Appears in "Evolutionary Computing: AISB Workshop", Ed: T.C. Fogarty, Springer-Verlag LNCS 865, pp1-16, 1994.

Formal Memetic Algorithms

Nicholas J. Radcliffe and Patrick D. Surry

Edinburgh Parallel Computing Centre
King's Buildings, University of Edinburgh
Scotland, EH9 3JZ

Abstract. A formal, representation-independent form of a memetic algorithm—a genetic algorithm incorporating local search—is introduced. A generalised form of N-point crossover is defined together with representation-independent patching and hill-climbing operators. The resulting formal algorithm is then constructed and tested empirically on the travelling sales-rep problem. Whereas the genetic algorithms tested were unable to make good progress on the problems studied, the memetic algorithms performed very well.

1 Motivation

The rôle of local search in the context of genetic algorithms and the wider field of evolutionary computing has been much discussed. The traditional view, which can be traced back to Holland (1975), has been that the primary search operator in evolutionary computing should be recombination. In its most extreme form, this view casts mutation and other local operators as mere adjuncts to recombination, playing auxiliary (if important) rôles such as keeping the gene pool well stocked and helping to tune final solutions. There have, however, long been advocates of a greater rôle for mutation, hill-climbing and local refinement. The arguments for serious consideration of operators other than recombination for primary search come in many forms and are inspired by widely differing applications. For example, Davis (1991) advocates *hybridisation* of genetic algorithms with domain-specific techniques for "real world" optimisation, by incorporating extra move operators. He regularly uses sophisticated decoders that make use, for example, of greedy algorithms and repair mechanisms. Ackley (1987) recommends *genetic hill-climbing*, in which crossover plays a rather less dominant rôle. Muehlenbein (1992) argues theoretically and Gorges-Schleuter (1989) provides empirical demonstrations that local search can play a key rôle, and Muehlenbein (1989) incorporates it as a fundamental component of his particular notion of a parallel genetic algorithm with a structured population. Meanwhile, the *Evolution Strategies* community has always placed more emphasis on mutation than crossover (Baeck *et al.*, 1991). Countless other advocates of a greater emphasis on non-recombinative elements of evolutionary search could be cited, especially from the ranks of those competing with domain-specific techniques.

Moscato & Norman (1992) have introduced the term *memetic algorithm* to describe evolutionary algorithms in which local search plays a significant part. This term is motivated by Richard Dawkins's notion of a *meme* as a unit of information that reproduces itself as people exchange ideas (Dawkins, 1976). A key difference exists between genes

and memes: before a meme is passed on, it is typically adapted by the person who transmits it as that person thinks, understands and processes the meme, whereas genes get passed on whole. Moscato and Norman liken this thinking to local refinement, and therefore promote the term "memetic algorithm" to describe genetic algorithms that use local search heavily.

The purpose of this paper is three-fold. The first aim is to formalise Norman and Moscato's memetic algorithms and to provide a unified framework for considering both memetic and genetic algorithms. The second aim is to use forma analysis (Radcliffe, 1991; 1994a) to devise further representation-independent operators to augment those previously developed in Radcliffe (1991, 1994a and 1994b). In particular, a generalised form of N-point crossover (GNX) will be defined, as will a general hill-climbing operator. This will allow the construction of a representation-independent (formal) memetic algorithm. The third aim is to investigate the application of the ideas developed to the travelling sales-rep problem (TSP) to test their efficacy.

2 Memetic Algorithms

The first task in this work is to provide a homogeneous formal framework for considering memetic and genetic algorithms. Informally, the idea exploited to achieve this is that if a (true) local optimiser is added to a genetic algorithm, and applied to every child before it is inserted into the population (including the initial population) then a memetic algorithm can be thought of simply as a special kind of "genetic" search over the subspace of local optima (figure 1). Recombination and mutation will usually produce solutions that are outside this space of local optima (and can thus be regarded as "damaged") but a local optimiser can then "repair" such solutions to produce final children that lie within this subspace, yielding a memetic algorithm. Section 2.1 formalises these notions and section 2.2 discusses when such memetic search might be more appropriate than genetic search.

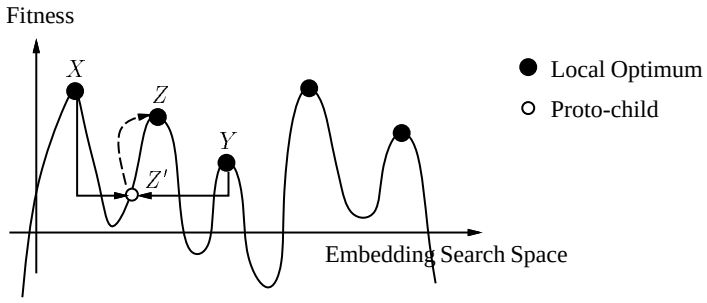


Fig. 1. Memetic algorithms search over the subspace of local optima within the embedding search space of all solutions. After recombination, the proto-child typically lies outside this subspace and a local optimiser is used to "repair" the child so that it lies at a local optimum. Here parents X and Y produce the proto-child Z', which is then optimised to produce the final child Z.

2.1 Formal Memetic Algorithms

Consider a search space $\mathcal{S}$ (of *phenotypes*) and a representation space $\mathcal{C}$ (of *genotypes*). Let

$$\rho : \mathcal{S} \longrightarrow \mathcal{C} \tag{1}$$

be the representation function which, given any solution in $\mathcal{S}$, returns the chromosome in $\mathcal{C}$ that represents it. It will be assumed throughout this paper that ρ is injective (so that every solution $s \in \mathcal{S}$ has a well-defined, unique chromosome $\rho(s) \in \mathcal{C}$ to represent it) but not that it is surjective, (so there may be chromosomes in $\mathcal{C}$ that do not correspond to any solution in $\mathcal{S}$). Let f be the fitness function, which it will be convenient to regard as a mapping

$$f : \mathcal{C} \longrightarrow \mathbb{R}^+ . \tag{2}$$

It will be assumed that the aim is to maximise fitness, and the set of global optima will be denoted $\mathcal{C}^\star \subset \mathcal{C}$.

