In more sophisticated genetic algorithms the crossover and mutation operators are generally uncoupled. In fact there may be a number of different crossover operators and a number of different mutation operators. When reproduction is to take place an operator is chosen from the list of operators and an appropriate number of chromosomes are taken from the population to serve as input for this operator.

The selection of operators can be performed in much the same way as the selection of chromosomes if an operator-fitness is assigned to each operator and roulette wheel selection is used to select an operator whenever required. Operator-fitness values can be chosen fixed or interpolated as the genetic algorithm progresses.

Interpolation of operator-fitness can be advantageous for many problems. If the initial population is largely random , i.e. it has a lot of diversity then crossover is the best operator to re-combine this diversity of schemata. Conversely, mutation does not have too much of a part to play in the initial generations as there is already a lot of diversity present in the population and the mutation operator is not needed to produce additional diversity.

After the algorithm has run for a number of generations crossover will have recombined the schemata present in the initial population to advantage. The mutation operator now becomes more important as it is required to perform its role as injector of diversity.

4.1.5 Population handling

The way in which a new population is created from child chromosomes is also a characteristic of a genetic algorithm. When the genetic algorithm starts there is an initial population and when it ends there is an evolved population of improved fitness. What is required is a policy for removing old (and inferior) population members from the population and replacing them by new and fitter offspring.

The elementary way of doing this is known as generational replacement. In the course of a generation the population of N_{pop} chromosomes is kept unchanged and N_{pop} new chromosomes are produced using the selection and reproduction mechanisms described in the previous sections. Once all N_{pop} new chromosomes have been produced the whole current generation is replaced by the new population. Hence the name generational replacement.

The advantage of this technique is its simplicity. Little or no computational overhead is incurred in population handling. Whether or not this is an important advantage depends on the problem at hand. Usually evaluation of the fitness function is by far the most computationally intensive part of the optimisation process and a little extra computational load incurred because of more sophisticated population handling will not make a significant difference to the net computational load.

What may well make a considerable difference in computation time is the fact that generational replacement does not tend to give the fastest convergence: more fitness function evaluations will be required to produce the same result. The best individual chromosome in a generation may not survive to the next generation and the best chromosome in this next generation may actually have a lower fitness value than this “old” best chromosome.

Another disadvantage of generational replacement is that storage space is required for an additional population. This may or may not be a problem depending on how large the individual chromosomes are, the magnitude of N_{pop} and how much storage space is conveniently available to the hardware on which the genetic algorithm is implemented.

A simple approach to improving population handling is the introduction of elitism. The best chromosome in a population is copied to the next generation. In order to prevent this chromosome from over-dominating as a super individual (Section 4.1.2) elitism should be coupled to either fitness windowing or some kind of fitness normalisation.

Another improvement in population handling is the introduction of a so called steady-state reproduction technique. This involves substituting a few new chromosomes at a time into the population instead of replacing the whole population at a stroke as with generational replacement. Typically the number of chromosomes which is replaced, n_{ss}, is small: 1 or 2. In fact n_{ss} is usually equal to the number of offspring produced by a reproduction operator. The n_{ss} worst (lowest fitness) members of the population are deleted and the n_{ss} new chromosomes are inserted into the population.

With steady-state reproduction the population changes gradually from generation to generation. Chromosomes with good schemata stand a better chance of surviving and propagating their successful genetic material into following generations.

The steady-state reproduction technique can be further sophisticated by disallowing the insertion of duplicate chromosomes into the population. This helps to keep diversity in the population and it can also help to keep successful chromosomes from over-dominating the population.

Disallowing duplicates has one drawback in the fact that a new chromosome will have to be compared to all chromosomes in the population before it can be inserted into that population. This introduces an additional computational load which may not be trivial. However, as stated earlier, this computational load will usually be considerably smaller that the one incurred because of the fitness function evaluations and the improvement in convergence speed (which means less fitness function evaluations) will usually more than compensate for the extra load because of chromosome comparisons.

4.2 Hybridisation for vector field estimation

The basic robust genetic algorithm which operates on bit string chromosomes with simple bit based reproduction operators will work for a very large variety of optimisation problems. While for a given problem it al almost certain to work it is also almost certainly not the best algorithm for solving a given problem.

Block matching techniques for vector field estimation already exist. Whilst not without their drawbacks they certainly do point the way towards the finding good vector fields. Little can be achieved by replacing these techniques with the robust genetic algorithm.

