

Digital Filter Design Using Multiple Pareto Fronts

Thorsten Schnier and Xin Yao
School of Computer Science
The University of Birmingham
Edgbaston, Birmingham B15 2TT, UK
Email: {T.Schnier,X.Yao}@cs.bham.ac.uk

Pin Liu
Marconi plc, Discovery Court
551-553 Wallisdown Road
Poole, Dorset BH12 5AG, UK
Email: pin.liu@marconi.com

Abstract

Evolutionary approaches have been used in a large variety of design domains, from aircraft engineering to the designs of analog filters. Many of these approaches use measures to improve the variety of solutions in the population. One such measure is clustering.

In this paper, clustering and Pareto optimisation are combined into a single evolutionary design algorithm. The population is split into a number of clusters, and parent and offspring selection, as well as fitness calculation, are performed on a per-cluster basis. The objective of this is to prevent the system from converging prematurely to a local minimum and to encourage a number of different designs that fulfil the design criteria.

Our approach is demonstrated in the domain of digital filter design. Using a polar coordinate based pole-zero representation, two different lowpass filter design problems are explored. The results are compared to designs created by a human expert. They demonstrate that the evolutionary process is able to create designs that are competitive with those created using a conventional design process by a human expert. They also demonstrate that each evolutionary run can produce a number of different designs with similar fitness values, but very different characteristics.

1 Introduction

Evolvable hardware (EHW) refers to one particular type of hardware whose architecture/structure and functions change dynamically and autonomously in order to improve its performance in performing certain tasks

⁰Part of the material presented in this paper was published in *Third NASA Workshop on Evolvable Hardware (EH 2001)*, 12-14 July 2001, Long Beach, California, pp 136-145

[14, 15]. The emergence of this new field in recent years has been influenced profoundly by the progresses in reconfigurable hardware and evolutionary computation.

This paper describes an evolutionary design approach to digital filter design. We have applied three different methods to achieve population diversity in our research, i.e., Pareto optimisation, fitness sharing and clustering. Our empirical studies show that the new approach can lead to improved designs of different characteristics.

The rest of this paper is organised as follows.

Section 2 introduces the evolutionary approach, and compares it with the conventional design approach. A detailed definition of the design problem is given in Section 3. The novel features of our algorithm are described in Section 4, where we also give an overview of other aspects of the implementation. Section 5 presents the experimental results. Finally, Section 6 concludes the paper with a brief summary and a few remarks.

2 Conventional and Evolutionary Digital Filter Design

Digital filter design has a number of features that make it very attractive as test problems for our research.

- The problem has engineering relevance. Digital filters play an important role in communication systems, often at the interface between digital and analog signal processing systems. Examples are mobile communications, speech processing, modems, etc.
- While the science of digital filter design is very well established and researched, there are no 'conventional' design procedures that lead to optimal designs with acceptable effort in the general case.

A survey of relevant journals reveals that digital filter design is an active area of research. For example, the leading *IEEE Transactions on Circuits and Systems II* published 12 papers on different approaches to filter design in only two years (1998-99), e.g., [27, 29, 34, 26, 32, 28], to name just a few.

- The design space for digital filters is well defined but large and complex. A well defined space facilitates comparison of different results. A large and complex design space will challenge our evolutionary system and will be good at evaluating our system's suitability in dealing with tough design problems.
- A quantitative measure of filter performance is generally available, providing a fitness measure for EAs that is relatively easy to compute. It also provides a straightforward metric in comparing different designs.

There are many different kinds of digital filters, depending on types of components used (e.g., linear or non-linear components), restrictions on interconnections (e.g., with or without feedback) and the intended characteristics of the filters. The difficulty in designing individual filters depends on the exact type of filters. For some, analytical methods are available. For others, approximation methods have to be used. In any case, different filters generally need different design approaches. A human designer specialised in designing one type of filters might not be able to design an optimal filter of a different type. Since no general design methodology is available, evolutionary design will be a good and automatic alternative to manual design.

2.1 Evolutionary Algorithms

EAs refer to a class of population-based stochastic search algorithms that are developed from ideas and principles of natural evolution. They include evolution strategies (ES) [16], evolutionary programming (EP) [17], and genetic algorithms (GAs) [18]. One important feature of all these algorithms is their population-based search strategy [19]. Individuals in a population compete and exchange information with each other in order to perform certain tasks. A general framework of EAs can be described by Figure 1 [19].

2.2 Advantages of Evolutionary Design

An evolutionary design approach offers a number of advantages over the conventional one used by human designers although there are some important issues that remain open.

First, the evolutionary design approach can explore a much wider range of design alternatives than those could be considered by human beings. This has been shown by many experiments in other design tasks, such as evolutionary design of neural networks [4, 5, 6, 7, 8], or of building architectures [9]. These experiments demonstrated how evolutionary techniques could be applied to evolving novel designs which were difficult to discover by human beings.

Second, the evolutionary design approach does not assume *a priori* knowledge of any particular design domain. It can be applied by users without resorting to domain experts. It can be used in domains where little *a priori* knowledge is available or where such knowledge is very costly to obtain.

Third, the evolutionary design approach is very flexible. It can deal with non-differential or even discontinuous objective functions. It can deal with various linear and nonlinear constraints as well as objectives. Its population-based nature makes it ideal in tackling multi-objective design problems. Although the evolutionary approach can work with little *a priori* domain knowledge, it can incorporate domain knowledge in the chromosome representation and search operators easily if such knowledge is available.

2.3 Limits of the Conventional Process

Designing digital filters, especially recursive filters, is not a straightforward problem. For certain design problems with particular characteristics, it is possible to mathematically derive the optimal filter configuration; but in the general case, no such method exists. Instead, a number of approximation methods have been developed, usually applicable only for a particular class of design problems.

There are two problems with this design method. First, for new problem classes, an approximation approach has to be developed first. For example, in [27], a process for design of stable IIR filters with equiripple passbands and peak-constrained least-squares stopbands is developed. As the title indicates, the class of filters that the method is applicable to is fairly limited. This is different from evolutionary approaches, which

1. Generate the initial population $G(0)$ at random, and set $i = 0$;
2. REPEAT
 - (a) Evaluate each individual in the population;
 - (b) Select parents from $G(i)$ based on their fitness in $G(i)$;
 - (c) Apply search operators to parents and produce offspring which form $G(i + 1)$;
 - (d) $i = i + 1$;
3. UNTIL 'termination criterion' is satisfied

Figure 1: A General Framework of Evolutionary Algorithms.

can generally be written for much larger classes of problems, allowing it to be used for many different filter design problems.

The second problem with the conventional approach is that, depending on the exact approach taken, the resulting design is likely to be suboptimal. For example, the approach taken in [27] is based on iterative quadratic programming method with linearised constraints with a least-square objective. The resulting design would most likely be suboptimal for three different reasons:

- Linearisation of constraints: All constraints have to be formulated as linear inequalities. Constraints that are not initially linear have to be linearised. To ensure that the linearised constraint still excludes all designs that initially violated the constraint, it has to exclude some viable designs (otherwise the linearised version would have to be identical to the original).
- Objective as least-square problem: the objective has to be implemented as a weighted least-square function. Often, this is not exactly the same as the actual design goal; for example in [27] the square error from the desired behaviour is used as objective for the quadratic programming, while the maximum deviation from the desired behaviour is used to compare the final designs with other filter designs. While the maximum deviation is the 'real' design goal, the accumulated weighted square error is used for the design algorithm.
- As noted in [27], the result of the iterated quadratic programming solver is generally only a subopti-

mal design, and generally depends on the starting point.

2.4 Top-Down versus Bottom-up Approach

In conventional digital filter design, a two-step process is usually chosen. In the first step, a mathematical description of the filter fulfilling the design criteria is derived. This description is then transformed into a hardware description in the second step. The two steps are very different in terms of difficulty, methods employed, and performance criteria. Similar to many conventional filter design papers [27], our work follows this top-down process, and the work reported here only deals with the more difficult first step. It produces a near optimal polynomial that can then be transformed into hardware implementation.

In the literature, a number of approaches to evolutionary digital filter design can be found that follow a bottom-up approach, where the evolutionary process is used directly to specify a filter in terms of building blocks and their connections. Generally, the building blocks used are very low-level, for example gate level or function block level [40, 37, 36, 35]. This bottom-up method has the advantage that a complete filter is evolved, and additional fitness criteria (e.g. sensitivity to parameter quantisation) can be used. On the other hand, the complexity of the evolved filters is limited, typically to second or third order filters. The work presented here is aimed at constructing filters of 'real world' complexity; the examples shown are order 12 and order 15 filters.

