Crossover operators for permutations equivalence between position and order-based crossover

Concepción Vidal[*], Gilberto Pérez, Felicidad Aguado and José María Molinelli

Department of Computer Science, University of A Coruña, A Coruña, Spain

Abstract

In the context of genetic algorithms, the use of permutation-based representations has worked out more conveniently than the classical binary encoding for some scheduling and combinatorial problems such as the Travelling Salesman Problem. In Aguado *et al.* (2007, Certified genetic algorithms: crossover operators for permutations, Vol. 4739 of *Lecture notes in Computer Science*, pp. 282–289), we implemented in Coq several genetic operators proposed in Davis (1991, *Handbook of Genetic Algorithms*) and Syswerda (1985, Schedule optimization using Genetic algorithms, *Handbook of Genetic Algorithms*, pp. 332–349) to deal with the chromosomes of problems where the individuals are encoded as permutations; in these cases we specifically implemented the so-called operators `pbx` and `obx`. In Aguado *et al.* (2007, Generalización de los cruces basados en el orden y en la posición. Una implementación verificada, In *Proceedings of CLEI 2007*), we define with an axiomatic implementation two new operators `gen_pbx` and `gen_obx` which generalize the previous ones. In this article, we formally specify the relation between these operators when restricted to the case of permutations without repetition. We also propose a new crossover operator which actually combines the genetic material from both parents in each child. Experimental results confirm that the use of one or another crossover makes no significant difference.

Keywords: Genetic algorithms, Coq, functional programming.

1 Introduction

The term evolutionary computation refers to some heuristic techniques more or less based on Darwin's idea that the genetic component of better adapted individuals is more likely to survive in future generations. It is included in a type of computing which makes use of concepts, principles and mechanisms underlying biological systems. Among evolutive techniques we have genetic algorithms (GAs) [6] which have been successfully applied in several fields: optimization, pattern recognition, evolving of neural networks, etc. Although there are many GAs in the literature, they all show some process of codification, evaluation and reproduction. A population of abstract representations called chromosomes evolves towards better solutions. The algorithm measures the worth of any individual (evaluation) with a fitness function. Finally some chromosomes are modified with crossover and eventually mutation to obtain a new generation of the population.

Although binary representation is the most common way of encoding, for some problems of combinatorial optimization, such as the Travelling Salesman Problem (TSP), the use of

*E-mail: concepcion.vidalm@udc.es

Vol. 19 No. 2, © The Author 2010. Published by Oxford University Press. All rights reserved.
For Permissions, please email: journals.permissions@oup.com
doi:10.1093/jigpal/jzq038 Advance Access published 12 July 2010

permutations have turned out to be a better choice considering the nature of the problem. However, in this case the classical crossover, which chooses a cutting point and then interchanges head and tail from both parents, can produce illegal offspring. To deal with this problem, several crossover operators (PBX, OBX) specifically adapted to this special representation are introduced in [3, 7].

In some previous papers [1, 2], we have formally implemented with the proof management system Coq [5] and the functional programming languages Ocaml and Haskell some crossover operators. We were chasing two goals: on the one hand, we thought it would be useful having a library with (certified) genetic operators that could be used in concrete problems instead of designing one operator for each specific problem (as it usually happens) and, on the other hand the use of formal methods and proof theory allows us to certify the correctness of our implementations and also prove some important properties satisfied by the proposed genetic operators. Notice that with Coq we can define functions and predicates, formulate theorems and software specifications, develop interactive proofs for these theorems and certify these proofs. The Coq system allows to automatically extract functional programs from the algorithmic content of the proofs.

Even though the crossover operators of [4, 7] were initially proposed for the crossover of permutations with non-repeated elements, in some optimization problems, such as scheduling, we need to consider permutations of possibly repeated elements. In this case, the general procedure of the PBX and OBX crossovers can be carried out in several ways. This is mainly due to the fact that, in the elimination process, more than one selection can be done when the elements to leave out appear more than once in the lists. In [2], we decided to generalize the operators defined in [1] using an axiomatic implementation based on the properties that characterized them. Moreover, these new functions verify the same specifications as those managed in [1] which are simply a particular case of those implemented later. Of course, when we only consider permutations without repetition, we recover the operators implemented in [1].

