RT/LSI-97-8-T

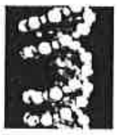
The PgaFrame - A frame for parallel genetic algorithms

Jordi Petit

Report LSI-97-8-T

PgaFrame

A frame for parallel genetic algorithms



Jordi Petit i Silvestre

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

http://www-lsi.upc.es/~jpetit/PgaFrame

Overview

This page documents the **PgaFrame** - a frame for parallel genetic algorithms. The PgaFrame offers to non-expert users the possibility to solve optimization problems via genetic algorithms on parallel computers. This frame has appeared as the natural fusion of two different projects: *PgaPack* and *Frames*. The PgaFrame combines the major features of these projects: it integrates the capabilities to specify genetic algorithms offered by the PgaPack library and the support of Frames to the easy and portable programming of parallel machines. In this way, a complex framework to develop genetic algorithms is achieved.

With PgaFrame a genetic algorithm can be considered as an extensible skeleton. Thus, the user only has to provide his problem dependent parameters to adapt it to his needs. This specification task is done through an easy-to-learn interface and an easy-to-use graphical user interface that do not require any knowledge on parallel programming: only some radio buttons must be selected, some entries filled and a few sequential functions in ANSI C written. The core of the genetic algorithm and its parallelization are completely transparent to the user. As a consequence, executing a genetic algorithm on a parallel system with the PgaFrame is as easy as filling a form. Moreover, due to the portability of the PgaPack library and the Frames tools, it can run efficiently on almost any parallel or sequential computer.

Introduction

Genetic algorithms

Genetic algorithms (GAs) are adaptative methods which may be used to solve search and optimisation problems. They are based on the genetic process of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection and "survival of the fittest", first clearly stated by Charles Darwin in *The Origin of Species*. By mimicking this process, GAs are able to "evolve" solutions to real world problems, if they have been suitably encoded. The basic principles of GAs were first laid down rigorously by Holland.

Genetic algorithms simulate those processes in natural populations that are essential to evolution. They work with a population of "individuals", each representing a possible solution to a given problem. Each individual is assigned a "fitness score" according to how good a solution to the problem is. The highly fit individuals are given opportunities to "reproduce", by "cross breeding" with other individuals of the population. This produces new individuals as "offspring", which share some features taken from each parent. The least fit members of the population are less likely to get selected for reproduction, and so "die out". The process is then repeated with the new generation.

Genetic algorithms have been used in numerical function optimisation, image processing, combinatorial optimisation, design, machine learning and much more areas.

The PgaPack library

PgaPack is a general-purpose, data-structure-neutral, parallel genetic algorithm library beeing developed at Argonne National Laboratory. Its key features on wich we are interested are:

- Runs on uniprocessors, parallel computers, and workstation networks.
- Binary-, integer-, real-, and character-valued native data types.
- Parameterized population replacement.
- Multiple choices for selection, crossover, and mutation operators.
- Easy integration of hill-climbing heuristics.
- Easy-to-use interface for novice and application users.
- Fully extensible to support custom operators and new data types.

PGAPack is written in ANSI C and uses the MPI message passing interface and thus should run on most uniprocessors, parallel computers, and workstation networks. It was developed by David Levine of the Mathematics and Computer Science Division at Argonne National Laboratory. [See the PgaPack Copyright notice]

Frames

Frames provide support for the programming of distributed-memory machines via a library of basic algorithms, data structures and so-called programming frames (or frameworks). The latter are skeletons with problem dependent parameters to be provided by the users. Frames focuses on re-usability and portability as well as on small and easy-to-learn interfaces. Thus, non-expert users will be provided with tools to program and exploit parallel machines efficiently. All these frames will be especially valuable and relevant to industrial cooperators.

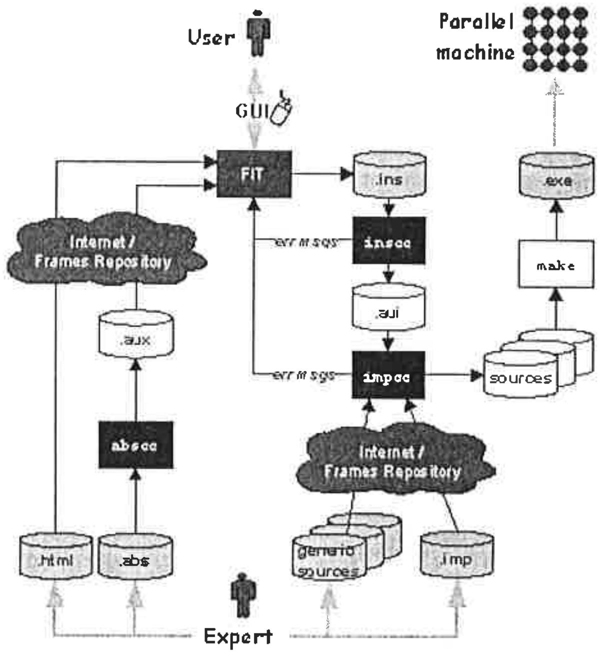
The main features of Frames are:

- Re-Usability
- Portability
- State-of-the-art techniques
- Efficiency
- Easy-to-learn

- Easy-to-use graphical interfaces.

