Chapter 3

Implementation

This chapter will consider the implementation of the algorithm developed during this dissertation. Most of the work involves bringing together a number of the techniques and theories discussed in Chapter 2. The principle of this dissertation is to assume that an impedance matching problem is simply an optimisation problem and to develop an algorithm to solve that problem.

Section 3.1 will consider the approach used during the development of the algorithm. Section 3.2 will then present the genetic algorithm used. The calculation of the fitness function values used by the genetic algorithm is covered in Section 3.3. Lastly, the local optimiser used is presented in Section 3.4.

3.1 Approach

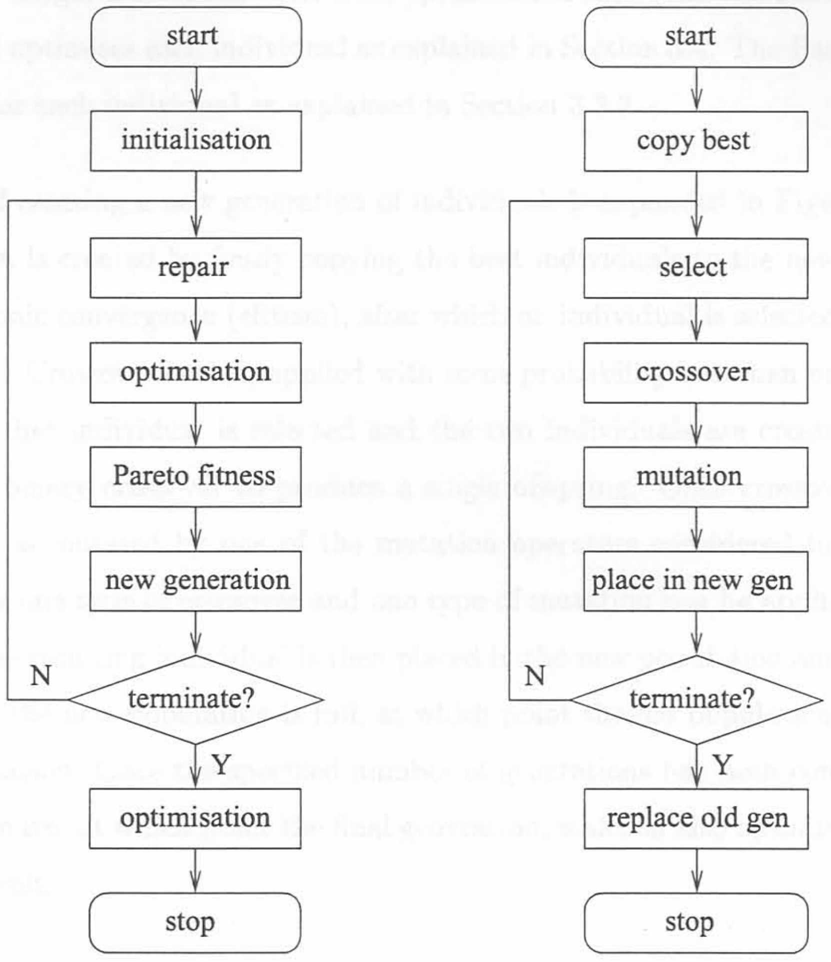
The approach used to develop the algorithm that is the basis of this dissertation is given in this section. This includes how an impedance matching problem can be considered as an optimisation problem, and why a genetic algorithm coupled with a local optimiser was used to solve that problem.

Impedance matching network design is the process of designing a circuit that matches one impedance to another with a specified gain. This can be viewed as an optimisation problem where the objective is to minimise the gain error given the source and load impedances, and the desired gain.

The problem of optimising a circuit for any purpose, including impedance matching, can be considered to consist of combinatorial, global, and local optimisation problems. The combinatorial part of the problem involves the structure of the circuit. This entails deciding between a number of circuit elements including inductors, capacitors and transmission lines, and whether a component should be in series or parallel. The global and local parts of the problem consider the component values. There are a number of local optima in an impedance matching problem, so a global optimisation algorithm is necessary to find the best of these. Once a global optimiser has found a point near the global optimum, a local optimiser is used to rapidly converge to the global optimum.

Of the optimisation algorithms considered in Section 2.1, genetic algorithms and simulated annealing can be used for both combinatorial and global optimisation. Using one of these algorithms would thus eliminate the need to implement separate combinatorial and global optimisation algorithms. Genetic algorithms were chosen ahead of simulated annealing because genetic algorithms use a population of individuals allowing more than one solution to a problem to be obtained. While genetic algorithms are very useful for combinatorial and global optimisation, their local optimisation properties are poor, so the hybrid system proposed by Renders and Flasse [34] was used to speed convergence. The implementation of the local optimiser is discussed further in Section 3.4.

A number of constants and options were implemented in this algorithm. Examples of constants include minimum and maximum component values, and crossover and mutation probabilities. Examples of options include the three different fitness functions discussed in Section 3.3. All the constants and options were implemented as constants that cannot be changed by the program and this has the drawback that all the constants and options must be set before the program is compiled. The major advantage of this approach is



(a) Complete algorithm. (b) New generation creation.

Figure 3.1: Flow charts.

that it results in faster code than the case where constants and options can be changed after compilation because the compiler can apply more optimisations to the code. A large number of tests were run during this dissertation, so program speed was considered more important than the problems involved with re-compiling the program every time a constant is changed.

A flow chart of the algorithm implemented here is given in Figure 3.1(a). Initialisation is the initialisation of the algorithm and is done randomly in this case. The repair stage eliminates any illegal individuals. The local optimisation algorithm calculates fitnesses and gradients, and optimises each individual as explained in Section 3.4. The Pareto-like fitness is calculated for each individual as explained in Section 3.3.3.