The approach to take here is that of hybridisation: take what we know of two ir more incongruous optimisation algorithms and create a new one which is better than either of the parent algorithms.

Vector field estimation is an optimisation problem. The objective is to produce as good a vector field as possible. In the design of any disparity estimator two issues must inherently be addressed:

- What is a vector field? How does the estimator take this into account?
- What is a “good” vector field? How does the estimator determine one?

In applying a genetic algorithm to a problem and in hybridising the genetic algorithm with existing algorithms for vector field estimation one of the first points to examine is that of the encoding. The bit string representation of the robust genetic algorithm can describe anything but encoding techniques appropriate to vector field estimation will favour the development of appropriate reproduction operators and it can also facilitate the integration of current algorithms into the genetic algorithm.

A full search blockmatcher is only conscious of the fact that it is dealing with vector fields in the sense that it stores vector data in a two dimensional array which is the vector field. The estimation algorithm itself functions by finding a best match for individual blocks. This best match for each of the individual blocks is also the definition of good for the vector field as a whole.

In this way the full search blockmatcher effectively ignores the way in which vectors appear in patches in vector fields. That is to say one moving object in a scene will have a single vector associated with it if the model of linear displacement without distortion holds. Even if the linear displacement model does not hold then the object displacement can be approximated as the linear displacement of several smaller objects. Objects in images are composed of blocks. The full search blockmatcher treats each block independently and assigns it an optimal vector in match criterion sense. Often this vector will be the correct vector for the object to which the block belongs. As described and illustrated in Chapter 3 however

decreasing block size brings with it a risk of noise based matches being made and areas with periodic texture content can introduce multiple local optima for the vector under examination.

The recursive blockmatcher still only works on individual block but it does take into account the fact that the vector for a block stands a good chance of being equal or very similar to the vector or vectors of a neighbouring block or blocks. It does so by using a limited vector search space in the vicinity of the vector or vectors determined for neighbouring blocks. This does indeed reduce the occurrence of spurious matches and also helps with periodic textures. There is one essential fallacy to this concept: neighbouring blocks either belong to the same object or they do not. The associated vectors will either be very similar or they may be completely different! Recursive schemes implicitly assume the former while the latter is certain to occur in any scene with more than one object. At the very least this introduces a convergence effect but particularly with disparity estimation where neighbouring objects can have very different disparities the estimator can be thrown completely off track.

So if a genetic algorithm for vector field estimation (blockmatching) is going to be an improvement over conventional techniques it will have to be more conscious of objects than these techniques. The place in the genetic algorithm into which to inject this consciousness is that of the reproduction operators of crossover and mutation.

At the very base of the genetic vector field estimation algorithm there is off course the encoding of a vector field.

The purpose of the mutation operator(s) is to introduce diversity into the population. Doing so on a random basis is always possible but in vector field estimation existing block matching techniques can also be used as a basis for mutation operators.

Existing block matching techniques do not feature anything like crossover operators as these operators are very particular to genetic algorithms. Because crossover is so important to the function of genetic algorithms it is crucial that any vector field encoding that is chosen should allow for the implementation of at least one meaningful crossover operator.

4.3 Vector field encoding

In designing an encoding technique it is important to decide what is being encoded. It is possible to define a chromosome as being a single vector belonging to a block. Thus aiming at a genetic algorithm version of a conventional blockmatcher. But as these blockmatchers were computationally realistic in the first place and operating on single vectors does not give sufficient scope for the exploitation of the properties

of the vector field this is not the best idea. To design an algorithm which operates on vector fields, chromosomes should describe a whole vector field and not single vectors.

Combining the idea of operating on a whole vector field with that of objects and associated vectors a chromosome could be formed by a list of objects with each object having associated with a shape, location and a displacement vector. This list would have to be of variable length as arbitrary scenes do not contain a predetermined number of objects. The storage required for each object would also get to be of variable length as objects will have different sizes and shapes.

Such an advanced vector field encoding strategy, whilst fascinating and very promising, poses formidable problems for the design and implementation of crossover and mutation operators. Design of such operators is possible but poses formidable implementation problems. The mere design of a good crossover operator for such encoding would be an extremely complex issue.

The vector field encoding technique used in this work is in fact a compromise. A whole vector field is encoded as a two dimensional array of vectors which are represented by integer numbers. One chromosome thus corresponds to one vector field. In this encoding strategy no specific account it taken of the fact that patches of vectors tend to have the same vector value. This awareness is used in the reproduction operators of the genetic algorithm.

