Chapter 1

CARTESIAN GENETIC PROGRAMMING AND THE POST DOCKING FILTERING PROBLEM

A. Beatriz Garmendia-Doval[1], Julian F. Miller[2], S. David Morley[3]

[1] *Galeon Software LTD, Las Rozas, Madrid, Spain*
beatrizagd@yahoo.co.uk

[2] *Department of Electronics, University of York, Heslington, YO10 5DD, UK*
jfm@ohm.york.ac.uk

[3] *Vernalis (R&D), Cambridge, UK. Present Address: Enspiral Discovery Ltd, Cambridge, UK*
D.Morley@vernalis.com

Abstract Structure-based virtual screening is a technology increasingly used in drug discovery. Although successful at estimating binding modes for input ligands, these technologies are less successful at ranking true hits correctly by binding free energy. This chapter presents the automated removal of false positives from virtual hit sets, by evolving a post docking filter using Cartesian Genetic Programming. We also investigate characteristics of CGP for this problem and confirm the absence of bloat and the usefulness of neutral drift.

Keywords: Cartesian Genetic Programming, Molecular Docking Prediction, Virtual Screening, Machine Learning, Genetic Programming, Evolutionary Algorithms, Neutral Evolution

1. Introduction

In this chapter we present the application of Cartesian Genetic Programming (CGP) to the real-world problem of predicting whether small molecules known as ligands will bind to defined target molecules. We have found CGP to be effective for this problem and it is currently in use in a commercial company. In addition to presenting a successful GP application we have investigated empirically a number of methodological issues that affect the performance and

characteristics of CGP. We have found, in accordance with previous studies of CGP on other problems, that neutral drift (see section 2) in the genotype can be highly beneficial. In addition, unlike some other forms of Genetic Programming we see very little bloat (even though we use many thousands of generations). The chapter consists of eight sections. In section 2 we describe the Cartesian Genetic Programming method and discuss some of its characteristics (some of which led us to adopt the technique). In section 3 we describe the ligand docking problem and how we implemented a CGP system for it. In section 4 we performed a large number of experiments to find optimum parameter settings and investigate how they influence the behaviour of CGP. In section 5 we examine empirically the relative performance and behaviour of an algorithm which utilizes neutral drift with one that doesn't. In section 6 we discuss the evolved post-docking filters and how we selected the best candidates using seeded libraries. In section 7 we examine the evolved filters on real data rather than idealised test sets. We end the chapter with our conclusions in section 8.

2. Cartesian Genetic Programming

Cartesian Genetic Programming (Miller and Thomson, 2000) is a graph based form of Genetic Programming that was developed from a representation for evolving digital circuits (Miller et al., 1997, Miller, 1999). In essence, it is characterized by its encoding of a graph as a string of integers that represent the functions and connections between graph nodes, and program inputs and outputs. This gives it great generality so that it can represent neural networks, programs, circuits, and many other computational structures. Although, in general it is capable of representing directed multigraphs, it has so far only been used to represent directed acyclic graphs. It has a number of features that are distinctive compared with other forms of Genetic Programming. Foremost among these is that the genotype can encode a non-connected graph (one in which it is not possible to walk between all pairs of nodes by following directed links). This means that it uses a many-to-one genotype-phenotype mapping to produce the graph (or program) that is evaluated. The genetic material that is not utilised in the phenotype is analogous to junk DNA. As we will see, mutations will allow the activation of this redundant code or de-activation of it. Another feature is the ease with which it is able to handle problems involving multiple outputs. Graphs are attractive representations for programs as they are more compact than the more usual tree representation since subgraphs can be used more than once.

CGP has been applied to a growing number of domains and problems: digital circuit design (Miller et al., 2000a, Miller et al., 2000b), digital filter design (Miller, 1999), image processing (Sekanina, 2004), artificial life (Rothermich and Miller, 2002), bio-inspired developmental models (Miller and Thomson,

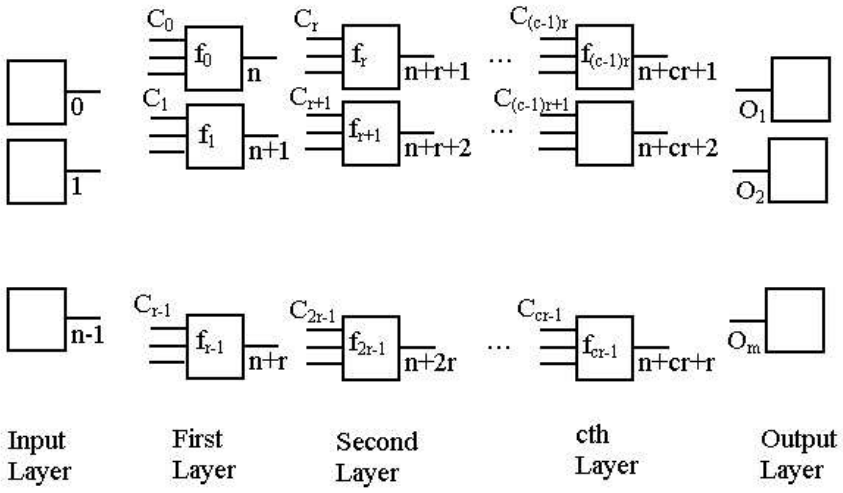

Figure 1.1. General form of Cartesian Program for an n input m-output function. There are three user-defined parameters: number of rows (r), number of columns (c) and levels-back (see text). Each node has a set of C_i connection genes (according to the arity of the function) and a function gene f_i which defines the nodes's function from a look-up table of available functions. On the far left are seen the program inputs or terminals and on the far right the program output connections O_i

2003 Miller, 2003, Miller and Banzhaf, 2003), evolutionary art (Ashmore, 2000) and has been adopted within new evolutionary techniques cell-based Optimization (Rothermich et al., 2003) and Social Programming (Voss, 2003, Voss and James C. Howland, 2003).