The process of creating a new generation of individuals is expanded in Figure 3.1(b). The new generation is created by firstly copying the best individuals to the new generation to ensure monotonic convergence (elitism), after which an individual is selected using tournament selection. Crossover is then applied with some probability less than one. If crossover is applied another individual is selected and the two individuals are crossed using either arithmetic or binary crossover to produce a single offspring. Once crossover is complete the individual is mutated by one of the mutation operators considered in Section 3.2.2. Note that only one type of crossover and one type of mutation can be applied to each new individual. The resulting individual is then placed in the new population and the process is repeated until the new population is full, at which point the old population is replaced by the new population. Once the specified number of generations has been considered the algorithm terminates at which point the final generation, which is also optimised, is returned as the final result.

Each of the steps shown in the flow chart in Figure 3.1 will be considered in more detail in the following sections.

3.2 Genetic Algorithm

The implementation of the genetic algorithm portion of this dissertation will be presented in this section. The initialisation, selection scheme, termination criterion, elitism, representation, and genetic operators that were used are covered.

The usual genetic algorithm practice of randomly initialising the population was used. Other possibilities considered were to initialise the population using a grid search or to seed the population with the results of other algorithms. The populations used are too small to make a grid initialisation practical because, for example, if an individual consists of six lumped elements there are $4^6 = 4096$ possible ways to choose those elements. While some of those combinations are redundant (for example two inductors in series), this type of initialisation is still impractical for reasonable population sizes. Seeding the population with the results of other algorithms would require the implementation of a large number of other algorithms, and while this is possible, it would be too time consuming for this dissertation.

Two options for determining the initial length of individuals were considered. The first option generates individuals with equal probabilities of being any length from one element to the maximum number of elements allowed, but the removal of illegal and redundant elements means that this scheme will actually favour the production of short individuals. The second possibility is to make all individuals as long as possible prior to removal of illegal and redundant elements. This is useful because the algorithm tends to favour the creation of short individuals due to removal of elements, and longer individuals represent more complex solutions which require more samples to arrive at a good result. For this reason, the second option was implemented.

Of the selection schemes considered in Section 2.2.2 tournament selection was used in this dissertation because it is very simple to implement and the bias towards good individuals can be adjusted. The algorithm was terminated after a fixed number of generations. Elitism was implemented as an option that can be set before compilation. When elitism is used the best individual (or individuals in the Pareto case) is copied to the new population unaltered.

The default genetic algorithm parameters are given in Table 3.1.

The representation used is considered in Section 3.2.1 and the genetic operators used are presented in Section 3.2.2.

Table 3.1: Genetic algorithm operator default probabilities.

Genetic Operator	Value
Arithmetic crossover	0.4
Binary crossover	0.3
Boundary mutation	0.03
Uniform mutation	0.02
Non-uniform mutation	0.1
Binary mutation	0.1

3.2.1 Representation

The representation used for this dissertation is a unique combination of the discrete and continuous parts of the problem. Additionally, the representation used for the continuous variables allows both binary and floating point operators to be used – something that has not been achieved before. This section will consider this representation with the emphasis on the new features.

The circuit configuration used was limited to ladder networks. This simplifies the representation because a ladder network can be represented as a string of circuit elements. Each individual thus consists of a number of circuit elements in order from load to source. The maximum number of circuit elements in each individual is set when the program is compiled. A ladder network can only have two elements in parallel at any point (a cross structure), so more than two parallel elements at a point are considered redundant and are eliminated. Similarly, in the purely lumped case, two capacitors or inductors in series or parallel are redundant and the second element is removed. Series elements are not eliminated in the distributed case and when discontinuities are used because there is no series redundancy in these cases. Another approach to dealing with redundant elements would have been to ignore them, but this can lead to difficulties in calculating a network's length and can cause poor combinations of elements to exist in abeyance (present in an individual,

Table 3.2: **Element Representation.**

Circuit element	Bits				
	29	28	27	26–13	12–0
parallel inductor	0	0	0	value (log)	
parallel capacitor	0	0	1	value (log)	
series inductor	0	1	0	value (log)	
series capacitor	0	1	1	value (log)	
parallel shorted stub	1	0	0	width (log)	length (linear)
parallel open stub	1	0	1	width (log)	length (linear)
series transmission line	1	1	0	width (log)	length (linear)
illegal	1	1	1		

but not used in that individual, although it might be used in future generations).

The full representation consists of 30 bits. Most desktop computers today use a 32 bit word length, so the use of 30 bits allows 2 bits for future expansion of the number of element types. The first three of these bits are used to encode the type of element and the remaining 27 bits encode the element's parameter values as shown in Table 3.2.

From Table 3.2 it is clear that bit 27 is 0 for inductive elements and 1 for capacitive elements, bit 28 is 0 for parallel elements and 1 for series elements, and bit 29 is 0 for lumped elements and 1 for distributed elements. The lumped elements only require one variable to represent their value, so they use the full remaining 27 bits for this purpose. Distributed elements require two values for both characteristic impedance (determined by the width in a microstrip system) and length, so 14 bits are used to represent the line width and 13 bits are used to represent the line length. The transmission line lengths are linearly distributed between the minimum and maximum specified values. The values of lumped elements vary logarithmically and the relationship between the width of a microstrip line and its characteristic impedance is approximately logarithmic, so a logarithmic representation was used in these cases. Binary values are distributed linearly, so a transformation was necessary.

The uniformly distributed binary values are converted to logarithmically distributed values using

$$r = y_{\min}\left(\frac{y_{\max}}{y_{\min}}\right)^{x} = y_{\min}e^{\left[x\ln\left(\frac{y_{\max}}{y_{\min}}\right)\right]} \tag{3.1}$$

where r is the representation of the variable y, $y_{\max}$ and $y_{\min}$ are the maximum and minimum values of y, and x is a uniformly distributed variable over the range [0, 1].

The first advantage of this representation is that the value of the variable y is constrained to the range $[y_{\min}, y_{\max}]$ allowing the algorithm to be constrained to realisable parameter ranges. The second advantage is that this allows the use of uniformly distributed variables to represent logarithmic parameters. This means that the value of x in (3.1) can be implemented as a binary string (a linear representation), allowing both the binary and floating-point genetic operators considered in Section 2.2.4 to be used on problems with floating-point variables.