3 The Design Problem

3.1 Mathematical Description of Linear Digital Filters

Any linear digital filter can be mathematically specified by a complex-numbered polynomial function, i.e., the transfer function (Equation 1). This polynomial function, i.e., Equation 1, can be rewritten as the quotient of two product terms with the numerator specifying the zeroes of the polynomial and the denominator specifying the poles. The function (i.e., Equation 2) usually has a scaling constant. The two descriptions are equivalent. It is easy to transform a pole-zero description to a polynomial description, but not vice versa. The frequency response can be derived from the transfer function by calculating the values for $z = e^{j\omega T}$, where $T = 2\pi/\omega_s$.

$$H(z) = \frac{\sum_{i=0}^n b_i z^{n-i}}{z^{(n-r)} \sum_{i=0}^r b_i z^{r-i}} \quad (1)$$

$$H(z) = b_0 * \frac{(z - z_{z0})(z - z_{z1}) \dots (z - z_{zn})}{z^{r-i} (z - z_{p0})(z - z_{p1}) \dots (z - z_{pi})} \quad (2)$$

where n is the number of zeros, i is the number of poles, and $r - i$ the number of poles at the origin [27].

It should be noted that not all transfer functions are generally realizable in hardware. Two main requirements have to be observed.

Real coefficients: The coefficients in the polynomial description directly translate to multipliers in the hardware implementation. Since it is very difficult to multiply a signal with a complex number, it is important that all coefficients are real. In terms of poles and zeroes, this can be achieved if all poles and zeroes are either real, or exist in conjugate-complex pairs (i.e. $a + jb$ and $a - jb$).

Stability: Because a general IIR filter has feedback loops, the output may grow without bounds (or in hardware until overflow), or oscillate. In a stable filter, a bounded input will always produce a bounded output. A filter is only stable if all poles are within the unit circle, i.e. $\|a + jb\| < 1$. While there are uses for unstable filters in specific applications, most filters are designed to be stable.

3.2 Performance Criteria and Constraints

Filter performance is usually multi-objective. There is no single universal criterion that applies to all filters.

The precise objectives depend on the type of filters and engineering constraints imposed by their applications. The two filters considered in our work are low-pass filters. An application example would be a filter in a telephone system with a corner frequency of 20kHz. Signals with a frequency below this frequency should pass the filter unmodified, while signals above should be suppressed. An ideal low-pass filter lets signals pass unchanged in the lower frequency region (passband), and blocks signals completely in the upper frequency region (stopband). In reality, a transition band is often located between passband and stopband. The goal of filter design is to minimise distortion of the signal in the passband and maximise suppression in the stopband. The transition band should be as narrow as possible.

To minimise distortion in the signal in the passband, two criteria have to be met. The first is that the amplitude of the frequency response in the passband should be as constant as possible. For example, all frequencies in a speech signal should be amplified by exactly the same amount.

The second criterion is that the phase in the passband has to be as linear as possible. In practice, the so-called ‘group delay,’ i.e., the first derivative of the phase $\frac{d\phi}{d\omega}$, is often used. The second criterion can therefore be stated as a constant group delay.

In the stopband, the design goal is generally to attenuate the signal as much as possible. Because the signal is attenuated, the phase and group delay of the signal in the stopband usually becomes unimportant.

For the transition band, constraints are rarely used.

Figure 2 shows a typical lowpass filter. The top half shows the amplitude and the lower half the group delay. The ideal behaviour is shown with thick lines. The ‘real’ behaviour (thin line) will be acceptable as long as it is within the shaded regions.

To facilitate comparison between our and other existing work, we follow the criteria used previously [27] whenever possible. That is, the following factors are considered:

1. weighted square error over the amplitude in the passband and stopband,
2. peak amplitude in the stopband,
3. maximum deviation from constant amplitude in the passband,
4. maximum deviation from the goal group delay in the passband, and

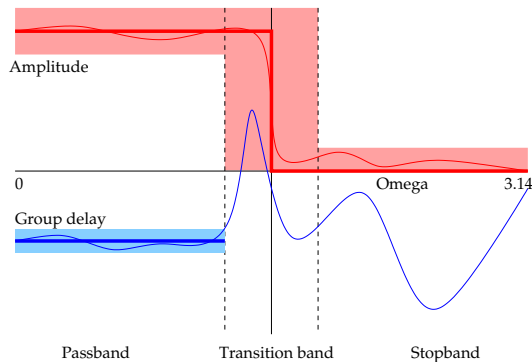


Figure 2: A lowpass filter which is used as the first test problem in our work.

5. stability.

The conventional design process adopted by human experts usually uses the first factor above as the optimisation objective and others as constraints within certain predefined values. By defining our representation so that the poles of any phenotype will always lie within the unit circle, we can guarantee stability of all filters produced by the evolutionary process.

4 The Filter Design Algorithm

The implementation of an evolutionary algorithm for filter design faces a number of challenges. The space of possible filters is very large, and individual parameters are tightly coupled, making it more likely for the algorithm to converge onto local, unsatisfactory sub-optima. For this reason, a number of new techniques have been introduced into the selection process. In this article, we will concentrate on the two main techniques, clustering and per-cluster self-adaptive constraints, which we will describe in detail in this section. We will also give an overview of all the other elements of the evolutionary algorithm at the end of this section; additional descriptions can be found in previous publications ([?, 31]).

4.1 Clustering Pareto Optimisation

As mentioned in Section 3.2, filter performance is generally multi-objective. A selection scheme based on Pareto fitness is a natural choice for our EA. In Pareto selection, any number of criteria can be used. Only a partial order among individuals, based on dominance,

needs to be established. One individual dominates another if and only if its fitness is higher than the other's according to at least one criterion and as good as the other's according to the rest of criteria. A population will usually contain a number of non-dominated individuals, which are referred to as the Pareto front. Among individuals in the Pareto front, it is not possible to say which one is better than the other.

Different individuals from different regions in the Pareto front can have very different genotypes. Pareto selection allows individuals at the Pareto front to co-exist as long as they are not dominated. Fitness sharing can help increase and maintain population diversity. However, these two techniques are not very good at helping dominated individuals to survive in a population. We are interested in some of the dominated individuals because they may be on different fitness peaks ("valleys" in our minimisation case) from those occupied by non-dominated individuals.

In order to allow more than one Pareto front to develop independently in the population, we have introduced a clustering operation. This operation will separate the population into a number of semi-independent sub-population according to their genotypes. Selection of parents and for survival is performed on a per-cluster basis, individuals can therefore survive and create offspring as long as they are on or near the Pareto front for that particular cluster.

The clusters in the population are initially established by applying the k-means algorithm to the random, initial population. The algorithm we implemented then distinguishes between an early 'exploration' phase and a main 'exploitation' phase. In the exploration phase, individuals will cover large parts of the search space, and we encourage exploration of the search space by allowing unrestricted crossover. During this phase, we periodically re-cluster the individuals using k-means clustering. At the end of the exploration phase, the population should have discovered a number of promising areas in the search space, and the algorithm switches to 'exploitation' mode. Here, crossover is restricted to parents of the same cluster only.

Clustering has its largest effect during the selection of the next generation from the current generation and the offspring. In this operation, the cluster memberships are established, and an individual's chance of survival depends very much on which cluster it is in. The following section describes the operation in detail.

4.1.1 Clustering Survivor Selection

Figure 3 shows a flowchart of the selection operation. It starts with the current population and the new offspring at the top. We assume the current population is already divided into clusters, and has its Pareto non-dominated set selected. The algorithm then proceeds in the following steps.