Let $\mathcal{Q}$ be a stochastic unary move operator over $\mathcal{C}$. It will be convenient for the moment to accommodate the stochastic element of such an operator through a *control set*, $\mathcal{K}_\mathcal{Q}$, from which a *control parameter* will be drawn to determine which of the (typically many) possible moves actually occurs. For example, in the case of mutation of binary strings, a binary mask might be used as the control parameter with the presence of a 1 at position i indicating that the ith bit should be mutated. The functional form for $\mathcal{Q}$ will then be

$$\mathcal{Q} : \mathcal{S} \times \mathcal{K}_\mathcal{Q} \longrightarrow \mathcal{S}. \tag{3}$$

A chromosome $x \in \mathcal{C}$ will be said to be *locally optimal with respect to $\mathcal{Q}$,* or *$\mathcal{Q}$-opt,* if no chromosome of higher fitness than x can be generated from it by a single application of $\mathcal{Q}$, i.e. if and only if

$$\forall \kappa \in \mathcal{K}_\mathcal{Q} : f(\mathcal{Q}(x, \kappa)) \leq f(x). \tag{4}$$

Let $\mathcal{C}_\mathcal{Q} \subset \mathcal{C}$ be the set of $\mathcal{Q}$-opt chromosomes in $\mathcal{C}$, i.e.

$$\mathcal{C}_\mathcal{Q} \triangleq \left\{ x \in \mathcal{C} \;\middle|\; x \text{ is } \mathcal{Q}\text{-opt} \right\} . \tag{5}$$

A genetic algorithm applied to the task of optimising f over $\mathcal{C}$ has some goal such as finding some or all optima in $\mathcal{C}^\star$ or making rapid improvements towards fitter chromosomes. It is clear that for any move operator $\mathcal{Q}$, all chromosomes in $\mathcal{C}^\star$ are $\mathcal{Q}$-opt, and thus $\mathcal{C}^\star \subset \mathcal{C}_\mathcal{Q}$. It would be perfectly satisfactory, therefore, to formulate the search instead over $\mathcal{C}_\mathcal{Q}$.

Given a representation space $\mathcal{C}$, a move operator $\mathcal{Q}$, and the subspace $\mathcal{C}_\mathcal{Q}$ of local optima as above, define a *hill-climber* to be any stochastic, parameterised operator that, given a chromosome $x \in \mathcal{C}$, returns a local optimum in $\mathcal{C}_\mathcal{Q}$. Thus a hill-climber $\mathcal{H}$ with control set $\mathcal{K}_\mathcal{H}$ is any function

$$\mathcal{H} : \mathcal{C} \times \mathcal{K}_\mathcal{H} \longrightarrow \mathcal{C}_\mathcal{Q} . \tag{6}$$

Notice that there is no requirement that the solution returned be in any sense "near" the starting solution, though of course this will often be the case in practice.

Typical genetic algorithms produce new chromosomes by recombination of two parents followed by some small level of mutation, so that if

$$\mathcal{X} : \mathcal{C} \times \mathcal{C} \times \mathcal{K}_{\mathcal{X}} \longrightarrow \mathcal{C} \tag{7}$$

is the recombination operator (with control set $\mathcal{K}_{\mathcal{X}}$), and

$$\mathcal{M} : \mathcal{C} \times \mathcal{K}_{\mathcal{M}} \longrightarrow \mathcal{C} \tag{8}$$

is the mutation operator (with control set $\mathcal{K}_{\mathcal{M}}$), the combined genetic reproductive function $\mathcal{R}_g$ would typically be given by the composition of mutation and recombination, $\mathcal{R}_g = \mathcal{M} \circ \mathcal{X}$, yielding

$$\mathcal{R}_g : \mathcal{C} \times \mathcal{C} \times \mathcal{K}_{\mathcal{M}} \times \mathcal{K}_{\mathcal{X}} \longrightarrow \mathcal{C}, \tag{9}$$

defined by

$$\mathcal{R}_g(x, y, \kappa_{\mathcal{M}}, \kappa_{\mathcal{X}}) \triangleq \mathcal{M}(\mathcal{X}(x, y, \kappa_{\mathcal{X}}), \kappa_{\mathcal{M}}). \tag{10}$$

If, however, $\mathcal{R}_g$ is further composed with a hill-climber $\mathcal{H}$ (with respect to some unary move operator $\mathcal{Q}$), and restricted to $\mathcal{C}_{\mathcal{Q}}$, a memetic reproduction function $\mathcal{R}_m \triangleq \mathcal{H} \circ \mathcal{M} \circ \mathcal{X}$ results:

$$\mathcal{R}_m : \mathcal{C}_{\mathcal{Q}} \times \mathcal{C}_{\mathcal{Q}} \times \mathcal{K}_{\mathcal{H}} \times \mathcal{K}_{\mathcal{M}} \times \mathcal{K}_{\mathcal{X}} \longrightarrow \mathcal{C}_{\mathcal{Q}}, \tag{11}$$

defined by

$$\mathcal{R}_m(x, y, \kappa_{\mathcal{H}}, \kappa_{\mathcal{M}}, \kappa_{\mathcal{X}}) \triangleq \mathcal{H}(\mathcal{M}(\mathcal{X}(x, y, \kappa_{\mathcal{X}}), \kappa_{\mathcal{M}}), \kappa_{\mathcal{H}}). \tag{12}$$

2.2 Decomposable Fitness Functions

While the general question of when it might be appropriate to use a memetic algorithm in preference to a genetic algorithm is beyond the scope of this paper, one special situation can be considered that seems likely to be relatively favourable to the memetic variety. This arises when the fitness function is *decomposable*, in the sense that computing the fitness of a solution given the fitness of another solution that is "close" to it (in the sense, informally, of having much genetic material in common with it) is significantly less computationally expensive than computing the fitness of a solution "from scratch". In the TSP, for example, computing the length of a tour that shares most of its edges with another tour whose length is already known is very much cheaper than computing the length of a general tour, so the fitness function is in that case decomposable. Contrariwise, when solving a system of non-linear equations, for example to compute the flow of gas through a pipe network, a small change in the chromosome can often have global effects and therefore computing the fitness of a chromosome is made no easier by knowing that of another similar chromosome. Given that in most real-world optimisation problems calculation of fitness accounts for almost all the time spent in a genetic algorithm, it seems likely that memetic algorithms will be at an advantage when the fitness function is decomposable, provided that the moves it makes while hill-climbing are "small".