An important consideration at this point is the decimal precision of the representation where values of $y_{\max}$, $y_{\min}$ and the number of bits used to represent x are given. The decimal precision of an arbitrary representation is

$$d = \left\lfloor \log\left(\frac{5}{err}\right)\right\rfloor + \lfloor \log(y)\rfloor \tag{3.2}$$

where

$$err = |y - r|, \tag{3.3}$$

d is the precision in number of decimal digits, and the $\lfloor\cdot\rfloor$ (floor) operator indicates that the argument must be rounded to the greatest integer value less than or equal to the argument. The first term of (3.2) gives the number of decimal digits precision after the decimal point and the second term gives the position of the most significant decimal digit. The value of the representation is given in (3.1), and the correct value of the variable is

$$y = y_{\min}\left(\frac{y_{\max}}{y_{\min}}\right)^{x_e} = y_{\min}e^{\left[x_e\ln\left(\frac{y_{\max}}{y_{\min}}\right)\right]} \tag{3.4}$$

where x_e is the value of x that would make the representation r exactly equal to the variable

y. The representation can be rewritten in terms of x and x_e as shown below.

$$x = \frac{\ln\left(\frac{r}{y_{\min}}\right)}{\ln\left(\frac{y_{\max}}{y_{\min}}\right)} \tag{3.5}$$

$$x_e = \frac{\ln\left(\frac{y}{y_{\min}}\right)}{\ln\left(\frac{y_{\max}}{y_{\min}}\right)} \tag{3.6}$$

The maximum difference between x and x_e is half of the smallest step in x. A binary value with n bits divides a range into $2^n - 1$ steps, so the maximum error is half this value:

$$|x_e - x| = \left|\frac{\ln\left(\frac{y}{y_{\min}}\right)}{\ln\left(\frac{y_{\max}}{y_{\min}}\right)} - \frac{\ln\left(\frac{r}{y_{\min}}\right)}{\ln\left(\frac{y_{\max}}{y_{\min}}\right)}\right| \leq \frac{1}{2\left(2^n-1\right)} . \tag{3.7}$$

This result can now be used to obtain the limits on the value of r as shown below.

$$\left|\ln\left(\frac{y}{y_{\min}}\right) - \ln\left(\frac{r}{y_{\min}}\right)\right| \leq \frac{\ln\left(\frac{y_{\max}}{y_{\min}}\right)}{2\left(2^n-1\right)} \tag{3.8}$$

$$\left|\ln\left(\frac{y}{r}\right)\right| \leq \frac{\ln\left(\frac{y_{\max}}{y_{\min}}\right)}{2\left(2^n-1\right)} \tag{3.9}$$

$$y \exp\left[-\frac{\ln\left(\frac{y_{\max}}{y_{\min}}\right)}{2\left(2^n-1\right)}\right] \leq r \leq y \exp\left[\frac{\ln\left(\frac{y_{\max}}{y_{\min}}\right)}{2\left(2^n-1\right)}\right] \tag{3.10}$$

$$y\left(\frac{y_{\max}}{y_{\min}}\right)^{\frac{-1}{2(2^n-1)}} \leq r \leq y\left(\frac{y_{\max}}{y_{\min}}\right)^{\frac{1}{2(2^n-1)}} \tag{3.11}$$

The values of the range limits do not differ much as can be seen by the fact that the exponents are both very close to zero when n is large. These values of r can now be substituted into (3.3) to give

$$err = \begin{cases} y - r \geq y - y\left(\frac{y_{\max}}{y_{\min}}\right)^{\frac{-1}{2(2^n-1)}} & \text{if } y \geq r \\ r - y \leq y\left(\frac{y_{\max}}{y_{\min}}\right)^{\frac{1}{2(2^n-1)}} - y & \text{if } y < r \end{cases} \tag{3.12}$$

and

$$y\left[1 - \left(\frac{y_{\max}}{y_{\min}}\right)^{\frac{-1}{2(2^n-1)}}\right] \leq err \leq y\left[\left(\frac{y_{\max}}{y_{\min}}\right)^{\frac{1}{2(2^n-1)}} - 1\right] . \tag{3.13}$$

As above, the limits are very similar when n is large, so this result can be approximated by the upper limit:

$$err \approx y \left[\left(\frac{y_{\max}}{y_{\min}} \right)^{\frac{1}{2(2^n-1)}} - 1 \right] . \tag{3.14}$$

This value can now be substituted into (3.2) to obtain the decimal precision of the result:

$$d \approx \left\lfloor \log \left(\frac{5}{y \left[\left(\frac{y_{\max}}{y_{\min}} \right)^{\frac{1}{2(2^n-1)}} - 1 \right]} \right) \right\rfloor + \lfloor \log (y) \rfloor \tag{3.15}$$

where the result is approximate because the value of err is approximate. In (3.15) the precision depends on the value of the variable y. The value of y can be removed by making the following approximation:

$$d \approx \left\lfloor \log \left(\frac{5}{y \left[\left(\frac{y_{\max}}{y_{\min}} \right)^{\frac{1}{2(2^n-1)}} - 1 \right]} \right) + \log (y) \right\rfloor - 1 \tag{3.16}$$

$$\approx \left\lfloor \log \left(\frac{5}{\left[\left(\frac{y_{\max}}{y_{\min}} \right)^{\frac{1}{2(2^n-1)}} - 1 \right]} \right) \right\rfloor - 1 . \tag{3.17}$$

The subtraction of one is necessary because there will be some value of y where the result is greater than the correct value given by (3.15) if n is large enough. This error cannot be greater than one because the sum of the fractional parts of both terms in the floor operator in (3.16) must be less than two.