1. *Assigning offspring to clusters.* Each individual in the offspring needs to be assigned to a single cluster. There are two ways of doing this, depending on how the individual was produced. If the offspring is the result of a mutation of a single parent, or of a crossover of two parents from the same cluster, it is simply assigned to the parent's cluster. If it was created from parents from different clusters (in the exploitation phase), the new cluster is selected based on genotype similarity. Each cluster has a 'representative', which is simply the arithmetic centre of the genotypes of the individuals in this cluster. Offspring with different parents will be assigned to the cluster whose centre is closest to it. If the distance to the closest centre is more than m (m is around 1.8 for most of our experiments) times the largest distance between any individual in the population and its associated cluster centre, a complete re-clustering of the population is triggered.
2. *Find non-dominated individuals in offspring.* This is done on a per-cluster basis; all individuals in the offspring are pairwise compared with all other individuals in the same cluster, and tested for Pareto domination. Any individual dominated by any other offspring is flagged as dominated. Individuals that violate at least one of the current constraints (see Section 4.2.1) are always flagged as dominated, independent of the other individuals.
3. *Merge non-dominated set:* The non-dominated sets of current population and offspring are merged on a per-cluster basis; individuals that become dominated during this step are removed from the non-dominated set.
4. *Shrink the Pareto front* In most implementations of Pareto selection with elitism, the number of individuals in the Pareto front will grow continuously through the run. It is necessary to periodically remove individuals from the Pareto front to prevent

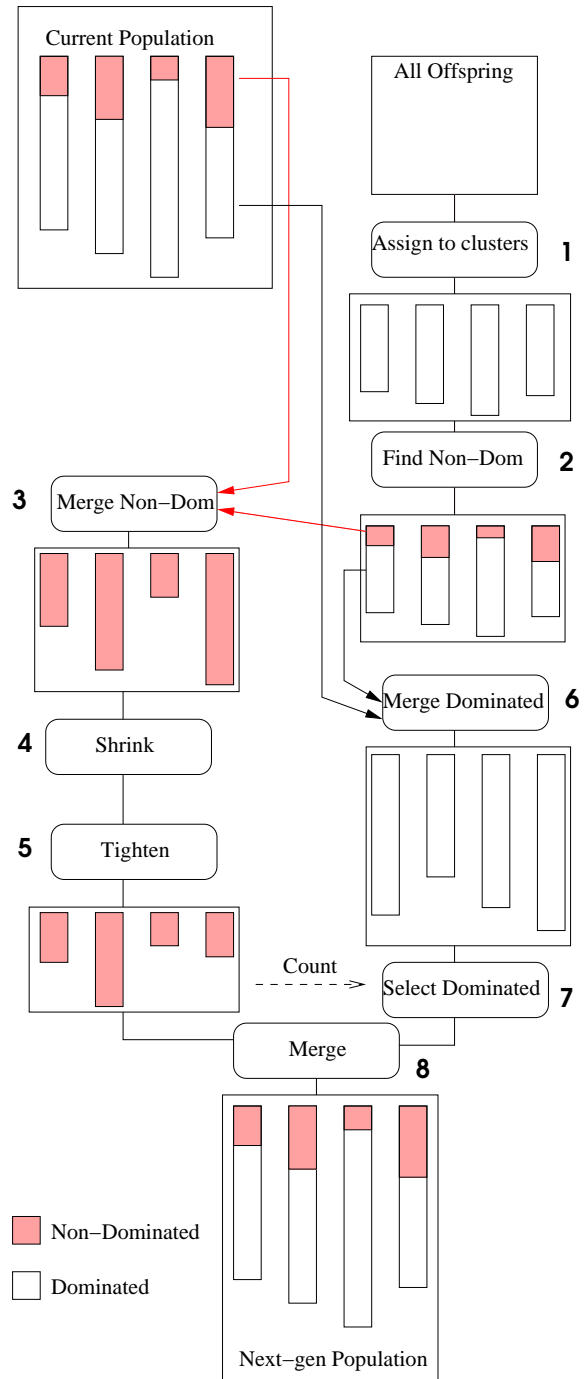


Figure 3: Flowchart of the selection of the next generation.

the front and the population from over-growing. Individuals are removed from the Pareto front of a cluster whenever the number of individuals in the front is above a pre-set threshold value (the same for all clusters).

Generally speaking, individuals should be removed whenever many very similar individuals can be found, because there is little incentive to keep very similar individuals. To achieve this, all individuals are 'paired': the two individuals which have the smallest genotypic distance are paired, then the two individuals with the next smallest distance are paired, etc. Each individual is allowed to be in only one such pair. Within each pair, one individual is removed. We can repeat this process until sufficient number of individuals have been removed. The number of individuals to be removed is a pre-set parameter in our system.

The decision about which individual of a pair to remove is based on the combined fitness of an individual. The better individual survives. The best individual in a cluster (in terms of combined fitness), and therefore also the best individual in the population, will never be removed from the population in the shrinking operation.

5. *Tightening Constraints* Using Pareto selection, individuals with poor fitness values can still survive as long as they are not dominated by any other individuals. For example, the final population can easily contain 'filters' that do not allow any signal to pass, because they maximise the signal suppression in the stopband and are very different from other filters in the same population. These extreme individuals may make up a large fraction of the individuals in the Pareto front and squeeze out promising individuals in the 'compromise' region of the front.

It is useful to have some kind of constraints for the individuals to restrict the number of such extreme individuals in a Pareto front. Unfortunately, setting these constraints is difficult. A harsh constraint will limit the number of extreme individuals, but early in a run only few, if any, individuals would be able to fulfill the constraints. To get around this problem, we have used a self-adaptive mechanism that adapts a constraint vector dynamically to the currently achieved fitness values in the population.

To concentrate evolutionary search on 'compromise' individuals, we use a 'Current Constraint Vector' (CCV). This vector has as many dimensions as there are fitness values, and contains the constraints currently applied to the fitness values. Individuals that violate any of these constraints will not be considered as non-dominated in the selection operations. They will not become part of the Pareto front, and will instead have to compete with the dominated individuals for the remaining places in the population.

To implement the self-adaptation, we initialise the CCV with the worst possible fitness values (positive infinity in our case of minimisation). We also define a second vector, the 'final constraints vector' (FCV). The values in it are pre-defined. They are the constraints that are applied at the end of a run. Every k generations, we compute a new (CCV) from the current CCV, the FCV, and the fitness values of the current population, using the following algorithm.

- (a) Find the worst individual for each of the fitness criteria.
- (b) For each of the fitness criteria, calculate the quotient of the value in the FCV to the current worst value in the population. The criterion that has the largest quotient is selected for tightening.
- (c) Sort all non-dominated individuals according to the criterion selected in the previous step.
- (d) Identify the worst m individuals and mark them for deletion (m is a pre-defined parameter, e.g. 0.5% of the population size).
- (e) Update the CCV by setting the value for the selected criterion to that achieved by the worst remaining individual.

The values in the CCV will shrink each time the above algorithm is run. The speed of shrinking depends on the progress of the evolution. Once a value in the CCV reaches that in the FCV, it will not be reduced further.

It is possible and has been observed in our experiments that the individual with best combined fitness may be removed from the population. This happens when this individual has extremely good values for some of its fitnesses, but also at least

one very bad fitness value. How likely this happens depends mainly on the number of individuals removed during each tightening. The speed of tightening is important. Fast tightening can improve the initial progress of evolution a lot, but may remove very promising individuals. Because promising individuals are generally compromises among different criteria, they are unlikely to be the best according to any single criterion and thus could be removed. In practice, it is sufficient to remove only a few individuals.

Another possible effect of constraint tightening is the removal of all individuals in a cluster especially when all individuals in that cluster have ‘extreme’ fitness values. When this happens, our algorithm will automatically increase the allowed size of the Pareto front of the remaining clusters. This is very useful because the algorithm can concentrate on the remaining clusters. In some sense, a limited degree of competition is introduced among clusters, poor clusters will be driven to extinction.

6. *Merge dominated sets.* All individuals from offspring and current population that are marked as ‘dominated’ are accumulated on a per-cluster basis.
7. *Selection between dominated individuals.* After we have considered all the non-dominated individuals for the next generation, any vacant places in the next generation will be filled up by dominated individuals. To decide which individuals survive in the population, we first divide the number of remaining places by the number of clusters. For each cluster, we then sort the individuals using niche count. The individuals with the smallest niche count (see Section 4.2.1) survive into the next generation. Other sorting criteria, like a weighted fitness, a combination of weighted fitness and niche count, or Pareto based selection schemes, are possible.
8. *Generate new population.* Surviving non-dominated and dominated individuals are combined into a new population.

4.2 Per-Cluster Parent Selection and Self-adaptive Fitness Sharing

As well as the selection for the next generation, we have also implemented versions of fitness sharing and parent selection that are optimised for a clustered population.