As the vector fields which are of interest in this work are disparity fields for stereo image pairs where the disparity can be modelled as completely horizontal the displacement has only one component and thus a chromosome becomes a two dimensional array of integers. The number of columns in this array becomes:

$$nvc = \frac{W_{pix}}{N_x} \quad , \tag{4.9}$$

the width of the images divided by the block width. Similarly the number of rows in this array becomes:

$$nvr = \frac{H_{pix}}{N_y} \quad , \tag{4.10}$$

the image height divided by the block height. Thus a chromosome, C, is described as:

$$C = \begin{bmatrix} d_h(1,1) & d_h(1,2) & .. & d_h(1,nvc-1) & d_h(1,nvc) \\ d_h(2,1) & .. & & & d_h(2,nvc) \\ .. & & .. & & .. \\ d_h(nvr-1,1) & & & .. & d_h(nvr-1,nvc) \\ d_h(nvr,1) & d_h(nvr,2) & .. & d_h(nvr,nvc-1) & d_h(nvr,nvc) \end{bmatrix} \quad (4.11)$$

4.4 Vector field fitness

Of the objective measures which are available for vector field evaluation such as prediction error, smoothness and vector field entropy prediction error is a computationally convenient choice as the prediction error for a whole image is nothing more than the sum of prediction errors for all the *nvc* x *nvr* blocks in the image. Also it is so that if crossover or mutation operators make localised changes to a chromosome (a vector field) then the fitness of the whole chromosome can be re-calculated by subtracting from the prediction error for the whole vector field the decrease in prediction error for those vectors which were actually changed.

The actual weighting function used in the prediction error is MSE.

In order to avoid the problems sketched in Section 4.1.2 it is also sensible to use linear normalisation of fitness as described in that same section.

4.5 Vector field population initialisation

The initial chromosome population can be randomly initialised in the style of the robust genetic algorithm. Each vector in each vector field can be set to a random value. The distribution of vectors chosen as uniform over the range of legal and feasible vector values.

A form of temporal recursion can be introduced by seeding the initial population with vector fields determined for previous frame pairs. This can speed convergence but also provides another reason for performing the fitness normalisation of Section 4.1.2. If this is not done then these previous vector fields, which in all likelihood give far lower prediction errors than any random vector field, will over-dominate the population as super individuals.

A similar idea would be to seed the initial population with a vector field generated by a full search blockmatcher. This idea is not in harmony with the use of a genetic algorithm coupled with prediction error as a fitness function. The full search blockmatcher generated vector field will produce the lowest possible prediction error (assuming it had a sufficiently large search range) and will thus always be the fittest individual chromosome in the population. In the first generation and in every subsequent generation. No matter how much the reproduction operators reflect the patch-wise constant nature of vector fields.

If such a full search type vector field is available and it is used to seed the population then after completion of the genetic algorithm it is the second fittest chromosome which should be selected as the output vector field. This vector field will have a prediction error which is a little higher than that of the full search generated vector field but it will have been generated by the reproduction operators which will result in a smooth vector field.

4.6 Vector field population handling

Generational replacement is seldom the optimum population handling strategy. As it will be necessary to maintain the population in a sorted order in order to efficiently implement fitness normalisation the additional overhead of a steady-state reproduction policy is minimal. When reproduction yields n_{ss} new chromosomes the n_{ss} worst performing chromosomes in the population are deleted to make space for these new chromosomes.

The question of whether to allow duplicates is less clear cut. Comparison of entire vector fields is a much less computationally expensive than determining the prediction error for a vector field but it certainly not a completely trivial overhead. As the chromosomes are large and there is much initial diversity comparison can be very quick because a single differing vector suffices to make two vector fields different. As the algorithm converges the chance of duplicate chromosomes occurring increases.

4.7 Vector field mutation

The mutation operator takes a single chromosome as input and produces a single chromosome as output. In the genetic vector field estimation algorithm a single chromosome is a vector field which links two images.

4.7.1 Random vector field mutation

The most trivial implementation of the vector field mutation operator is the random vector field mutation operator which is little different from the basic bit mutation operator used on bit strings.

For each of the *nvc* x *nvr* vectors in a chromosome *C* a probability test is performed. A random number *r* is generated according to a uniform distribution on [0,1>. If for a certain element of *C* it happens that $r < p_m$ then this element of C is set to a random vector value which is generated according to a uniform distribution over the range of legal vector values.