The component ranges used are given in Table 3.3, where λ is the wavelength of a transmission line and h is the height of a microstrip substrate. The component ranges considered during parameter tuning were intentionally chosen to be very large to give worst-case results when the algorithm was tested in Chapter 4. The transmission line impedances are the minimum and maximum impedances obtained when the effective dielectric constant of a microstrip line can vary from 1 to 10, and the width of the microstrip line can vary from 0.01 to 100 times the substrate height. The number of decimal digits' precision for the logarithmically distributed variables is at least 3 in all cases and the linearly distributed variables have step sizes that are smaller than 1/8000 of the total range, so the accuracy of

Table 3.3: Component Ranges.

Component Type	Minimum Value	Maximum Value	Precision (decimal digits)
Lumped	10^{-15} F or H	10^{-6} F or H	6
Transmission line length	0.01 λ	3 λ	step: $3.650 \times 10^{-6}\lambda$
Transmission line impedance	1 Ω	400 Ω	3
Transmission line length	2 h	100 h	step: $1.196 \times 10^{-2}h$
Transmission line width	0.1 h	10 h	3

Table 3.4: Microstrip substrate parameters.

Problems	Relative dielectric constant (ε_r)	Height (mm)
2, 3, 4, 5	1	3
7, 8	2	0.1
1, 6	5	0.25
9, 10	10	0.5

the representation is seen to be good. The minimum microstrip line length was chosen to ensure that there is a reasonable separation between parallel elements to minimise coupling. A transmission line length of zero was not used because this leads to infinite impedances for parallel open stubs.

The substrates used for the microstrip problems are given in Table 3.4 for the ten test problems considered in Section 4.1 on page 106. The substrate relative dielectric constants were chosen to give a range of typical values, and the substrate heights were chosen to be approximately one hundredth of a wavelength in the substrate at the highest frequency. All elements are assumed to be lossless and the conductor is assumed to be infinitesimally thin. The via diameter was equal to half the substrate height for each problem.

3.2.2 Genetic Operators

The genetic operators used in this dissertation are covered in this section. One of the unique features of this dissertation is that all the operators in Section 2.2.4 except simple crossover are used. A description of how the operators are applied is given below followed by the implementation details of the crossover and mutation operators.

A flow chart of the algorithm is given in Figure 3.1 on page 79. An individual is selected before the genetic operators are applied. Two individuals are required for crossover so a second individual is selected if crossover is applied. Mutation is applied to the individual generated by crossover. Crossover and mutation are applied with some probability less than one so not all individuals are modified by these operators. Individuals that are not modified simply pass through the operator unaltered and are applied to the next stage of the algorithm. While a number of crossover and mutation operators are implemented, only one crossover and one mutation operator can be applied to each individual.

3.2.2.1 Crossover

Binary and arithmetic crossover were implemented and their operation is described below. The two parents used were not allowed to be identical in this dissertation to increase the genetic diversity of the population.

The representation consists of a number of 30 bit elements in each individual. Variable elements were allowed by using a crossover scheme similar to that used in the messy genetic algorithms considered in Section 2.2.6. If the parents are given by

$$\begin{array}{ccc|c|cccc} & & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 \\ y_6 & y_5 & y_4 & y_3 & y_2 & y_1 & y_0 & \end{array} \tag{3.18}$$

where x_i and y_j represent circuit elements and the cross point is marked by $|\cdot|$, then the offspring is given by

$$x_5 \; \big| \; a \; \big| \; y_2 \quad y_1 \quad y_0 \tag{3.19}$$

where a represents some combination of x_4 and y_3. To implement this process a cross point must be chosen for each parent. Care must be taken to ensure that the offspring is not longer than the maximum allowable length by limiting the position of the cross point in the second parent.

An option to always choose the cross point in the second parent so as to make the offspring as long as possible was implemented. This option allows the variable length genetic algorithm to function as a normal fixed length genetic algorithm by making all individuals as long as possible when Pareto optimality is not used. This option also helps to ensure that there are more long individuals than short individuals. This is necessary because longer individuals represent a more difficult problem than short individuals and thus require more processing.

Binary crossover is implemented as above with a being generated by performing binary crossover on the two parents' elements at that position. The cross point is chosen uniformly from before the first bit to after the last bit of the element representation. This allows a to be generated solely by the one parent, eliminating the need for a simple crossover operator. It is possible for binary crossover to generate the illegal elements, so a repair operator that eliminates illegal elements was implemented.

Arithmetic crossover is very similar to binary crossover except that the generation of a is different. The circuit element type of a is the same as the circuit element type of the first parent. The element values are then crossed using (2.29) on page 44. If two transmission lines are crossed, then the length and the width are crossed independently using the same weight (r). In any other case bits 26–0 of the representation are assumed to contain one value and are crossed accordingly. The weight used for crossover can vary from 0 to 1 so a can be generated by only one of the parents as with binary crossover. This again eliminates the need for a simple crossover operator.

3.2.2.2 Mutation

The mutation operators in this case all work in the same basic way. An element in the individual is chosen and is then mutated. As with crossover, only one type of mutation can operate on each individual.

Binary mutation works by inverting bits in an individual. One of the elements of the individual is chosen, and then one of the bits of that element is inverted. Binary mutation can generate illegal individuals, but these are deleted by the repair algorithm.

Non-uniform mutation is described in Section 2.2.4.1. An element of an individual is chosen and then mutated. Both the width and length of a distributed element are mutated independently, but with the same parameters.

Boundary mutation sets an element's value to either its minimum or maximum value with equal probability. In this case it is not necessary to consider lumped and distributed elements separately.

Uniform mutation reinitialises an element of an individual by assigning a random element type and value to that element.

3.3 Fitness Calculation

The calculation of the fitness of each individual will be considered in this section. Lumped, distributed, and mixed lumped and distributed networks are possible. Further a number of discontinuity models were implemented and a type of multi-objective ranking similar to Pareto ranking was used. Each of these points is considered below starting with a discussion of the error functions used.

3.3.1 Error Functions

The calculation of the fitness starts at the load and calculates the input impedance by taking each element into account. The final result of a fitness calculation in this dissertation uses one of four types of error, namely Voltage Standing Wave Ratio (VSWR), Maximum Relative Deviation (MRD), or the gain error in decibels (dB). In each case the maximum value of the error with frequency is used as the fitness value, so this is a minimax problem.

