Affectation automatique des relecteurs de la conférence PPSN VI : un exemple d'utilisation du language EASEA

Pierre Collet, Evelyne Lutton, Marc Schoenauer

▶ **To cite this version:**

Pierre Collet, Evelyne Lutton, Marc Schoenauer. Affectation automatique des relecteurs de la conférence PPSN VI : un exemple d'utilisation du language EASEA. [Research Report] RR-4177, INRIA. 2001, pp.32. inria-00000850

HAL Id: inria-00000850

https://inria.hal.science/inria-00000850

Submitted on 24 Nov 2005

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Affectation automatique des relecteurs de la conférence PPSN VI : un exemple d'utilisation du language EASEA

Pierre COLLET — Evelyne LUTTON — Marc SCHOENAUER

N° 4177

February 2001

THÈME 4

ISSN 0249-6399

INRIA
ROCQUENCOURT

Affectation automatique des relecteurs de la conférence PPSN VI : un exemple d'utilisation du language EASEA

Pierre COLLET*, Evelyne LUTTON†, Marc SCHOENAUER‡

Thème 4 — Simulation et optimisation
de systèmes complexes
Projets Fractales

Rapport de recherche n° 4177 — February 2001 — 29 pages

Résumé : Ce rapport présente un algorithme qui a été effectivement utilisé par les organisateurs de la conférence PPSN VI pour affecter ses relecteurs aux papiers soumis, ce qui peut être considéré comme un problème de satisfaction de contraintes. Le but n'est pas ici de présenter une méthode révolutionnaire et compétitive de résolution du problème de satisfaction de contraintes, mais plutôt de montrer comment on peut programmer de façon simple un tel problème à l'aide du language de spécification d'algorithmes évolutionnaires EASEA.

Mots-clés : algorithmes évolutionnaires, algorithmes génétiques, satisfaction de contraintes

* CMAPX, École Polytechnique, 91128 Palaiseau cedex, France, Pierre.Collet@polytechnique.fr
† Projet FRACTALES, INRIA, B.P. 105, 78153 Le Chesnay cedex, France, Evelyne.Lutton@inria.fr
‡ CMAPX, École Polytechnique, 91128 Palaiseau cedex, France,Marc.Schoenauer@polytechnique.fr

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 —Télécopie : +33 1 39 63 53 30

PPSN VI Reviewers and Papers: an EASEA Match

Abstract: This paper presents an algorithm which has been used by PPSN VI organisers to allocate papers to reviewers, which is typically a Constraint Satisfaction Problem. Its aim is not to present a new revolutionary competitive method to solve CSPs, but rather to show how such a problem can be simply implemented using EASEA, a language designed specifically to write evolutionary algorithms.

Key-words: evolutionary algorithms, genetic algorithms, Constraint Satisfaction Problem

1 Introduction

Matching papers and reviewers has always been a major cause of baldness amongst conference organisers, unless they manage to avoid this chore by delegating the job to someone else, rewarded with some honorific title.

The quality of the conference somehow depends on the nice match between papers and reviewers: judging a paper can rapidly become problematic if the subject falls outside of the reviewer's field of competence. Honest reviewers will tell the organiser they cannot evaluate the paper, causing delays due to the necessary redirection while shy ones will probably give an average mark, unless the paper is utterly unreadable.

A bad match between reviewers and papers therefore has at least two main consequences:

1. embarrassed reviewers, wishing they had not accepted to review papers for this conference in the first place.
2. and above all, a boring conference, where uninteresting papers have been selected by incompetent reviewers.

This paper presents a simple evolutionary algorithm written with EASEA v0.35 trying to match hundreds of papers with hundreds of reviewers, mainly according to keywords and a couple of constraints. Rather than trying to compete with state of the art CSP-solving methods, this paper aims to:

- demonstrate how easily such an algorithm can be written with EASEA v0.35,
- cut down the budget of hair lotion in research teams.

2 Presentation of the EASEA programming language

While many important fields in computer science have their specific languages (FORTRAN, C/C++, LISP, PROLOG, SMALLTALK, ...), not speaking of complex applications such as databases spread-sheets, and web-browsers which have also developed their own language (!), EA programmers remain with inadapted general purpose languages.

Unfortunately, EAs are not that straightforward to implement and the lack of any specialised language forces users to reinvent the wheel every time they want to write a new program.

One way to speed up the process is to use one of the many existing evolutionary libraries. All is for the best as they offer very powerful tools provided ... one is fluent enough with constructors, copy-constructors, destructors and such niceties involved by relatively low-level object languages.

The next hurdle is then to learn how to use the library, to understand the intricate data structures and to memorise the necessary several hundred object types, functions and variables and the way they are inter-related. This can be quite time consuming when all major evolutionary libraries are written in C++ or JAVA and make full use of object programming.

All in all, many physicists, chemists, mathematicians and other scientists who otherwise would be capable of writing relatively elaborate functions in C, FORTRAN or LISP are denied experimentation of evolutionary algorithms due to the sheer complexity of their implementation. When they go through the long and difficult process of writing their own evolutionary programs, their results are barely comparable due to thoroughly different programming techniques and languages, which is a great obstacle to scientific cooperation and emulation.

The aim of EASEA (EAsy Specification of Evolutionary Algorithms) is to hide this complexity behind a high-level language, allowing scientists to concentrate on evolutionary algorithms, rather than on their implementation.

EASEA v0.6 is available on the net at: `http://www-rocq.inria.fr/EVO-Lab/`.

2.1 Mode of operation

Rather than throwing down the drain all the man-years already spent in the development of evolutionary libraries by recreating our own, EASEA is designed to *reuse* already existing libraries.

This means that object files resulting from the EASEA compilation of `.ez` source files are *C++ source files*, using the objects of an existing evolutionary library. The resulting C++ file is in turn compiled and linked with the library to finally produce an executable file implementing the evolutionary algorithm specified in the original `.ez` file.

Two libraries have been chosen to start with: GAlib —a widely used C++ genetic library [2]— and EO (Evolving Objects [5]) initially developed at the University of Granada (Spain) within the EVONET [4] framework.

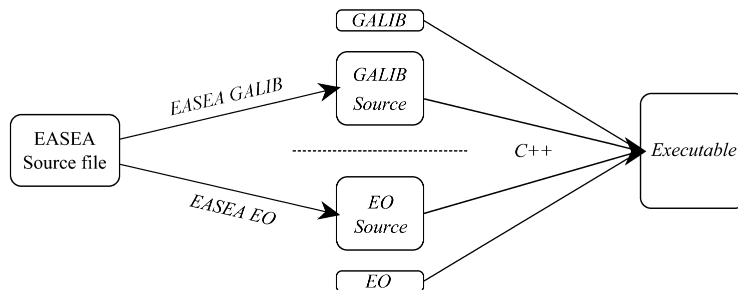

FIG. 1 – *EASEA mode of operation.*

The EASEA–GAlib compiler uses for input an ascii file with a `.ez` suffix. Its output is a GAlib or an EO C++ source file containing calls to GAlib or EO functions and objects. The resulting C++ file must then be compiled by a C++ compiler and linked with the GAlib or EO library (cf. figure 1). The produced executable implements the evolutionary algorithm described in the original EASEA source file.

3 Matching Papers and Reviewers

To begin with, all authors and reviewers were given the same list of fixed keywords to choose from. All the necessary information is in fact stored in four databases:

1. Among other fields, the reviewers database contains their identification number, their name, their e-mail address and a set of keywords listing the fields in which they are competent.
2. The interesting fields of the papers database are the identification number, the title, the list of authors, their e-mail addresses and a list of keywords related to the paper.

For the first time this year, PPSN VI organisers have decided to let reviewers chose which papers they would rather review and which they would rather not. This policy added two extra files:

3. A file containing the reviewer's identification number along with a list of paper IDs he/she would rather review
4. A second file containing the reviewer's Id along with a list of paper IDs he/she would rather not review.