Frames are constructed for different target machines and common programming environments (like PVM or MPI). The focus, however, is on distributed-memory machines. Frames will be adapted optimally to the target systems, contain efficient state-of-the-art programming techniques, and therefore increase the acceptance of parallel computing.

The Frames model is depicted in the next picture. In this model, each generated target executable is built from three specifications, an *abstract level specification*, an *instance level specification* and an *implementation level specification*.



Frames is a work-package of the AlCom-IT project. It is described in the paper *Programming Frames for the Efficient use of Parallel Systems* (1997) written by Thomas Römke from the Paderborn Center for Parallel Computing and Jordi Petit i Silvestre from the Departament de Llenguatges i Sistemes Informàtics (UPC).

The PGA Frame

The PFA frame is a modular frame (or framework, or toolbox) for genetic algorithms on distributed memory systems. The idea of *programming frames* augmentes the PgaPack library with graphical editors for specifying genetic parameters that customize the generic algorithm to the user needs. Crossover and mutation operators, selection policies, termination criteria and much other parameters can be set with this graphical tool. The parallelization is based on MPI which guarantees a portable code. Thanks to Frames, customizing or extending a genetic algorithm is very easy.

Following the Frames approach, the PgaFrame is made up of three different specifications. The first specification, also called the *abstract specification*, is used to define the parameters of the problem. These are the parameters that latter must be filled by the user in the second specification, also called the *instance specification*. A new instance for a genetic algorithm can be generated by simply changing this instance specification. In the third specification (*implementation specification*), the implementors of the PgaFrame have coded (using

PgaPack) the implementation of the generic genetic algorithm. The result of merging this generic code with the instance filled by the user consists in a set of programs and a makefile.

The elements of the PgaFrame (called PGA) are:

- An abstract specification
- A graphical tool to let user specify instances
- Examples of instance specifications (TSP)
- An implementation specification
- The PgaPack library.

The abstract level

Let us now describe the abstract level of the PGA frame. The main task in this level is to define the properties of the generic genetic algorithm. The declaration of these parameters is done using the abstract specification language defined in Frames, which is simply a list of standard ANSI C type definitions with some enhancements.

We remark that all the parameters are strongly sequential and portable: they do not contain parallel constructs nor aspects of any target machine. This will ensure that they can be reused over different systems. Along with the declaration of the parameters, the abstract specification manages their documentation, which is presented in the *User Documentation* page in HTML format.

The PGA frame contains a lot of parameters. However, only four of them are not optional; the rest contain default values that do not need to be changed except for extra custumization. The parameters are structured in the following set of families:

Fundamental parameters:
Contains the information about the type of the chromosomes, the direction in which to improve the evaluation function, the evaluation function itself and preprocessing and postprocessing functions, which can be used to input and output data.

Population replacement:
Allows to select how many individuals will be keep in each iteration, how many will reproduce, how they will replace the old ones, etc. In particular, the PGA frame supports *generational replament GAs* (GRGA), *steady-state GAs* (SSGA), elitist selection,...

Stopping criteria:
Three predefined stopping rules are offered, the user can parametrize them or write his own stopping rule.

Initialization:
Specificies how to create the new individuals. Predefined operators exist, but user with spetial needs can write their own initialization functions.

Selection:
Usual selection techniques can be parametrized.

Crossover:
Four popular operators can be selected and parametrized. If they are not enough, the user can provide his own crossover operator.

Mutation:

Again, popular operators can be selected and parametrized. If they are not enough, the user can provide his own mutation operator.

Fitness:

In order to avoid premature convergence, fitness can be adapted.

Restart:

If requested, a new generation can be created perdiocally by filling the population with mutations of the best current string.

Report options:

Many kind of statistics (on-line and off-line) can be gathered.

Miscellaneous functions:

Includes random number seeding, user-defined printing of strings, and version information.