In its original formulation CGP was represented as a directed Cartesian grid of nodes in which nodes were arranged in layers (rows) and it was necessary to specify the number of nodes in each row and the number of columns. The nodes in each column were not allowed to be connected together (rather like a multilayer perceptron neural network). In addition an additional parameter was introduced called *level-back* which defined how many columns back a node in a particular column could connect to. The program inputs were arranged in an "input layer" on the left of the array of nodes. This is shown in figure 1.1

It is important to note that in many implementations of CGP (including this one) the number of rows (r) is set to one. In this case the number of columns (c) becomes the *maximum* allowed number of nodes (user defined). Also the parameter *levels-back* can be chosen to be any integer from one (in which case, nodes can only connect to the previous layer) to the maximum number of nodes (in which case a node can connect to *any* previous node). It should be noted that the output genes can be dispensed with by choosing the program outputs to be taken from the m rightmost consecutive nodes (when only one row is used).

The Cartesian genotype (shown below) is a string of integers. C_i denotes in general a set of connection points that the inputs to the node are connected. Each node also has a function, f_i chosen from a list of available functions (defined by the user). Sometimes it happens that the node functions in the function list have different arities (so the cardinality of C_i varies). Usually this is handled (as in this work) by setting the node arity to be the maximum arity that appears in the function list. Nodes with functions that require less inputs than the maximum ignore the extra inputs.

$$C_0, f_0, C_1, f_1, \ldots, C_{cr-1}, f_{cr-1} O_0, O_1, \ldots, O_m$$

If the graphs encoded by the Cartesian genotype are directed then the range of allowed alleles for C_i are restricted so that nodes can only have their inputs connected to either program inputs or nodes from a previous (left) column. Function values are chosen from the set of available functions. Point mutation consists of choosing genes at random and altering the allele to another value provided it conforms to the above restrictions. The number of genes that can be mutated is chosen by the user (usually defined as a percentage of the total number of genes in the genotype). Although the use of crossover is not ruled out, most implementations of CGP (including this one) only use point mutation.

We emphasize that there is no requirement in CGP that all nodes defined in the genotype are actually used (i.e. have their output used in the path from program output to input). This means that there is a many-one genotype phenotype mapping. Although the genotype is of fixed size the phenotype (the program) can have any size up to the maximum number of nodes that are representable in the genotype.It should also be observed that although a particular genotype may have a number of such redundant nodes they cannot be regarded as purely non-coding genes, since mutation may alter genes "downstream" of their position that causes them to be activated and code for something in the phenotype, similarly, formerly active genes can be deactivated by mutation.

When Cartesian genotypes are initialised one finds that many of the nodes are inactive. In many CGP implementations on various problems it is often found that this figure changes relatively little. Thus it is clear that during evolution many mutations have no effect on the phenotype (and hence do not change the fitness of the genotype). We refer genotypes with the same fitness as being neutral with respect to each other. A number of studies (mainly on Boolean problems) have shown that the constant genetic change that happens while the best population fitness remains fixed is very advantageous for search (Miller and Thomson, 2000, Vassilev and Miller, 2000, Yu and Miller, 2001, Yu and Miller, 2002). In the results section of this chapter we will show that such neutral search is also highly beneficial for the ligand docking problem.

To date no work on CGP has required any action to deal with bloat. Bloat is not observed even when enormous genotypes are allowed. Miller investigated (

Miller, 2001) this phenonmenon in CGP and found it to be intimately connected with the presence of genes that can be activated or deactivated. He argued that when the fitness of genotypes is high it becomes more likely that *equally good* genotypes will be favourably selected. In tree-based GP models most equally good phenotypes differ from one another in useless (bloated) code sections, and they will be strongly selected for when fit. This, unfortunately, propagates the spread of such useless code but paradoxically compresses the useful code (Nordin and Banzhaf, 1995). On the other hand, in CGP, the increased proportion of *genetically different but phenotypically identical* code is able to exist without harm (i.e. it does not have to be processed as it is not in the phenotype). It is as if the bloat can exist in the form of genetically redundant code that resides in the genotype (but bounded by the fixed genotype size) but not in the phenotype. This has the side effect of *reducing* the size of the phenotype without requiring any parsimony pressure.

Evolutionary Algorithm

The evolutionary algorithm used for all experiments is that recommended in Miller and Thomson, 2000. It is a simplified (1+4) Evolution Strategy (Schwefel, 1965) for evolutionary search, i.e. one parent with 4 offspring (population size 5). The algorithm is described as follows:

1 Generate initial population of 5 individuals randomly;

2 Evaluate fitness for each individual in the population;

3 Select the best of the 5 in the population as the winner;

4 Carry out point-wise mutation on the winning parent to generate 4 offspring;

5 Construct a new generation with the winner and its 4 offspring;

6 Select a winner from the current population using the following rules:

   (a) If there are offspring that have a better fitness than the parent has, the best offspring becomes the winner.

   (b) *Otherwise, if there are offspring which have the same fitness as the parent then one is randomly selected and becomes the winner (NDEA)*

   (c) else the parent remains the winner.

7 Go to step 4 unless the maximum number of generations has reached.

The evolutionary strategy can be mistaken for a form of hillcimbing. However it should be remembered that the application of the mutation operator

causes a sampling of a whole distribution of phenotypes. A single gene change *can* cause an enormous change in the phenotype, however when the genotype is quite fit, in most cases it will only cause little change (as large change is likely to be deleterious). Thus we can see that the genotype representation in CGP allows a very simple mutation operator to sample a large range of phenotypes. If the neutral drift is not allowed in selection of the genotype to be promoted to the next generation, the step emphasized (NDEA - neutral drift evolutionary algorithm) is removed. We refer to such an algorithm as simply an EA. If this is done, the only way a genotype can supplant its parent is by having a superior fitness. Some have argued that allowing neutral drift is equivalent to using a higher mutation rate in an EA (Knowles and Watson, 2002). In results later we show empirically that this is not the case for the problem studied here, this accords with previous work reported on Boolean problems (Yu and Miller, 2001).