3.1 Matching criteria

- A *basic* match between papers and reviewers supposes that at least two constraints be satisfied:
  1. Each paper must be examined by N reviewers.
  2. A reviewer must not review a paper of which he is an author.
- *Nicely* matching papers and reviewers involves maximising an evaluation function, depending on the following criteria:
  1. If 156 papers are to be reviewed by 178 reviewers and if each paper should be reviewed $N = 3$ times, this means that each reviewer should be given an average of $155 \times 3/178 = 2.61$ papers to review. It is understandable that we should try to avoid having too many reviewers with 0 or 6 papers to review.
  2. So as to avoid biases, it is preferable that reviewers should not know authors personally. Although this is difficult to determine considering the little information available in the databases, we can attempt to minimise such "risks" thanks to the e-mail fields.

     The last field of an e-mail address allows us to determine the country while the last but one usually contains the institution name. Our matching algorithm will then avoid as much as possible to assign a reviewer a paper in which an author's e-mail address has the same last two fields than the reviewer's.

Of course, hotmail or yahoo users will be prevented to review papers written by other hotmail or yahoo users, but that is another story.

3. As reviewers have taken the pain to select which papers they would like or would not like to review based on the paper title and abstract, the algorithm should try to take their preference into account.
4. Finally, the program should try to match papers and reviewers who have a maximum of keywords in common.

In fact, experience has shown that criterion 4 should have been more important than criterion 3: strongly refusing to match a reviewer and a paper he has refused to read had the side effect of considerably diminishing the number of papers a reviewer could read within his own field.

4 Algorithmic choices

4.1 Satisfying constraints

Constraints handling is a crucial problem for EA in general. However, one should distinguish between Constrained Optimization Problems (COP), in which the goal is to minimize some objective function while satisfying a few (generally numerical) constraints, and Constraint Satisfaction Problems (CSP), that involve a large number of generally boolean constraints (typically several thousands) and in which the goal is to find either a point of the search space satisfying all constraints, or to minimize the number of violated constraints. Of course this is not a strict disctinction, and many problems involve both numerical and boolean constraints, with some objective function to minimize, but the number and nature of the constraints is the primary criterion that determines the choice of a particular method to tackle a problem involving constraints.

Several methods have been proposed for COPS (see e.g. [9] for a complete review of these methods), and for CSPs (e.g. [7, 6]), including:

1. methods based on preserving feasability of solutions, for example using specialized operators which transform valid individuals into other valid ones, or repair operators (projection on the feasible region),

2. methods based on penalty functions (weighted sum of the constraints violations included in the fitness function),

The second solution has the advantage of being very simple to implement, although it considerably enlarges the search space with bad solutions, meaning that many generations will be devoted to sieving out unviable solutions.

The first solution presents the advantage of considerably reducing the search space to viable solutions. Its main disadvantage is that it is more complex to implement, leading

to slower genetic operators (the mutator must make sure the mutation does not result in a non-viable solution) but above all, if the problem has no solution, the evolutionary algorithm will stop when trying to initialise the first individual of the first generation ... which is not of much help to the conference organisers.

A closer look to our constraints will however show that they are of different nature: while constraint number 2 (a reviewer must not review a paper of which he is an author) alone can directly lead to a deadlock (one paper co-authored by all reviewers), constraint number 1 (each paper must be examined by N reviewers) alone can always be satisfied, provided there are at least N reviewers.

Our implementation will take a bit of both solutions: our genome structure will enforce constraint number 1, and we will let the evolutionary algorithm try to satisfy constraint number 2 with help of weighted penalty functions. Another justification for using such a strategy is that such penalty functions allow "smoother" fitness functions, which clearly tend to make the job easier for the GA (see for example [8] for theoretical considerations about fitness irregularities influence).

Moreover, we chose to implement the unwillingness of reviewers to be assigned a paper as a supplementary constraint, so that their choice is not neglected by the algorithm. The initialising and mutator functions make sure that they do not assign a paper to a reviewer who has expressly mentioned his reluctance to review it.

4.2 Structure of a genome

Knowing N (number of reviewers per paper), the simplest structure capable of representing a solution is an array of P papers to which N reviewers are assigned, resulting in a bidimensional $P \times N$ array. This has the advantage of automatically satisfying constraint number 1.

4.3 Genetic operators

4.3.1 Initialisation function

Each paper in the array is randomly assigned N reviewers (constraint 1 is satisfied, while constraint 2 will be taken care of by the evaluation function).

As stated before, we try to take into account the reviewers's point of view by exclusively initialising papers with reviewers who have expressed their willingness to review the paper. If less than three reviewers have expressed their interest, we select reviewers at random among those who have not refused to review the paper. It appears that this decision was a bad one, as this has seriously reduced the number of possible papers for a reviewer within his field.

4.3.2 Mutation function

Thanks to the remarkable robustness of evolutionary algorithms, we can allow ourselves to use the following (rather crude) mutation function:

```
for all papers in the genome:
  if tosscoin(pMut) returns true,
    randomly choose N new reviewers for the current paper among those
    who have not refused to review the paper.
```

4.3.3 Cross-over function

To keep it simple, a single point cross-over can be easily defined as follows :
Let `parent1` and `parent2` be the two genomes out of which `child1` and `child2` must be generated, and let `L` be the locus where the cross-over will take place:

- `child1` will inherit papers 0 to $L-1$ from `parent1` and will inherit papers L to $P-1$ from `parent2`.
- `child2` will inherit papers 0 to $L-1$ from `parent2` and will inherit papers L to $P-1$ from `parent1`.

4.4 Fitness function

The fitness function has two aims: maximising the quality of the solution, and making sure constraint 2 is satisfied.

4.4.1 Satisfying constraint 2

Here again, to keep things simple, the easiest way to ensure that constraint 2 is not busted is to punish the genome with a -1000 penalty for every offending match between paper and reviewer. If in fact the punishment is applied whenever the last two fields of the authors' e-mail addresses are found in the matched reviewers' e-mails, this automatically takes care of constraint 2 while at the same time trying to avoid possible existing connexions between authors and reviewers.

4.4.2 Fulfilling reviewers' preferences

As the unwillingness of reviewers to review a paper has already been taken into account in the initialisation and mutation functions, there remains to take into account their willingness to review a paper. This is easily done by giving a bonus of 10 points whenever the case occurs.
Maximising the quality of the solution is however more subtle, as more parameters come into consideration.

4.4.3 Matching keywords

Some reviewers have neglected to give a list of keywords describing the fields in which they are competent. If we give bonus points whenever a paper keyword matches a reviewer keyword, reviewers with no keywords will receive less papers to review than the others, which is somewhat unfair to those who have taken the pain to correctly fill up the form.

Conversely, fitness is biased by the number of keywords given for a paper: papers with 1 keyword cannot get many points from keywords match compared to a paper described with 5 keywords.

Hence, a simple boolean fitness (+1 per matched keyword) would not give balanced results.
We therefore chose to:

- proportionally grade from 0 to 10 the number of paper keywords that are matched by reviewer keywords, independently of the number of paper keywords.
- give an average bonus of 5 points to a reviewer who has given no keyword list (as if he had matched half the paper keywords).
- give a penalty of -10 to a match between a reviewer with a keyword list with 0 paper keyword match to discourage ill-matching.

4.4.4 Evening out the number of papers per reviewer

Finally, we should try to avoid rewarding hyper-competent reviewers with many more papers to review than the others. As it is easy to calculate the average number of papers which should be attributed per reviewer ($P \times N/$ number of reviewers), a penalty of the cube of the difference over this average is given, and a penalty of -5 is given per paper under this average (it is preferable to have a small number of papers per reviewer than the opposite).

5 Implementation with EASEA v0.35

The implementation in EASEA v0.35 is very straightforward. The `ppsn.ez` source file is decomposed in different sections each introduced by a different keyword :

User Specific: This section contains preprocessor directives, extern variables declarations, structures declarations for data storage as well as a couple of string comparison functions.