VSWR is a well known error function for measuring the quality of a match, and is calculated from the reflection coefficient:

$$\text{VSWR} = \frac{1+|\Gamma|}{1-|\Gamma|} \tag{3.20}$$

where Γ is the reflection coefficient calculated using

$$\Gamma = \frac{Z_L - Z_0^*}{Z_L + Z_0} \tag{3.21}$$

where Z_0 is the complex normalising impedance, and Z_L is the other impedance at the junction under consideration [3]. The highest VSWR value over the frequency range of interest is used as the fitness. VSWR can only be used where the objective is to perfectly match the load and source impedances because its definition does not include a gain specification. The MRD and decibel errors overcome this limitation by using the transducer power gain (G_T) to determine the error.

The transducer power gain is defined as the ratio of the power delivered to the load to the power available from the source [3]. The magnitude of S_{11} squared is equal to the ratio of the power reflected by the network to the power available from the source [3] when the network is terminated by its normalising impedances. All the power entering a lossless network must emerge at the output, and a passive network cannot add any power to a signal. This means that the transducer gain of a passive, lossless network is the proportion of the power available from the source that is not reflected back to the source, and is given by

$$G_T = 1 - |S_{11}|^2 \tag{3.22}$$

$$= 1 - |\Gamma|^2 \,. \tag{3.23}$$

Both ports are guaranteed to be terminated in their normalising impedances in this algorithm because the source and load impedances of the required design are used as the normalising impedances. The equivalence of $|S_{11}|$ and $|\Gamma|$ can be shown by using the definition of Γ given in (3.21) and the equation for S_{11}:

$$S_{11} = \frac{Z_L - Z_0^*}{Z_L + Z_0} \,. \tag{3.24}$$

The difference between the MRD and decibel errors lies in how they use the transducer gain.

MRD is calculated using

$$\text{MRD} = \left| \frac{G_T(f)}{G_{Topt}(f)} - 1 \right| \tag{3.25}$$

where $G_T(f)$ and $G_{Topt}(f)$ are the actual and desired values of transducer gain at frequency f respectively [2]. The drawback of this error function is that gain values above and below the desired gain are treated differently because of the nonlinearity introduced by the absolute value function. For example values of 0.5 and 1.5 for $G_T(f)/G_{Topt}(f)$ both give a MRD value of 0.5 despite the fact that the first case represents a gain error of 3 dB and the second only 1.76 dB. This nonuniformity means that errors below the desired gain will be better optimised than errors above the desired gain although the difference will decrease when the error is small.

The bias inherent in MRD errors is avoided by using the gain error in decibels. This error is calculated using

$$dB_{err} = \left| 10 \log \left(\frac{G_T(f)}{G_{Topt}(f)} \right) \right| \tag{3.26}$$

where dB_{err} is the gain error in decibels.

Conversions between these error functions will now be derived so that results with different error functions can be compared. The case where $G_T(f)/G_{Topt}(f)$ is greater than one has to be considered separately from the case where it is less than one because of the absolute value functions in the definitions of MRD and gain error.

The derivation of decibel error in terms of MRD proceeds by solving (3.25) in terms of $G_T(f)/G_{T_{opt}}(f)$ and the substituting these into (3.26) to obtain the decibel error in terms of MRD:

$$dB_{err} = \begin{cases} 10\log\left(1+\text{MRD}\right) & \text{if } \frac{G_T(f)}{G_{T_{opt}}(f)} \geq 1 \\ 10\log\left(1-\text{MRD}\right) & \text{if } \frac{G_T(f)}{G_{T_{opt}}(f)} < 1 \end{cases} . \tag{3.27}$$

This result allows conversions from MRD to decibel error.

The conversion from decibel error to MRD also involves two cases. The equation for decibel error (3.26) is solved for $G_T(f)/G_{T_{opt}}(f)$ and the result is substituted into (3.25) to give

$$\text{MRD} = \begin{cases} 10^{\frac{dB_{err}}{10}} - 1 & \text{if } \frac{G_T(f)}{G_{T_{opt}}(f)} \geq 1 \\ 1 - 10^{\frac{-dB_{err}}{10}} & \text{if } \frac{G_T(f)}{G_{T_{opt}}(f)} < 1 \end{cases} . \tag{3.28}$$

This result allows conversions from decibel error to MRD.

VSWR is only defined for the case where a perfect match is attempted, so $G_{T_{opt}}(f)$ is always equal to one because a perfect match for a passive network corresponds to a gain of one. This means that $G_T(f)/G_{T_{opt}}(f)$ is always less than or equal to one because $G_T(f)$ cannot be greater than one. Thus, only one case needs to be considered in the conversions between VSWR, and MRD and decibel error.

The conversion from decibel error to VSWR involves converting the decibel error to a reflection coefficient using (3.23) and (3.26) to obtain

$$|\Gamma| = \sqrt{1 - 10^{\frac{-dB_{err}}{10}}} . \tag{3.29}$$

This value is then substituted into (3.20) to obtain VSWR in terms of decibel error.

The reflection coefficient is obtained in terms of MRD using (3.23) and (3.25) and this result is then substituted into (3.20) to obtain

$$\text{VSWR} = \frac{1+\sqrt{\text{MRD}}}{1-\sqrt{\text{MRD}}} . \tag{3.30}$$

This result allows conversions from MRD to VSWR.

Converting from VSWR to MRD and decibel error involves solving (3.20) in terms of $|\Gamma|$, using this value in (3.23) to calculate $G_T(f)$, and substituting the result into (3.25) and (3.26). The results after simplification are:

$$\text{MRD} = 1 - \left(\frac{\text{VSWR}-1}{\text{VSWR}+1}\right)^2 \tag{3.31}$$

and

$$dB_{err} = -10\log\left[\frac{4\text{VSWR}}{(\text{VSWR}+1)^2}\right] \tag{3.32}$$

where the minus sign in (3.32) is due to the fact that the logarithm is negative because its argument is less than one.