In this chapter, we present evidence later, that shows that fixing the output gene to be the rightmost node is sometimes advantageous. This accords with findings on other problems (Yu and Miller, 2001, Yu and Miller, 2002). It is important to note that CGP is continuing to develop and recently a form of automatically defined functions has been implemented that promises to make the technique more powerful (Walker and Miller, 2004).

3. Docking

Structure-based virtual screening (Lyne, 2002) is an increasingly important technology in the hit identification (identification of compounds that are potentially useful as drugs) and lead optimisation (process of refining the chemical structure of a hit to improve its drug characteristics) phases of drug discovery. The goal of structure-based virtual screening is to identify a set of small molecules (ligands) that are predicted to bind to a defined target macromolecule (protein or nucleic acid). Through the combination of fast molecular docking algorithms, empirical scoring functions and affordable computer farms, it is possible to virtually screen hundreds of thousands or even millions of ligands in a relatively short time (a few days). The output from the docking calculation is a prediction of the geometric binding conformation of each ligand along with a score that represents the quality of fit for the binding site. Only a small fraction of the top-scoring virtual hits (typically up to 1000) then are selected for experimental assay validation. If successful, this virtual hit set will be significantly enriched in bioactive molecules relative to a random selection and will yield a number of diverse starting points for a medicinal chemistry 'hit-to-lead' programme.

Although many factors contribute to the success of virtual screening, a critical component is the scoring function employed by the docking search algorithm.

Whilst reasonably effective at reproducing the binding geometries of known ligands, empirical scores are less successful at ranking true hits correctly by binding free energy. This is a natural consequence of the many approximations made in the interests of high throughput and, as such, all virtual hit sets contain false positives to a greater or lesser extent. Many of these false positives can be removed manually by visual inspection of the predicted binding geometries by an expert computational chemist, but this is a time consuming process.

There have been previous studies that used Genetic Algorithms to improve the coefficients of the scoring function (Smith et al., 2003). Also Böhm (Stahl and Böhm, 1998) developed an empirical postfilter for the docking program FlexX using penalty functions. Here we present the results of our initial attempts to apply Cartesian Genetic Programming techniques to automate the removal of false positives from virtual hit sets.

Virtual Screening

At Vernalis rDock (Morley et al., 2004) and its predecessor RiboDock (Afshar and Morley, 2004) were developed as docking platforms that can rapidly screen millions of compounds against protein and RNA targets.

During docking rDock tries to minimise the total score: $S_{total} = S_{inter} + S_{intra} + S_{restraint}$ where S_{inter} stands for the sum of all the intermolecular scoring functions, S_{intra} is the ligand intramolecular term and $S_{restraint}$ is a penalty term that considers the deviation from certain restraints, for instance when part of the ligand is outside the docking cavity.

Using this score rDock searches for the best conformation for a given ligand over a given docking site. At the end, rDock stores the ligands for which a conformation with a low enough score has been found. These are the ligands that will be considered virtual hits.

Filtering

Once all the hits are found, the value of the score is no longer meaningful. The score is good enough to compare two different conformations of a given ligand, but not good enough to accurately rank order different ligands.

rDock outputs the score and its constituents. rDock also outputs additional descriptors for both the ligand and the target docking site such as molecular weight, number of aromatic rings, charge, number of atoms, etc., that are not used directly during docking. This information is used in an ad hoc manner by the computational chemists to filter out manually the virtual hits, often on a per-target basis, for example to ensure a desired balance between polar and apolar interaction scores. We have explored the use of Genetic Programming techniques to automatically evolve more complex, target-independent post-docking filters (Garmendia-Doval et al., 2003).

Implementation

In our experiments we used a single row of 200 nodes. We chose the levels-back parameter to be 100 and we counted the input variables as the first nodes. All nodes have three inputs and one output. So if their true arity is lower, the extra inputs are ignored. The operations implemented can be seen in table 1.1.

The input to the program is the data returned by rDock. There are components to S_{inter}, S_{intra}, and $S_{restraint}$, ligand descriptors and docking site descriptors. Some of these descriptors are explained in detail in table 1.2.

Apart from the input variables, there is also a pool of 15 constants introduced as program inputs. Each time the CGP is run, 13 of them will be created at random. The other 2 are the constants 0.0 and 1.0. A random constant is equal to $a * 10^b$ where a is a float (with just one decimal place) number between -10 and 10 and b is an integer between -5 and 5.

In total there were 66 input variables, although a given filter did not have to use all of them. On average 10 to 15 variables were used by individual filters.

Training Set

We assembled a set of 163 targets, such that for each of them there is a structure available of the target and of a native ligand, a compound which is experimentally known to dock into that target.

Each of the 163 ligands have been docked against each of the targets. If the scoring function used in docking were perfect, then the lowest (best) score for each target would be obtained for the native ligand bound.

As our current ability to calculate physical properties is quite limited, the native ligand only ranks first in a few cases. Therefore, this cross-docking set contains a large number of false positives. These can be used to drive a genetic program to evolve appropriate filters based on various calculated properties.

From the targets for which the corresponding native ligand ranks in the 9th position or higher, 30 were chosen at random. The training set is then these 30 targets, where the native ligands are considered hits and the ligands with a higher rank are considered misses.