In this work, we prove that for this type of permutations, the two crossover operators based on order and position are closely related. Specifically, for each pair of chromosomes p and q, the crossover `gen_pbx` with an ℓ pattern of p and q is the crossover `gen_obx` of p and q with another pattern that depends on ℓ and, on p and q. Besides, since the original idea of the classical crossover operator was to combine genetic material from both parents in each child, we propose a new crossover operator denoted by `gen_POBX` which in fact makes use, in the second child, of the genetic material from the first parent that was not used in the first child. We verify by experimental proofs that there are no significant differences in the use of one or other type of crossover in the case of permutations without repetition, but we think that our proposal is more faithful to the biologically inspired original crossover.

2 Background

Let D be a finite, non-empty and non-unary set. The chromosomes will be represented as finite chains (lists) of D elements, for which the type `list D` of Coq will be used. To characterize the permutations in `list D`, the predicate `permutation` found in the Coq standard libraries will be used (here we will write $p \approx q$ for `permutation p q`). Chains of bits (`list bit` in Coq) will be used to represent the patterns that determine each crossover operator, `bit` being a set of two elements (here denoted as 0 and 1). Similarly, we recursively define the function `ext` that, given a p list and an ℓ crossover pattern, recovers the list obtained when extracting from p the alleles corresponding to the positions where the ℓ pattern has *ones*.

```
Fixpoint ext (l: (list bit)) (p: (list D)) {struct l}: (list D):=
    match l with
    | nil => nil
    | a :: t => match a with
                |zero => match p with
                         |nil => nil
                         | _ :: lt => ext t lt
                         end
                |one  => match p with
                         |nil => nil
                         | h :: lt => h :: ext t lt
                         end
                end
    end.
```

In addition to the properties of this function obtained in [1], we now prove others that will be used in the proof of the results in later sections; among these we highlight the following.

Lemma 2.1.
Let $p \in \mathit{list}\,D$ and $\ell \in \mathit{list\,bit}$. If $\mathit{length}(p) \leq \mathit{length}(\ell)$, then $(\mathtt{ext}\,\ell\,p) {+}{+} (\mathtt{ext}\,\overline{\ell}\,p) \approx p$.

One of the most important concepts was that of 'inclusion' of lists. We will use the implementation ($\preceq$) of [2], based on the definitions of permutation and concatenation, included in the Coq standard libraries.

Definition 2.2.
Let $s,t \in \mathit{list}\,D$. We say that t *is included in* s, and it will be represented by $t \preceq s$, if there exists $r \in \mathit{list}\,D$, such that, $t{+}{+}r \approx s$.

```
Definition inclusion (t s: (list D)) := {r:(list D) | (permutation (t ++ r) s)}.
```

A function that obtains a difference between two lists s and t is also needed. The definition of this function (`diff`) is axiomatically made from the also axiomatic definition of `pos` [2]. Given two lists s and t, the aim was for $\mathtt{pos}\,s\,t$ to be a list of *zeros* and *ones* having the same length as s (Axiom 1) and such that, if $t \preceq s$, we wanted $\mathtt{pos}\,s\,t$ to have *ones* at s places where the t elements are (Axiom 2).

Definition 2.3.
If $s,t \in \mathit{list}\,D$, then $(\mathtt{pos}\,s\,t) \in \mathit{list\,bit}$ verifies:

Axiom 1
$\mathit{length}(\mathtt{pos}\,s\,t) = \mathit{length}(s)$

Axiom 2
if $t \preceq s$, then $(\mathtt{ext}\,(\mathtt{pos}\,s\,t)\,s) \approx t$

```
 Variable pos: (list D) -> (list D) -> (list bit).
 Axiom axpos1: forall (s t: (list D)), length (pos s t) =  length s.
 Axiom axpos2: forall (s t: (list D)), inclusion t s -> permutation (ext
(pos s t) s) t.
```

It was proven that, if $t \preceq s$, the number of *ones* of $\mathtt{pos}\,s\,t$ coincides with the length of t. Now we implement the function `diff`.