3.3.2 Impedance Calculation

This section will consider the calculation of the input impedance used to calculate the values of the error functions given in Section 3.3.1. The calculation of the input impedance, the effect of losses, lumped and distributed elements, and discontinuities are presented below.

The impedances required to calculate the values of the error functions described in Section 3.3.1 are calculated starting at the load impedance. Each element is then used to modify the impedance until the input impedance of the system is obtained. This input impedance is then used to calculate the input reflection coefficient from which the fitness is calculated. The only major change to this approach occurs when discontinuities are used. In this case the effect of parallel elements cannot simply be included when they are encountered because the discontinuities affect the parallel elements' contributions to the circuit, and the discontinuity parameters depend on all circuit elements forming the discontinuity. The solution used in this dissertation is to only modify the impedance when a series element is encountered. The effects of any parallel elements and discontinuities are then simultaneously accounted for.

All elements used are assumed to be lossless. The main reason for this approach is that it allows simplified calculation of the gain of the matching network as discussed in Section 3.3.1. The error encountered by using the assumption that all elements are lossless is usually very small, and is adequate for a synthesis technique such as the one presented here.

The lumped elements used in this dissertation are simple capacitors and inductors which can be connected either in series or in parallel. The models used do not include losses or resonant effects which arise due to the parasitic elements that are present in all components [2]. The main reason for this is that the magnitude of the parasitic effects change with component value and type. Accounting for parasitics is however an extremely important process and would have to be done for any practical design.

The distributed elements in this dissertation are either perfect transmission lines or microstrip transmission lines. Series lines, and open and short circuited stubs are allowed. In the case of perfect transmission lines, the same characteristic impedance is used at all frequencies and dispersion is not present, allowing the characteristic impedance of the line to be specified directly. Microstrip lines are not perfect TEM transmission lines leading to dispersion which causes the characteristic impedance to vary with frequency. This means that the characteristic impedance of a microstrip line cannot be uniquely specified and the line widths have be specified instead.

One of the most important properties of the algorithm developed during this dissertation is the inclusion of dispersion and discontinuity effects during synthesis. This allows these effects to be used by the algorithm to obtain better results than algorithms that attempt to minimise these effects or ignore them altogether. The discontinuity models used can thus have a large effect on the quality of the results obtained. The discontinuities considered are open-ends, via holes to ground, width steps, T-junctions, and crosses.

3.3.3 Pareto-Like Fitness

One of the major advantages of maintaining a population of solutions is that a number of solutions to a problem can be obtained. The problem is to find a way of identifying multiple solutions that are of high quality. Pareto optimality, which was considered in Section 2.1.1.3, is one solution to this problem because it allows a number of objectives to be optimised simultaneously. This dissertation uses a simplified implementation of Pareto optimality to rank solutions.

The two objectives used for the Pareto-like optimality criterion are the fitness (error) and the length of an individual. The error functions used in this dissertation are considered in Section 3.3.1, and the length of an individual is the number of circuit elements comprising that individual. The use of the length of an individual as one of the objectives is intuitive because a good five element solution can usually be created by adding one element to a good four element solution and optimising the result. Using a multi-objective fitness function thus means that a number of solutions can be compared, and the performance of the algorithm should improve.

Pareto optimality works on the principle of dominated and non-dominated solutions. The fitness is then assigned by giving each layer of non-dominated individuals a new fitness as described in Section 2.1.1.3. The implementation of this process can be complicated, so a simpler Pareto-like ranking system was used in this dissertation.

The population is first sorted according to the two objective functions' values so that the final order has the lengths from shortest to longest, and each length further is sorted so that the error values vary from low to high. The fitness is then assigned by using a fitness of zero for the first individual of each length, and increasing the fitness by one for each subsequent individual of the same length. This has the effect of ranking each length independently to ensure that comparisons between individuals of different lengths are fair.

The main difference between this approach and the Pareto approach is that solutions are only considered to be dominated by other solutions of the same length. The largest differences between the fitnesses assigned by this approach and the Pareto approach will be seen near the beginning of a run, when some individuals will have fitnesses that differ from the pure Pareto case because their Pareto ranking is affected by individuals of other lengths. Near the end of a run these fitness differences should be negligible because longer individuals will give smaller errors than shorter individuals meaning that the Pareto ranking of an individual is almost always only affected by individuals of the same length.

This approach only works because there are a small number of lengths, and each length generally gives better results than shorter lengths. This approach will not work where two continuous variables are used.

3.4 Local Optimiser

This section will consider the implementation of the local optimiser used in this dissertation. The decision as to which local optimisation algorithm to use will be considered first because this determines whether gradient calculations and line searches are necessary or not. Then gradient calculation is considered followed by a description of the line search algorithm used.

3.4.1 Algorithm

The choice of local optimisation algorithm is very important to this dissertation because it determines whether gradient calculations and line searches are necessary, and it can affect the performance of the algorithm. This section will consider the choice of local optimisation algorithm.

The first major consideration is that the objective function used in this dissertation is a minimax function. This leads to gradient discontinuities and limits the usefulness of methods that assume that the objective function is quadratic. Another consideration is that the genetic algorithm will probably find a point near an optimum, so the local optimiser does not have to be exceptionally good. The calculation of the gradient is difficult because of the presence of discontinuity models which do not have simple gradient functions, so an algorithm that does not require many gradient calculations would be preferable.

Simplex methods have the disadvantage that they are known to be slow for large problems [12]. The direction set and conjugate gradient methods assume the function is quadratic which is a poor approximation to a minimax problem. The quasi-Newton methods also use a quadratic approximation, but have the additional disadvantage that the calculations can involve large matrices. The leap-frog algorithm has the limitation that a large number of gradient calculations are required.

Eventually the steepest descent algorithm was chosen. Steepest descent is known to be a poor local optimiser, but the problems usually only arise when the current solution is far from an optimum. As mentioned above, the genetic algorithm should provide a solution that is close to an optimum, so these problems should not be significant. Steepest descent has the additional advantage that it is very simple to implement, but it has the drawback that gradient information is required.