This section is a space of freedom for the user, written in pure C++.

Initialisation function: As indicated by its name, this section contains the body of an initialisation function immediately called by the `main` function. In `ppsn.ez`, it is used to load the defined structures with data found in the papers and reviewers databases.

This function is also written in pure C++.

Classes: This section contains the different classes needed by the genome and the genome itself:

```
Match { int reviewer[3]; }
Genome { Match paper[200]; }
```

As EASEA v0.35 cannot yet handle multidimensional arrays, we define the genome as an array of 200 papers of type `Match`. By doing so, the second reviewer of paper 23 is accessed by `paper[22].reviewer[1]`.

Standard functions: Here come all genetic operators, namely:

1. The initialisation function (matching all papers with random reviewers).
2. The crossover function (transcribing in C++ the behaviour described in section 4.3.3).
3. The mutation function (transcribing in C++ the behaviour described in section 4.3.2).
4. The evaluation function (transcribing in C++ the behaviour described in section 4.4).

Run parameters: This section speaks for itself:

```
Population size : 40
Number of generations : 5000
Mutation probability : 0.2
Crossover probability : 1
Genetic engine : SteadyState
```

Thanks to EASEA, the end-user only has to write in C++ the bodies of the "interesting" functions of an evolutionary algorithm. EASEA takes care of wrapping them into a GALib program.

A side effect is that end users do not need to know about constructors, destructors, copy-constructors and all similar niceties one must take into account when using an object-oriented language. EASEA takes care of that part. The C++ code used in `ppsn.ez` is purely procedural. The functions implementing the initialisation function, the genetic operators and the fitness function have nothing to do with object-oriented programming. They are similar to what they would have looked like had they been written in pure C, but with a C++ syntax.

The question that may legitimely arise is "Why C++?" The answer is simple: the underlying evolutionary libraries called by EASEA are written in C++, hence this choice. Subsequent versions of EASEA will allow users to write their functions in the language of their choice (FORTRAN for instance) and link them at compile time to produce an executable file.

Hence, default methods specific to C++ are transparently added for the user classes (`Match` and `Genome`), as well as an automatic genome display function. Once a satisfying prototype has been elaborated with EASEA, the end-user can switch to the generated `ppsn.cpp` file to specialise the display function for instance.

EASEA can therefore be used as a primer, creating a working prototype destined to be refined afterwards.

6 Results

6.1 GA output

The results presented in this paper were obtained with the real PPSN VI databases containing 155 papers and 178 reviewers. As it was first decided that 3 reviewers should read each paper, the average number of papers per reviewer is 2.61.

As GALib does not handle negative fitness evaluations, we offset results by 100,000 so as to allow "negative" scores.

By default, the EASEA-generated genome display method is automatically called twice: the first time on generation 0 on a newly created individual initialised by the user-written `initialiser` method, and the second time on the best genome, after the last generation has been evolved.

The run we will analyse (which has been used by the conference organisers to distribute the papers) shows that the evaluation of a first-generation individual returns 22,716. This is the result of the following sum : 100,000 + `Keyword_Contribution` $(-27,190)$ + `Willing_Contribution` (4,200) + `Distribution_Contribution` $(-54,294)$.

- The `Keyword_Contribution` value is extremely negative because we have attributed a penalty of $-1,000$ to each reviewer whose institution is the same as the institution of one of the paper's authors. A careful examination of the output file shows that 28 such matches occurred, which gives a real keyword contribution of +810. This good result is explained by the fact that the initialisation function chose for each papers those of the reviewers who had wanted to review them. Such a criterion is sure to also select reviewers with the greatest number of keywords in common with the paper.
- The `Willing_Contribution` value is maximal as the initialisation function has deliberately chosen reviewers among those who had been willing to review the papers whenever possible, regardless of the papers per reviewer distribution.
- The `Distribution-Contribution` value is therefore extremely negative, as reviewers who did not express their choices have not been selected by default. Figure 3 shows that 65 reviewers out of 178 have been given 0 papers to review and that 13 reviewers were given more than 9 papers to review, hence the very bad result.

The best genome of the 5 millionth generation of 40 individuals (obtained in 41 hours and 40 minutes on a PENTIUM II 300 Mhz) yields a result of 99,900 which can be decomposed in : 100,000 + `Keyword_Contribution` (368) + `Willing_Contribution` (500) + `Distribution_Contribution` (-968).

- The `Keyword_Contribution` value is now positive, but a careful analysis of the results shows that no $-1,000$ penalty has been given. This means that constraint 2 is fully satisfied but this also means that the real keyword contribution dropped from 810 down to 368. This drop is explained by the even reshuffle of papers among reviewers who

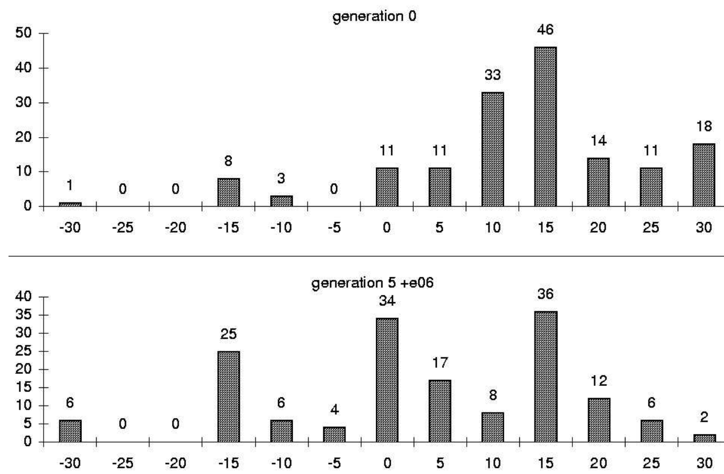

FIG. 2 – *Keyword match*

have not expressly wished to review the papers (see `Distribution_Contribution` below) and by the fact that allowing reviewers to refuse papers has introduced an negative bias: Reviewers were allowed to refuse papers exclusively among the list of papers having at least one keyword in common with their list of keywords. This means that only potentially good solutions w.r.t. keyword match have been rejected!

The histogram of keyword match values is shown in figure 2 for generation 0 and generation 5 million.

- For similar reasons, the `Willing_Contribution` value drops down to 500, meaning that only 50 papers were given to reviewers who had asked to review them.

- The `Distribution-Contribution` value is the one responsible for the previous bad results, as it has broken many optimal matches on the ground that the repartition of papers among reviewers was unbalanced. As this was nevertheless a very important criterion, it was given a great weight which explains why its value rose from $-54,294$ to -968. This is still a negative value as the distribution is not optimal (all reviewers do not have 2.61 papers to review) although it is much better than the one of the "greedy" solution of generation 0 (figure 3).

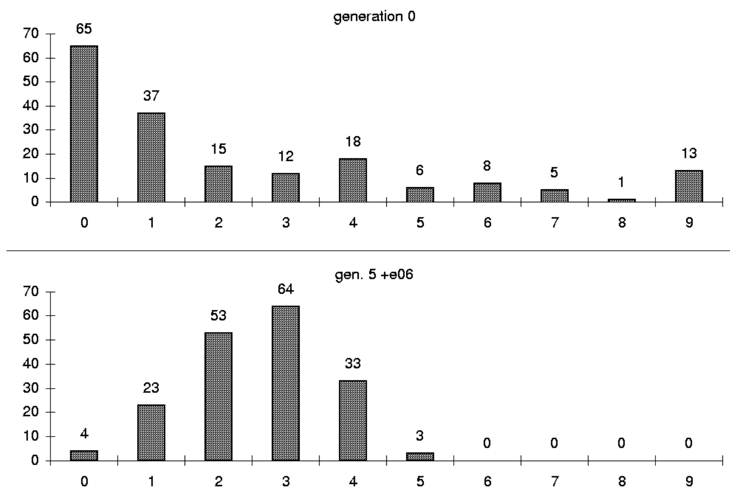

FIG. 3 – *Papers per reviewer distribution*

Extensive results cannot be given for obvious reasons of confidentiality. We will however provide:

- the ppsn.ez source file, in Appendix A.
- the EASEA-generated ppsn.cpp that has been slightly refined (modified display functions), in Appendix B.