4.2.1 Self-adaptive Fitness Sharing

Analysing the runs using fitness sharing and clustering, we noted that within each cluster, the population was quickly converging towards a small area of search space. Because each cluster only represents a fraction of the overall population, the number of individuals in the cluster is small, and convergence is therefore more likely to occur prematurely.

To counteract this, we use a niche-count based fitness sharing **ref to horn etc.**. The niche count produces a measure of how many other individuals ‘inhabit’ the same niche in the population. To do this, distances are computed between individuals and individuals already in the population. This distance is based on a comparison of the genotypes, by computing the geometric distance of the locations of the corresponding poles and zeroes. Since distance calculation is computationally expensive, each individual is only compared to a small random sample of individuals in the population in our EA. This is somewhat similar to implicit fitness sharing [42]. If the distance between two individuals as calculated above is below a threshold (i.e., the sharing radius), the *niche count* of the individual is increased by a value inversely proportional to the distance. The niche count will be normalised by the number of samples this individual has been compared with.

This niche count is used in selection of both parents and next generation (see Section 4.1.1), individuals with higher niche counts are less likely to become offspring and to survive into the next generation.

An important parameter in niche-based fitness sharing is the niche radius. This radius defines the size of the niche, only individuals with a distance less than the niche radius are counted in the niche count. The size of the niche radius is therefore critical for the success of niche based fitness sharing. If the value is set too low, very few individuals will be defined as ‘sharing a niche’; the niche count therefore provides little information about the distribution of the individuals. Similarly, with a niche size too high, most of the individuals would share one or a few niches, again the niche count provides little information about the individual distribu-

tion.

The optimal niche radius depends on the distance measure employed, and on the actual distribution of individuals in the search-space. During any particular run, this distribution will change, as the population moves from the random distribution of the initial population towards areas of the search space with good fitness. This in itself is not a problem, as the premature convergence only becomes a concern once the population has concentrated on a few areas. The niche radius is usually set so that it is optimal for this phase of the evolutionary run.

In our cluster based implementation, each cluster contains only one or a few major fitness peaks, and the size of these peaks in the fitness landscape varies from cluster to cluster. As a result, the distribution of the individuals in each cluster varies. The optimal niche size needs to vary too. In order to have different niche sizes for different clusters, we have implemented a new self-adaptive per-cluster fitness sharing algorithm.

In this algorithm, each cluster uses a different niche size. After the creation of new individuals, the new individuals are evaluated. For each individual, the niche count is then computed by comparing it to a number of individuals from the population. The individuals are only compared to other individuals in the same cluster, using the cluster-specific niche radius. The niche count has to be recalculated for all individuals in the population, not just for the new individuals, because the new individuals will have changed the overall distribution of the population. While computing the new shared fitness, the algorithm also computes the average distance between individuals in each cluster. From these values, the new niche radius is computed, simply as a fraction (for example 50%) of the average distance.

4.2.2 Per-cluster Selection

Observing individual runs, it was apparent that some clusters with a large number of high-fitness individuals often produce a large fraction of the offspring than other clusters in the next generation. By having more offspring, these clusters would then also have a higher chance of improving the fitness of their individuals, leading to a positive feedback situation where some clusters would eventually be starved of offspring. With a population-wide niche size, the fitness sharing would partially counteract this process, but with a per-cluster niche computation, different measures are necessary.

For this reason, parent selection has been modified

to be performed on a per-cluster basis. Each non-empty cluster receives an equal share of the number of parents. These parents are selected using the per-cluster niche counts. As described in Section 4.1, each run generally has an exploration and an exploitation phases. During the exploration phase, crossover between individuals from different clusters is allowed. Per-cluster parent selection produces a list of parents for each cluster. To allow for cross-cluster genetic operations, the parents from the individual clusters are randomly merged into a single list before being passed to the genetic operators.

4.2.3 Computational Cost

Re-clustering a whole population using the k-means algorithm is computationally expensive and cannot easily be distributed over multiple processors. However, it is only required in the early exploration phase of the algorithm where the evolutionary run discovers promising regions, and is only run a few times during this time. Fitness evaluations require a large number of floating point operations and are therefore expensive, compared to this the additional complexity of the selection mechanism does not add a significant amount of computational cost. The other computationally expensive part of the algorithm is fitness sharing, because it requires distance computation between fairly long (e.g. 13 points in polar space) genotypes. However, the clustering reduces this a lot, as only distance measures to individuals in the same cluster are required. Since in our algorithm nearly all operations proceed on a per-cluster basis, it is also possible to distribute the algorithm by associating each cluster to a separate process.

4.3 Other Implementation Details

This section will give an overview of our implementation of the remaining parts of the evolutionary algorithm. More detailed descriptions can be found in a previous publication ([?]).

4.3.1 Chromosome Representation

As described in Section 3.1, the transfer function is generally given in one of the two forms: a polynomial or a pole-zero description of the filter. Because of the direct relationship between the transfer function and frequency response, poles and zeroes in the pole-zero form of the transfer function (Equation 2) can be directly interpreted: a pole near the current frequency amplifies

the signal, a zero attenuates it. Since poles and zeroes are complex numbers, their locations in the complex plane can be naturally expressed in polar coordinates. Under such coordinates, the angle directly specifies the frequency at which the pole or zero is active, and the distance from the origin indicates its strength. Our study [30] has shown the effectiveness of such a polar representation for certain class of real-valued problems. The detailed results were reported earlier in [30].

In short, a polar coordinate based representation of poles and zeroes has the following advantages.

- It can represent all linear IIR filters.
- a suitable genotype-phenotype transformation can automatically add the conjugate opposite to complex poles and zeroes, thereby ensuring that the phenotype will always be realizable.
- Stability can be guaranteed by restricting the radius of poles to $[0.0 \dots 1.0]$.
- Locality is preserved. That is, similar genotypes will have similar frequency responses.
- The search space is relatively smooth since small changes in a genotype will cause small changes in the frequency response and therefore in the fitness of the genotype in most cases.

The transfer function of a filter is represented by a sequence of paired real-value numbers, where each pair indicates the angle and radius of the polar coordinates of a pole or zero. An additional pair of real-valued numbers encode the scaling parameter b_0 . Each pair of real-valued numbers is called a gene in our study.

4.3.2 Evolutionary Search Operators

We have proposed both crossover and mutation operators for our chromosome representation based on our previous work on evolutionary programming [20, 21, 43]. The effectiveness of the operators has been shown both analytically and experimentally elsewhere [20, 30].

Crossover Both uniform and two-point crossover have been implemented, with the later used in most experiments. As radius-angle pairs are closely coupled, it does not seem to make sense to allow crossover to separate them. Crossover points are therefore limited to be between pairs only. In other words, two parents can only be crossed over between genes, not within a gene.

Mutation For mutation, we use an operator built on Cauchy mutation [20, 21]. We decide on a random basis which genes in a genotype are mutated. For each gene that is mutated, a random value is created using a Cauchy distribution with a given strategy parameter η [20, 21], and added to the value of the gene. Because radius and angle have different ranges, different strategy parameters are used.

4.3.3 Fitness Computation

Fitness evaluation is a challenging issue in design, because a design task is usually multi-objective and because it is often difficult to quantify the quality of a design. Fitness evaluation is done in three steps in our work.

Genotype-Phenotype transformation First, a phenotype is computed from the genotype. This mainly involves translating the polar representation of poles and zeroes into Cartesian representation, and application of scaling.

Generating the Frequency Response The frequency response is derived by sampling the transfer function at regular intervals. The filter response for a single frequency can be calculated easily by computing $H(z)$ with $z = e^{j\omega}$.

Computation of Fitness Values We compute a number of fitness values from the frequency response. Fitness values are computed separately for passband (including transition band), and stopband. We compute all the performance measures used by the human designers (Section 3.2); plus a number of additional values. Not all values are used in all runs, the values used in the runs are:

1. *Passband amplitude deviation*: maximum deviation from constant attenuation in dB.
2. *Passband amplitude constraint violation*: accumulated 'punishment' value for all samples where the amplitude is outside the allowed band, with a power of 4 increase with increasing violation.
3. *Passband delay deviation*: maximum deviation from goal group delay.

4. *Passband delay constraint violation*: accumulated 'punishment' value for all samples where the amplitude is outside the allowed band, with a power of 8 increase with increasing violation.
5. *Stopband amplitude deviation*: maximum amplitude in dB.
6. *Passband amplitude constraint violation*: accumulated 'punishment' value for all samples where the amplitude is outside the allowed band, with a power of 4 increase with increasing violation.