3.4.2 Gradient Calculation

Gradient calculation is required for the implementation of most local optimisation algorithms including the steepest descent algorithm used here. The accurate calculation of the gradient is essential to obtain good results and this section describes the gradient calculation used in this dissertation.

The main problem encountered with gradient calculations in this dissertation is the presence of discontinuities. The models for these discontinuities are complex precluding the calculation of closed form gradient functions. This meant that finite differences had to be used.

Finite differences were used to calculate the gradient as described in Section 2.1.3.3. The values of distributed elements parameters were changed by one to obtain the maximum accuracy. An adjustment of 10^{-6} of the maximum value was used for lumped elements because the extremely high precision of the representation could lead to problems with rounding errors if the step size used is too small.

3.4.3 Line Search

The implementation of the line search for this dissertation is considered in this section. The Fibonacci line search algorithm was used. The motivations for this choice, and the modifications made are considered in this section.

As mentioned previously, a polynomial approximation is a poor fit to minimax functions, and this means that interpolation methods considered in Section 2.1.2.3 will not produce good results. This narrows the choice to the Fibonacci and golden ratio line search algorithms. The only advantage of the golden ratio line search is that the number of steps does not have to be set in advance. The number of line search steps was implemented as a constant in this algorithm to minimise the number of optimisation steps wasted on poor solutions. Good solutions will tend to survive to the next generation of a genetic algorithm allowing them to be optimised further, while poor solutions will die off, limiting the computations wasted on that solution. Thus the Fibonacci line search was chosen because it gives a smaller error than the golden ratio search for a given number of steps.

The next problem is bracketing a minimum. The quadratic interpolation method considered in Section 2.1.2.1 will not produce good results because a minimax function is used. A heuristic technique was thus implemented. Three points are required to determine whether an optimum has been bracketed, and these three points should preferably be chosen so that they form three of the four points used by the Fibonacci search.

The initial step size in the direction of the search in a constant. A third point is now

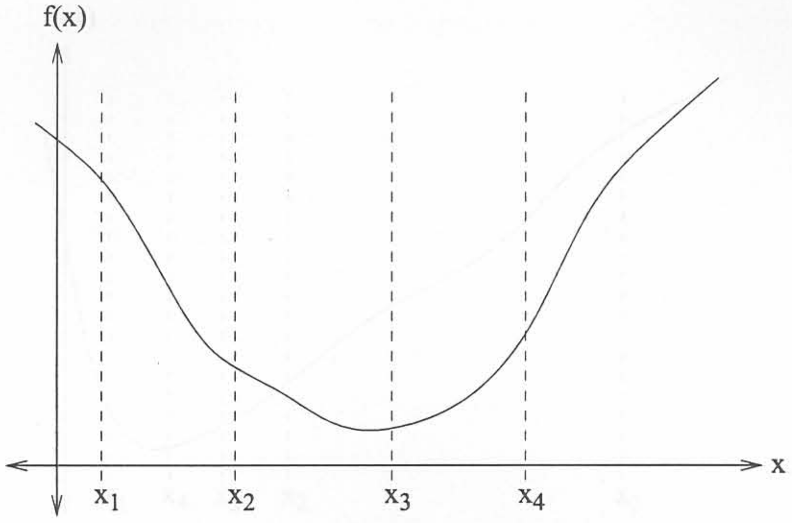


Figure 3.2: Line search range extension.

required between the starting and ending points of the range. This point is generated by using the point from the Fibonacci search that is closest to the starting point of the current range, giving points x_1, x_2, and x_3 in Figure 3.2. The range must be extended if the middle point has an objective function value that is worse than the objective function value at the end point of the range. The range extension is accomplished by using the previous range middle point as the new starting point and the previous end point as the new middle point, giving points x_2 and x_3 in Figure 3.2. A new end point must now be determined to ascertain whether an optimum is now bracketed (point x_4 in Figure 3.2). The maximum number of times this process may be repeated is set as a constant. Each range extension step only requires one calculation and the last three points generated from three of the four points required by the Fibonacci line search. In other words, point x_5 would have to be added to points x_2, x_3, and x_4 in Figure 3.2 to start a Fibonacci line search.

It is possible that the procedure described above can bracket more than one optimum, but the Fibonacci search only works in the case where a single optimum is bracketed. This problem is overcome by checking that the two middle points of the Fibonacci line search are better than the endpoints. If this is not true the line search is terminated and the best point found up to that point is returned. Note that only one of the middle points has to be

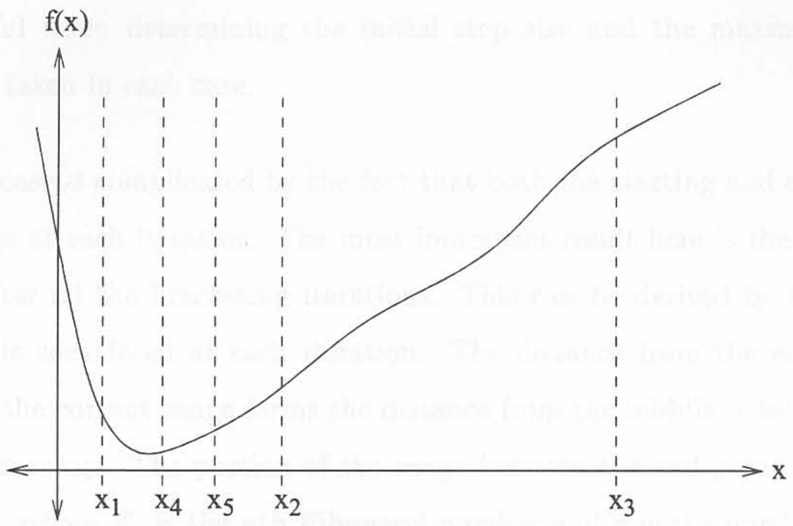


Figure 3.3: Line search range reduction.

checked at each step because the other middle point is carried over from the previous step.