6.2 Final result

The previously described result was passed to the conference organisers who used the automatic assignment as a basis for the final assignment.

Starting from this basis, for each paper, they have looked into the keyword match and all reviewers who were willing to review the paper. If for some paper, the assignment was mostly 0 (no keyword match) or -1 (no keyword suggested), they have replaced at least one reviewer with another one, either willing or with a keyword match. At the same time, they have tried to balance the number of papers per reviewer.

Finally, according to them, "[the final assignment] is almost the GA-generated result, with some knowledge of experts for each paper."

After this was done, the organising committee finally decided to manually add one more reviewer per paper for security, in case of defective reviewers, rising the average number of papers per reviewer to 3.50.

7 Conclusion

The very simple `ppsn.ez` program has been written both to write a first real-world application with EASEA, and to help out organisers of PPSN VI in the painstaking chore of matching papers and reviewers, which up to now was done by hand and necessitated a couple of full-time human days. The presented result (5 million generations of 40 individuals) also needed a couple of days to complete, but with less human sweat.

Although a second pass was needed to polish the raw results of the algorithm, the organisers who were in charge with papers assignment kindly asked to keep this code for future conferences, which leads us to think that the results of this program have been of some help to them, although there is certainly much room for improvement.

EASEA v0.6 is now capable to use the EO evolutionary library developed within EVONET. The next step will be for EASEA v0.7 to use the DREAM library written in JAVA.

8 Acknowledgements

The authors would like to thank the PPSN VI organisers for their help and for their kind analysis of the results of this algorithm.

We would also like to thank the other organisers of PPSN VI, as well as Daniel BOUILLOT, Jean-Michel GENEVOIS, Thierry PROST, who have accepted to run of the presented Genetic Algorithm on their personal computers, in addition to the computers of the *Projet Fractales*, the computers of the EEAAX laboratory at *École Polytechnique* and especially Jacques TISSERAND, whose personal computer came out with the best result, which was used by the PPSN VI organising committee.

Références

[1] *ALex & AYacc home page:* http://www.bumblebeesoftware.com, Bumblebee Software Ltd.

[2] *GAlib home page:* http://www.mit.edu/people/moriken/doc/galib, MIT.

[3] *EVO-Lab home page :* http://www-rocq.inria.fr/EVO-Lab/ (also containing now EASEA v0.6).

[4] The EVONET *mirror home page:* http://www.evonet.polytechnique.fr.

[5] *EO home page:* http://eodev.sourceforge.net/, Granada University.

[6] A.E. Eiben and Z. Ruttkay, Self-adaptivity for Constraint Satisfaction: Learning Penalty Functions, in *ICEC96*, 1996, IEEE Service Center, pp 258-261.

[7] J.K. Hao, P. Galinier and M. Habib, Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes, *Revue d'Intelligence Artificielle*, 1999, vol 13 No 2, pp 283-324.

[8] E. Lutton, "Genetic Algorithms and Fractals," in *Evolutionary Algorithms in Engineering and Computer Science*, John Wiley & Sons, 1999.

[9] Z. Michalewicz, M. Schoenauer, "Evolutionary Algorithms for Constrained Parameter Optimization Problems," Evolutionary Computation v4, pp1-32, 1997.

[10] M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, H.-P. Schwefel (Eds), "Parallel Problem Solving from Nature," PPSN VI 6th International Conference, Paris, France, September 16-20, 2000. Proceedings. Springer Verlag, Lecture Notes in Computer Science 1917, 2000.