Definition 2.4.
Let $s,t \in \mathit{list}\,D$, we define $\mathtt{diff}\,s\,t = \mathtt{ext}\,\overline{(\mathtt{pos}\,s\,t)}\,s$.

```
Definition diff (s t: (list D)) := ext (complement (pos s t)) s.
```

The main property of the previous function is shown in the following result.

THEOREM 2.5.
Let $s, t \in \mathit{list}\, D$ such that $t \preceq s$, then

$$t{+}{+}\mathtt{diff}\, s\, t \approx s.$$

```
Lemma perm_diff: forall (s t: (list D)),
inclusion t s -> permutation s (t ++ (diff s t)).
```

3 Position and order-based crossover

In [1], we extended the original definitions (for permutations without repetition) of [7] for crossover operators based on position and order to the case of any permutation. Later, in [2], we generalized the previous definitions. In the implementation proposed, we used the function subs that, given two lists p and q and an ℓ crossover pattern, recovers the list obtained, substituting in the list p the elements at locations marked with 1 in ℓ for the elements of q from left to right. The implementation in Coq of the function subs is the following:

```
Fixpoint subs (l: (list bit))(p q: (list D)){struct p}: (list D):=
    match p with
    | nil => nil
    | hp :: tp => match q with
                  | nil => p
                  | hq :: tq  => match l with
                                 | nil => p
                                 | hl :: tl => match hl with
                                               |zero => hp :: (subs tl tp q)
                                               |one  => hq :: (subs tl tp tq)
                                               end
                                 end
                  end
    end.
```

Given two chromosomes p and q, with $p \approx q$ and crossover pattern ℓ, in the position-based crossover (gen_pbx), the idea is to respect the positions of the p alleles indicated with the ones of ℓ and to complete the rest of the chromosome using the missing alleles, keeping the relative order they show in q. Then, in the order-based crossover (gen_obx), the aim is to keep the relative order of the alleles indicated with the ones of the ℓ pattern in q and to complete the chromosome using the rest of the alleles at the position occupied in p. Figure 1 shows with an example how each of the operators for permutations without repetition works.

DEFINITION 3.1.
Let $p, q \in \mathit{list}\, D$ and $\ell \in \mathit{list}\, \mathit{bit}$, we define

$$\mathtt{gen\_pbx}\, \ell\, p\, q = \mathtt{subs}\, \overline{\ell}\, p\, (\mathtt{diff}\, q\, (\mathtt{ext}\, \ell\, p))$$
$$\mathtt{gen\_obx}\, \ell\, p\, q = \mathtt{subs}\, (\mathtt{pos}\, p\, (\mathtt{ext}\, \ell\, q))\, p\, (\mathtt{ext}\, \ell\, q).$$

Now we summarize in a theorem the results proved in [2] used to verify that the proposed implementations satisfy the required specifications.

THEOREM 3.2.
Let $p, q \in \mathit{list}\, D$, with $p \approx q$, and let $\ell \in \mathit{list}\, \mathit{bit}$. Then, we have

(1) $\texttt{ext}\,\ell\,(\texttt{gen\_pbx}\,\ell\,p\,q)=\texttt{ext}\,\ell\,p$
(2) $\texttt{ext}\,\overline{\ell}\,(\texttt{gen\_pbx}\,\ell\,p\,q)=\texttt{diff}\,q\,(\texttt{ext}\,\ell\,p)$, if $length(\ell)=length(p)$
(3) $\texttt{ext}\,(\texttt{pos}\,p\,(\texttt{ext}\,\ell\,q))(\texttt{gen\_obx}\,\ell\,p\,q)=\texttt{ext}\,\ell\,q$
(4) $\texttt{ext}\,(\texttt{pos}\,p\,(\texttt{ext}\,\ell\,q))(\texttt{gen\_obx}\,\ell\,p\,q)=\texttt{diff}\,p\,(\texttt{ext}\,\ell\,q)$
(5) $\texttt{gen\_obx}\,\ell\,p\,q\approx p$ and $\texttt{gen\_pbx}\,\ell\,p\,q\approx p$

4 Equivalence between gen_pbx and gen_obx for permutations without repetition

In the paper [2], we showed that there is only one possible implementation for the functions `gen_pbx` and `gen_obx` when only permutations without repetition are considered; as we can expect, this implementation coincides with that carried out in [1].
Now we will see that in this case of permutations without repetition, given two chromosomes, the crossover based on an order of a particular pattern corresponds with the crossover based on the position having another pattern (that depends on the two chromosomes and on the initial pattern).