This is the abstract specification:

```
/**
  * Abstract frame PGA (Basic version)
  *
  * Jordi Petit i Silvestre, 1997
  * Departament de Llenguatges i Sistemes Informatics
  * Universitat Politecnica de Catalunya
  *
  * http://www-lsi.upc.es/~jpetit/PgaFrame
  *
  */


ABSTRACT FRAME PGA;

DECLARATION

    /* Enumeration types */
    TYPE Boolean = enum (False, True);
    TYPE DataTypeEnum = enum (Binary, Integer, Real, Character);
    TYPE OptDirectionEnum = enum (Maximize, Minimize);
    TYPE ReplacementTypeEnum = enum (ReplaceBest, ReplaceRandom, ReplaceRandomNoRep);
    TYPE MutationCrossoverEnum = enum (MutationCrossoverOR, MutationCrossoverAND);
    TYPE CrossoverTypeEnum = enum (CrossoverOnePt, CrossoverTwoPt, CrossoverUniform, CrossoverUser);
    TYPE FitnessMappingTypeEnum = enum (FitnessRaw, FitnessRanking, FitnessNormal);
    TYPE MinimizationFitnessTypeEnum = enum (FitnessMinCMax, FitnessMinReciprocal);
    TYPE SelectionTypeEnum = enum (SelectionProportional, SelectionSUS,
                   SelectionTournament, SelectionProbTournament);
    TYPE InitializationTypeEnum = enum (InitZero, InitBinRandom, InitCharLower, InitCharUpper,
                   InitCharMixed, InitPermutation, InitRange, InitPercent, InitUser);
    TYPE MutationTypeEnum = enum (MutationConstant, MutationRange, MutationUniform,
                   MutationGaussian, MutationPermutation,MutationUser);

    /* Out types (privates) */
    OUT TYPE PGAContext;
    OUT TYPE FILE;

    /* Pointer equivalent yypes */
    TYPE PGAContextPtr = PGAContext*;
    TYPE FILEPtr = FILE*;
    TYPE charPtrPtr = char**;

    /* Fundamental parameters */
    CONSTANT DataTypeEnum DataType;
    CONSTANT OptDirectionEnum Direction;
    FUNCTION double Evaluate (PGAContextPtr ctx, int p, int pop);
    FUNCTION int Preprocess (int argc, charPtrPtr argv);
    OPTIONAL PROCEDURE Postprocess (int argc, charPtrPtr argv, PGAContextPtr ctx);
    OPTIONAL STATEMENT UserCode ();

    /* Population replacement */
    OPTIONAL CONSTANT int PopulationSize = "100";
    OPTIONAL CONSTANT int ReplacementValue = "10";
    OPTIONAL CONSTANT ReplacementTypeEnum ReplacementType = "ReplaceBest";
    OPTIONAL CONSTANT MutationCrossoverEnum MutationCrossover = "MutationCrossoverOR";
    OPTIONAL CONSTANT Boolean AllowDuplicates = "False";
    OPTIONAL FUNCTION int DuplicateChecking (PGAContextPtr ctx, int p1, int pop1, int p2, int pop2);

    /* Stopping criteria */
    OPTIONAL CONSTANT Boolean UserDefinedStopRule = "False";
    OPTIONAL CONSTANT Boolean StopMaxIter = "True";
    OPTIONAL CONSTANT Boolean StopNoChange = "False";
    OPTIONAL CONSTANT Boolean StopTooSimilar = "False";
    OPTIONAL CONSTANT int MaxIterValue = "1000";
    OPTIONAL CONSTANT int MaxNoChangeValue = "100";
    OPTIONAL CONSTANT int MaxSimilarValue = "95";
    OPTIONAL FUNCTION int StopCondition (PGAContextPtr ctx);

    /* Initialization */
    OPTIONAL CONSTANT InitializationTypeEnum InitializationType;
    OPTIONAL CONSTANT double BinInitProbBit1 = "0.5";
    OPTIONAL CONSTANT int IntInitRangeLow = "0";
    OPTIONAL CONSTANT int IntInitRangeHigh = "ChromosomeLength";
    OPTIONAL CONSTANT double RealInitRangeLow = "0.0";
    OPTIONAL CONSTANT double RealInitRangeHigh = "1.0";
    OPTIONAL PROCEDURE Initialize (PGAContextPtr ctx, int p, int pop);

    /* Selection */
    OPTIONAL CONSTANT SelectionTypeEnum SelectionType = "SelectionTournament";
    OPTIONAL CONSTANT double ProbabilisticTournamentProb = "0.6";

    /* Crossover */
    OPTIONAL CONSTANT CrossoverTypeEnum CrossoverType = "CrossoverTwoPt";
    OPTIONAL CONSTANT double CrossoverProb = "0.85";
    OPTIONAL CONSTANT double UniformCrossoverProb = "0.6";
    OPTIONAL PROCEDURE Crossover (PGAContextPtr ctx, int c1, int c2, int c_pop, int p1, int p2, int p_pop)

    /* Mutation */
    OPTIONAL CONSTANT double MutationProb = "0.001";
    OPTIONAL CONSTANT MutationTypeEnum MutationType;
    OPTIONAL CONSTANT Boolean BoundMutation = "True";
    OPTIONAL CONSTANT int IntMutationValue = "1";
    OPTIONAL CONSTANT double RealMutationValue = "0.1";
    OPTIONAL FUNCTION int Mutation (PGAContextPtr ctx, int p, int pop, double mr);

    /* Fitness */
    OPTIONAL CONSTANT FitnessMappingTypeEnum FitnessMappingType = "FitnessRaw";
    OPTIONAL CONSTANT double MaxFitnessRank = "1.2";
    OPTIONAL CONSTANT MinimizationFitnessTypeEnum MinimizationFitnessType = "FitnessMinCMax";
    OPTIONAL CONSTANT double MinimizationMultiplier = "1.01";

    /* Restart */
    OPTIONAL CONSTANT Boolean ApplyRestartOperator = "False";
    OPTIONAL CONSTANT int RestartFrequency = "50";
    OPTIONAL CONSTANT double RestartAlleleChangeProb = "0.5";

    /* Report options */
    OPTIONAL CONSTANT Boolean OnlineAnalysis = "False";
    OPTIONAL CONSTANT Boolean OfflineAnalysis = "False";
    OPTIONAL CONSTANT Boolean WorstEvaluation = "False";
    OPTIONAL CONSTANT Boolean AverageEvaluation = "False";
    OPTIONAL CONSTANT Boolean HammingDistance = "False";
    OPTIONAL CONSTANT Boolean StringItself = "False";
    OPTIONAL CONSTANT int FrequencyValue = "10";
    OPTIONAL PROCEDURE EndOfGeneration (PGAContextPtr ctx);

    /* Miscellaneous */
    OPTIONAL CONSTANT int RandomSeed;
    OPTIONAL CONSTANT Boolean PrintContextVariable = "False";
    OPTIONAL CONSTANT Boolean PrintPGAVersionNumber = "False";
    OPTIONAL CONSTANT Boolean PrintPGAFrameVersionNumber = "False";
    OPTIONAL PROCEDURE PrintString (PGAContextPtr ctx, FILEPtr f, int p, int pop);

DOCUMENTATION
    DOC(PGA) = URL "http://www-lsi.upc.es/~alcom-it/frames/FramesRepositories/PGA/PGA.html";

END PGA.
```