Appendix A: ppsn.ez file

```
/*_________________________________________________________

    ppsn.ez

    This file specifies an evolutionary algorithm trying to
    match papers and reviewers

    Pierre COLLET (Pierre.Collet@inria.fr)
    03/02/00
__________________________________________________________*/

User specific :

#define MAX_PAPERS 8
#define PAP_KEYW 5
#define PAP_AUTH 5
#define PAP_INST 5

#define MAX_REVIEWERS 12
#define REV_KEYW 5
#define REV_WILL 5
#define REV_UNWILL 5

#define REV_PER_PAPER 3

int PAPERS, REVIEWERS;

struct PaperInfo {char Title[50];
       int Keyword[PAP_KEYW];
       char Author[PAP_AUTH][50];
       char Institution[PAP_INST][50];
      } PAPER[MAX_PAPERS];

struct ReviewerInfo {char Name[50];
       int Keyword[REV_KEYW];
       char Institution[50];
       int Willing[REV_WILL];
       int Unwilling[REV_UNWILL];
      } REVIEWER[MAX_REVIEWERS];

FILE *fpPapers, *fpReviewers;

int mystricmp(char *string1, char *string2){
  for (int i=0; string1[i]&&string2[i];i++){
    if (tolower(string1[i])<tolower(string2[i])) return -(i+1);
    if (tolower(string1[i])>tolower(string2[i])) return i+1;
  }
  if (string2[i]) return  -(i+1);
  if (string1[i]) return  i+1;
  return 0;
}

Initialisation function:

  int i,j;

  fpPapers=fopen("Papers.txt","r");
  fpReviewers=fopen("Review.txt","r");

  for (i=0;i<MAX_PAPERS;i++) {
    strcpy(PAPER[i].Title,"");
```

```
    for (j=0;j<PAP_KEYW;j++) PAPER[i].Keyword[j]=999;
    for (j=0;j<PAP_AUTH;j++) strcpy(PAPER[i].Author[j],"");
    for (j=0;j<PAP_INST;j++) strcpy(PAPER[i].Institution[j],"");
  }
  for (i=0;i<MAX_REVIEWERS;i++) {
    strcpy(REVIEWER[i].Name,"");
    for (j=0;j<REV_KEYW;j++) REVIEWER[i].Keyword[j]=999;
    strcpy(REVIEWER[i].Institution,"");
    for (j=0;j<REV_WILL;j++) REVIEWER[i].Willing[j]=999;
    for (j=0;j<REV_UNWILL;j++) REVIEWER[i].Unwilling[j]=999;
  }
  for (i=0;i<MAX_PAPERS;i++) {
    if (fscanf(fpPapers,"%[^\n]",PAPER[i].Title)==EOF) break; getc(fpPapers);
    for (j=0;j<PAP_KEYW;j++) {
        fscanf(fpPapers,"%d",&(PAPER[i].Keyword[j]));
        if(getc(fpPapers)=='\n') break;
        }
    for (j=0;j<PAP_AUTH;j++) {
        fscanf(fpPapers,"%[^,\n]",PAPER[i].Author[j]);
        if(getc(fpPapers)=='\n') break;
        }
    for (j=0;j<PAP_INST;j++) {
        fscanf(fpPapers,"%[^,\n]",PAPER[i].Institution[j]);
        if(getc(fpPapers)=='\n') break;
        }
  }
  PAPERS=i;
  for (i=0;i<MAX_REVIEWERS;i++) {
    if (fscanf(fpReviewers,"%[^\n]",REVIEWER[i].Name)==EOF) break; getc(fpReviewers);
    for (j=0;j<REV_KEYW;j++) {
        fscanf(fpReviewers,"%d",&(REVIEWER[i].Keyword[j]));
        if(getc(fpReviewers)=='\n') break;
        }
    fscanf(fpReviewers,"%[^,\n]",REVIEWER[i].Institution); getc(fpReviewers);
    for (j=0;j<REV_WILL;j++) {
        fscanf(fpReviewers,"%d",&(REVIEWER[i].Willing[j]));
        if(getc(fpReviewers)=='\n') break;
        }
    for (j=0;j<REV_UNWILL;j++) {
        fscanf(fpReviewers,"%d",&(REVIEWER[i].Unwilling[j]));
        if(getc(fpReviewers)=='\n') break;
        }
  }
  REVIEWERS=i;
  fclose(fpPapers); fclose(fpReviewers);

Classes :

Match {int reviewer[3];}

Genome {Match paper[8];}

Standard functions :

Genome::initialiser : // "initializer" is also accepted
  int i,j,k,again=0;
  for (i=0;i<PAPERS;i++)
    for (j=0;j<REV_PER_PAPER;j++)
      do {
        again=0;
        Genome.paper[i].reviewer[j]=(int) random(0,REVIEWERS);
        for (k=0;k<j;k++) if (Genome.paper[i].reviewer[k]==Genome.paper[i].reviewer[j]) again++;
      } while (again) ;
```

```
Genome::crossover : // Must return the number of concerned children
  int i,j,GeneratedChildren=0;
  int pos=(int) random(0,PAPERS-1); // Picks a random site named pos

  if (&child1){
    child1<=parent1;
    for(i=pos;i<PAPERS;i++)
      for(j=0;j<REV_PER_PAPER;j++)
        child1.paper[i].reviewer[j]=parent2.paper[i].reviewer[j];
    GeneratedChildren++;
  }

  if (&child2){
    child2<=parent2;
    for(i=pos;i<PAPERS;i++)
      for(j=0;j<REV_PER_PAPER;j++)
        child2.paper[i].reviewer[j]=parent1.paper[i].reviewer[j];
    GeneratedChildren++;
  }

return GeneratedChildren;

Genome::mutator : // Must return the number of mutations as an int
  int i,j,k,again,nbMut=0;
  for (i=0;i<PAPERS;i++)
    if (tossCoin(PMut)){
      for(j=0;j<REV_PER_PAPER;j++)
        do {
          again=0;
          Genome.paper[i].reviewer[j]=(int) random(0,REVIEWERS);
          for (k=0;k<j;k++) if (Genome.paper[i].reviewer[k]==Genome.paper[i].reviewer[j]) again++;
        } while (again) ;
      nbMut++;
    }

  if (nbMut==0) identicalGenome=true;  // saves evaluation time
  return nbMut;

Genome::evaluator : // Must return the score as a positive double
  int i,j,k,l,eval=100000;
  int papPerRev[MAX_REVIEWERS];

  for (i=0;i<REVIEWERS;papPerRev[i++]=0);
  for (i=0;i<PAPERS;i++)
    for(j=0;j<REV_PER_PAPER;j++){
//-------------------------------------------------------------------------
// if the keywords of the paper match the keywords of the reviewer
      for(k=0;k<REV_KEYW;k++)
        for(l=0;l<PAP_KEYW;l++)
          if (   (REVIEWER[Genome.paper[i].reviewer[j]].Keyword[k]==PAPER[i].Keyword[l])
              &&(REVIEWER[Genome.paper[i].reviewer[j]].Keyword[k]!=999)
              &&(PAPER[i].Keyword[l]!=999)
              ) eval++;

// if paper and reviewer come from the same institution
      for(k=0;k<PAP_INST;k++)
        if (!stricmp(REVIEWER[Genome.paper[i].reviewer[j]].Institution,PAPER[i].Institution[k])) eval-=1000;

// if the reviewer has been willing to review the paper
      for(k=0;k<REV_WILL;k++)
        if (REVIEWER[Genome.paper[i].reviewer[j]].Willing[k]==i) eval++;
```

```
// if the reviewer has been unwilling to review the paper
      for(k=0;k<REV_UNWILL;k++)
        if (REVIEWER[Genome.paper[i].reviewer[j]].Unwilling[k]==i) eval--;

// Reviewers should review have an average of REV_PER_PAPERxPAPERS/REVIEWERS papers to review
      papPerRev[Genome.paper[i].reviewer[j]]++;

//-----------------------------------------------------------------------------
    }

  for (i=0;i<REVIEWERS;i++) eval-=abs(papPerRev[i]-REV_PER_PAPER*PAPERS/REVIEWERS);

  return (double)(eval<0 ? 0 : eval);

Run parameters :
  Population size : 100          // PSize
  Number of generations : 100   // NbGen
  Mutation probability : 0.3    // PMut
  Crossover probability : 1     // PCross
  Genetic engine : SteadyState

End of genome file.
```