There are some special cases to be taken into account when implementing these, especially to ensure that the conversion into dB does not produce a negative fitness value for amplitudes larger than 1.0 in fitness 5.

Combined Fitness From the fitness values computed, we create a single combined fitness value as a weighted average. We have experimented with different selection mechanisms, some of which use this value; the ones described and used in the result presented here only use the combined fitness value at a single point: to decide which of two individuals to delete when shrinking the Pareto front (Section 4.1.1). The combined fitness is also used to generate a measure of performance of the evolutionary process during runs.

5 Experimental Results

Extensive experimental studies have been carried out using different evolutionary algorithms (as described in previous sections) and parameter settings. To save space, we only report those results that we think are interesting in this section. The results produced by our evolutionary system will be compared against those generated by the human expert [27] in order to evaluate the quality of evolved filter designs and gauge the strength and weakness of our evolutionary system. However, the goal is not to beat the human expert with more than a decade experience in filter design, but to evaluate what our evolutionary system can and cannot do at the moment.

In all the results given below, 300 samples have been used in the passband and 200 in the stopband. To conduct a fair comparison, fitness values have been computed for the designs given by the human expert [27] using exactly the same sampling and fitness computation methods as those used in our evolutionary system.

Because of sampling and rounding, the computed fitness values for the filters are similar to but not exactly the same as those reported by the human expert [27].

5.1 Two Test Problems

Both test problems used in our experiments are lowpass filters [27] with slightly different numbers of poles and zeroes, cutoff frequencies, and goals for delays and amplitude. The two problem cases are defined as follows.

Problem Case 1: $\omega_p = 0.2$, $\omega_a = 0.28$, maximum amplitude deviation 0.1dB, minimum stopband attenuation 43dB, group delay = 11 samples with maximum deviation 0.35, order 15 with 7 zero pairs, 1 single zero, 2 pole pairs, 1 pole single, 10 poles at the origin.

Problem Case 2: $\omega_p = 0.25$, $\omega_a = 0.3$, maximum amplitude deviation 0.3dB, minimum stopband attenuation 32dB, group delay = 9 samples with maximum deviation 0.5, order 12 with 6 zero pairs, no single zeros, 5 pole pairs, 1 pole single, 1 poles at the origin.

Genotypes representing individuals require 12 pairs of numbers for Case 1 and 13 pairs of numbers for Case 2. When computing the fitness for the human design in Case 1, it was noted that the amplitude curve seemed to be slightly too high. When the value for b_0 was modified from -0.00046047 as given in the paper [27] to -0.000456475, the fitness value becomes very similar to that given in the paper [27]. We think there is a typo or genuine mistake in the published paper [27]. We will use the corrected value in all our performance comparisons in this report.

5.2 Performance of Our Evolutionary System

Because the design problems are multi-objective, there is generally no single best design that has the best fitness values according to all objectives. Instead, there will be a set of good designs (at the Pareto front) as a result of evolution.

To demonstrate the range and quality of designs evolved by our system, we have picked a number of individuals from the final population by taking the individual with the lowest weighted sum over the fitness values, using a number of different weight sets. Table 1 shows these designs for Case 1. For comparison, the

first row in the table indicates the constraints for the design, and the second row shows the performance of the filter created by the human expert [27]. Apart from the fitness values, the table also shows of which cluster in the population this individual is a member.

Several observations can be made immediately from Table 1. Firstly, the design by the human expert actually violates the constraints slightly. This is likely to be due to sampling differences between our fitness calculation and the calculation used in [27]. The fitness values reported there differ slightly from ours and are within the constraints. In any case, the design by the human expert is very close to the constraints in both passband and stopband amplitude objectives.

Secondly, a number of designs evolved by our system satisfy all constraints, i.e., those selected by the weight sets 1, 4, 5, 6 and 7. Four of these five designs are actually better than the design by the human expert according to all criteria.

Thirdly, our evolutionary system provide not just a single solution as a human designer would do, but a whole population of designs that differ in their emphasis on different criteria. For example, the designs in Table 1 range between 0.04 dB and 0.134 dB in passband amplitude deviation, between 0.288 and 0.44 samples in deviation from constant group delay, and between 49 dB and 52 dB in stopband attenuation. Such diversity represents different trade-offs in the designs, which satisfy all constraints, and gives the user of our evolutionary system to compare and choose the one that fits his/her situation best.

A more detailed performance comparison between the expert design and one evolved design is shown in Figure 4 and Figure 5. As can be seen, the evolved design achieves noticeably better suppression in the stopband (Figure 4(a)) and performs similarly to the expert design in terms of passband amplitude and passband delay variations. The evolved design is slightly better in terms of passband amplitude (Figure 4(b)), but slightly worse in terms of passband delay variation (Figure 5(b)).

Table 2 shows some designs from one run of our evolutionary system for Case 2. In this case, the design by the human expert does satisfy all three constraints, as do some evolved designs. Although there is no evolved design that is better than the design by the human expert according to all three objectives, some evolved designs are very similar in performance to the expert design.

The run presented in Table 1 was run for 35000 generations. The run presented in Table 2 was run for

45000 generations. Both used a population size of 8000 and took 2-3 days in real clock time on a single Athlon PC.

5.3 Examples of Evolved Design

For comparison purposes, Figures 6 and 7 show the two filters designed by the human expert [27]. The left of each figure shows the filter response. The top curve is the amplitude. The lower curve indicates the group delay of the filter. The vertical lines indicate $\omega = \omega_p, \omega = 0.25, \omega = \omega_a$; for case 2 $\omega_p = 0.25$ therefore only two lines appear. Both amplitude and group delay curves use linear scales. On the right hand side of the figures, the poles and zeroes of the transfer function are shown. Poles are indicated by crosses and zeroes by circles.

Figures 8 and 9 show two evolved designs from Table 1 for Problem Case 1. Figures 10 to 11 show two evolved designs from Table 2 for Problem Case 2.

The pole-zero diagrams can be compared quite easily to examine the differences between different designs. The evolved designs have noticeably different pole-zero positions from those in the human design. It is worth noting that all the evolved designs presented here were from a single run of our evolutionary system. The evolutionary approach enables us to obtain a population of candidate designs, not just one.

The ability of our evolutionary system in discovering different novel designs can also be seen from the evolved filters for Problem Case 2, where both evolved designs (Figures 10 and 11) are fairly different from the design by the human expert and also different from each other.

5.4 Per-cluster fitness sharing

In order to show the effect of the per-cluster self-adaptive niche radius, we have collected data on the share radii used during a number of runs. Figure 12 shows the share radii of 20 clusters at different stages of a typical evolutionary run. It shows two important properties we are after. First, the niche size for a cluster changes as evolution progresses; and second, they develop differently for different clusters.

<i>Weightset / Cluster</i>	<i>Passband Amplitude Deviation</i>	<i>Passband Delay Variation</i>	<i>Stopband Attenuation</i>
Constraints	<0.1 dB	<0.35 samples	>43 dB
Human	0.103 dB	0.293 samples	42.98 dB
(weightset 1) cluster 6	(10.0) 0.085 dB	(10.0) 0.289 samples	(10.0) 49.75 dB
(weightset 2) cluster 3	(100.0) 0.048 dB	(10.0) 0.409 samples	(10.0) 52.63 dB
(weightset 3) cluster 3	(2000.0) 0.048 dB	(10.0) 0.409 samples	(10.0) 52.63 dB
(weightset 4) cluster 6	(10.0) 0.087 dB	(100.0) 0.2884 samples	(10.0) 50.00 dB
(weightset 5) cluster 6	(10.0) 0.092 dB	(2000.0) 0.2881 samples	(10.0) 49.75 dB
(weightset 6) cluster 6	(10.0) 0.085 dB	(10.0) 0.289 samples	(100.0) 49.75 dB
(weightset 7) cluster 1	(10.0) 0.073 dB	(10.0) 0.350 samples	(2000.0) 50.76 dB
(weightset 8) cluster 11	(10.0) 0.134 dB	(10.0) 0.441 samples	(20000.0) 52.798 dB

Table 1: Some results from a single run for Problem Case 1. Weights used for selection are indicated in brackets.

