München, 1976

[Meh84] K.Mehlhorn: Data Structures and Algorithms 1:
Sorting and Searching
Serie EATCS Monographs on Theoretical Computer Science
Springer Verlag
1.Auflage, Berlin, 1984

[Meh88] K.Mehlhorn: Datenstrukturen und effiziente Algorithmen
Band 1: Sortieren und Suchen
B.G.Teubner Verlag

2.Auflage, Stuttgart, 1988
vgl.[Meh84]

[Moo74] A.M.Mood, F.A.Graybill, D.C.Boes: Introduction to the Theory of Statistics
Verlag McGraw-Hill
3. Auflage, Auckland, 1974

[Mue73] H.Müller-Merbach: Operations Research
Methoden und Modelle der Optimalplanung
Verlag Franz Vahlen
3. Auflage, München, 1973

[Mue78] H.Müller: Diskrete Algebraische Strukturen I
Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung (Informatik) an der Friedrich Alexander Universität Erlangen Nürnberg
Band 6, Nummer 5
4.Auflage, Erlangen, 1978

[Mue79] H.Müller: Diskrete Algebraische Strukturen II
Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung (Informatik) an der Friedrich Alexander Universität Erlangen Nürnberg
Band 8, Nummer 2
4.Auflage, Erlangen, 1979

[Pre89] W.H.Press, B.P.Flannery, et al.:Numerical Recipes
The Art of Scientific Computing
(Fortran Version)
Cambridge University Press
1.Auflage, Cambridge, 1989

[Sch83] G.Schlageter, W.Stucky: Datenbanksysteme:
Konzepte und Modelle
Teubner Studienbücher Informatik
2.Auflage, Hagen, 1983

[Tar91] O.Taraszow: Optimierung von Topologien verteilter Systeme und andere verwandte Strukturprobleme
Vortrag an der Fachhochschule Köln, Abt. Gummersbach
Gummersbach, 1991

[Ull83] J.D.Ullman: Principles of Database Systems
Computer Science Press
1.Auflage, Rockville, 1983

[Wed76] H.Wedekind, T.Härder: Datenbanksysteme II
Wissenschaftsverlag, Bibliographisches Institut
1.Auflage, Erlangen, 1976

[Wir81] N.Wirth: Compilerbau
Teubner Studienbücher Informatik
2.Auflage, Zürich, 1981

[Wir83] N.Wirth: Algorithmen und Datenstrukturen
Pascal- Version
Teubner Leitfäden und Monographien der Informatik
3.Auflage, Stuttgart, 1983

[Wir85] N.Wirth, K.Jensen: Pascal
User Manual and Report
Springer-Verlag
3.Auflage, New York, 1985

[XOp89] X/Open Gruppe: X/Open Portability Guide, Stand 1988
Band 1: XSI Commands and Utilities
Band 2: XSI System Interfaces and Headers
Band 3: XSI Supplementary Definitions
Band 4: Programming Languages
Band 6: Window Management
Band 7: Networking Services
Verlag Prentice Hall
1.Auflage, Englewood Cliffs, 1989