We will denote as *listnr* D the set of lists of D elements where there are no repeated elements. We implement in Coq this type of lists as follows:

```
Inductive listnr: list D -> Prop :=
  listnr_nil: listnr nil
  | listnr_cons: forall a l, ~ In a l -> listnr l -> listnr (a :: l).
```

The following lemmas will be used in the proof of Theorem 4.6.

Lemma 4.1.
Let $q\in listnr\,D$ and $\ell\in list\,bit$ such that $length(q)=length(\ell)$. Then, we have:

(1) $\texttt{pos}\_\,q\,(\texttt{ext}\,\ell\,q)=\ell$.
(2) $\texttt{ext}\,\overline{\ell}\,q=\texttt{diff}\,q\,(\texttt{ext}\,\ell\,q)$.

Lemma 4.2.
Let $s,\ t\in list\,D$ such that $t\in listnr\,D$ and $s\preceq t$, then

$$\texttt{pos}\,t\,s=\overline{\texttt{pos}\,t\,(\texttt{diff}\,t\,s)}.$$

Lemma 4.3.
Let $p,x,y\in list\,D$ such that $p\in listnr\,D$, $x\approx y$ and $x\preceq p$, then $\texttt{pos}\,p\,x=\texttt{pos}\,p\,y$.

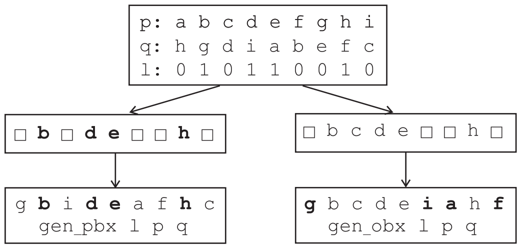

Fig. 1. The crossover operators based on position and order for permutations without repetition.

Lemma 4.4.
Let $p, q, r \in \mathit{list}\, D$. If $p \approx q$ and $r \preceq p$, then

$$(\texttt{diff}\, p\, r) \approx (\texttt{diff}\, q\, r).$$

Lemma 4.5.
Let $p, q \in \mathit{listnr}\, D$ and $\ell \in \mathit{list\, bit}$ such that $p \approx q$ and $\mathit{length}(q) = \mathit{length}(\ell)$. Then,

(1) $\texttt{pos}\, p\, (\texttt{ext}\, \ell\, q) = \overline{\texttt{pos}\, p\, (\texttt{ext}\, \overline{\ell}\, q)}$.
(2) $\texttt{pos}\, q\, (\texttt{ext}\, (\texttt{pos}\, p\, (\texttt{ext}\, \ell\, q))\, p) = \ell$.

(1) Applying Lemmas 4.1, 4.4, 4.3 and 4.2, we obtain:
$\texttt{pos}\, p\, (\texttt{ext}\, \ell\, q) = \texttt{pos}\, p\, \texttt{diff}\, q\, (\texttt{ext}\, \overline{\ell}\, q)) = \texttt{pos}\, p\, (\texttt{diff}\, p\, (\texttt{ext}\, \overline{\ell}\, q)) = \overline{\texttt{pos}\, p\, (\texttt{ext}\, \overline{\ell}\, q)}$.
(2) Applying the previous section, the definition of `diff` and Lemmas 4.4, 4.3 and 4.1, we obtain:

$$\begin{array}{lcl} \texttt{pos}\, q\, (\texttt{ext}\, (\texttt{pos}\, p\, (\texttt{ext}\, \ell\, q))\, p) & = & \texttt{pos}\, q\, (\texttt{ext}\, (\overline{\texttt{pos}\, p\, (\texttt{ext}\, \overline{\ell}\, q)})\, p) \\ \texttt{pos}\, q\, (\texttt{diff}\, p\, (\texttt{ext}\, \overline{\ell}\, q)) & = & \texttt{pos}\, q\, (\texttt{diff}\, q\, (\texttt{ext}\, \overline{\ell}\, q)) \\ \texttt{pos}\, q\, (\texttt{ext}\, \ell\, q) & = & \ell. \end{array}$$

The following theorem proves that, if we fix a pair of chromosomes, the map which changes the pattern ℓ of one crossover (`gen_obx`) to the pattern of the other crossover (`gen_pbx`) is bijective.