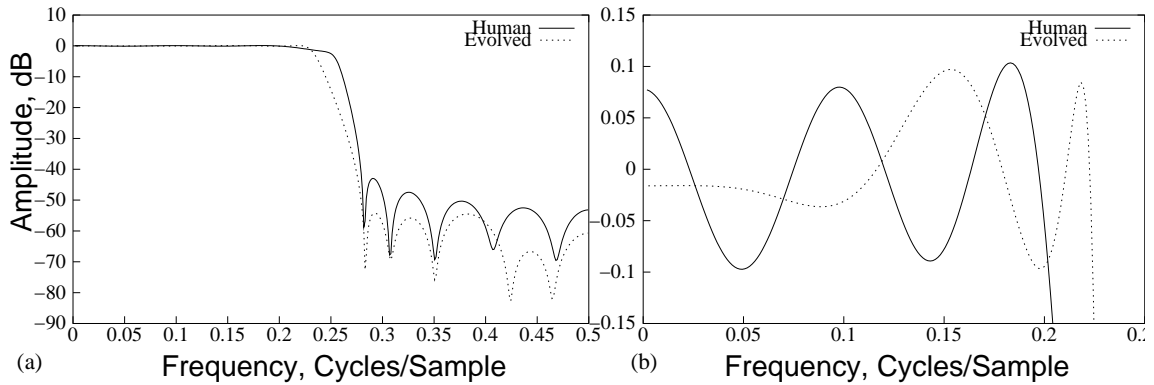


Figure 4: Comparison of amplitudes between the design by the human expert [27] and an evolved design (selected using weightset 7 for problem case 1); (a) full view, (b) detail view of the passband behaviour.

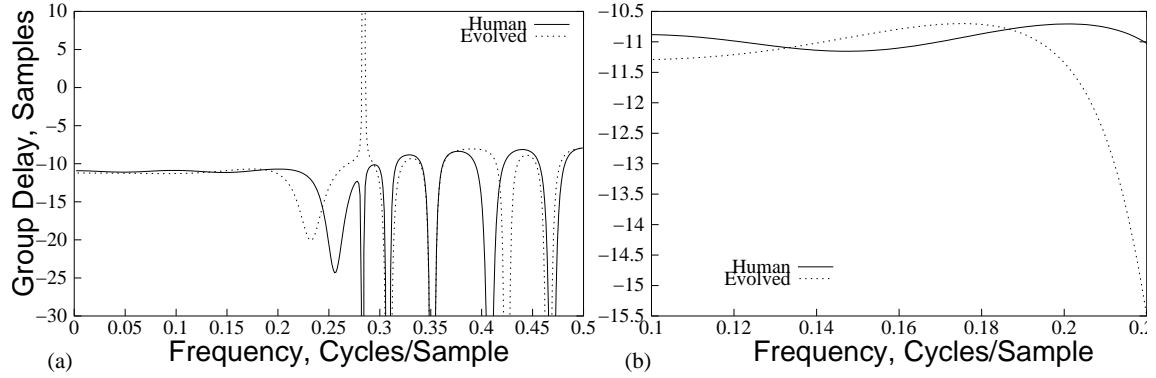


Figure 5: Comparison of delays between the design by the human expert [27] and an evolved design (selected using weightset 7 for problem case 1); (a) full view, (b) detail view of the passband behaviour.

<i>Weightset / Cluster</i>	<i>Passband Amplitude Deviation</i>	<i>Passband Delay Variation</i>	<i>Stopband Attenuation</i>
Constraints	<0.30 dB	<0.50 samples	>32 dB
Human	0.2708 dB	0.437 samples	33.11 dB
(weightset 1) cluster 8	(10.0) 0.267 dB	(10.0) 0.495 samples	(10.0) 35.58 dB
(weightset 2) cluster 8	(100.0) 0.267 dB	(10.0) 0.495 samples	(10.0) 35.58 dB
(weightset 3) cluster 17	(2000.0) 0.189 dB	(10.0) 1.544 samples	(10.0) 18.53 dB
(weightset 4) cluster 8	(10.0) 0.284 dB	(100.0) 0.491 samples	(10.0) 35.58 dB
(weightset 5) cluster 24	(10.0) 0.521 dB	(2000.0) 0.423 samples	(10.0) 33.44 dB
(weightset 6) cluster 8	(10.0) 0.267 dB	(10.0) 0.495 samples	(100.0) 35.58 dB
(weightset 7) cluster 10	(10.0) 0.305 dB	(10.0) 0.548 samples	(2000.0) 38.79 dB
(weightset 8) cluster 10	(10.0) 0.305 dB	(10.0) 0.548 samples	(20000.0) 38.79 dB

Table 2: Some evolved designs for Problem Case 2. Weights used for selection are indicated in brackets.

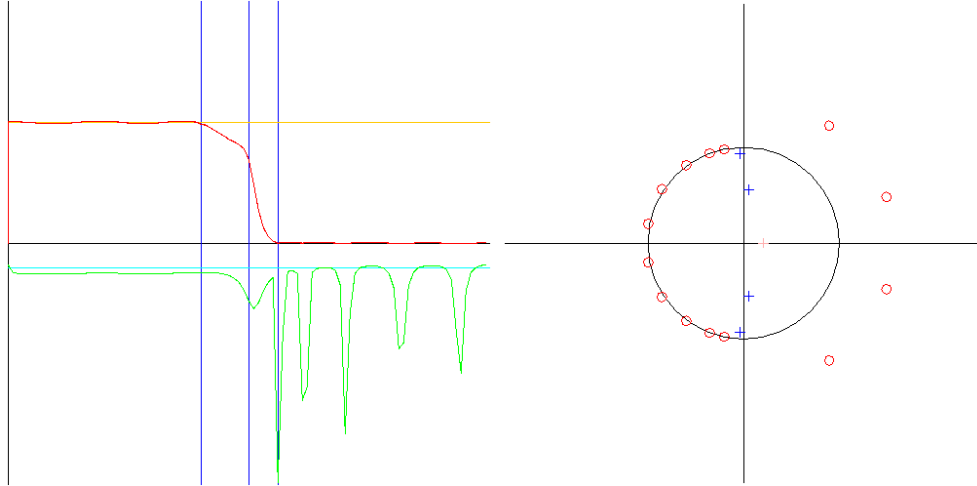


Figure 6: The filter designed by the human expert for Problem Case 1 [27]. The left part shows the filter response, where the top curve indicates the amplitude and the lower one the group delay. The three vertical lines indicate $\omega = \omega_p, \omega = 0.25, \omega = \omega_a$. Both amplitude and group delay curves use linear scales. The right part of the figure shows the poles and zeroes of the transfer function, where poles are indicated by crosses and zeroes by circles.

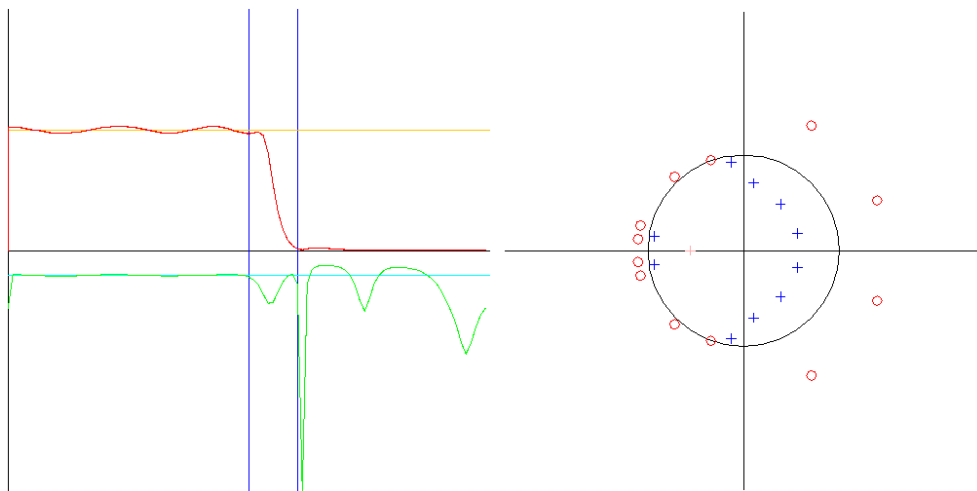


Figure 7: The filter designed by the human expert for Problem Case 2 [27].