Fitness Function

The CGP system implemented evolves numerical functions. For each input (i.e., docking score components of a given ligand over a given docking site, together with the ligand and docking site descriptors), a float number is returned. It was decided to interpret these numbers in the following manner:

$f < 0$ represents a hit

$f >= 0$ represents a miss

For each protein in the training set there is one hit (native ligand) and a list of size between 0 and 8 with misses (ligands that score better than the native

Name	*#Args*	*Description*
$+$	2	Addition
$-$	2	Subtraction
$*$	2	Multiplication
$/$	2	$div(a,b) = \begin{cases} a & \text{if } \lvert b\rvert < 0.000001 \\ a/b & \text{otherwise} \end{cases}$
log	1	$\log(a) = \begin{cases} 0 & \text{if } \lvert a\rvert < 0.000001 \\ \log(\lvert a\rvert) & \text{otherwise} \end{cases}$
exp	1	$\exp(a) = \begin{cases} 0 & \text{if } a < -200 \\ \exp(200) & \text{if } a > 200 \\ \exp(a) & \text{otherwise} \end{cases}$
if	3	$if(a,b,c) = \begin{cases} b & \text{if } a > 0 \\ c & \text{otherwise} \end{cases}$
Random constant	0	The first time this command is called for a given node, it will create a new random constant. That remains the value of the node for the rest of the program, unless a mutation operator changes the operation of the node.

Table 1.1. Operators

ligand). The fitness function counts the number of proteins for which the hit was recognised as hit ($f < 0$) and at least $\frac{2}{3}$ of the misses were recognised as misses ($f >= 0$).

4. Experiments investigating CGP behaviour with parameter variation

For all the following experiments, the results are the average of 100 runs. Every 500 generations the best individual and its program size was stored. The program size is understood as the phenotype size, i.e., the number of nodes, including the input variables, that are present in the function/program represented by the genotype. We are using the NDEA version of the evolutionary strategy discussed in section 2.

Genome Sizes

Figure 1.2 is a comparison of results with different genome sizes. For all of them, the levels-back parameter was set equal to the genome size.

Examining the plot of average best of population fitness versus number of generations (Figure 1.3) we see that even after 10,000 generations the fitness is still improving. More interestingly we see that the maximum allowed number of nodes provides a good ordering of this behaviour: the larger the allowed number of nodes the higher the average best fitness. However, it looks like much larger genotypes would offer diminishing improvements over smaller,

provided the allowed size is large enough. The growth in phenotype size is fairly rapid initially but settles down to a very small growth. It should be noted that even when 1000 active nodes are allowed the average best size eventually settles at about 43, leaving 957 inactive nodes. Despite this enormous level of redundancy in the genotype we find the evolutionary algorithm described is very effective.

Levels-Back

When the levels-back parameter is varied we see that with low values (25 out of a possible 200) the performance of the algorithm is much poorer and the program size is very much larger. Interestingly we find that intermediate values of local graph connectivity give the best results (levels-back 75 and 100).

Output Node

In the implementation used for the docking problem, the output of the filter was taken to be one of the CGP nodes taken at random. This output node could