3 Representation-Independent Operators

The principal complication that arises in defining representation-independent operators is that some combinations of gene values may be incompatible. While this is most obviously a problem for recombination operators, it is also a serious consideration for other move operators. As forma analysis (Radcliffe, 1991) has been developed, significant efforts have been made to define representation-independent operators and to understand and classify the kinds of representations that can arise in evolutionary search. In particular, one representation-independent mutation operator—binomial minimal mutation (BMM; Radcliffe, 1994b)—and three representation-independent recombination operators—random respectful recombination (R^3; Radcliffe, 1991), random transmitting recombination (RTR; Radcliffe, 1992), and random assorting recombination (RAR; Radcliffe, 1994a)—have been developed. It is not necessary to revisit all of these for the purposes of this paper, but it is necessary to define some recombination operator and some mutation operator. BMM and RAR will therefore be reviewed briefly in sections 3.2 and 3.3, after which, in section 3.4, a further representation-independent recombination operator will be introduced—generalised N-point crossover (GNX). Attention will then be turned to generalised memetic operators, with a consideration of representation-independent *patching* operators in section 3.5 and representation-independent *hill-climbing* in section 3.6. Before any of this can be achieved, however, it is first necessary to introduce a distinction between two kinds of formal representations—genetic and allelic.

3.1 Genetic Representations and Allelic Representations

The notions of genes and alleles are very familiar, but need to be defined rather carefully for present purposes. A distinction will be drawn between *genetic* representations and *allelic* representations. A formal genetic representation is precisely a formal version of the familiar string composed of genes, and should cause little confusion. It will be assumed that a genetic representation consists of a string of n *genes*, numbered 1 to n, and that each gene takes on values from some (typically but not necessarily finite) set $\mathcal{A}_i$. Thus in the case of a genetic representation, the representation space will be assumed to have the form

$$\mathcal{C} = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n, \tag{13}$$

so that a chromosome is formally a vector of gene values. The only complication with respect to the typical case is that, as before, it will not be assumed that all members of $\mathcal{C}$ correspond to solutions in the search space $\mathcal{S}$, so some combinations of gene values may be "illegal".

A formal *allele* in the context of a genetic representation will be considered to be an ordered pair consisting of a gene and one of its possible values, so that a chromosome $x = (x_1, x_2, \ldots, x_n)$ has alleles $(1, x_1), (2, x_2), \ldots, (n, x_n)$. This formulation of alleles, so far from being new, was suggested in Holland (1975), albeit with different motivation.

There are situations in which a suitable genetic representation of the form described above is not straightforwardly available. In such situations, it may be appropriate to

drop the requirement that genes be defined, working instead with an *allelic representation* (Radcliffe, 1994b). In such an allelic representation, instead of being a vector, a chromosome is a *set* whose elements are drawn from some universal set $\mathcal{A}$. In order to qualify as a formal allelic representation, all that is necessary is that the representation function ρ of equation 1 be injective, as required previously, and that $\mathcal{C}$ be a subset of $\mathbb{P}(\mathcal{A})$, where $\mathbb{P}(\mathcal{A})$ denotes the *power set* (set of all subsets) of $\mathcal{A}$. Again, there is no requirement that all members of $\mathcal{C}$ represent solutions in $\mathcal{S}$.

More concretely, consider representations of the TSP based on edges (city-to-city links). If these edges are considered to be directed, then a genetic representation is arrived at simply by letting the ith gene take the value of the city visited after city i, so that $(4, 3, 1, 2)$ represents the tour that goes from city 1 to city 4, to city 2, to city 3, to city 1 (sic). If, however, the edges are considered to be undirected (so that the 3–2 edge and the 2–3 edge are equivalent) it is no longer straightforward to identify genes, because each city is connected to two others. In this case, one approach is simply to let $\mathcal{A}$ be the set of all possible edges and represent a tour by the set of (undirected) edges it contains. The tour represented by $(4, 3, 1, 2)$ in the directed-edge representation is then represented by $\{$ 1–4, 2–4, 2–3, 1–3 $\}$ in the undirected edge representation, where the edges have all been written with the lower-numbered city first the emphasize their directionless nature.

It is obviously trivial to construct an allelic representation from a genetic representation by taking $\mathcal{A}$ to be the set of all alleles, so that (referring to equation 13)

$$\mathcal{A} = \bigcup_{i=1}^{n} \mathcal{A}_i. \tag{14}$$

Under this scheme a solution is represented simply by its set of (formal) alleles, so that $(4, 3, 1, 2)$ in the directed edge representation gives rise to $\{(1, 4), (2, 3), (3, 1), (4, 2)\}$ in the allelic representation. This motivates the term "allelic representation", and the members of $\mathcal{A}$ will henceforth be referred to as alleles whether they are alleles in the sense of ordered pairs of gene values from a genetic representation or simply members of a given set $\mathcal{A}$ used directly to construct an allelic representation.

It is only slightly less obvious that given an allelic representation it is also easy to construct a genetic representation from it by creating for each member of $\mathcal{A}$ a binary gene that takes the value 1 if the (allelic) chromosome contains that allele and 0 if it does not. It should be noted, however, that such an induced genetic representation is very different from the initial allelic representation, so much so that if an allelic representation is then constructed from the (induced) genetic representation it will be quite different, in general, from the original allelic representation. For this reason, most of the operators introduced below are defined with respect to allelic representations, which allows them to be used for (natural) allelic representations or genetic representations without complication.