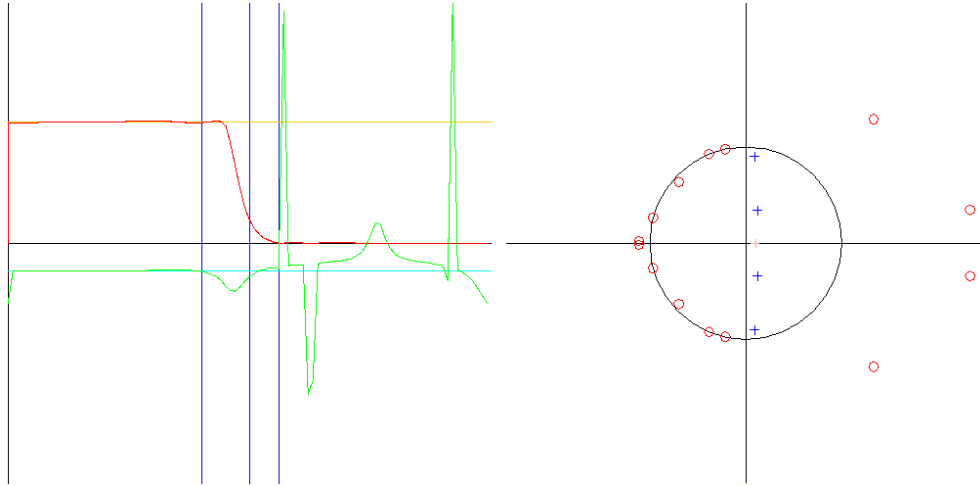


Figure 8: Evolved design for Problem Case 1 based on weight set (1) in Table 1.

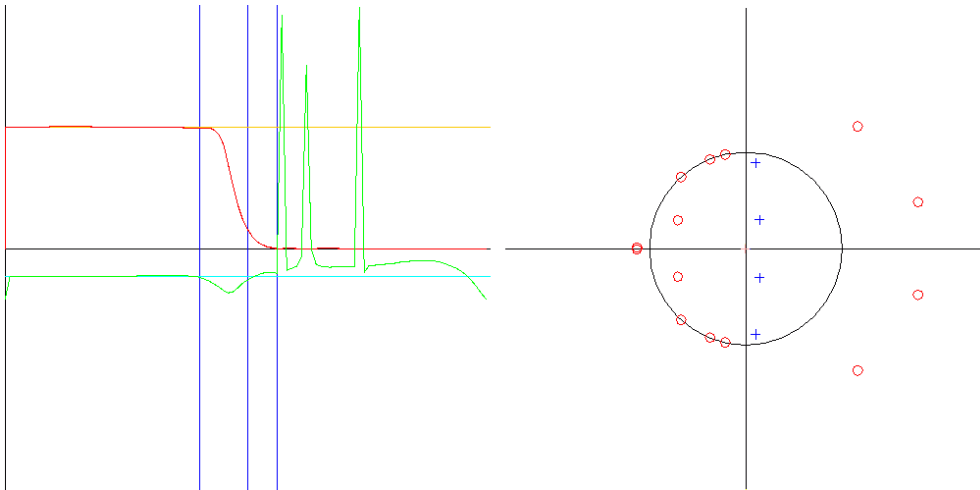


Figure 9: Evolved design for Problem Case 1 based on weight set (2) in Table 1.

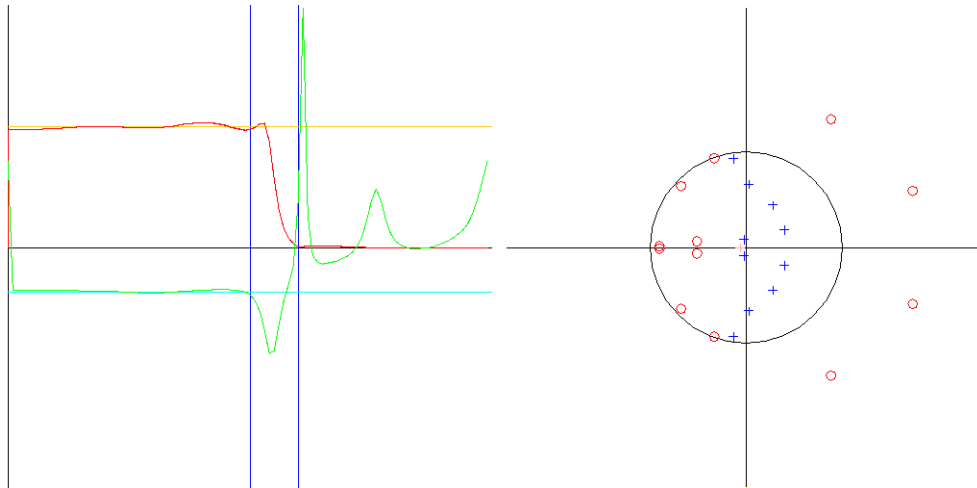


Figure 10: Evolved design for Problem Case 2 based on weight set (1) in Table 2.

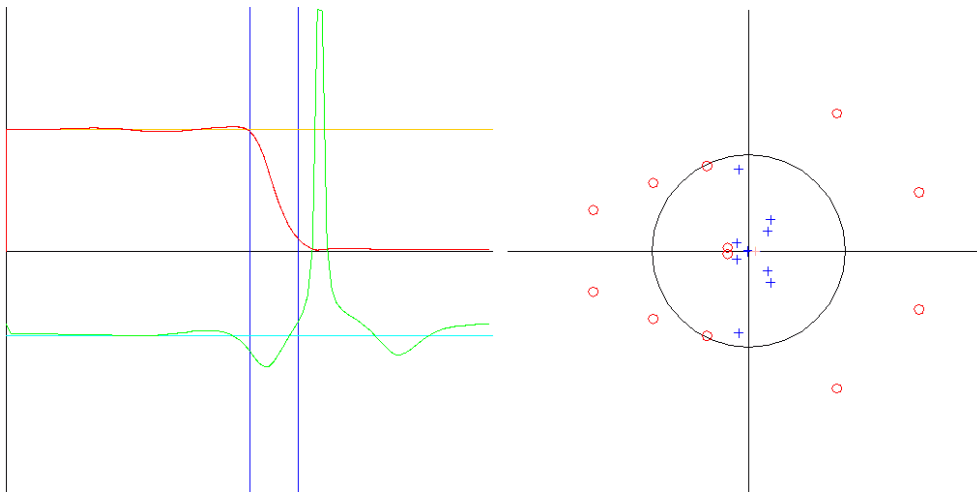


Figure 11: Evolved design for Problem Case 2 based on weight set (3) in Table 2.

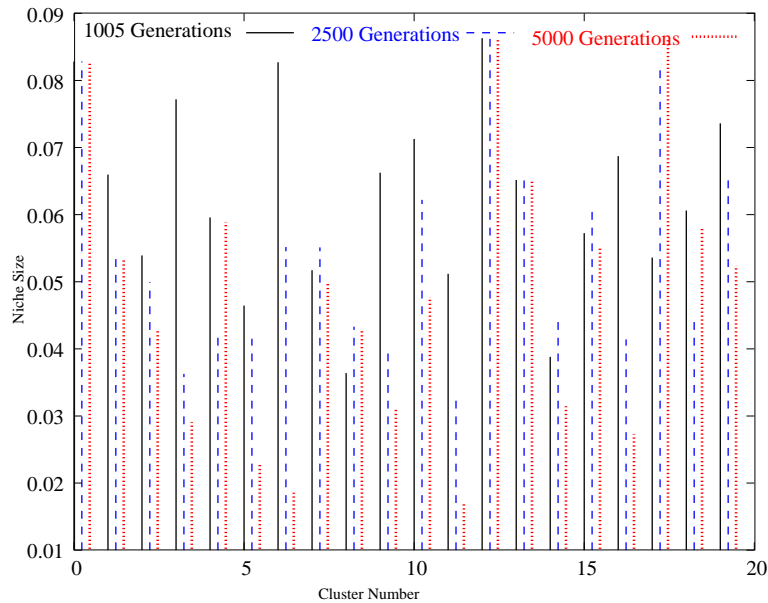


Figure 12: Niche sizes of 20 clusters at the beginning, middle and end of an evolutionary run.

6 Conclusions

The work presented here has shown that Pareto optimisation combined with clustering can be used to produce a variety of results that otherwise would be difficult to achieve. Clustering allows the process to explore different regions in the search space without direct competition, while Pareto optimisation with fitness sharing allows for an efficient, multi-objective exploitation of the different search spaces. The self-adaptive constraint mechanism that we introduced allows the algorithm to exclude unacceptable designs from the search space, without restricting the search process any more than necessary.