For each $n \in \mathbb{N}$, we will denote

$$(\texttt{list bit})_n = \{\ell \in (\texttt{list bit});\, \mathit{length}(\ell) = n\}.$$

Theorem 4.6.
Let $p, q \in \mathit{listnr}\, D$ and $\ell \in \mathit{list\, bit}$ such that $p \approx q$ and $\mathit{length}(q) = \mathit{length}(\ell)$. The map:

$$f_{p,q} : (\texttt{list bit})_n \to (\texttt{list bit})_n$$

defined by $f_{p,q}(\ell) = \texttt{pos}\, p\, (\texttt{ext}\, \overline{\ell}\, q)$ is bijective with inverse

$$f_{p,q}^{-1}(\ell) = \texttt{pos}\, q\, (\texttt{ext}\, \overline{\ell}\, p) = f_{q,p}(\ell).$$

Moreover, for each pattern ℓ, we have that:

$$\begin{array}{c} \texttt{gen\_obx}\, \ell\, p\, q = \texttt{gen\_pbx}\, f_{p,q}(\ell)\, p\, q \\ \texttt{gen\_pbx}\, \ell\, p\, q = \texttt{gen\_obx}\, f_{p,q}^{-1}(\ell)\, p\, q \end{array}$$

First, we can apply the Lemma 4.5 and state that

$$\texttt{pos}\, q\, (\texttt{ext}\, f_{p,q}(\ell)\, p) = \overline{\ell}.$$

Furthermore:

$$\begin{array}{ll}
\texttt{gen\_obx}\,\ell\,p\,q & = \\
\texttt{subs}\,(\texttt{pos}\,p\,(\texttt{ext}\,\ell\,q))\,p\,(\texttt{ext}\,\ell\,q) & = \\
\texttt{subs}\,\overline{(\texttt{pos}\,p\,(\texttt{ext}\,\overline{\ell}\,q))}\,p\,(\texttt{ext}\,\ell\,q) & = \\
\texttt{subs}\,\overline{f_{p,q}(\ell)}\,p\,(\texttt{ext}\,\overline{\overline{\ell}}\,q) & = \\
\texttt{subs}\,\overline{f_{p,q}(\ell)}\,p\,(\texttt{ext}\,\overline{(\texttt{pos}\,q\,(\texttt{ext}\,f_{p,q}(\ell)\,p))})\,q & = \\
\texttt{subs}\,\overline{f_{p,q}(\ell)}\,p\,(\texttt{diff}\,q\,(\texttt{ext}\,f_{p,q}(\ell)\,p)) & = \\
\texttt{gen\_pbx}\,f_{p,q}(\ell)\,p\,q &
\end{array}$$

In order to show that $f_{p,q}$ is bijective, note that:

$$\begin{array}{rcl}
f_{p,q}(f_{p,q}^{-1}(\ell)) & = & f_{p,q}(\texttt{pos}\,q\,(\texttt{ext}\,\overline{\ell}\,p)) \\
\texttt{pos}\,p\,(\texttt{ext}\,\overline{(\texttt{pos}\,q\,(\texttt{ext}\,\overline{\ell}\,p))})\,q & = & \texttt{pos}\,p\,(\texttt{ext}(\texttt{pos}\,q\,(\texttt{ext}\,\ell\,p)))\,q \\
 & = & \ell.
\end{array}$$

Reciprocally, we prove that $f_{p,q}^{-1}(f_{p,q}(\ell)) = \ell$. Finally, the only thing left is taking into account that:

$$\texttt{gen\_pbx}\,\ell\,p\,q = \texttt{gen\_pbx}\,(f_{p,q}(f_{p,q}^{-1}(\ell))\,p\,q = \texttt{gen\_obx}\,f_{p,q}^{-1}(\ell)\,p\,q$$