To reduce possible confusion, genetic chromosomes will be denoted with lower case letters x, y, z and allelic chromosomes will take upper case letters X, Y, Z.

3.2 Binomial Minimal Mutation (BMM)

In a general representation it will often not be possible to use standard gene-wise mutation because the new allele chosen may well be incompatible with other alleles in

the chromosome. For convenience it will here be assumed that chromosomes have a fixed number n of alleles, though it is simple to relax this restriction. Allelic representations will be considered, so that a chromosome will be taken to be a set of exactly n alleles from $\mathcal{A}$. A distance measure, D, between two chromosomes can then be introduced, and will be taken to be the number of alleles present in one chromosome but not the other:

$$D(X,Y) \triangleq n - |X \cap Y|. \tag{15}$$

Y will be said to be a *minimal mutation* of X if and only if there is no other chromosome in $\mathcal{C}$ closer than Y to X with respect to D. Thus the set $M_D(X)$ of minimal mutations of X is given by

$$M_D(X) = \left\{ Y \in \mathcal{C} \;\middle|\; \forall Z \in \mathcal{C} \setminus \{X\} : \; D(X,Y) \leq D(X,Z) \right\}, \tag{16}$$

where $\setminus$ denotes set subtraction. For example, in the undirected edge representation for the TSP, any tour that can be constructed from another by reversing some section of it is one of its minimal mutations, because reversing a section involves breaking only two edges and there is no pair of tours that differ by only one edge (figure 2).

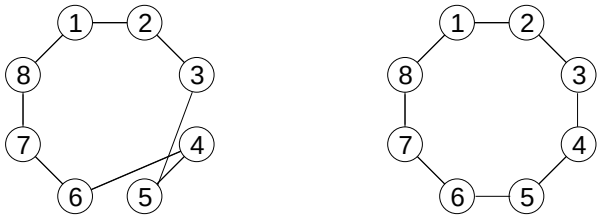


Fig. 2. In the undirected edge representation, the left-hand tour is { 1–2, 2–3, 3–5, 4–5, 4–6, 6–7, 7–8, 1–8 } and the right-hand tour is represented by { 1–2, 2–3, 3–4, 4–5, 5–6, 6–7, 7–8, 1–8 }. In this representation, these two tours are minimal mutations of each other, because they differ by exactly two edges and no pair of distinct tours, in this representation, differ by fewer than two edges.

The *binomial minimal mutation* operator (BMM) takes a single parameter p_m, which specifies the probability of performing each possible minimal mutation. A number k of mutations to perform is then selected from the binomial distribution $B(n, p_m)$, where n, as above, is the number of alleles in each chromosome. This choice ensures that in the case of orthogonal representations BMM's behaviour mimics that of conventional gene-wise mutation. A sequence of k chromosomes is generated, each of which is a minimal mutation of the previous, the first in the sequence being the chromosome to be mutated and the last being the resultant chromosome. Thus if

$$\hat{\mathcal{M}} : \mathcal{C} \longrightarrow \mathcal{C} \tag{17}$$

is a stochastic operator that returns a randomly (uniformly) chosen member of $M_D(X)$, BMM is its kth iterate:

$$\text{BMM}(X, p_m) \triangleq \hat{\mathcal{M}}^k(X), \tag{18}$$

where

$$k \sim B(n, p_m). \tag{19}$$

Note that this operator does not exclude the possibility that subsequent mutations reverse earlier ones, but in practice the likelihood of this is low for small values of p_m. (In the case of orthogonal representations, this is the only difference between BMM and conventional gene-wise mutation.) Note also that this operator is really only appropriate provided that any (legal) point in the representation space can be reached by a finite sequence of minimal mutations from any other point, so that BMM satisfies the requirements of ergodicity (Radcliffe, 1991).

3.3 Random Assorting Recombination (RAR)

Random assorting recombination (RAR_w) may be viewed as a generalisation of uniform crossover, though this was not its genesis. Informally, it proceeds to choose alleles from those of the parents, inserting them in the child when it can, and discarding them otherwise. If the parents' alleles become exhausted before the child is fully specified, its remaining alleles are set either at random (from among the legal combinations) or by some form of *patching*. As with uniform crossover, locus has no effect on the likelihood that a group of alleles will be inherited, and—neglecting the fact that alleles from one parent are known to be compatible, whereas those from different parents may not be—the number of alleles taken from each parent is binomially distributed. Indeed, in the limit of orthogonal genetic representations (those in which all allele patterns are legal) RAR_w reduces to uniform crossover (with parameter half).

RAR_w takes a parameter w that specifies a relative weighting between alleles common to the parents and those that are present only in one. $\text{RAR}_w(X, Y)$ begins by assigning to each allele $a \in X \cup Y$ a weight $W(a)$ given by

$$W(a) = \begin{cases} w, & \text{if } a \in X \cap Y, \\ 1, & \text{otherwise.} \end{cases} \tag{20}$$

It then initialises an empty child $Z_0 = \varnothing$ and selects an allele a_0 from $\mathcal{G}_0 \triangleq X \cup Y$, with probability proportional to its weight. This allele is added to the proto-child to form Z_1. The following process is then repeated for steps indexed by i:

Repeat until $\mathcal{G}_i = \varnothing$:

1. Let $\mathcal{G}_i \triangleq \mathcal{G}_{i-1} \setminus a_i$.
2. Choose a new allele a_i from $\mathcal{G}_i$ with probabilities proportional to the weights of the alleles in $\mathcal{G}_i$.
3. Let $Z_i \triangleq \begin{cases} Z_{i-1} \cup \{a_i\}, & \text{if } a_i \text{ is compatible with those in } Z_{i-1}, \\ Z_{i-1}, & \text{otherwise.} \end{cases}$
4. $i \leftarrow i + 1$.