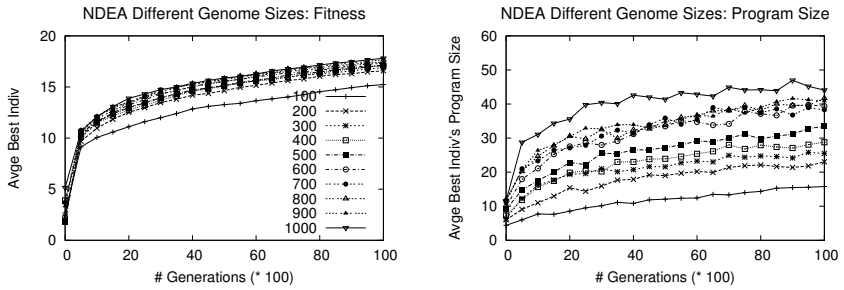

Figure 1.2. Comparison Genome Sizes

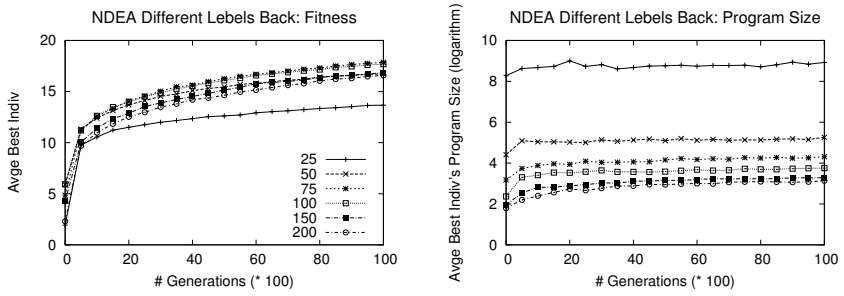

Figure 1.3. Comparison Levels-Back

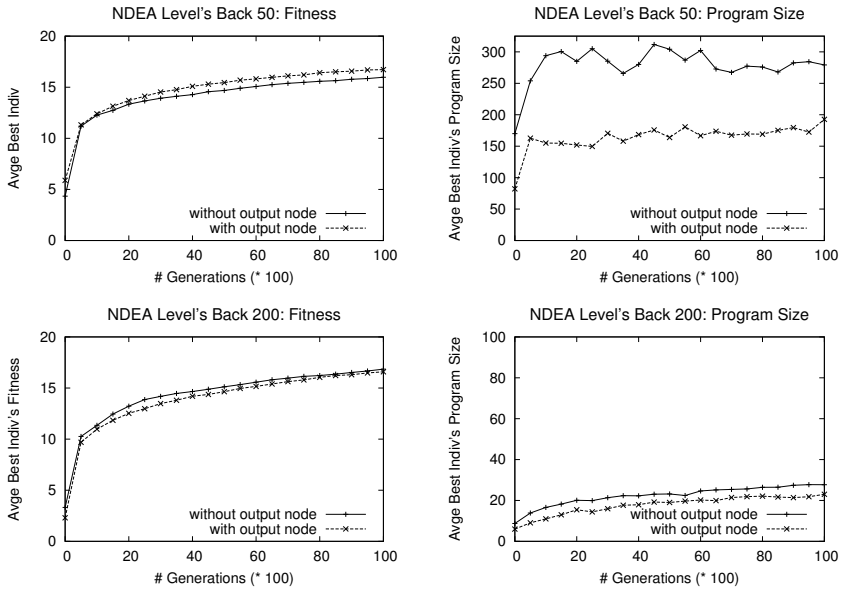

Figure 1.4. Comparison with/without output node: 50, 200

afterwards be mutated during the CGP run. Another option is to take always the last node as the output node without possibility of mutating it, i.e., it is taken out of the genome. A comparison of these two implementations was done using for both of them with 200 nodes, 0.08 mutation rate and NDEA algorithm. A mutation rate of 0.08 means that for every mutation operation, each gene has a 0.08 probability of being mutated. With 200 nodes each one represented by 4 genes (3 inputs and the operator), it means on average 64 genes will be mutated. The parameter levels-back was modified to be 50, 75, 100, 150 and 200. The results for 50 and 200 can be seen in Figure 1.4. It is clear from the results that the performance of the evolutionary algorithm is not greatly affected by whether the program has a fixed output node or whether it is subject to evolution. However having no output gene appears to give better results when the levels-back parameter is large. Even though there is little difference in fitness improvement the average size of the best programs is very different especially with smaller values of levels-back. The weakness of the correlation between fitness improvement and the presence or absence of an output gene was unexpected as it has been found that in Boolean function search the performance is much more reliably good when the program output is taken from the rightmost node. This is because it can sometimes happen by chance that the best individual in the initial population has a small phenotype length. This means that nearly all mutations affect redundant code thus leading

to trapping in a sub-optimum. The output gene is unlikely to hit by mutation and so sometimes one has to wait for many generations for a larger phenotype to be created. The continuous nature of the data may be the reason why the presence or absence of an output gene is of minor importance.

5. Experiments comparing NDEA vs. EA

In the next set of experiments (Figures 1.5) we compare the performance of the evolutionary algorithm with and without neutral drift and also the behaviour of both scenarios with varying amounts of mutation. It is immediately clear that at mutation rates below 0.3 NDEA is superior to the EA. With high mutation rates (>=0.3) the behaviour of the two algorithms is similar both in fitness and program size. Fitness stagnates at about 12 and program size randomly varies around 22 active nodes (out of 200). The behaviour of the NDEA when the mutation rates are much lower is very different. Firstly we see a continuous improvement in fitness with time which is still growing after 10,000 generations. Secondly the improvement in fitness is accompanied by a steady growth in program size. It is interesting that the optimal mutation rate also produces the strongest growth in active code. The rate of code growth drops eventually. This indicates that if evolution was continued longer the active code would stabilize at about 60 nodes (for the best mutation rate). It is also noteworthy that the program growth stabilizes much earlier without neutral drift and that there is much less variation in program sizes (lower variance). The graphs show very clearly that neutral drift is not equivalent to turning neutral drift off and allowing higher mutation rates.

6. Application of NDEA over Docking

In the initial implementation of CGP for the docking problem, a series of experiments were conducted in which system parameters such as the structure of the matrix, mutation rate, etc. were varied, although not in such detail as the experiments shown in sections 4 and 5. At that time it was not possible to conduct very rigorous tests because of the severe time restrictions associated with the business environment, although another reason was caused by this being a classification problem. The fitness function in the CGP implementation is based on the result of applying the current filter on the training set. Since we are considering a classification problem, our aim is to maximize the classification accuracy over the test set. Our goal was not to find the global optimum for the training set as this would have almost surely been equivalent to overfitting and would have produced a filter that would have performed poorly over new data. Because of this, once a system capable of finding good local optima was identified, the system parameters were fixed to be the following: mutation

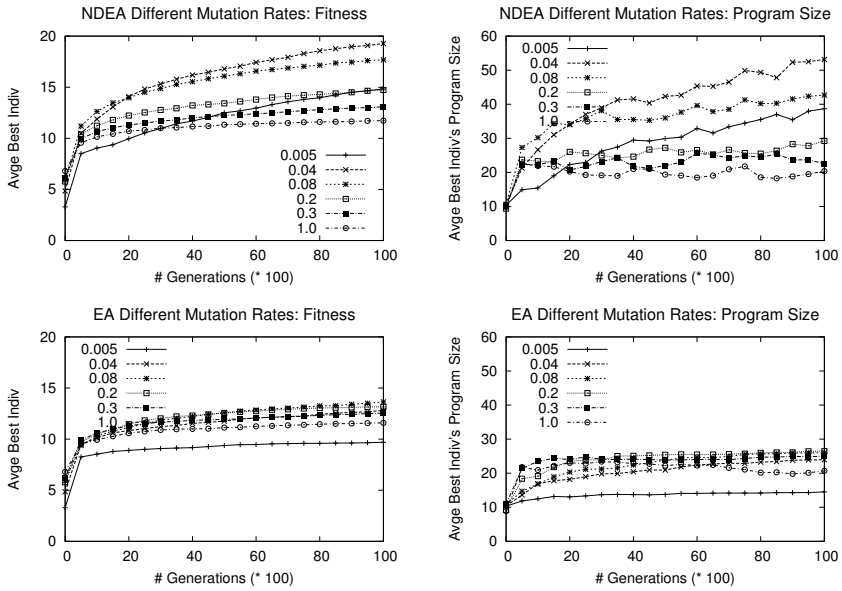

Figure 1.5. Comparison of NDEA vs. EA

rate 0.08, genome size 200 nodes and levels-back 100. From the results of the experiments described one can see that it was a good enough choice.