The instance level

In the second level (instance level) of the Frames approach, the user is asked to bind values to the abstract parameters. This part is therefore the inter-changeable part of the PGA frame. The generic genetic algorithm is converted in this level in a pourpose specific genetic algorithm by simply instancying the abstract parameters. The values binded to the parameters are expected to be given in ANSI C and without parallel constructs. This is why the user is not supposed to have knowledge in parallel programming.

The user can specify his/her instance in three different ways:

- Using the textual instance specification language defined in Frames, which is simply a list of assignments of values to the abstract parameters.

- Using a graphical user interface we have designed *adhoc* for the PGA frame (called **xpga**).
- Using the Frames Instantiator Tool (FIT), ie the graphical user interface automatically generated by the Frames tools from the abstract specification.

Specifying the instance level textually is teddious, borring and error-prone, so we claim that, in general, users should use one of the two graphical interfaces available.

The **xpga** GUI has been expressely implemented to work with the PGA frame. Due to this knowledge, it is very practical, because it integrates the set of parameter families in different windows that can be displayed independently. It has been implemented using the Tcl/Tk programming language and thus it can be executed on almost any machine running Unix, Windows 95 or Macintosh.

The **Frames Instantiator Tool** is a generic tool to instantiate any frame, and thus, can work with the PGA frame. With this graphical tool, the user can easily access the Frames Repositories, input the values for the abtract parameters, display their corresponding documentation, and launch the compilation process. Since this tool is written in the Java programming language, it can be ran within a WWW browser such as Netscape. Of course, the FIT does not look so nice as the **xpga** because the GUI is generated automatically, but it brings some advantages as generality, portability and uniformity.

Last but not least, in the second level, the **inscc** tool of the Frames system carries out a type checking of the given instance against the abstract definition in order to verify that all the assignments are legal.

The implementation level

The implementation level contains the sources of the generic genetic algorithm and the rules on how to modify these sources to get an implementation that complies with the given values in the instance level. The PgaPack library is also provided in this level. The **impcc** tool of Frames processes all this information in order to create a directory with the new generated sources and a makefile. The user is then ready to compile them and execute them in his/her favourite computer, provided that an MPI implementation is available (for example, MPICH)

Internally, the PgaPack library uses a master/slave strategy in order to execute concurrently the genetic operators. This is sound, because most of the time used by the program is expended in computing the evaluation function for different strings. As there is any dependence with these calculations, their parallel executions brings high speedups.

An example: solving the TSP with the PgaFrame

As a simple demonstrator for the PGA frame, let us make a quick specification to approximate the Traveling Salesman Problem (TSP). The emphasys in this section is pedagogic, to show how to use the PgaFrame. It is not by any means a work of research about solving the TSP with genetic algorithms.

In the TSP, a salesman must visit *n* cities. Modeling the problem as a complete graph with *n* vertices, we can say that the salesman wishes to make a **tour** visiting each city exactly once and finishing at the city he starts from. There is a cost *M(i,j)* to travel from city *i* to city *j*, and the saleman wishes to make the tour whose total cost is minimum, where the total cost is the sum of the individual costs along the edges of the tour. Since this is an NP-complete problem, we will not try to find the minimum tour but one tour close to the minimum. And we will do that using genetic algorithms.

Encoding and the genetic operators

In our genetic algorithm, a possible tour will be encoded in chromosomes whose alleles will be integers and will represent a permutation of 0 to *n-1* (a tour of the cities using C language). The evaluation function will be the length of the tour. We will initialize chromosomes at random with permutations, make mutations as flips permutations and use the PMX crossover operator. As global variables we will have the integer `n` and the matrix `M` of inter-city disances. In the preprocessing phase, we will read the matrix of distances beetwen cities and in the postprocess we will output the best tour found so far. Of course, we want to minimize the evaluation function.