In step 3 above, "compatible" means that there exists a solution in $\mathcal{S}$ whose representative in $\mathcal{C}$ has all the alleles in Z_{i-1} and also a_i.

At this stage it is possible that the child will be completely specified, but in general this will not be the case. If it is not, a patching algorithm must be used to complete

the child. The most general way to achieve this is to select randomly (uniformly) from the chromosomes that include all the alleles in the proto-child constructed thus far. Section 3.5 introduces more sophisticated memetic patching operators.

3.4 Generalised N-point Crossover (GNX)

In constructing a generalised form of N-point crossover, it is convenient to consider only genetic representations. The difficulty in applying conventional crossover operators is that not all combinations of gene values are legal. Let $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_N\}$ be a set of cross points, with $0 < \ell_1 < \ell_2 < \cdots < \ell_N < n$. This breaks a parent (genetic) chromosome x into $N + 1$ segments

$$(x_1, x_2, \ldots, x_{\ell_1 - 1}),\ (x_{\ell_1}, x_{\ell_1 + 1}, \ldots, x_{\ell_2 - 1}),\ \ldots,\ (x_{\ell_N}, x_{\ell_N + 1}, \ldots, x_n), \qquad (21)$$

and breaks up the second parent y into corresponding segments.

GNX proceeds by picking a random order to visit the $N + 1$ segments from alternate parents, and within each segment "tests" each allele in a random order. An allele is "tested" by seeing whether it can be placed in the child—whether it is compatible with those alleles that have already been placed in it. If compatible, the new allele is inserted, otherwise it is discarded. Because in general after this process has terminated the child will still be incomplete, the process is then repeated with the alternating untested segments from the parents, again visiting these in a random order and testing the alleles within them in random sequence. If the child is still incomplete after this, the child is completed at random or by patching in the same way as for RAR. The general pattern of progress of GNX is shown in figure 3.

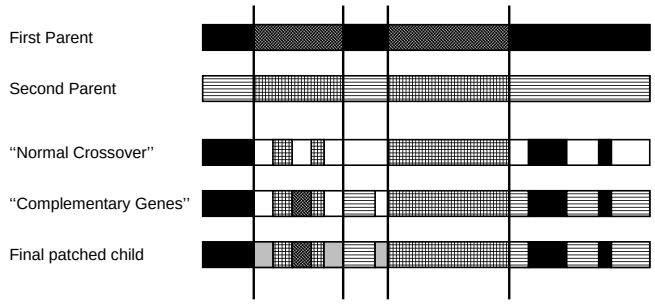


Fig. 3. GNX first copies gene value from alternating segments of the parent chromosomes, visiting the segments and testing the genes within these segments in a random order. Gene values are copied to the child only if they are compatible with those already present. For genes not able to be assigned by this process, alleles from the unused (complementary) segments of the parent genomes are then tested, again in random sequence, for inclusion. Genes still not assigned after this process are assigned either at random, from the set of legal combinations, or by some heuristic or other patching procedure.

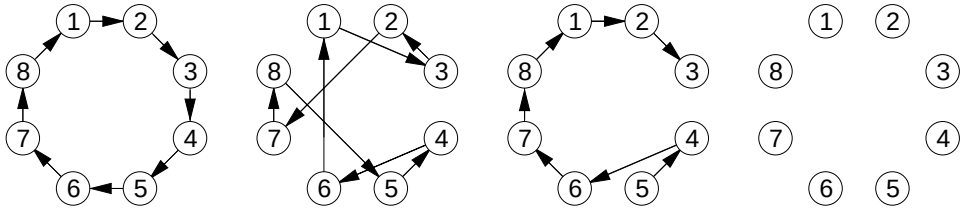


Fig. 4. Two parent tours, one possible partial child they can produce under GNX, and an empty grid for the reader to use while following through the example.

An example using the TSP may help to clarify this. In order to follow this example, the reader may find it helpful to try connecting up the "empty" tour in figure 4 to understand the edge acceptances and rejections. Consider the directed edge representation for the TSP and G2X with cross points 3 and 6 with parents as shown in figure 4 and given by

$$\begin{aligned} x &= (2,3,4 \mid 5,6,7 \mid 8,1), \\ y &= (3,7,2 \mid 6,4,1 \mid 8,5). \end{aligned} \tag{22}$$

(Recall that in this representation the ith gene represents a directed edge from city i to city x_i.)

Suppose the permutation of the segments chosen is $(3,2,1)$. Then the third segment of x (visited first) will be inserted whole, giving edges $7 \rightarrow 8$ and $8 \rightarrow 1$ (or the proto-child $(\square, \square, \square, \square, \square, \square, 8, 1)$). Then alleles in segment 2 from y will be tested in a random order, say $5 \rightarrow 4, 6 \rightarrow 1, 4 \rightarrow 6$ and the first and third (in this case) will be accepted, giving the proto-child $(\square, \square, \square, 6, 4, \square, 8, 1)$. The first segment of x is then tested, and the edges $1 \rightarrow 2$ and $2 \rightarrow 3$ will be accepted giving $(2,3,\square, 6, 4, \square, 8, 1)$. This completes the first phase.

The untested segments are then visited in random order, say first $(5,6,7)$ from x, then $(3,7,2)$ from y, and finally $(8,5)$ from y. During this process only the edge $6 \rightarrow 7$ will be accepted, giving the proto-child $(2,3,\square, 6, 4, 7, 8, 1)$.

Since this child is still incomplete, it must be patched. In this case however, only one legal chromosome (with directed edges) has the required allele pattern, namely $(2,3,5,6,4,7,8,1)$, so it would be the result of the cross.

3.5 Patching by Forma Completion

Both RAR and GNX produce (in general) partially-specified children that then need to be completed in some manner. In the case of GNX, it is reasonably natural to think of the partially completed child as a schema. In the case of RAR (which works with allelic representations) this is less natural, but a child is specified precisely by a set of alleles it should contain, and such a specification qualifies as a *forma*—a set of chromosomes sharing certain alleles. A schema may be viewed as a special case of a forma, applicable to the case of genetic representations. The question that arises for both RAR and GNX is thus how to choose a child from a given forma. The "default" method is to choose

one randomly but two other methods of patching (or "completing") formae to produce children will be considered.