As with all such mutation operators p_m should be chosen low. In order for the random vector field mutation operator to change an average of two vectors in a typical vector field a value of 0.0003 is appropriate.

The function of the random vector field mutation operator is to maintain diversity in the population. In itself it will not generate good vector fields with any speed. Because the diversity in the initial population is already high the operator fitness of the random vector field mutation operator should start low and increase to a modest value as the generations progress.

4.7.2 Hybrid vector field mutation

In having one vector field, which links two images, as output vector field mutation is similar to complete conventional vector field estimators. The difference lies in the fact that vector field mutation also has a vector field as input. In thinking of conventional vector field estimators this is still not such a large difference because it simply means that there is an "initial guess" vector field available.

There are two things which can be input into a hybrid vector field mutation operator:

- The "looking for best matches" behaviour of conventional block matching techniques.
- Knowledge of the patch-wise constant nature of vector fields. Best matches belong not so much to single blocks but to homogenous clusters of blocks, to objects in the scene.

These two thoughts lead to a hybrid vector field mutation operator which works as follows:

1. A random location (i, j) is chosen within the vector field. i according to a uniform distribution over $[1, nvr]$ and j according to a uniform distribution over $[1, nvc]$.
2. A vector search range is examined to see if a better value (producing a lower prediction error) can be found for $d_h(i,j)$. If so then this new value is substituted for $d_h(i,j)$ if not then $d_h(i,j)$ is left unmodified (per definition because then $d_h(i,j)$ must itself already be the best match vector).
3. $d_h(i,j)$ is recursively propagated to the left $(i, j\text{-}1)$, right $(i, j+1)$, up $(i\text{-}1, j)$ and down $(i+1,j)$ to those locations within C where it would not increase the local prediction error and a value equal to $d_h(i,j)$ is not already present.
4. If recursive propagation was not possible, i.e. if only the vector at location (i,j) was changed then $d_h(i,j)$ is reset to its original value so that the mutation operator has no effect on C.

The best match search nature of conventional blockmatchers is embodied in step (2) of this algorithm where a best match is looked for by calculating the prediction error for a number of candidate vectors and selecting the one which gives the lowest prediction error.

Steps (3) and (4) reflect the piece-wise constant nature of vector fields. The vector value found in step (2) is tried for neighbouring blocks which may belong to the same object. Spurious matches for a single block only are rejected in step (4).

This hybrid vector field mutation operator will never increase the prediction error associated with a vector field and it will also (except in the pathological case) improve the smoothness and homogeneity of the vector field.

A sample run of this hybrid vector field mutation operator is illustrated in Figure 4.10 for a small example vector field which has *nvr* 6 and *nvc* = 8.

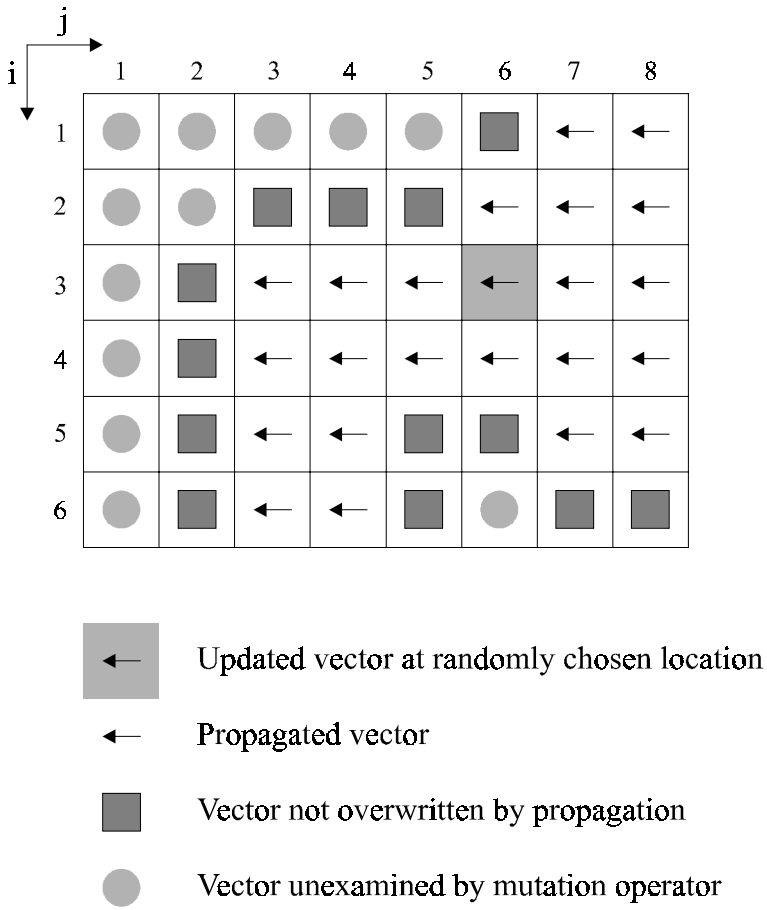

Figure 4.10: Example of hybrid vector field mutation.