5 Position and order crossover revisited

In GA, crossover involves combining elements from two parent chromosomes into (usually) two child chromosomes. The child chromosomes for the position and the order crossover operators proposed in [4] and generalized in [2] are:

$$\texttt{gen\_PBX}\,\ell\,p\,q = (\texttt{gen\_pbx}\,\ell\,p\,q, \texttt{gen\_pbx}\,\ell\,q\,p)$$

$$\texttt{gen\_OBX}\,\ell\,p\,q = (\texttt{gen\_obx}\,\ell\,p\,q, \texttt{gen\_obx}\,\ell\,q\,p)$$

where p and q are any pair of parent chromosomes and ℓ is any binary pattern.

The original idea of the biologically inspired crossover operator was to combine the genetic material from both parents into the two children. So, to define the second child, we should consider the genetic material from both parents not used in the first one (child). If we reanalyse the definition of `gen_PBX`, we notice that the first child $\texttt{gen\_pbx}\,\ell\,p\,q$ has the alleles of $\texttt{ext}\,\ell\,p$ in the positions where the pattern ℓ has ones. The rest of the alleles $(\texttt{diff}\,p\,(\texttt{ext}\,\ell\,p))$ appear in this first child following the order they have in q. If we want the second child to inherit the genetic material not used from both p and q, we expect the alleles of $\texttt{ext}\,\ell\,p$ to maintain the positions and order they have in q. The rest of the alleles must appear in the order they have in p. Because of this, we propose a new crossover operator `gen_POBX` defined by:

$$\texttt{gen\_POBX}\,\ell\,p\,q = (\texttt{gen\_pbx}\,\ell\,p\,q, \texttt{gen\_obx}\,\overline{\ell}\,q\,p).$$

You can see an example in Figure 2 showing how it works.

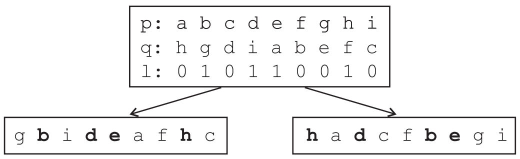
p: a b c d e f g h i
q: h g d i a b e f c
l: 0 1 0 1 1 0 0 1 0
g **b** i **d** **e** a f **h** c
h a **d** c f **b** **e** g i

FIG. 2. The offspring obtained with the new crossover operator.

If we denote by $\ell^* = \mathtt{pos}\, q\, (\mathtt{ext}\, \ell\, p)$, we have to verify

(1) $\mathtt{ext}\, \ell\, p = \mathtt{ext}\, \ell\, (\mathtt{gen\_pbx}\, \ell\, p\, q)$ and $\mathtt{ext}\, \bar{\ell^*}\, q = \mathtt{ext}\, \bar{\ell}\, (\mathtt{gen\_pbx}\, \ell\, p\, q)$
(2) $\mathtt{ext}\, \ell^*\, q = \mathtt{ext}\, \ell^*\, (\mathtt{gen\_obx}\, \bar{\ell}\, q\, p)$ and $\mathtt{ext}\, \bar{\ell}\, p = \mathtt{ext}\, \bar{\ell^*}\, (\mathtt{gen\_obx}\, \bar{\ell}\, q\, p)$

The first property follows from Theorem 3.2 and the fact that

$$\mathtt{diff}\, q\, (\mathtt{ext}\, \ell\, p) = \mathtt{ext}\, \overline{\mathtt{pos}\, q\, (\mathtt{ext}\, \ell\, p)}\, q.$$

With regard to the second child, first note (Theorem 4.6) that:

$$\mathtt{gen\_obx}\, \bar{\ell}\, q\, p = \mathtt{gen\_pbx}\, f_{q,p}(\bar{\ell})\, q\, p = \mathtt{gen\_pbx}\, \ell^*\, q\, p.$$