One option is to choose the best solution in the forma. In general, this would be prohibitively expensive, but if the number of unspecified alleles were small and the fitness function were decomposable this method can reasonably be considered. This method will be called *globally optimal forma completion.*

A more practical method in many circumstances is to find a local optimum within the forma. With *locally optimal forma completion,* this is achieved by completing the forma at random and then testing minimal mutations that remain within it in sequence, accepting any that are better. This process continues until there is no minimal mutation within the forma that is better than the current solution.

3.6 Allelic Hill-Climbing

In section 2.1, a hill-climber (with respect to a move operator $\mathcal{Q}$) was defined to be any operator $\mathcal{H}$ having the functional form given in equation 6:

$$\mathcal{H} : \mathcal{C} \times \mathcal{K}_{\mathcal{H}} \longrightarrow \mathcal{C}_{\mathcal{Q}}. \qquad (6 \text{ bis})$$

It is easy to construct a hill-climber from $\mathcal{Q}$ by repeatedly applying $\mathcal{Q}$ to the chromosome to be optimised, cycling through all the control parameters from $\mathcal{K}_{\mathcal{Q}}$. There is considerable freedom in exactly how such a hill-climber operates. In particular, there are many ways to decide when to accept a move and in which order to cycle through the control parameters. The hill-climber constructed here will be *greedy,* that is, it will accept any improvement generated by the operator immediately (and will never backtrack), as opposed, for example, to testing all control parameters and then accepting the move that generates the biggest improvement.

The order in which to test the control parameters in $\mathcal{K}_{\mathcal{Q}}$ is more open. Any order will suffice provided that all parameters are tested (preferably without repetition) but a fixed order will afford the operator considerably less freedom than a random order. Testing the parameters in a totally random order (excluding only repetition) is by far the most appealing theoretically, and will almost certainly show the best performance because it minimises the correlations between applications of $\mathcal{H}$. In practice, however, this requires the generation of a very large number of (pseudo-) random numbers, and maintenance of a list of the moves that have been tested (or of those that remain to be tried). Nevertheless, this form of hill-climbing is sufficiently important to be named, and will be referred to as *ideal greedy hill-climbing.*

Many compromises between a totally random and a fixed order of sampling $\mathcal{K}_{\mathcal{Q}}$ across applications of $\mathcal{Q}$ are possible. Two in particular will be considered. The first is to construct a random permutation of the control parameters to $\mathcal{Q}$ when the hill-climber is invoked, and to sample these in sequence. When an application of $\mathcal{Q}$ yields an improvement (and is therefore accepted), a new starting point within the permuted values is chosen, but the parameters are then sampled in the same order. This scheme will be called *rotated cyclic greedy hill-climbing.* A minor augmentation of this scheme involves also exchanging a randomly chosen pair of control parameters in the permutation when a move is accepted. This scheme will be called *rotated transposed cyclic greedy hill-climbing.*

For the purposes of the formal memetic algorithm that is the subject of this paper, the move operator defining local optimality will be minimal mutation ($\hat{\mathcal{M}}$).

4 Empirical Setting: Application to the TSP

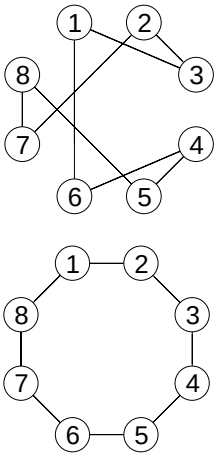
Earlier sections having constructed the formal components required for a memetic algorithm, the present section seeks to construct an instantiation for comparison with its genetic counterpart and for simple experimentation with its coarser parameters. The problem tackled will be the travelling sales-rep problem (TSP) because this has a decomposable fitness function (and should therefore be favourable to memetic search), is well-known, and is relatively hard for evolutionary techniques. Most studies of large TSP instances have previously concluded that augmentation with local search is essential, (e.g. Verhoeven *et al.*, 1992; Gorges-Schleuter, 1989) and the aim here is not to achieve good performance as such but rather to understand how well an unembellished implementation of a formal memetic algorithm can work in this problem domain. For this reason, a modest but non-trivial TSP instance is used for these experiments.

Previous studies by Whitley *et al.* (1989) and Radcliffe (1994b) have provided evidence, both theoretical and empirical, that undirected edges are a relatively suitable basis for a representation of this problem, so the undirected edge representation discussed above will be used. This provides a small difficulty, however, in that the undirected edge representation is allelic, but it would be interesting to apply the GNX operator to this problem. This difficulty can be overcome as follows. In the context of alignment for crossover (only), the alleles (edges) making up a chromosome X will be arranged in the order they are visited in the tour, followed in a consistent direction, starting from city 1, to form a corresponding genetic chromosome x. This guarantees that every city occurs exactly once at the "start" of an edge, and gives appropriate linkage to adjacent edges. When brought together for crossover, the alleles in the second parent are then re-ordered to align with those in the first parent, and this allows GNX to proceed sensibly (figure 5). This borrows from the original view provided in Holland (1975) of crossover as a locus-independent operator in the context of re-linking operators such as inversion.

It may seem as if the effect of this procedure is to manipulate the edges as directed, but it is important to appreciate that this is not the case, for when an edge is tested for compatibility with those in the proto-child, no account is taken of its direction. Thus the direction of the tour affects only the order in which alleles from the parents are tested for inclusion in the child, not the direction in which they occur in the child.