Appendix B: ppsn.cpp file

```
//*********************************************
//
//  ppsn.cpp
//
//  C++ file generated by EASEA-GALIB v0.4
//
//*********************************************

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <iostream.h>
#include <fstream.h>
#include <ga/ga.h>

// User Specific
/*___________________________________________________________

    ppsn.ez

    This file specifies an evolutionary algorithm trying to
    match papers and reviewers

    Pierre COLLET (Pierre.Collet@inria.fr)
    03/02/00
___________________________________________________________*/

// User Functions and Declarations

#define MAX_PAPERS 148
#define PAP_KEYW 10
#define PAP_AUTH 25
#define PAP_INST 10

#define MAX_REVIEWERS 179
#define REV_KEYW 20
#define REV_WILL 5
#define REV_UNWILL 5

#define REV_PER_PAPER 3

int PAPERS, REVIEWERS;

struct PaperInfo {int Id;
       char Title[300];
       char Keyword[PAP_KEYW][100];
       char Author[PAP_AUTH][100];
       char Institution[PAP_INST][100];
      } PAPER[MAX_PAPERS];

struct ReviewerInfo {int Id;
       char Name[100];
       char Keyword[REV_KEYW][100];
       char Institution[100];
       int Willing[REV_WILL];
       int Unwilling[REV_UNWILL];
      } REVIEWER[MAX_REVIEWERS];

FILE *fpPapers, *fpReviewers;

char  mytolower(char c) {
```

```
  if  ((c>=65)&&(c<=91)) c-=64;
  return c;
}

int mystricmp(char *string1, char *string2){
  int i;
  for (i=0; string1[i]&&string2[i];i++){
    if (mytolower(string1[i])<mytolower(string2[i])) return -(i+1);
    if (mytolower(string1[i])>mytolower(string2[i])) return i+1;
  }
  if (string2[i]) return  -(i+1);
  if (string1[i]) return  i+1;
  return 0;
}

// Initialisation function

void EASEAInitFunction(){

  int i,j,k,lastPaper,currentPaper;
  char *p,szTemp[1000];

  fpPapers=fopen("papers.txt","r");
  fpReviewers=fopen("review.txt","r");

  for (i=0;i<MAX_PAPERS;i++) {
    strcpy(PAPER[i].Title," ");
    for (j=0;j<PAP_KEYW;j++) strcpy(PAPER[i].Keyword[j]," ");
    for (j=0;j<PAP_AUTH;j++) strcpy(PAPER[i].Author[j]," ");
    for (j=0;j<PAP_INST;j++) strcpy(PAPER[i].Institution[j]," ");
  }
  for (i=0;i<MAX_REVIEWERS;i++) {
    REVIEWER[i].Id=97834;
    strcpy(REVIEWER[i].Name," ");
    for (j=0;j<REV_KEYW;j++) strcpy(REVIEWER[i].Keyword[j]," ");
    strcpy(REVIEWER[i].Institution," ");
    for (j=0;j<REV_WILL;j++) REVIEWER[i].Willing[j]=999;
    for (j=0;j<REV_UNWILL;j++) REVIEWER[i].Unwilling[j]=999;
  }
  lastPaper=i=-1;

  while (i+1<MAX_PAPERS){
    if (fscanf(fpPapers,"| %d | ",&currentPaper)==EOF)break;
    if (currentPaper!=lastPaper) { // We have a new paper
      i++;
      lastPaper=PAPER[i].Id=currentPaper;
      if (fscanf(fpPapers,"%[^|] | ",PAPER[i].Title)==0) fscanf(fpPapers," | ");
      if (fscanf(fpPapers,"%[^|] | ",PAPER[i].Author[0])==0) fscanf(fpPapers," | ");
      if (fscanf(fpPapers,"%[^|] | ",szTemp)==0) fscanf(fpPapers," | ");
      for (p=&(szTemp[strlen(szTemp)-1]);(*p==' ')&&(p!=&szTemp[0]);p--); *(p+1)=0;
      if (p!=&szTemp[0]) {
        for (;(*p!='.')&&(p!=&szTemp[0]);p--); for (p--;(*p!='.')&&(*p!='@')&&(p!=&szTemp[0]);p--);
        strcpy(PAPER[i].Institution[0],p+1);
      }
      for (j=0;j<PAP_KEYW;j++) {
        fscanf(fpPapers," %[^,|]",&(PAPER[i].Keyword[j]));
        for (k=strlen(PAPER[i].Keyword[j])-1;(PAPER[i].Keyword[j][k]==' ')&&(k>0);k--);
                  PAPER[i].Keyword[j][k+1]=0;// remove trailing spaces
        if(getc(fpPapers)=='|') break;
      }
    }
    else {// another line for the same paper
      if (fscanf(fpPapers,"%[^|] | ",szTemp)==0) fscanf(fpPapers," | ");
```

```
      for (k=0;(k<PAP_AUTH);k++)     // looking for an empty space
        if ((PAPER[i].Author[k])[0]==' ') break;
      if (fscanf(fpPapers,"%[^|] | ",PAPER[i].Author[k])==0) fscanf(fpPapers," | ");
      if (fscanf(fpPapers,"%[^|] | ",szTemp)==0) fscanf(fpPapers," | ");
      for (p=&(szTemp[strlen(szTemp)-1]);(*p==' ')&&(p!=&szTemp[0]);p--); *(p+1)=0;
      if (p!=&szTemp[0]) {
        for (;(*p!='.')&&(p!=&szTemp[0]);p--); for (p--;(*p!='.')&&(*p!='@')&&(p!=&szTemp[0]);p--);
        strcpy(PAPER[i].Institution[k],p+1);
      }
      fscanf(fpPapers,"%[^\n]",szTemp);
    }
  getc(fpPapers);
  }
  PAPERS=i+1;
  for (i=0;i<MAX_REVIEWERS;i++) {
    if (fscanf(fpReviewers,"| %d | ",&(REVIEWER[i].Id))==EOF)break;
    if (fscanf(fpReviewers,"%[^|] | ",REVIEWER[i].Name)==0) fscanf(fpReviewers," | ");
    if (fscanf(fpReviewers,"%[^|] | ",szTemp)==0) fscanf(fpReviewers," | ");
    for (p=&(szTemp[strlen(szTemp)-1]);(*p==' ')&&(p!=&szTemp[0]);p--); *(p+1)=0;
    if (p!=&szTemp[0]) {
      for (;(*p!='.')&&(p!=&szTemp[0]);p--);
      for (p--;(*p!='.')&&(*p!='@')&&(p!=&szTemp[0]);p--);
      strcpy(REVIEWER[i].Institution,p+1);
      }
    for (j=0;j<REV_KEYW;j++) {
      fscanf(fpReviewers," %[^,|]",&(REVIEWER[i].Keyword[j]));
      for (k=strlen(REVIEWER[i].Keyword[j])-1;(REVIEWER[i].Keyword[j][k]==' ')&&(k>0);k--);
          REVIEWER[i].Keyword[j][k+1]=0;// remove trailing spaces
      if(getc(fpReviewers)=='|') break;
    }
  getc(fpReviewers);
  }
  REVIEWERS=i;
  fclose(fpPapers); fclose(fpReviewers);

}

// User Classes

class Match {
public:
// Default methods for class Match
  Match(){  // Constructor
  }
  Match(Match &EASEA_Var) {  // Copy constructor
    {for(int EASEA_Ndx=0; EASEA_Ndx<3; EASEA_Ndx++)
       reviewer[EASEA_Ndx]=EASEA_Var.reviewer[EASEA_Ndx];}
    Id=EASEA_Var.Id;
    {for(int EASEA_Ndx=0; EASEA_Ndx<3; EASEA_Ndx++)
       ReviewerId[EASEA_Ndx]=EASEA_Var.ReviewerId[EASEA_Ndx];}
    {for(int EASEA_Ndx=0; EASEA_Ndx<3; EASEA_Ndx++)
       KwMatch[EASEA_Ndx]=EASEA_Var.KwMatch[EASEA_Ndx];}
    KeywContribution=EASEA_Var.KeywContribution;
  }
  ~Match() {  // Destructor
  }
  Match& operator<=(Match &EASEA_Var) {  // Operator<=
    if (&EASEA_Var == this) return *this;
    {for(int EASEA_Ndx=0; EASEA_Ndx<3; EASEA_Ndx++)
       reviewer[EASEA_Ndx] = EASEA_Var.reviewer[EASEA_Ndx];}
    Id = EASEA_Var.Id;
    {for(int EASEA_Ndx=0; EASEA_Ndx<3; EASEA_Ndx++)
       ReviewerId[EASEA_Ndx] = EASEA_Var.ReviewerId[EASEA_Ndx];}
```

```
    {for(int EASEA_Ndx=0; EASEA_Ndx<3; EASEA_Ndx++)
       KwMatch[EASEA_Ndx] = EASEA_Var.KwMatch[EASEA_Ndx];}
    KeywContribution = EASEA_Var.KeywContribution;
  return *this;
  }

  bool operator==(Match &EASEA_Var) const {  // Operator==
    {for(int EASEA_Ndx=0; EASEA_Ndx<3; EASEA_Ndx++)
       if (reviewer[EASEA_Ndx]!=EASEA_Var.reviewer[EASEA_Ndx]) return gaFalse;}
    if (Id!=EASEA_Var.Id) return gaFalse;
    {for(int EASEA_Ndx=0; EASEA_Ndx<3; EASEA_Ndx++)
       if (ReviewerId[EASEA_Ndx]!=EASEA_Var.ReviewerId[EASEA_Ndx]) return gaFalse;}