Now, and again applying Theorem 3.2, it holds that:

$$\mathtt{ext}\, \ell^*\, (\mathtt{gen\_obx}\, \bar{\ell}\, q\, p) = \mathtt{ext}\, \ell^*\, (\mathtt{gen\_pbx}\, \bar{\ell^*}\, q\, p) = \mathtt{ext}\, \ell^*\, q.$$

Finally, and taking into account Lemma 4.5:

$$\begin{array}{lcl} \mathtt{ext}\, \bar{\ell^*}\, (\mathtt{gen\_obx}\, \bar{\ell}\, q\, p) & = & \mathtt{ext}\, \bar{\ell^*}\, (\mathtt{gen\_pbx}\, \ell^*\, q\, p) \\ \mathtt{diff}\, p\, (\mathtt{ext}\, \ell^*\, q) & = & \mathtt{ext}\, \overline{\mathtt{pos}\, p\, (\mathtt{ext}\, \ell^*\, q)}\, p \\ & = & \mathtt{ext}\, \bar{\ell}\, p. \end{array}$$

6 Experimental results

For experimental proofs, we have chosen the TSP with 17 cities and the code extracted from Coq for the crossover process. We used a GA with crossover probability 0.6, mutation probability 0.02 and with tournament selection. We can see the best fitness and the fitness average for two trials of different population sizes (300 and 500) for 20 generations in Figure 3 and for 25 generations in Figure 4. These results corroborate that the use of one or another crossover makes no relevant difference.

7 Conclusions

In this article, we have formally specified the similarity between the two crossover operators of [7] when permutations without repetition are considered. We have defined a bijective map which allows us to express each of the previous operators in terms of the other one. Moreover, we have proposed a new crossover operator (gen_POBX) more faithful to the original idea of the classical crossover. Experimental results confirmed that there are no significant differences in the behaviour of any of the crossover operators. In order to carry

Population size 300		
Crossover	Best Fitness	Fitness Average
PBX	-2085	-2288.673
OBX	-2085	-2276.866
POBX	-2085	-2274.313
Population size 500		
Crossover	Best Fitness	Fitness Average
PBX	-2085	-2259.3122
OBX	-2085	-2262.996
POBX	-2085	-2251.36

FIG. 3. Experimental results for the TSP with 17 cities running 20 generations.

Population size 300		
Crossover	Best Fitness	Fitness Average
PBX	-2085	-2273.106
OBX	-2085	-2272.066
POBX	-2090	-2285.860
Population size 500		
Crossover	Best Fitness	Fitness Average
PBX	-2085	-2222.512
OBX	-2085	-2247.0086
POBX	-2085	-2229.644

FIG. 4. Experimental results for the TSP with 17 cities running 25 generations.

on with our certified genetic library, we are also interested in the formal implementation of more genetic operators specifically designed for other type of representations (appearing in scheduling, genetic programming, design of artificial neural networks architectures, etc.).

Acknowledgement

This work was supported by Xunta de Galicia-Spain (INCITE08-PXIB105159PR).

References

[1] F. Aguado, J. L. Doncel, J. M. Molinelli, G. Pérez, C. Vidal, and A. Vieites. In *Proceedings of 11th International Conference on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, February 12–16, 2007.* In Vol. 4739 of *Lecture Notes in Computer Science*, pp. 282–289. Springer, Berlin/Heidelberg, 2007.

[2] F. Aguado, J. M. Molinelli, G. Pérez, C. Vidal, and A. Vieites. Generalización de los cruces basados en el orden y en la posición. Una implementación verificada, In *Proceedings of CLEI 2007*, University of Costa Rica, (ISBN: 978-9968-9678-9-1), 2007.

[3] L. Davis. Applying adaptive algorithms to epistatic domains. In *IJCAI85, Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 162–164. Morgan Kaufmann Publishers Inc., 1985.

[4] L. Davis. *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[5] INRIA, *The Coq proof assistant*. Available at: http://coq.inria.fr.

[6] M. Mitchell. *An Introduction to Genetic Algorithms*, MIT Press, 1996.

[7] G. Syswerda. Schedule optimization using genetic algorithms. In *Handbook of Genetic Algorithms*, L. Davis ed., pp. 332–349, Van Nostrand Reinhold, 1991.

Received 26 February 2010