The evaluation of a tour simply consists in adding the lengths between the cities that make it up it. This is done in the `Evaluate` function. Here the `PGAGetIntegerAllele` function is used to read a detemined allele of a chromosome.

The pre-process consists on filling the `n` and `M` variables with the information provided through a file, whose name will be the first argument (`argv[1]`). In the post-process function, we will suppose that a file named `argv[2]` will hold the 2D coordinates of the cities, will order them according to the best tour found and will save them in a file named `argv[3]`. We remark that the genetic algorithm we will build will not be restricted to two dimensional TSP problems. We output the solution in this way only because with 2D instances the tours can be easily drawn with tools such as `gnuplot`.

Let us now explain the mutation operator of user function `Mutation`. Given a chromosome, this function flips a coin according to the mutation probability `mr`. If the mutation is accepted, two alleles choosen at random in the interval `0..n-1` are exchanged.

The crossover function we will apply is the so-called PMX operator. This operator applied to two tours A and B does the following: A position i is chosen uniformly at random from `0..n-1`, designating a transposition T=(A[i],B[i]). This transposition is applied to yield two children AT and BT. This is the basic operation. The full-blown crossover operator in fact picks two positions `j<=k` uniformly at random and performs the above operation on positions `j` to `k`. As an example, consider two tours represented by the chromosomes A=(1 4 3 2 0 5 7 6) and B=(6 7 5 0 3 1 4 2). Applying the previous operator to position 3, we obtain A'=(1 4 3 0 2 5 7 6) and B'=(6 7 5 2 3 1 4 0).

The rest of the instance we show in the next subsection contains the implementation of the functions we have just described.

Textual instance

We can give as instance the following file (**`TSP.ins`**):

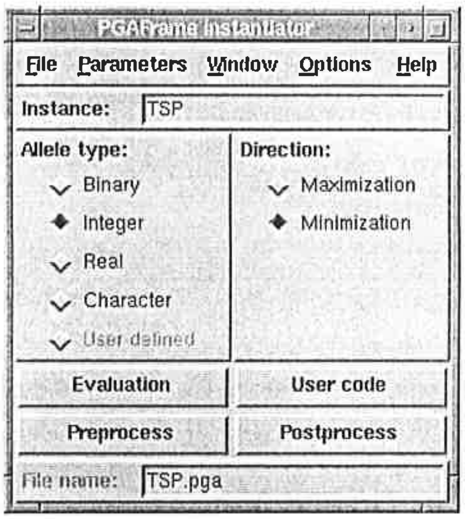
```
FRAME TSP IS PGA (MPI)

        CONSTANT DataType = Integer;
        CONSTANT Direction = Minimize;

        FUNCTION Evaluate = double Evaluate (PGAContextPtr ctx, int p, int pop) {
```

```
        int i,a,b;
        double s=0;

        for ( i=0; i<n; i++ ) {
                a=PGAGetIntegerAllele(ctx,p,pop,i);
                b=PGAGetIntegerAllele(ctx,p,pop,(i+1)%n);
                s+=M[a][b];
        }
        return s;
};

FUNCTION Preprocess = int Preprocess (int argc, charPtrPtr argv) {
        int i,j;
        FILE *f;

        f=fopen(argv[1],"r");
        fscanf(f,"%i",&n);
        for (i=0; i<n; i++) for (j=0; j<n; j++)
                fscanf(f,"%lf",&M[i][j]);
        fclose(f);
        return n;
};

PROCEDURE Postprocess = Postprocess (int argc, charPtrPtr argv, PGAContextPtr ctx) {
        int p,i,j;
        double x[200],y[200];
        FILE *f;

        p=PGAGetBestIndex(ctx, PGA_OLDPOP);

        f=fopen(argv[2],"r");
        for (i=0; i<n; i++) {
                fscanf(f,"%lf",&x[i]);
                fscanf(f,"%lf",&y[i]);
        }
        fclose(f);

        f=fopen(argv[3],"w");
        for (i=0; i<n; i++) {
                j=PGAGetIntegerAllele(ctx,p,PGA_OLDPOP,i);
                fprintf(f,"%lf %lf\n",x[j],y[j]);
        }
        j=PGAGetIntegerAllele(ctx,p,PGA_OLDPOP,0);
        fprintf(f,"%lf %lf\n",x[j],y[j]);
        fclose(f);
};

STATEMENT UserCode =  {
        int n;                  /* Size of the problem */
        double M[200][200];     /* Distances between cities */
};


/* Population replacement */
CONSTANT PopulationSize = 1000;
CONSTANT ReplacementValue = 900;


/* Initialization (integer) */
CONSTANT InitializationType = InitPermutation;

/* Crossover */
PROCEDURE Crossover = Crossover (PGAContextPtr ctx, int p1, int p2,
    int p_pop, int c1, int c2, int c_pop) {
        int i,j,k,l,t1,t2;

        j=PGARandomInterval(ctx,0,n-1); k=PGARandomInterval(ctx,0,n-1);
        if (k<j) {i=j; j=k; k=i;}
        PGACopyIndividual(ctx,p1,p_pop,c1,c_pop);
        PGACopyIndividual(ctx,p2,p_pop,c2,c_pop);

        for (i=j; i<=k; i++) {
                t1=PGAGetIntegerAllele(ctx,c1,c_pop,i);
                t2=PGAGetIntegerAllele(ctx,c2,c_pop,i);

                for (l=0; PGAGetIntegerAllele(ctx,c1,c_pop,l)!=t2; l++);
                PGASetIntegerAllele(ctx,c1,c_pop,l,t1);
                for (l=0; PGAGetIntegerAllele(ctx,c2,c_pop,l)!=t1; l++);
                PGASetIntegerAllele(ctx,c2,c_pop,l,t2);
                PGASetIntegerAllele(ctx,c1,c_pop,i,t2);
                PGASetIntegerAllele(ctx,c2,c_pop,i,t1);
}       };

CONSTANT CrossoverType = CrossoverUser;


/* Mutation (integer) */

CONSTANT MutationProb = 0.05;
CONSTANT MutationType = MutationUser;
FUNCTION Mutation = int Mutation (PGAContextPtr ctx, int p, int pop, double mr) {
        int m=0,i,j,a1,a2;
        if (PGARandomFlip(ctx,mr)) {
                i=PGARandomInterval(ctx,0,n-1); j=PGARandomInterval(ctx,0,n-1);
```

```
                        if (i!=j) {
                                m++;
                                a1=PGAGetIntegerAllele(ctx,p,pop,i);
                                a2=PGAGetIntegerAllele(ctx,p,pop,j);
                                PGASetIntegerAllele(ctx,p,pop,i,a2);
                                PGASetIntegerAllele(ctx,p,pop,j,a1);
                }       }
                return m;
        };

END TSP.
```