The design results show that it is possible to create a variety of results that are competitive with a conventionally derived design, but allow the user to choose between designs with different characteristics.

ACKNOWLEDGEMENT — This research is generously supported by a grant from Marconi Communications, Ltd. The leadership and support of John Evans (from Marconi) is gratefully acknowledged.

References

- [1] M. P. Fourman, "Compaction of symbolic layout using genetic algorithms," in *Proc. of the First Int'l Conf. on Genetic Algorithms and Their Applications* (J. J. Grefenstette, ed.), pp. 141–153, Carnegie-Mellon University, 1985.
- [2] J. Cohoon and W. Paris, "Genetic placement," in *Proc. of Int'l Conf. on CAD*, pp. 422–425, IEEE Press, New York, NY 10017-2394, 1986.
- [3] R. M. Kling and P. Banerjee, "ESP: Placement by simulated evolution," *IEEE Trans. on CAD*, vol. CAD-8, pp. 245–256, 1989.
- [4] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 694–713, 1997.
- [5] X. Yao and Y. Liu, "Towards designing artificial neural networks by evolution," *Applied Mathematics and Computation*, vol. 91, no. 1, pp. 83–90, 1998.
- [6] X. Yao and Y. Liu, "Evolving artificial neural networks through evolutionary programming," in *Evolutionary Programming V: Proc. of the Fifth Annual Conference on Evolutionary Programming* (L. J. Fogel, P. J. Angeline, and T. Bäck, eds.), (Cambridge, MA), pp. 257–266, The MIT Press, 1996.
- [7] X. Yao and Y. Liu, "Evolutionary artificial neural networks that learn and generalise well," in *1996 IEEE International Conference on Neural Networks, Washington, DC, USA, Volume on Plenary, Panel and Special*

- Sessions, pp. 159–164, IEEE Press, New York, NY, 3-6 June 1996.
- [8] Y. Liu and X. Yao, “A population-based learning algorithm which learns both architectures and weights of neural networks,” *Chinese Journal of Advanced Software Research (Allerton Press, Inc., New York, NY 10011)*, vol. 3, no. 1, pp. 54–65, 1996.
- [9] M. A. Rosenman, “An evolutionary model for non-routine design,” in *Proc. of the Eighth Australian Joint Conference on Artificial Intelligence (AI’95)* (X. Yao, ed.), pp. 363–370, World Scientific Publ. Co., Singapore, 1995.
- [10] J. He and X. Yao, “Drift analysis and average time complexity of evolutionary algorithms,” *Artificial Intelligence*, vol. 127, pp. 57–85, March 2001.
- [11] X. Yao and Y. Liu, “Scaling up evolutionary programming algorithms,” in *Evolutionary Programming VII: Proc. of the 7th Annual Conference on Evolutionary Programming* (V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, eds.), vol. 1447 of *Lecture Notes in Computer Science*, (Berlin), pp. 103–112, Springer-Verlag, 1998.
- [12] A. Antoniou, *Digital Filters - Analysis, Design and Applications*. McGraw Hill International, 2 ed., 193.
- [13] T.W.Parks and C.S.Burrus, *Digital Filter Design*. Topics in Digital Signal Processing, John Wiley & Sons, 1987.
- [14] X. Yao, “Following the path of evolvable hardware,” *Communications of the ACM*, vol. 42, no. 4, pp. 47–49, 1999.
- [15] X. Yao and T. Higuchi, “Promises and challenges of evolvable hardware,” *IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 29, no. 1, pp. 87–97, 1999.
- [16] H.-P. Schwefel, *Evolution and Optimum Seeking*. New York: John Wiley & Sons, 1995.
- [17] D. B. Fogel, *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*. New York, NY: IEEE Press, 1995.
- [18] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.
- [19] X. Yao, ed., *Evolutionary Computation: Theory and Applications*. Singapore: World Scientific Publishing Co., 1999.
- [20] X. Yao, Y. Liu, and G. Lin, “Evolutionary programming made faster,” *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 82–102, July 1999.
- [21] X. Yao and Y. Liu, “Fast evolution strategies,” *Control and Cybernetics*, vol. 26, no. 3, pp. 467–496, 1997.
- [22] K.-H. Liang, X. Yao, C. Newton, and D. Hoffman, “A new evolutionary approach to cutting stock problems with and without contiguity,” *Computers and Operations Research*, 2001. to appear.
- [23] X. Yao and P. Darwen, “An experimental study of N-person iterated prisoner’s dilemma games,” *Informatika*, vol. 18, pp. 435–450, 1994.
- [24] P. J. Darwen and X. Yao, “Speciation as automatic categorical modularization,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 2, pp. 101–108, 1997.
- [25] D. B. Fogel and P. J. Angeline. Guidelines for a suitable encoding. In T. Bäck, D. B. Vogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, chapter C1.7:1. IOP Publishing and Oxford University Press, 1997. Release 97/1.
- [26] Y. V. Joshi and S. Roy. Design of multiple notch filters based on all-pass filters. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 46(2):134–138, 1999.
- [27] W. S. Lu. Design of stable iir digital filters with equiripple passbands and peak-constrained least-squares stopband. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 46(11):1421–1426, 1999.
- [28] W.-S. Lu and T.-B. Deng. An improved weighted least-squares design for variable fractional delay fir filters. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 46(8):1035–1040, 1999.
- [29] S. Nordebo. Semi-infinite linear programming: A unified approach to digital filter design with time- and frequency-domain specifications. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 46(6):765–775, 1999.
- [30] T. Schnier and X. Yao. Using multiple representations in evolutionary algorithms. In *Congress on Evolutionary Computation CEC 2000*, volume 1, pages 479–487. IEEE Neural Network Council, IEEE, 2000.
- [31] T. Schnier and X. Yao. Evolutionary design calibration. In *The 4th International Conference on Evolvable Systems: From Biology To Hardware (ICES 2001)* Tokyo, Japan, October 2001, pages 26–37. Springer
- [32] I. W. Selesnick. Low-pass filter realizable as all-pass sums: Design via a new flat delay filter. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 46(1):40–49, 1999.
- [33] Statsoft. Cluster analysis, 2000.
- [34] K. Surma-aho and T. Saramäki. A systematic technique for designing approximately linear phase recursive digital filters. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 46(7):956–963, 1999.
- [35] K. Uesaka and M. Kawamata. Synthesis of low coefficient sensitivity digital filters using genetic programming. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, volume 3, pages 307–310, 1999.
- [36] K. Uesaka and M. Kawamata. Synthesis of low-sensitivity second-order digital filters using genetic programming with automatically defined functions. *IEEE Signal Processing Letters*, 7(4):83–85, Apr. 2000.

- [37] P. B. Wilson and M. D. Macleod. Low implementation cost IIR digital filter design using genetic algorithms. In *IEE/IEEE Workshop on Natural Algorithms in Signal Processing*, pages 4/1–4/8, Chelmsford, U.K., 1993.
- [38] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results (revised version). Technical Report TIK-Report 10, Institut für Technische Informatik und Kommunikation-snetze, ETH Zürich, 1999.
- [39] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4) pages 257-271, November 1999.
- [40] J. Miller. Digital filter design at gate-level using evolutionary algorithms. In W. Banzhaf et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, pages 1127–1134, San Francisco, CA, 1999. Morgan Kaufmann.
- [41] P. Darwen and X. Yao, “A dilemma for fitness sharing with a scaling function,” in *Proc. of the 1995 IEEE Int'l Conf. on Evolutionary Computation (ICEC'95)*, Perth, Australia, pp. 166–171, IEEE Press, New York, NY 10017-2394, 1995.
- [42] P. Darwen and X. Yao, “Every niching method has its niche: fitness sharing and implicit sharing compared,” in *Parallel Problem Solving from Nature (PPSN IV)* (H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds.), vol. 1141 of *Lecture Notes in Computer Science*, (Berlin), pp. 398–407, Springer-Verlag, 1996.
- [43] K.-H. Liang, X. Yao, and C. Newton, “Evolutionary search of approximated n -dimensional landscapes,” *International Journal of Knowledge-Based Intelligent Engineering Systems*, vol. 4, pp. 172–183, July 2000.