5 Results and Discussion

Empirical studies were undertaken using the Reproductive Plan Language RPL2 (Surry & Radcliffe, 1994) running on super-scalar SPARC processors. The problem instance used was the 100-city Krolak 'C' problem from TSPLIB (Reinelt, 1990). All experiments used a panmictic population of size 100 with elitism, non-generational ("one-at-a-time") update, binary tournament selection with parameter 0.7, binary tournament replacement with parameter 0.7, recombination with probability 1.0 and mutation probability p_m of 0.02. The weight used for RAR was 2.0, and the number of cross points used



Parent 1
Unordered: { 7–8, 2–7, 1–6, 8–5, 4–5, 1–3, 6–4, 2–3 }
Ordered around tour: (1–3, 3–2, 2–7, 7–8, 8–5, 5–4, 4–6, 6–1)
Aligned "Chromosome": (3, 2, 7, 8, 5, 4, 6, 1)

Parent 2
Unordered: { 1–2, 6–7, 7–8, 1–8, 6–5, 3–2, 4–3, 4–5 }
Ordered around tour: (1–2, 2–3, 3–4, 4–5, 5–6, 6–7, 7–8, 8–1)
Aligned with parent 1: (1–2, 3–4, 2–3, 7–8, 8–1, 5–6, 4–5, 6–7)
Aligned "Chromosome": (2, 4, 3, 8, 1, 6, 5, 7)

Fig. 5. In order to use GNX, which requires a *genetic* representation, with the undirected edge representation, which is *allelic*, a special form of alignment must be performed, creating a "pseudo-genetic representation". Each parent is re-ordered so the the order and sense of the edges are taken by following the tour around. The second parent is then further re-ordered to align it with the first. "Pseudo-genomes" that GNX can then manipulate are then created. Notice, however, that the sense of an edge may be reversed by GNX when it inserts it in the child tour.

for GNX was 2. Experiments were conducted using random and $\hat{\mathcal{M}}$-opt (i.e. minimal-mutation-based) patching, and also using Karp's heuristic (Lawler, 1985), which is specific to the TSP, for comparison. In the memetic experiments, rotated cyclic greedy hill-climbing was applied both to the initial population and before children were inserted into the population. Comparisons with random search and with repeated generation of 2-opt solutions are also shown. The results are shown in figures 6–8, and are on the basis of "wall-clock" time. In all cases, the length of the best tour in the population is plotted, normalised by the length of the optimum tour. Different algorithms are run for different numbers of updates to give broadly comparable total run times. It should, however, be noted that the implementations of the operators used are not tuned, and the implementation of RPL2 itself is still under beta test at the time of writing, so times should be taken as indicative rather than definitive.

As expected, given the decomposable nature of the evaluation function and the large number of possible alleles for the TSP, the memetic algorithms all significantly out-performed their genetic counterparts. Note particularly that with the exception of the algorithms using Karp stitching, all implementations are direct instantiations of the formal, representation-independent algorithms discussed above. The choice of patching algorithm has a large effect on the genetic algorithms, with the $\hat{\mathcal{M}}$-based and Karp stitching providing substantially higher performance, whereas the choice of recombination operator has little effect. Conversely, for memetic search the choice of patching

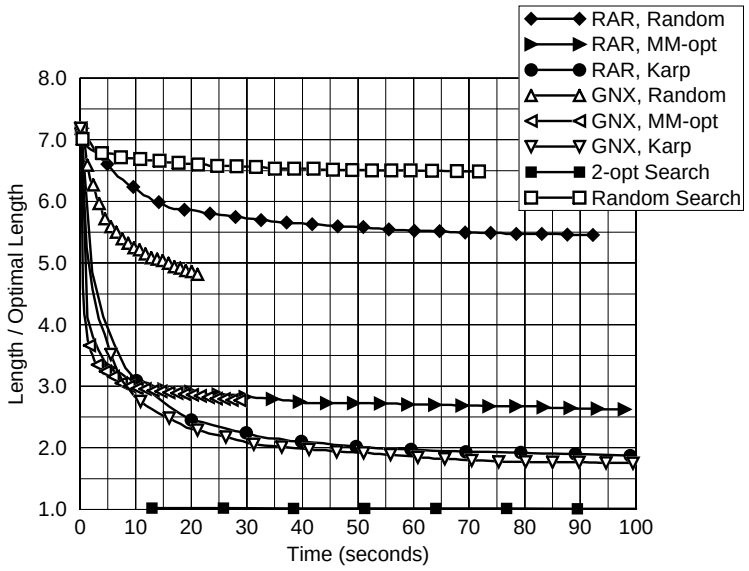


Fig. 6. The performance of genetic algorithms on the 100 city Krolak C TSP is shown. The length of the best solution in the population (relative to the length of the optimal tour) is shown as a function of wall-clock time. Error bars are not shown as they are smaller than the tick marks. For comparison, random search (the top line) is shown, as is the performance achieved by repeatedly generating 2-opt solutions (bottom line). Notice that the genetic algorithm is not remotely competitive with random search over 2-opt solutions. Ticks are placed every 5 generations (500 updates) except for the non-adaptive searches. The ticks for random search over 2-opt solutions occur every 100 updates and those for pure random search every 2,500 updates.

algorithm has relatively little influence over performance, but here G2X is significantly superior to RAR_2. Notice also that genetic algorithms fail by a large margin to match the performance achieved by repeatedly generating 2-opt solutions.

The patching results can be understood since in the memetic case the local search will be able to fix any poor patches, whereas this ability is not present in the genetic algorithm. The results for the two recombination operators seem to indicate that for this problem (in the context of the particular reproductive plans chosen) G2X is genuinely superior to RAR_2. The only cases in which this superiority is not exhibited are the genetic runs with good patching, but here the results suggest that it is the patching that is performing almost all the useful search, masking any distinction between the recombination operators' performances.