A similar procedure was implemented to reduce the range before the commencement of the Fibonacci line search because the genetic algorithm and repeated optimisations should ensure that the current point is close to the local optimum. The range reduction search starts in the same way as the bracketing search with a step being taken in the direction of the line search and a middle point being determined, giving points x_1, x_2, and x_3 in Figure 3.2. If the middle point is worse than the range starting point then the current middle point is taken as the new range end point and the procedure is repeated, giving points x_1, x_2, and x_4 in Figure 3.2. As with the bracketing search, each iteration only requires one calculation and the last three points generated form three of the four points used by the Fibonacci search. In other words, point x_5 would have to be added to points x_1, x_2, and x_4 in Figure 3.2 to start a Fibonacci line search. The maximum number of times this procedure may be repeated is limited. This range reduction algorithm decreases the range faster than a Fibonacci line search when the optimum is very close to the starting point.

The size of the final ranges and the steps taken by the bracketing and range reduction searches is useful when determining the initial step size and the maximum number of iterations to be taken in each case.

The bracketing case is complicated by the fact that both the starting and ending points of the range change at each iteration. The most important result here is the maximum step that is taken after all the bracketing iterations. This can be derived by first calculating the range that is considered at each iteration. The distance from the end point to the middle point of the current range forms the distance from the middle point to the starting point of the next range. The portion of the range between the end point and the middle point is F_{p-1}/F_p where F_n is the nth Fibonacci number and p is the number of iterations in the Fibonacci search [12]. The ratio of the of the total range to the distance from the middle point to the starting point is F_p/F_{p-2}. These results can then be used to find the relationship between the current range and the previous range:

$$I_{i+1} = \frac{F_p}{F_{p-2}}\frac{F_{p-1}}{F_p}I_i \tag{3.33}$$

$$= \frac{F_{p-1}}{F_{p-2}}I_i \tag{3.34}$$

where I_i denotes the range at iteration i. The range at any iteration can be calculated by applying (3.35) recursively:

$$I_i = \left(\frac{F_{p-1}}{F_{p-2}}\right)^i I_0 \ . \tag{3.35}$$

The total step size can now be calculated using this result and ignoring overlaps between iterations. The portion of each range that is not an overlap is F_{p-1}/F_p. The total step size is now given by

$$I_i = I_0 + I_0\frac{F_{p-1}}{F_p}\sum_{i=1}^{n}\left(\frac{F_{p-1}}{F_{p-2}}\right)^i . \tag{3.36}$$

The first term is the size of the initial step that is always taken. The second term is the sum of the step sizes at each iteration which are calculated from the portion of the range that is not an overlap at each iteration.

The range reduction case is a simple application of the formulae that are used by the Fibonacci line search. The portion of the total range that is between the starting and middle points of the range for the first iteration of a Fibonacci search is given by F_{p-2}/F_p

Table 3.5: Default local optimiser parameters.

Parameter	Value
Initial step size	$\frac{1}{8000}$
Local optimiser steps	2
Line search steps	3
Range extension steps (maximum)	5
Range reduction steps (maximum)	5

where F_n is the nth Fibonacci number and p is the number of iterations in the Fibonacci search [12]. At each iteration of the range reduction algorithm the previous range will be reduced by this factor. The final range as a fraction of the initial range is thus given by

$$I_m = \left(\frac{F_{p-2}}{F_p}\right)^m I_0 \tag{3.37}$$

where I_m is the range after m range reduction iterations. This is seen to be much faster than the Fibonacci search in this case because the Fibonacci search reduces the range by a factor of F_p after p iterations.

The parameters used in the final algorithm are given in Table 3.5. The initial step size was chosen to give a change of approximately one in the smallest range used in the representation (transmission line length). The number of line search steps was chosen to be the smallest allowable value for a Fibonacci line search because the genetic algorithm is the primary search algorithm and accurate line searches are thus not required. The number of range reduction steps were chosen to give a minimum step size of 3.91×10^{-6} of the maximum value used in the representation, which corresponds to an 18 bit accuracy. This accuracy might appear to be too low, but it corresponds to four significant digits when the component range is 10^{-15} to 10^{-6}. The total range after range extension with five range extension steps is 2.44×10^{-3} of the maximum value used in the representation.

The last important consideration is that a line search can generate points that violate the constraints. In this case the constraints specify the maximum and minimum values of the variables and are thus inequality constraints. If a line search generates a point that violates constraints, all the variables whose constraints are violated are set to either their maximum or minimum values as required. The main drawback of this approach is that it changes the line search direction. A way of avoiding this problem is to move backwards along the search direction until all constraints are satisfied. However this will mean that points on a constraint boundary that have a gradient that points into the infeasible region will not be optimised by the line search. This problem is avoided by the approach described above.

3.5 Summary

The implementation of the impedance matching algorithm developed during the course of this work was given in this chapter.

Firstly, an overview of the approach followed was presented. The reasons for using a hybrid genetic algorithm, and a flow chart of the algorithm were given.

The genetic algorithm implemented was then considered in detail with special attention being paid to the representation and operators used. The representation used is unique because it allows both binary and floating-point genetic operators to be used, and equations for calculating the decimal precision of the representation were derived.

The error functions, impedance calculation, and Pareto-like optimality were reviewed to show how the fitness of an individual was calculated. Equations for the conversions between VSWR, MRD, and decibel errors were derived, and the default algorithm parameters were given. The calculation of the input impedance of a network includes microstrip dispersion and discontinuity effects. A new type of Pareto-like optimality was used to allow multiple solutions with different lengths to be synthesised simultaneously.

The choice of local optimiser, and the gradient calculation and line search algorithms used were discussed. Steepest descent was used because it is very simple and robust, and the combination with a genetic algorithm should overcome its limitations. Gradients were calculated using finite differences because the complex discontinuity models used mean that the derivation of analytic expressions would be impractical. A Fibonacci line search was used as the basis for the line search algorithm, with extensions being made to allow a minimum to be bracketed.

The performance of the algorithm presented above will be considered in Chapter 4.