In the example of Figure 4.10 the location within the vector field which is randomly picked by the mutation operator is (3,6). A best matching vector is determined for this vector location and it is successfully propagated to 22 vector field locations where it would not increase the local prediction error. Twelve locations within the vector field of the example are never examined because they would not be considered by the recursive propagation process which was stopped at thirteen vector field locations.

4.8 Vector field crossover

In vector field crossover schemata are defined as contiguous clusters of vectors within a vector field which share the same vector value(s). Hopefully these clusters of vector will corresponds to objects in the scene.

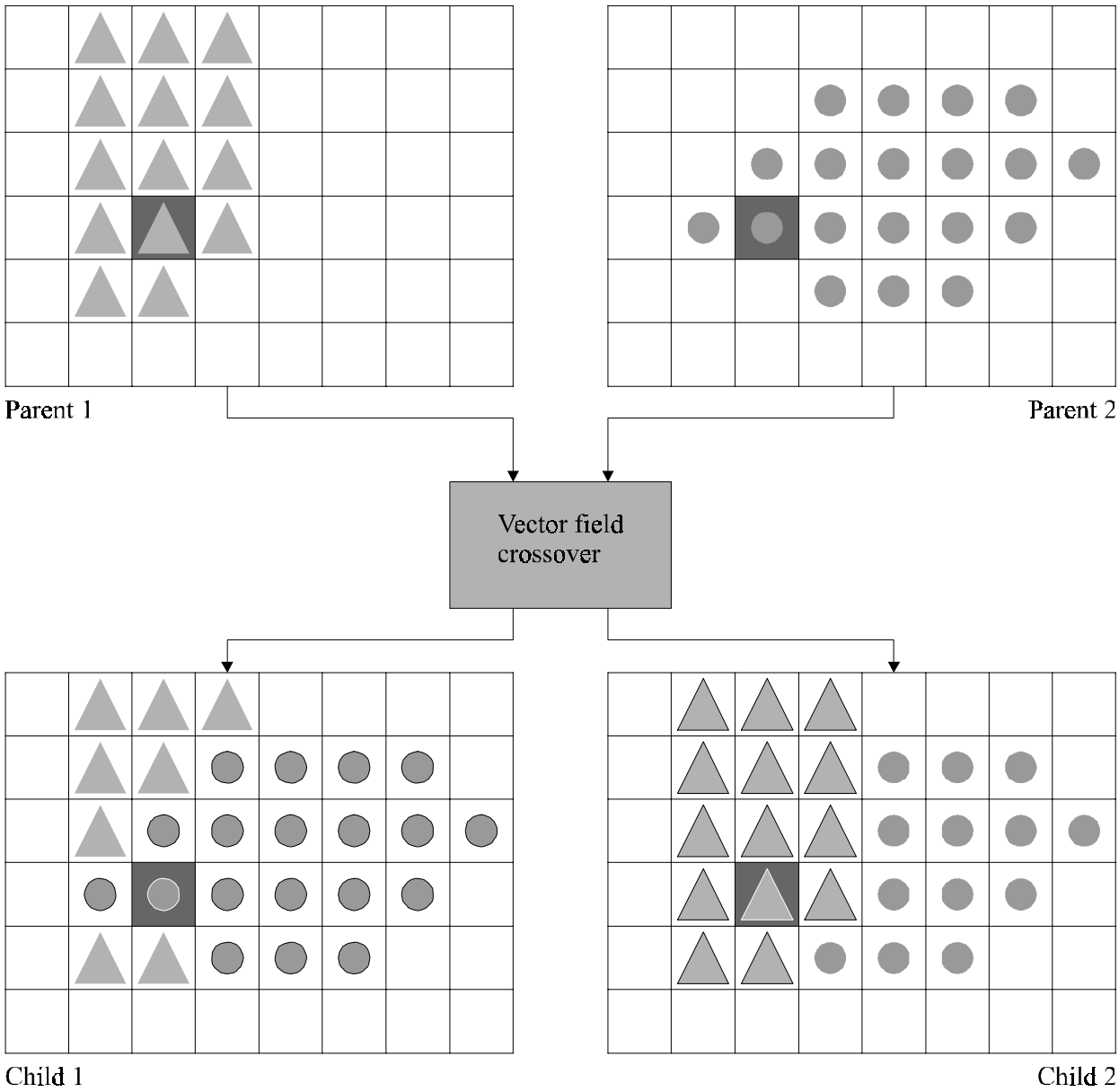

Figure 4.11: Example of vector field crossover.

The following vector field crossover operator will exchange such schemata for two parent chromosomes *C* and *D*:

1. A random location (*i*, *j*) is chosen, *i* according to a uniform distribution over [1, *nvr*] and *j* according to a uniform distribution over [1, *nvc*].
2. In parent vector field *C* the contiguous patch of vectors around (*i*, *j*) which has the same vector value of $d_{h,C}(i,j)$ is determined. Similarly in parent vector field *D* the contiguous patch of vectors around (*i*, *j*) which has the same vector value of $d_{h,D}(i,j)$ is determined.

3. The patch of vectors found for C is copied to D and the patch of vectors determined for D is copied to C.

This vector field crossover operator will propagate successful vector field patches throughout the population. It is effectively blind to prediction error except for the fact that the parent chromosomes were chosen with a bias to high fitness, i.e. a bias to low prediction error.

A sample run of this vector field crossover operator is illustrated in Figure 4.11. The random location used in this instance was (4,3). Triangle marks denote the vector patch which was determined in parent vector field 1 while circles mark the vector patch which was determined in parent vector field 2. In the child vector fields the appropriate super-impositions have been made.

4.9 Operator fitness setting

Three reproduction operators have been discussed: random vector field mutation, hybrid vector field mutation and vector field crossover. When a new chromosome or new chromosomes are to be produced one of these operators should be picked as described in Section 4.1.4. In order to use the mechanism there described it is necessary to assign necessary to assign each operator an operator-fitness. These operator-fitness values can be chosen fixed or parametrised over the generations.