Using the GUIs for specifying the instance

Instead of writing this file, a user could have used the **xpga** tool that generates it automatically. The user task would then be limited to fill the entries of the GUI. For instance, the general parameters and the population strategy have been specified in these windows:

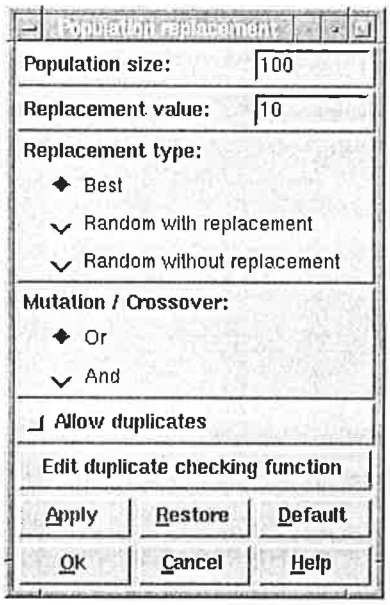


Instance: TSP

Allele type: Binary [✓ Integer] Real Character User defined

Direction: Maximization [✓ Minimization]

File name: TSP.pga



Population size: 100

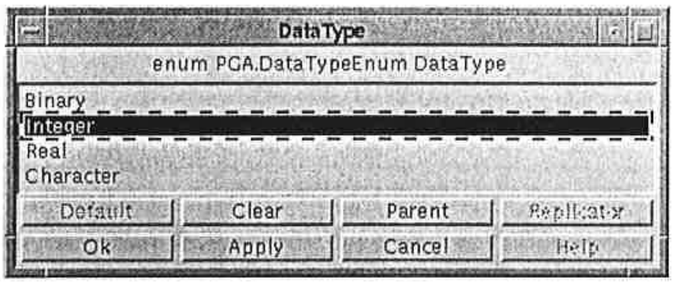
Replacement value: 10

Replacement type: [✓ Best] Random with replacement Random without replacement

Mutation / Crossover: [✓ Or] And

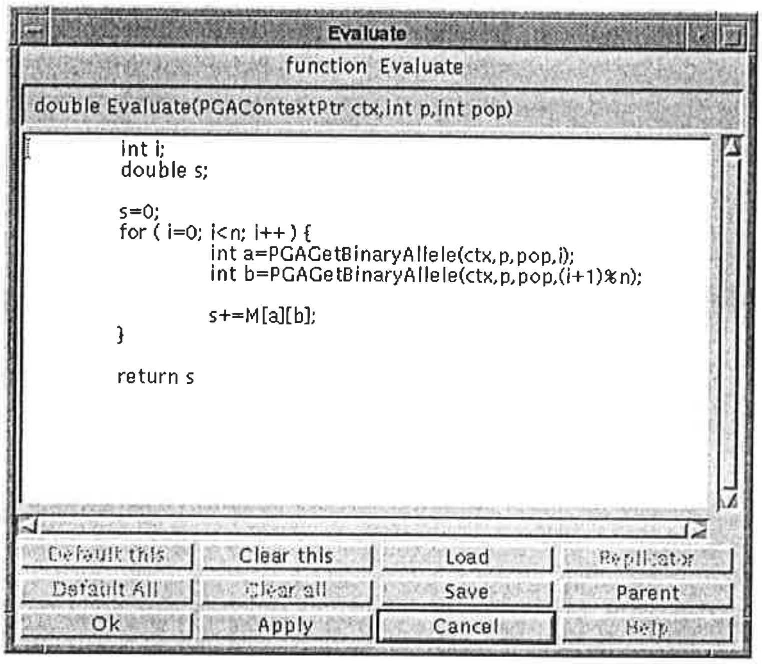
Allow duplicates

Another alternative would have been used the Frames Instantiator Tool. The look of this tool is as follows:

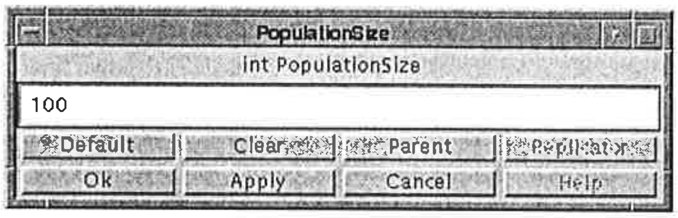


enum PGA.DataTypeEnum DataType

Binary
[✓ Integer]
Real
Character



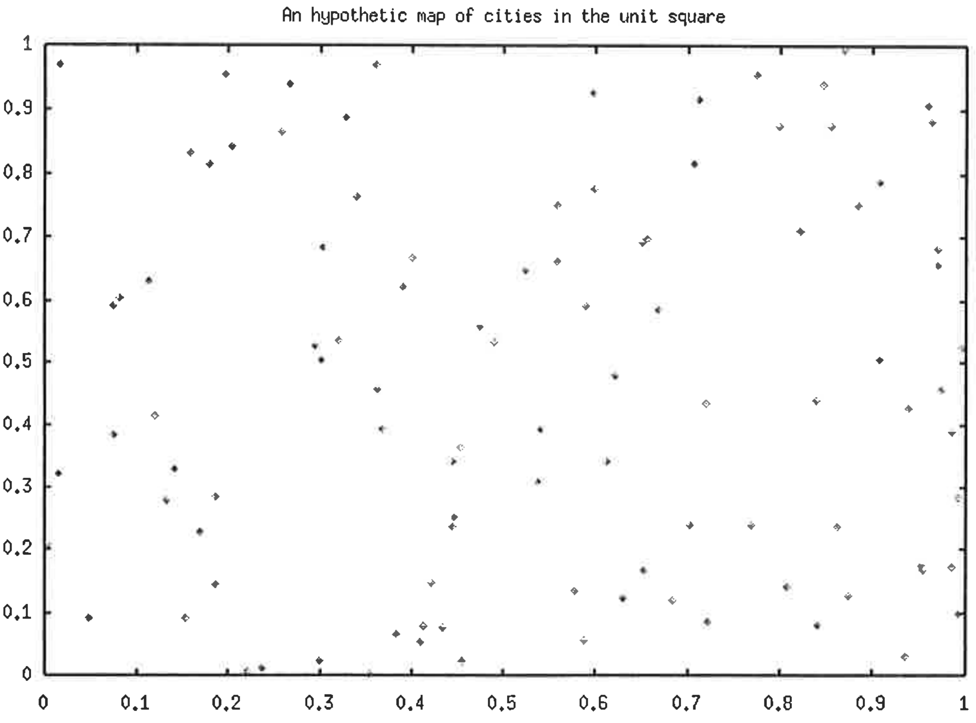
```
double Evaluate(PGAContextPtr ctx,int p,int pop)
        int i;
        double s;

        s=0;
        for ( i=0; i<n; i++ ) {
                int a=PGAGetBinaryAllele(ctx,p,pop,i);
                int b=PGAGetBinaryAllele(ctx,p,pop,(i+1)%n);

                s+=M[a][b];
        }

        return s
```



```
100
```