Test Set

The test set corresponded to the rest of the cross-docking matrix, i.e., the 133 proteins left after removing the 30 that were used for training. The reason for the test set being so much larger than the training set was due to the fact that only half of the matrix was available at the beginning of the project. Once the other half was made available, it was added directly to the test set.

CGP was run several hundred times and the filters that performed best over the test set were chosen. These were then further tested over our validation set.

Seeded Libraries

Seeded libraries are one of a number of methods that have been developed to assess the performance of virtual screening programs. A seeded library for a given target is a library of drug-like compounds that includes several native ligands known to bind to that target. All the compounds are docked into the given target using the docking platform. The output conformations are then sorted by the score. The ability of the virtual screening software to distinguish

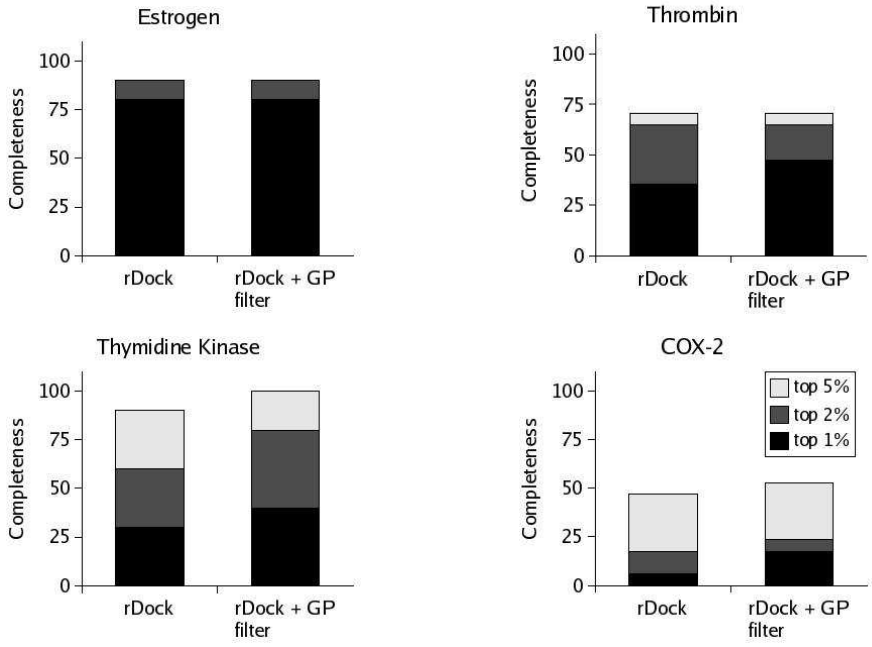

Figure 1.6. Seeded Libraries

between the native ligands and the non native ones can then be measured, typically by calculating the percentage of native ligands found in the top $x\%$.

We tested the best filters over four seeded libraries, and the most promising was chosen. This filter can be seen in figure 1.7. The variables used by this filter are described in table 1.2 and the results obtained in the training set and the test set can be seen in table 1.3.

Figure 1.6 shows the top 1% (black), top 2% (dark gray) and top 5% (light gray) completeness for these four seeded libraries, where completeness is defined as the percentage of true actives retrieved in the slice. We had 10 native ligands for proteins Estrogen and Thymidine Kinase, and 17 for proteins Thrombin and COX-2. The first column shows the results of rDock. The second column shows the results of applying the figure 1.7 filter.

For Estrogen, there is not a great improvement using the filter, as rDock already produces very good results and is therefore difficult to improve upon them.

Thrombin shows a nice improvement as some of the native ligands that were previously found on the top 2% are now found on the top 1%. Similarly for COX-2, all the native ligands found before in the top 2% are now found in the top 1%.

Finally Thymidine Kinase gives the best results as more native ligands are found in the top 1% (40% vs. 30%), more in the top 2% (80% vs. 60%) and again more in the top 5% (100% vs. 90%) where all the native ligands are found.

Best Filter

There were several filters that worked well for the cross-docking matrix but most of them did not generalise well for the seeded libraries. They either filtered out native ligands or, most commonly, they filter almost nothing out. However, we found the following filter to generalise quite well.

```
log(SCORE.INTRA.VDW.raw - 0.9913) *
   exp(SCORE.INTER.AROM.narom *
      exp(SCORE.INTER.POLAR.nhbd) * LIG_POS_CHG)
+ 684 *
   if  SCORE.INTER.REPUL.nhba > 0   then
      LIG_NEG_CHG
   else
      SCORE.INTER.VDW.nrep / SITE_PERC_AROMATOMS
   end
-
 if  (SCORE.INTER.POLAR.nhbd -
     SCORE.INTRA.REPUL.raw + LIG_TOT_CHG )  > 0   then
    SITE_NLIPOC
 else
    exp(SITE_NEG_CHG) - log(LIG_NHBD)
 end
```

Figure 1.7. Best filter found to date

It should be emphasised that the cross-docking matrix is a quite different experiment from the seeded libraries. The fact that this filter is able to filter out true misses while maintaining most of the true hits in both experiments is quite encouraging and is relatively safe to infer that somehow it has found some general trends in the data.

Although it is difficult to understand exactly what the filter is doing, the filter combines intermolecular score components (as used during docking) with both protein and ligand properties in a chemically meaningful way. For example, highly strained conformations (SCORE.INTRA.VDW.raw) and steric clashes between ligand and target (SCORE.INTER.VDW.nrep) are more likely to be rejected.