    {for(int EASEA_Ndx=0; EASEA_Ndx<3; EASEA_Ndx++)
       if (KwMatch[EASEA_Ndx]!=EASEA_Var.KwMatch[EASEA_Ndx]) return gaFalse;}
    if (KeywContribution!=EASEA_Var.KeywContribution) return gaFalse;
  return gaTrue;
  }

  bool operator!=(Match &EASEA_Var) const {return !(*this==EASEA_Var);} // operator!=

  friend ostream& operator<< (ostream& os, const Match& EASEA_Var) { // Output stream insertion operator
    {os << "Array reviewer : ";
     for(int EASEA_Ndx=0; EASEA_Ndx<3; EASEA_Ndx++)
       os << "[" << EASEA_Ndx << "]:" << EASEA_Var.reviewer[EASEA_Ndx] << "\t";}
    os << "\n";
    os <<  "Id:" << EASEA_Var.Id << "\n";
    {os << "Array ReviewerId : ";
     for(int EASEA_Ndx=0; EASEA_Ndx<3; EASEA_Ndx++)
       os << "[" << EASEA_Ndx << "]:" << EASEA_Var.ReviewerId[EASEA_Ndx] << "\t";}
    os << "\n";
    {os << "Array KwMatch : ";
     for(int EASEA_Ndx=0; EASEA_Ndx<3; EASEA_Ndx++)
       os << "[" << EASEA_Ndx << "]:" << EASEA_Var.KwMatch[EASEA_Ndx] << "\t";}
    os << "\n";
    os <<  "KeywContribution:" << EASEA_Var.KeywContribution << "\n";
    return os;
  }

  friend istream& operator>> (istream& is, Match& EASEA_Var) { // Input stream extraction operator
    {for(int EASEA_Ndx=0; EASEA_Ndx<3; EASEA_Ndx++)
       is >> EASEA_Var.reviewer[EASEA_Ndx];}
    is >> EASEA_Var.Id;
    {for(int EASEA_Ndx=0; EASEA_Ndx<3; EASEA_Ndx++)
       is >> EASEA_Var.ReviewerId[EASEA_Ndx];}
    {for(int EASEA_Ndx=0; EASEA_Ndx<3; EASEA_Ndx++)
       is >> EASEA_Var.KwMatch[EASEA_Ndx];}
    is >> EASEA_Var.KeywContribution;
    return is;
  }

// Class members
  int reviewer[3];
  int Id;
  int ReviewerId[3];
  int KwMatch[3];
  int KeywContribution;
};


// User Genome
class ppsnGenome : public GAGenome {
// Default methods for class ppsnGenome
public:
```

```
  GADefineIdentity("ppsnGenome", 251);
  static void Initializer(GAGenome&);
  static int Mutator(GAGenome&, float);
  static float Comparator(const GAGenome&, const GAGenome&);
  static float Evaluator(GAGenome&);
  static int Crossover(const GAGenome&, const GAGenome&, GAGenome*, GAGenome*);
public:
  ppsnGenome::ppsnGenome() :GAGenome(Initializer, Mutator, Comparator){
    evaluator(Evaluator); crossover(Crossover);
  }
  ppsnGenome(const ppsnGenome & orig) {
    copy(orig);
  }
  ~ppsnGenome() {
// Destructing pointers
  }
  ppsnGenome& operator<=(const GAGenome &);
  virtual GAGenome *clone(GAGenome::CloneMethod) const ;
  virtual void copy(const GAGenome & c);
  virtual int equal(const GAGenome& g) const;
  virtual int read(istream & is);
  virtual int write(ostream & os) const ;

// Class members
  int NbPapersPerReviewerContribution;
  int Distribution[10];
  int NbPapersPerReviewer[179];
  Match paper[148];
};

ppsnGenome& ppsnGenome::operator<=(const GAGenome & arg){
  copy(arg);
  return *this;
}

void ppsnGenome::copy(const GAGenome& g) {
  if(&g != this){
    GAGenome::copy(g);    // copy the base class part
    ppsnGenome & genome = (ppsnGenome &)g;
// Memberwise copy
    NbPapersPerReviewerContribution=genome.NbPapersPerReviewerContribution;
    {for(int EASEA_Ndx=0; EASEA_Ndx<10; EASEA_Ndx++)
       Distribution[EASEA_Ndx]=genome.Distribution[EASEA_Ndx];}
    {for(int EASEA_Ndx=0; EASEA_Ndx<179; EASEA_Ndx++)
       NbPapersPerReviewer[EASEA_Ndx]=genome.NbPapersPerReviewer[EASEA_Ndx];}
    {for(int EASEA_Ndx=0; EASEA_Ndx<148; EASEA_Ndx++)
       paper[EASEA_Ndx]=genome.paper[EASEA_Ndx];}
  }
}

GAGenome*ppsnGenome::clone(GAGenome::CloneMethod) const {
  return new ppsnGenome(*this);
}

int ppsnGenome::equal(const GAGenome& g) const {
  ppsnGenome& genome = (ppsnGenome&)g;
// Default diversity test (required by GALIB)
  if (NbPapersPerReviewerContribution!=genome.NbPapersPerReviewerContribution) return 0;
  {for(int EASEA_Ndx=0; EASEA_Ndx<10; EASEA_Ndx++)
     if (Distribution[EASEA_Ndx]!=genome.Distribution[EASEA_Ndx]) return 0;}
  {for(int EASEA_Ndx=0; EASEA_Ndx<179; EASEA_Ndx++)
     if (NbPapersPerReviewer[EASEA_Ndx]!=genome.NbPapersPerReviewer[EASEA_Ndx]) return 0;}
  {for(int EASEA_Ndx=0; EASEA_Ndx<148; EASEA_Ndx++)
```

```
     if (paper[EASEA_Ndx]!=genome.paper[EASEA_Ndx]) return 0;}
  return 1;
}

float ppsnGenome::Comparator(const GAGenome& a, const GAGenome& b) {
  ppsnGenome& sis = (ppsnGenome &)a;
  ppsnGenome& bro = (ppsnGenome &)b;
  int diff = 0;
// Default genome comparator (required by GALIB)
  if (sis.NbPapersPerReviewerContribution!=bro.NbPapersPerReviewerContribution) diff++;
  {for(int EASEA_Ndx=0; EASEA_Ndx<10; EASEA_Ndx++)
     if (sis.Distribution[EASEA_Ndx]!=bro.Distribution[EASEA_Ndx]) diff++;}
  {for(int EASEA_Ndx=0; EASEA_Ndx<179; EASEA_Ndx++)
     if (sis.NbPapersPerReviewer[EASEA_Ndx]!=bro.NbPapersPerReviewer[EASEA_Ndx]) diff++;}
  {for(int EASEA_Ndx=0; EASEA_Ndx<148; EASEA_Ndx++)
     if (sis.paper[EASEA_Ndx]!=bro.paper[EASEA_Ndx]) diff++;}
  return (float)diff;
}

int ppsnGenome::read(istream & is) {
// Default read command (required by GALIB)
  is >> NbPapersPerReviewerContribution;
  {for(int EASEA_Ndx=0; EASEA_Ndx<10; EASEA_Ndx++)
     is >> Distribution[EASEA_Ndx];}
  {for(int EASEA_Ndx=0; EASEA_Ndx<179; EASEA_Ndx++)
     is >> NbPapersPerReviewer[EASEA_Ndx];}
  {for(int EASEA_Ndx=0; EASEA_Ndx<148; EASEA_Ndx++)
     is >> paper[EASEA_Ndx];}
  return is.fail() ? 1 : 0;
}

int ppsnGenome::write(ostream & os) const {
// Default write command (required by GALIB)
  os << "NbPapersPerReviewerContribution:" << NbPapersPerReviewerContribution << "\n";
  {os << "Array Distribution : ";
   for(int EASEA_Ndx=0; EASEA_Ndx<10; EASEA_Ndx++)
     os << "[" << EASEA_Ndx << "]:" << Distribution[EASEA_Ndx] << "\t";}
  os << "\n";
  {os << "Array NbPapersPerReviewer : ";
   for(int EASEA_Ndx=0; EASEA_Ndx<179; EASEA_Ndx++)
     os << "[" << EASEA_Ndx << "]:" << NbPapersPerReviewer[EASEA_Ndx] << "\t";}
  os << "\n";
  {os << "Array paper : ";
   for(int EASEA_Ndx=0; EASEA_Ndx<148; EASEA_Ndx++)
     os << "[" << EASEA_Ndx << "]:" << paper[EASEA_Ndx] << "\t";}
  os << "\n";
  return os.fail() ? 1 : 0;
}


// Standard Functions

void ppsnGenome::Initializer(GAGenome& g) {
  ppsnGenome & genome = (ppsnGenome &)g;
 // "initializer" is also accepted
  int i,j,k,again=0;
  for (i=0;i<PAPERS;i++){
    genome.paper[i].KeywContribution=0;
    genome.paper[i].Id=PAPER[i].Id;
    for (j=0;j<REV_PER_PAPER;j++)
      do {
        again=0;
        genome.paper[i].reviewer[j]=(int) GARandomDouble(0,REVIEWERS);
```

```
        for (k=0;k<j;k++) if (genome.paper[i].reviewer[k]==genome.paper[i].reviewer[j]) again++;
      } while (again) ;
    genome.paper[i].ReviewerId[j]=REVIEWER[genome.paper[i].reviewer[j]].Id;
    }

genome._evaluated=gaFalse;
}