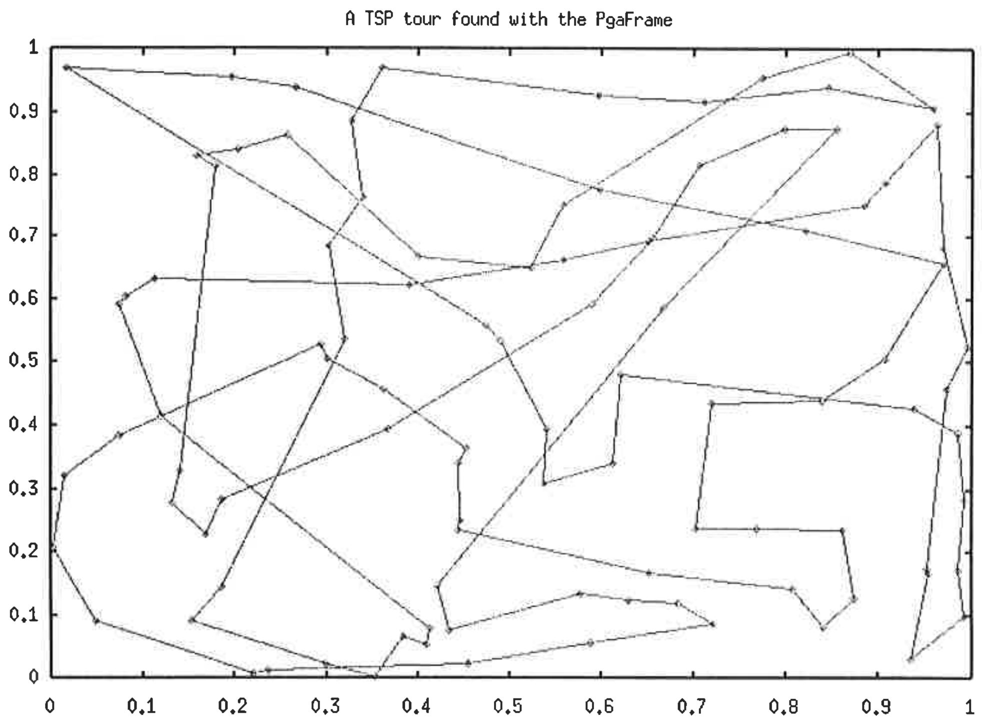
The results

We ran the previous genetic algorithm on the input given by the following map (100 cities):



The best tour we found is shown in the next figure. The tour doest not look quite optimal, but is nevertheless a "good" solution. It must be remarked that, even if this inputs instance satisfies the triangle inequality, the genetic program we have developed does not use this information.

More information about the TSP and genetic algorithms can be found, for example, in [8].



Future work

This is the first version of the PgaFrame. It is also the second complex frame that has been developed following the Frames approach and its tools. As such, it is still a current working project and new enhancements are planned. Some of them are the following:

- User-defined datatypes.
- Creation of a new function parameter `SetUp` in order to modify the instance given at compile time at running time. For example, in this way it would be possible to adapt the size of the population respect to the input.
- Support to hibridization.
- Explicit usage of the main loop, ie over the steps performed in a single iteration of the GA.
- More convenient support for input/output. As soon as this will be included in Frames, the PGA will support it.

Summary

Lots of people involved in the algorithmic community (or in other fields where optimization methods are needed) are willing to apply genetic algorithms to their problems. However, they cannot invest a large effort to develop a truly efficient, parallel and portable program that could satisfy their needs. Thanks to the PgaFrame, experts from these fields but not in parallel programming nor in evolutionary algorithms can easily specify a genetic program and run it on the most powerfull parallel computers.

Acknowledgements

This work is heavily based in the PgaPack library and in the Frames project.
I would like to thank Thomas Römke, Jordi Giribet, Josep Díaz and Maria Serna for their help in preparing this work.

More information

Information about the PgaFrame will be keep on-line at `http://www-lsi.upc.es/~jpetit/PgaFrame` and at `http://www-lsi.upc.es/~alcom-it/frames/FramesRepository/PGA`.

Jordi Petit i Silvestre is recheable by electronic mail as `jpetit@lsi.upc.es`.

References

1. ***Programming Frames for the Efficient use of Parallel Systems*** (1997)
   - Thomas Römke / Jordi Petit i Silvestre
   - TR-001-97 (PC² Paderborn) / LSI-97-9-R (LSI Barcelona)
   - frames.ps.gz (PostScript + gzip)
2. ***The Frames Poster*** (1997)
   - Jordi Petit i Silvestre / Thomas Römke
   - LSI-97-1-T
   - poster1.ps.gz (PostScript + gzip)
   - poster1.eps.gz (EncapsulatedPostScript + gzip)
   - poster1.a1.eps.gz (DIN A1 - EncapsulatedPostScript + gzip)
3. ***FIT: A Frame Instantiator Tool - Current Status*** (1996)
   - Jordi Petit i Silvestre / Jordi Giribet Perich / Thomas Römke / Uwe Dralle
   - LSI-96-14-T
   - fit1.ps.gz (PostScript + gzip)
4. ***Frames: Progress report and documentation*** (1996)
   - Thomas Römke / Jordi Petit i Silvestre
   - http://www.uni-paderborn.de/~alcom-it/PReport (HTML)
5. ***User Guide to the PGAPack Parallel Genetic Algorithm Library*** (1996)

- David Levine
- user_guide.ps (Postscript)

6. *An Overview of Genetic Algorithms: Part 1, Fundamentals* (1993)

   - David Beasley / David R. Bull / Ralph R. Martin
   - ga_overview1.ps (Postscript)

7. *An Overview of Genetic Algorithms: Part 2, Reseach topics* (1993)

   - David Beasley / David R. Bull / Ralph R. Martin
   - ga_overview2.ps (Postscript)

8. *The Traveling Salesman Problem, A Study in Local Optimization* (1993)

   - David S. Johnson / Lyle A. McGeoch
   - (Postscript)

9. *PgaFrame - User documentation* (1993)

   - Jordi Petit i Silvestre
   - http://www-lsi.upc.es/~alcom-it/frames/FramesRepository/PGA/PGA.html (HTML)