Finally it should also be noted that the only simplifications done over the original filter output by the CGP program and this filter were replacing the expression $\exp(-0.0087)$ for 0.9913 and the expression $-(900 * -0.76)$ for

Table 1.2. Descriptions of the Variables used by the best filter

variables	description
SCORE.INTRA.VDW.raw	sum of the intraligand van der Waals forces
SCORE.INTRA.REPUL.raw	sum of the intraligand repulsive polar contacts
SCORE.INTER.AROM.narom	number of aromatic rings involve in aromatic interactions
SCORE.INTER.POLAR.nhbd	number of ligand hydrogen bond donors involved in polar interactions
SCORE.INTER.REPUL.nhba	number of ligand hydrogen bond acceptors involved in repulsive polar interactions
SCORE.INTER.VDW.nrep	number of ligand atoms with overall repulsive van der Waals interactions (steric clash)
LIG_NEG_CHG	sum of formal negative charges of the ligand
LIG_NHBD	number of hydrogen bond donors in ligand
LIG_TOT_CHG	total formal charge of the ligand
SITE_PERC_AROMATOMS	percentage of atoms that are aromatic in the target site
SITE_NLIPOC	number of non-polar carbons in the site
SITE_NEG_CHG	sum of formal negative charges of the site

Table 1.3. Results of Best Filter for training set and test set

		Correctly Classified	Incorrectly Classified
Training Set			
	Native Ligands	28	2
	Non-Native Ligands	36	21
Test Set			
	Native Ligands	89	44
	Non-Native Ligands	1946	2417

684. Some parenthesis that were not necessary were also removed to make it more readable. As reported in Miller, 2001, in all the programs found by CGP for this problem, there was "either very weak program bloat or *zero bloat*"

7. Results with Real Data

All the previous results shown were obtained over idealised test sets used routinely to measure docking performance. As a final validation we have applied the filter in figure 1.7 to real virtual screening data from docking campaigns performed at Vernalis, specifically against an oncology target protein, HSP90.

From an initial docking library of around 700000 compounds, a total of around 40000 virtual hits were identified over several docking campaigns against HSP90. Around 1500 of the virtual hits were selected by a computational chemist for experimental assay using a variety of ad hoc post filters, and knowledge and experience of the target protein, in a process taking around a week.

Table 1.4. HSP90

	Manual post-docking hits	+GP post-docking filter	Reduction
rDock virtual hits	39908	28374	−29%
Compounds assayed	1467	1409	−4%
True actives	30	27	−10%

Thirty of the assayed compounds were confirmed as real hits, in that they showed significant activity against HSP90.

The filter shown in figure 1.7 was applied to the virtual hits (see Table 1.4) and was able to remove 29% of the original unfiltered hits, whilst only removing 4% of the compounds manually selected for assay. Three of the true actives were also removed.

The GP-derived filter therefore shows very good agreement with the manual filtering process, in that the filter passes almost all of the original assayed compounds, but is able to reduce automatically the initial size of the data set by almost 30%. This provides further evidence that the filter is generalising across docking targets quite distinct from those in the training and test sets.

The filter is currently being used and tested with each new docking campaign, with very good results. It promises to be a useful additional tool in the computational chemist's armoury of post-docking filters.

8. Conclusions

Removal of false positives after structure-based virtual screening is a recognised problem in the field. This chapter describes what we believe is the first attempt at using Genetic Programming to evolve a post-docking filter automatically. We found the simple 1+4 evolutionary strategy with neutral drift to be very effective and also confirmed that for this real world problem, program bloat was not a problem.

The cross docking matrix used for training and evolving post-docking filters is quite different from the seeded libraries and the HSP90 data. The post-docking filter chosen from the ones found by the GP platform is filtering out consistently bad compounds in all cases, while retaining interesting hits. We can say that it is generalising over the data. The HSP90 data is the first real data on which the filter has been tested and the results are very promising. This

filter is now being used as standard in all the projects in the company. Early results confirm its usefulness.

The GP platform offers immediately a pragmatic, automated post-docking filter for cleaning up virtual hit sets. It can be easily applied again for different descriptors or scoring functions.

Longer-term the filters found may offer a way of "boot-strapping" docking scoring function improvements, by identifying non-obvious, yet systematic, defects in the scoring function.

This technique is also not specific to docking programs, and we plan to apply it in the near future for other problems where a list of variables and descriptors is available and there is a need for a generic filter.

References

Afshar, Mohammad and Morley, S. David (2004). Validation of an empirical rna-ligand scoring function for fast flexible docking using ribodock(r). J. Comput.-Aided Mol. Design, accepted.

Ashmore, Laurence (2000). An investigation into cartesian genetic programming within the field of evolutionary art. Technical report, Final year project, http://www.gaga.demon.co.uk/evoart.htm, Department of Computer Science, University of Birmingham.

Garmendia-Doval, A. Beatriz, Morley, S. David, and Juhos, Szilvester (2003). Post docking filtering using cartesian genetic programming. In Liardet, P., Collet, P., Funlupt, C., Lutton, E., and Schoenauer, M., editors, *Proceedings of the 6th International Conference on Artificial Evolution*, pages 435–446.

Knowles, Joshua D. and Watson, Richard A. (2002). On the utility of redundant encodings in mutation-based evolutionary search. In J.-J. Merelo Guervós, P. Adamidis, H.-G. Beyer, J.-L. Fernández-Villacañas, H.-P. Schwefel, editor, *Parallel Problem Solving from Nature - PPSN VII, 7th International Conference, Granada, Spain, September 7-11, 2002. Proceedings*, number 2439 in Lecture Notes in Computer Science, LNCS, page 88 ff. Springer-Verlag. Keywords: Technique::Comparisons of representations, Theory of EC::Fitness landscape.

Lyne, Paul D. (2002). Structure-based virtual screening: an overview. *Drug Discovery Today*, 7(20):1047–1055.