The operator fitness for generation g are:

- Random vector field mutation: $O_0(g)$.
- Hybrid vector field mutation: $O_1(g)$.
- Vector field crossover: $O_2(g)$.

In order for a random number on [0,1> to be directly useable for roulette wheel operator selection the operator-fitness are chosen so that for all generations g:

$$\sum_{i=0}^{2} O_i(g) = 1 \,. \tag{4.12}$$

The random vector field mutation operator is not particularly good at improving vector fields. It is just there to maintain diversity in the population and thus should have a low operator-fitness. In early generations it is in fact superfluous as there is more than sufficient diversity already present in the population. Reflecting this thought the operator-fitness of the random vector field mutation operator can be made linearly increasing over the generations. Starting at zero and increasing to a modest value like 0.01 after N_{gen} generations:

$$O_0(g) = \frac{g-1}{N_{gen}-1} \cdot 0.01 \tag{4.13}$$

The hybrid vector field mutation operator and the vector field crossover operator each play an important role in improving the population. In early generations the hybrid vector field mutation operator will be the more important. It produces patches of vectors where first there was only random diversity. Initially there are only few vector patches in the population for the vector field crossover operator to propagate. In later generations the vector field crossover operator will come to play an increasingly important role as it comes to combine good vector patches between vector fields. Operator-fitness values for O_1 and O_2 which reflect this are:

$$O_1(g) = \frac{1 - O_o(g)}{2} + 0.20 - 0.40 \cdot \frac{g-1}{N_{gen} - 1} \tag{4.14}$$

and:

$$O_2(g) = \frac{1 - O_o(g)}{2} - 0.20 + 0.40 \cdot \frac{g-1}{N_{gen} - 1}. \tag{4.15}$$

4.10 Experiments

In order to actually be able to test the genetic vector field estimator a number of parameters must be set and some policy decisions have to be made.

Finding “optimal” setting for all the possible parameters and choices involved would be a complex optimisation problem in itself. The settings described in this Section were found to work well with the DISTIMA test sequence material and with the “mirror” test sequence.