int ppsnGenome::Crossover(const GAGenome& a, const GAGenome& b, GAGenome* c, GAGenome* d) {
  ppsnGenome& mom = (ppsnGenome &)a;
  ppsnGenome& dad = (ppsnGenome &)b;
  ppsnGenome& sis = (ppsnGenome &)*c;
  ppsnGenome& bro = (ppsnGenome &)*d;
  if(&bro) bro._evaluated=gaFalse;
  if(&sis) sis._evaluated=gaFalse;
 // Must return the number of concerned children
  int i,j,GeneratedChildren=0;
  int pos=(int) GARandomDouble(0,PAPERS-1); // Picks a GARandomDouble site named pos

  if (&bro){
    bro<=dad;
    for(i=pos;i<PAPERS;i++)
      for(j=0;j<REV_PER_PAPER;j++)
        bro.paper[i].reviewer[j]=mom.paper[i].reviewer[j];
    GeneratedChildren++;
  }

  if (&sis){
    sis<=mom;
    for(i=pos;i<PAPERS;i++)
      for(j=0;j<REV_PER_PAPER;j++)
        sis.paper[i].reviewer[j]=dad.paper[i].reviewer[j];
    GeneratedChildren++;
  }

return GeneratedChildren;

}

int ppsnGenome::Mutator(GAGenome& g, float pmut) {
  ppsnGenome & genome = (ppsnGenome &)g;
  genome._evaluated=gaFalse;
 // Must return the number of mutations as an int
  int i,j,k,again,nbMut=0;
  for (i=0;i<PAPERS;i++)
    if (GAFlipCoin(pmut)){
      for(j=0;j<REV_PER_PAPER;j++)
        if (GAFlipCoin(1-pmut)) do {
          again=0;
          genome.paper[i].reviewer[j]=(int) GARandomDouble(0,REVIEWERS);
          for (k=0;k<j;k++) if (genome.paper[i].reviewer[k]==genome.paper[i].reviewer[j]) again++;
        } while (again) ;
      nbMut++;
    }

  if (nbMut==0) genome._evaluated=gaTrue;  // saves evaluation time
  return nbMut;

}

float ppsnGenome::Evaluator(GAGenome & c) {
  ppsnGenome & genome = (ppsnGenome &)c;
 // Must return (float) the score as a positive double
```

```
  int contrib,i,j,k,l,matches,NbPapKwds,eval=100000;

  for (i=0;i<REVIEWERS;genome.NbPapersPerReviewer[i++]=0);
  for (i=0;i<10;genome.Distribution[i++]=0);

  for (i=0;i<PAPERS;i++) {
    contrib=0;
    genome.paper[i].Id=PAPER[i].Id;
    for(j=0;j<REV_PER_PAPER;j++){
      genome.paper[i].ReviewerId[j]=REVIEWER[genome.paper[i].reviewer[j]].Id;

//----------------------------------------------------------------------------
// if the keywords of the paper match the keywords of the reviewer
      matches=0;
      for (NbPapKwds=0;(NbPapKwds<PAP_KEYW)&&((PAPER[i].Keyword[NbPapKwds])[0]!=' ');NbPapKwds++);
          // Determines the nb of keywords for the current paper.
      if ((REVIEWER[genome.paper[i].reviewer[j]].Keyword[0])[0]==' '){
        contrib+=5; // If the reviewer has no keyword list, we suppose this is equivalent to an average match (5/10)
        genome.paper[i].KwMatch[j]=-1; // We signal the fact that the reviewer has no keywords by indicating "-1"
      }
      else {
        for(k=0;k<REV_KEYW;k++){
          if ((REVIEWER[genome.paper[i].reviewer[j]].Keyword[k])[0]==' ') break;
          for(l=0;l<PAP_KEYW;l++){
            if ((PAPER[i].Keyword[l])[0]==' ') break;
            if (!mystricmp(REVIEWER[genome.paper[i].reviewer[j]].Keyword[k],PAPER[i].Keyword[l]) ) matches++;
          }
        }
        if (matches==0) contrib-=10; // we discourage ill matching
        else contrib+=(matches*10)/NbPapKwds; // With this calculation,  a reviewer matching all the paper keywords gets 10/10
        genome.paper[i].KwMatch[j]=matches;
      }  // and a reviewer matching half of the keywords gets only 5/10

// if paper and reviewer come from the same institution
      for(k=0;k<PAP_INST;k++)
        if (!mystricmp(REVIEWER[genome.paper[i].reviewer[j]].Institution,PAPER[i].Institution[k])) contrib-=1000;

// if the reviewer has been willing to review the paper
//      for(k=0;k<REV_WILL;k++)
//        if (REVIEWER[genome.paper[i].reviewer[j]].Willing[k]==i) contrib++;

// if the reviewer has been unwilling to review the paper
//      for(k=0;k<REV_UNWILL;k++)
//        if (REVIEWER[genome.paper[i].reviewer[j]].Unwilling[k]==i) contrib--;

      genome.NbPapersPerReviewer[genome.paper[i].reviewer[j]]++;

//----------------------------------------------------------------------------
    }
  eval +=contrib;
  genome.paper[i].KeywContribution=contrib;
  }

  contrib=0;
// Reviewers should have an average of REV_PER_PAPERxPAPERS/REVIEWERS papers to review
  for (i=0;i<REVIEWERS;i++){
    genome.Distribution[genome.NbPapersPerReviewer[i]]++;
    if (REV_PER_PAPER*PAPERS/REVIEWERS<1) {
      if (genome.NbPapersPerReviewer[i]>1) contrib -= 10*(genome.NbPapersPerReviewer[i] -1);
    }
    else{
      j=genome.NbPapersPerReviewer[i]-REV_PER_PAPER*PAPERS/REVIEWERS;
      if (j>0) contrib -= (int)pow(5,j);   // Marc's suggestion
```

```
      if (j<0) contrib +=5*j;
    }
  }
  eval += contrib;
  genome.NbPapersPerReviewerContribution=contrib;

  return (float) (float) (double)(eval<0 ? 0 : eval);


}



int main(int argc, char *argv[]){
  int i;

  GARandomSeed(0);

// Checks whether we've been given a seed to use (for testing purposes).
  for(int ii=1; ii<argc; ii++) {
    if(strcmp(argv[ii++],"seed") == 0) {
      GARandomSeed((unsigned int)atoi(argv[ii]));
    }
  }

// Parse the command line for arguments.

  for(i=1; i<argc; i++){
    if(strcmp("seed", argv[i]) == 0){
      if(++i < argc) continue;
      continue;
    }
    else {
      cerr << argv[0] << ":  unrecognized argument: " << argv[i] << "\n\n";
      cerr << "valid arguments are standard GAlib arguments.\n";
      exit(1);
    }
  }


EASEAInitFunction();


  ppsnGenome genome;
  GASteadyStateGA ga(genome);
  ga.populationSize(40); // how many individuals in the population
  ga.nGenerations(10000);    // number of generations to evolve
  ga.pMutation((float)0.200000);    // likelihood of mutating new offspring
  ga.pCrossover((float)1.000000);   // likelihood of crossing over parents

  genome.initialize();
  cout << "Score of a generation 0 genome: " << genome.Evaluator(genome) << "\n";
  cout << "Contents of the genome:\n" << genome << endl;

  ga.evolve();
  cout << "\nBest genome score : " << (ga.statistics().bestIndividual()).evaluate() << endl;
  cout << "Contents of the genome :\n" << ga.statistics().bestIndividual() << "\n";

  exit(0);
  return 0;
}

// If your compiler does not do automatic instantiation (e.g. g++ 2.6.8),
```

```
// then define the NO_AUTO_INST directive.  This will force the instantiation
// of the template classes that we use.  For some compilers (e.g. metrowerks)
// this must come after any specializations or you'll get 'multiply-defined'
// errors when you compile.
#ifdef NO_AUTO_INST
#include "GAList.cpp"
#include "GAListGe.cpp"
#if defined(__GNUG__)
template class GAList<int>;
template class GAListGenome<int>;
#else
GAList<int>;
GAListGenome<int>;
#endif
#endif


// That's all folks !
```

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399