Miller, Julian (2001). What bloat? cartesian genetic programming on boolean problems. In Goodman, Erik D., editor, *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 295–302, San Francisco, California, USA.

Miller, Julian F. (1999). An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In Banzhaf, Wolfgang, Daida, Jason, Eiben, Agoston E., Garzon, Max H., Honavar, Vasant,

Jakiela, Mark, and Smith, Robert E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1135–1142, Orlando, Florida, USA. Morgan Kaufmann.

Miller, Julian F. (2003). Evolving developmental programs for adaptation, morphogenesis, and self-repair. In Banzhaf, Wolfgang, Christaller, Thomas, Dittrich, Peter, Kim, Jan T., and Ziegler, Jens, editors, *Advances in Artificial Life, ECAL 2003, Proceedings*, volume 2801 of *Lecture Notes in Artificial Intelligence*, pages 256–265. Springer.

Miller, Julian F. and Banzhaf, Wolfgang (2003). Evolving the program for a cell: from french flags to boolean circuits. In Kumar, Sanjeev and Bentley, Peter J., editors, *On Growth, Form and Computers*, pages 278–301. Academic Press.

Miller, Julian F., Job, Dominic, and Vassilev, Vesselin K. (2000a). Principles in the evolutionary design of digital circuits-part I. *Genetic Programming and Evolvable Machines*, 1(1/2):7–35.

Miller, Julian F., Job, Dominic, and Vassilev, Vesselin K. (2000b). Principles in the evolutionary design of digital circuits-part II. *Genetic Programming and Evolvable Machines*, 1(3):259–288.

Miller, Julian F. and Thomson, Peter (2000). Cartesian genetic programming. In Poli, Riccardo, Banzhaf, Wolfgang, Langdon, William B., Miller, Julian F., Nordin, Peter, and Fogarty, Terence C., editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 121–132, Edinburgh. Springer-Verlag.

Miller, Julian F. and Thomson, Peter (2003). A developmental method for growing graphs and circuits. In Tyrrell, Andy M., Haddow, Pauline C., and Torresen, Jim, editors, *Evolvable Systems: From Biology to Hardware, Fifth International Conference, ICES 2003*, volume 2606 of *LNCS*, pages 93–104, Trondheim, Norway. Springer-Verlag.

Miller, Julian F., Thomson, Peter, and Fogarty, Terence (1997). Designing electronic circuits using evolutionary algorithms arithmetic circuits: A case study. In Quagliarella, D., Périaux, J., Poloni, C., and Winter, G., editors, *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science. Recent Advances and Industrial Applications*. John Wiley and Sons.

Morley, S. David, Juhos, Szilveszter, and Garmendia-Doval, A. Beatriz (2004). in preparation.

Nordin, Peter and Banzhaf, Wolfgang (1995). Complexity compression and evolution. In Eshelman, L., editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, Pittsburgh, PA, USA. Morgan Kaufmann.

Rothermich, Joseph A. and Miller, Julian F. (2002). Studying the emergence of multicellularity with cartesian genetic programming in artificial life. In Cantú-Paz, Erick, editor, *Late Breaking Papers at the Genetic and Evolution-*

ary Computation Conference (GECCO-2002), pages 397–403, New York, NY. AAAI.

Rothermich, Joseph A., Wang, Fang, and Miller, Julian F. (2003). Adaptivity in cell based optimization for information ecosystems. In Press, IEEE, editor, *Proceedings of the 2003 Congress on Evolutionary Computation*, pages 490–497, Camberra.

Schwefel, H. P. (1965). Kybernetische evolution als strategie der experimentelen forschung in der stromungstechnik. Master's thesis, Technical University of Berlin.

Sekanina, Lukas (2004). *Evolvable Components: From Theory to Hardware Implementations*. SpringerVerlag.

Smith, Ryan, Hubbard, Roderick E., Gschwend, Daniel A., Leach, Andrew R., and Good, Andrew C. (2003). Analysis and optimization of structure-based virtual screening protocols (3). new methods and old problems in scoring function design. *J. Mol. Graphics Mod.*, 22:41–53.

Stahl, Martin and Böhm, Hans-Joachim (1998). Development of filter functions for protein-ligand docking. *J. Mol. Graphics Mod.*, 16:121–132.

Vassilev, Vesselin K. and Miller, Julian F. (2000). The advantages of landscape neutrality in digital circuit evolution. In *Proceedings of the Third International Conference on Evolvable Systems*, pages 252–263. Springer-Verlag.

Voss, Mark S. (2003). Social programming using functional swarm optimization. In *IEEE Swarm Intelligence Symposium (SIS03)*.

Voss, Mark S. and James C. Howland, III (2003). Financial modelling using social programming. In *FEA 2003: Financial Engineering and Applications*, Banff, Alberta.

Walker, James A. and Miller, Julian F. (2004). Evolution and acquisition of modules in cartesian genetic programming. In Keijzer, Maarten, O'Reilly, Una-May, Lucas, Simon M., Costa, Ernesto, and Soule, Terence, editors, *Proceedings of the Seventh European Conference on Genetic Programming*, volume 3003 of *LNCS*, pages 187–197. Springer-Verlag.

Yu, Tina and Miller, Julian (2001). Neutrality and the evolvability of boolean function landscape. In Miller, Julian F., Tomassini, Marco, Lanzi, Pier Luca, Ryan, Conor, Tettamanzi, Andrea G. B., and Langdon, William B., editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 204–217, Lake Como, Italy. Springer-Verlag.

Yu, Tina and Miller, Julian F. (2002). Needles in haystacks are not hard to find with neutrality. In Foster, James A., Lutton, Evelyne, Miller, Julian, Ryan, Conor, and Tettamanzi, Andrea G. B., editors, *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, volume 2278 of *LNCS*, pages 13–25, Kinsale, Ireland. Springer-Verlag.