The population size is a very important parameter for genetic vector field estimation because each chromosome consists of an entire vector field. Having very large populations in the order of thousands of chromosomes would work prohibitively in the sense that the storage requirements would be too large. In practice the algorithm will just work with a population size of 30. Better results are obtained with a population size of 100 while there is little gain to be made by having a population size larger than that.

The steady-state reproduction policy produces faster convergence than does a generational replacement policy. Checking for duplicates speeds convergence in terms of the number of generations required but slows it down in terms of the execution time taken in simulation. Therefore no checking for duplicates was performed.

Seeding of the population with a vector field generated for a previous image pair produced little improvement. Because linear normalisation of fitness was performed it didn’t have considerable negative effects either. Considering this no such seeding was performed and the vector field estimator was allowed to work purely on the image pair at hand. Initialisation of the population is performed by filling with

random vector values on the same range as was used by the block matchers of Chapter 3: -16…+39.

The choice of operator fitness trajectories did have significant effect on the number of generations required for convergence. The trajectories used are plotted in Figure 4.12.

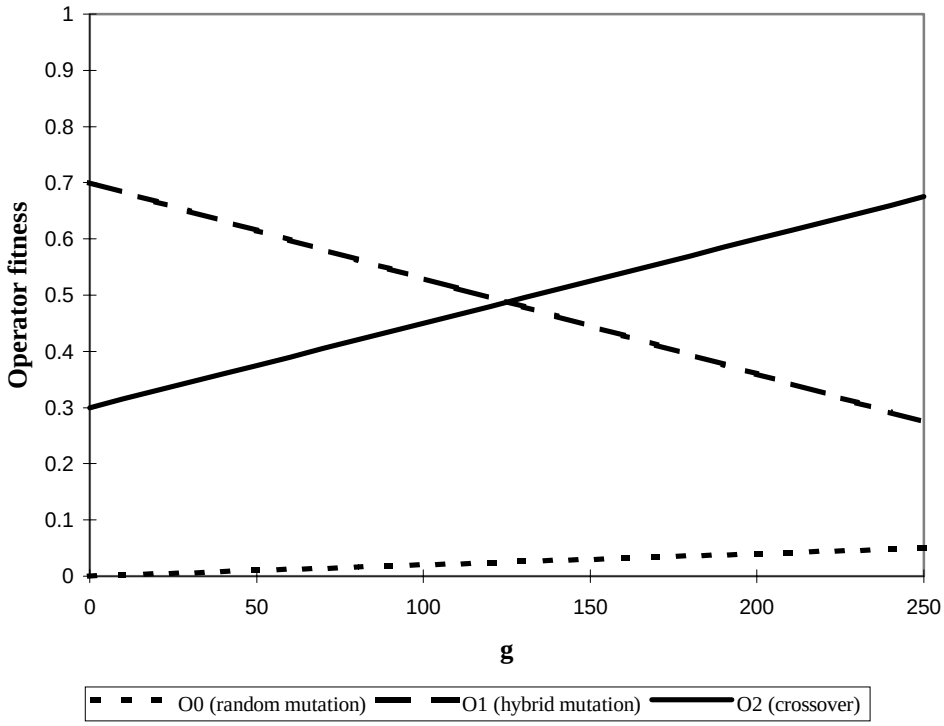

Figure 4.12: Operator fitness trajectories.

Operator fitness of hybrid mutation should start relatively high and end relatively low. The operator fitness of crossover should do just the opposite: start low and end high. In this way good results are obtained on all test material after 250 generations at most.

For comparison with the results presented in Chapter 3 experiments were run on the sequence “mirror” at 8x8 block size on a frame.

The accuracy of the genetic vector field estimator on the sequence mirror (at the various noise levels) is plotted in Figure 4.13. The accuracy of the full search block matcher when there is no noise present is practically equalled. In fact this accuracy is still attained at a 30 dB noise level. At 20 dB the number of correctly estimated vectors drops to the order of 90% but that is still a doubling of what is achieved by the full search estimator.

The vector field entropy is plotted in Figure 4.14. At low noise levels it is slightly lower then the level achieved with either full search or recursive blockmatchers. At the 20 dB noise level however the entropy of the genetically estimated disparity

field is markedly lower then what is achieved with either of the other two block matchers.