Clearly the problem instance chosen is rather easy for memetic search. (The other Krolak 100 city problems have also been tested, with very similar results.) To emphasize this point further, when a 3-opt-based hill-climber was tested, an initial population of 50 solutions was found to contain two copies of the optimum. The problem was, nevertheless, appropriate for this study given the level of difficulty it provided for genetic search. Extensive efforts will now be made to tackle larger problems with memetic

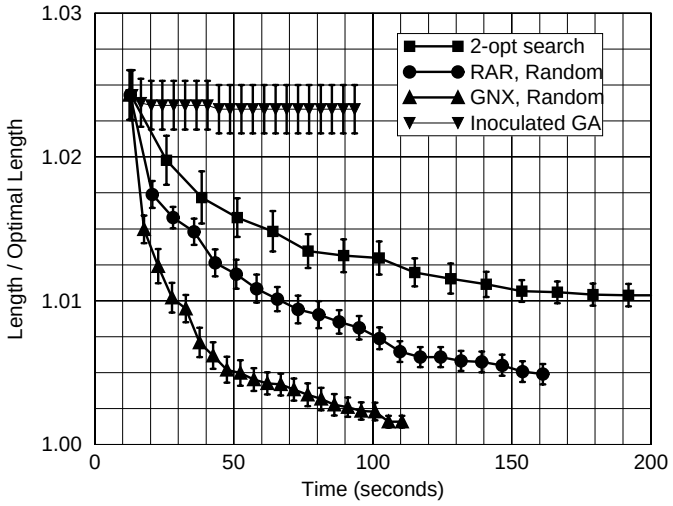


Fig. 7. The performance of memetic algorithms with random patching on the 100 city Krolak C TSP is shown. The top line shows the performance of the best genetic algorithm (i.e. using G2X and Karp stitching) inoculated with a starting population of randomly generated 2-opt solutions. The second highest line is the same as the bottom line of the previous graph, i.e. shows the performance of random search over 2-opt solutions, but notice the massively expanded scale on the y-axis. The bottom two lines show that G2X significantly out-performs RAR_2 on this problem. Tick marks are shown every generation (100 updates) except in the case of the inoculated genetic algorithm, where they are every five generations (500 updates), and error bars indicate standard errors.

search, using structured population models, parallelism and still more sophisticated operators; indeed, this work has already commenced.

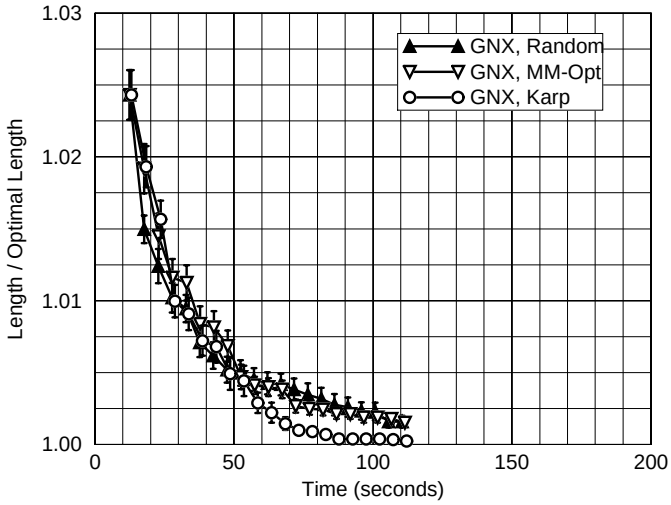


Fig. 8. The performance variation of memetic algorithms using G2X as a function of patching method is shown for the 100 city Krolak C TSP. Notice that the effect of the patching is rather small, in contrast to the large effect it has on the genetic algorithm (figure 6), though Karp stitching still performs best. A similar pattern emerges if RAR is used (not shown), but the performance for each patching method is worse than with G2X. Ticks are shown every generation (100 updates) and error bars indicate standard errors.

References

D. H. Ackley, 1987. *A connectionist machine for genetic hillclimbing*. Kluwer Academic Press, Boston.

Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel, 1991. A survey of evolution strategies. In *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo).

Lawrence Davis, 1991. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold (New York).

Richard Dawkins, 1976. *The Selfish Gene*. Oxford University Press (Oxford).

Martina Gorges-Schleuter, 1989. ASPARAGOS: an asynchronous parallel genetic optimization strategy. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 422–427. Morgan Kaufmann (San Mateo).

John H. Holland, 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press (Ann Arbor).

E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, 1985. *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimisation*. Wiley.

Pablo Moscato and Michael G. Norman, 1992. A "memetic" approach for the travelling salesman problem — implementation of a computational ecology for combinatorial optimisation on message-passing systems. In *Proceedings of the International Conference on Parallel Computing and Transputer Applications*. IOS Press (Amsterdam).

H. Mühlenbein, 1989. Parallel genetic algorithms, population genetics and combinatorial optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 416–421. Morgan Kaufmann (San Mateo).

Heinz Mühlenbein, 1992. How genetic algorithms really work. part I: Mutation and hillclimbing. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature, 2*. Elsevier Science Publishers/North Holland (Amsterdam).

Nicholas J. Radcliffe, 1991. Equivalence class analysis of genetic algorithms. *Complex Systems*, 5(2):183–205.

Nicholas J. Radcliffe, 1992. Genetic set recombination. In Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*. Morgan Kaufmann (San Mateo, CA).

Nicholas J. Radcliffe, 1994a. The algebra of genetic algorithms. *To appear in Annals of Maths and Artificial Intelligence*.

Nicholas J. Radcliffe, 1994b. Fitness variance of formae and performance prediction. Technical report, To appear in Foundations of Genetic Algorithms 3.

Gerhard Reinelt, 1990. TSPLIB.

Patrick D. Surry and Nicholas J. Radcliffe, 1994. RPL2: A language and parallel framework for evolutionary computing. In *Parallel Problem Solving from Nature III (to appear)*.

M. G. A. Verhoeven, E. H. L. Aarts, E. van de Sluis, and R. J. M. Vaessens, 1992. Parallel local search and the travelling salesman problem. In R. Männer and B. Manderick, editors, *Parallel Problem Solving From Nature, 2*, pages 543–552. Elsevier Science Publishers/North Holland (Amsterdam).

Darrell Whitley, Timothy Starkweather, and D'Ann Fuquay, 1989. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo).

This article was processed using the LaTeX macro package with LLNCS style