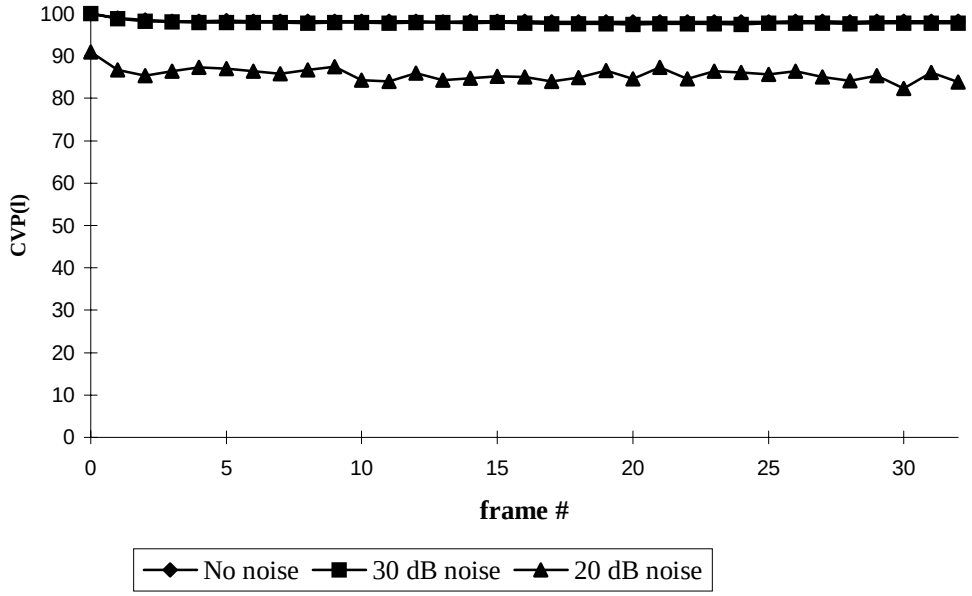

Figure 4.13: Accuracy of genetic vector field estimation.

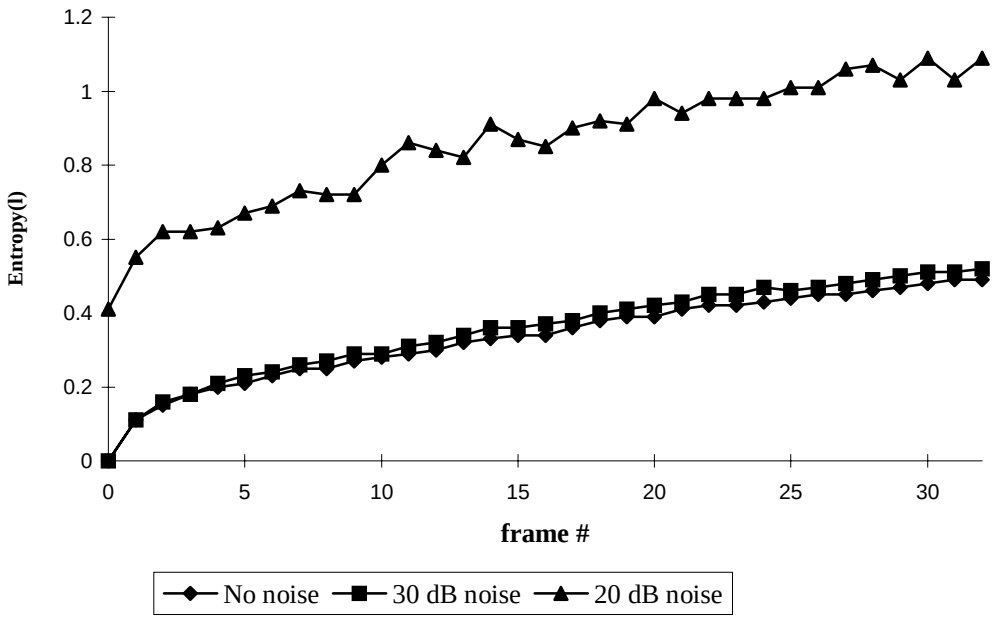

Figure 4.14: Entropy of genetically estimated vector fields.

The disparity field estimated for “aqua” is shown in Figure 4.15. Only near the bottom of the field are there some locations left where single vectors survive from initial random initialisation or random vector field mutation. The vector field shows good correspondence with the actual scene content. The spurious outlier matches of the full search blockmatcher are not present while there are also no convergence

artefacts as happens with the recursive blockmatcher. At 1.8 bit/vector the entropy of this disparity field is lower than the entropy of the vector fields estimated by either full search (3.7 bit/vector) or recursive search (2.7 bit/vector).

Figure 4.15: Genetically estimated disparity field "for aqua".