Methods to Evolve Legal Phenotypes

Tina Yu and Peter Bentley

Department of Computer Science, University College London,
Gower Street, London WC1E 6BT, UK.
T.Yu@cs.ucl.ac.uk P.Bentley@cs.ucl.ac.uk

Abstract. Many optimization problems require the satisfaction of constraints in addition to their objectives. When using an evolutionary algorithm to solve such problems, these constraints can be enforced in many different ways to ensure that legal solutions (phenotypes) are evolved. We have identified eleven ways to handle constraints within various stages of an evolutionary algorithm. Five of these methods are experimented on a run-time error constraint in a Genetic Programming system. The results are compared and analyzed.

1. Introduction

Constraints form an integral part of every optimization problem, and yet they are often overlooked in evolutionary algorithms (Michalewicz, 1995b). A problem with constraints has both an objective, and a set of restrictions. For example, when designing a VLSI circuit, the objective may be to maximize speed and the constraint may be to use no more than 50 logic gates. It is vital to perform constraint handling with care, for if evolutionary search is restricted inappropriately, the evolution of good solutions may be prevented.

In order to explore the relationship between constraints and evolutionary algorithms, this paper presents an evolutionary framework in which the search space and solution space are separated. In this framework, a genotype represents a point in the search space and is operated on by the genetic operators (crossover and mutation). A phenotype represents a point in solution space and is evaluated by the fitness function. The result of the evaluation gives the fitness of the phenotype, and by implication, of the underlying genotype.

In the same way that phenotypes are evaluated for fitness, not genotypes, it is the phenotypes which must satisfy the problem constraints, not the genotypes (although their enforcement may result in the restriction of some genotypes). However, unlike the fitness evaluation, constraints can be enforced at any point in the algorithm to attain legal phenotypes. As will be described later, they may be incorporated into the genotype or phenotype representations, during the seeding of the population, during reproduction, or handled at other stages.

There are two main types of constraint: the *soft constraint* and the *hard constraint*. Soft constraints are restrictions on phenotypes that should be satisfied, but will not always be. Such constraints are often enforced by using penalty values to lower fitnesses. Illegal phenotypes (which conflict with the constraints) are permitted to exist as second-class, in the hope that some portions of their genotypes will aid the search for fit phenotypes (Michalewicz, 1995b). Hard constraints, on the other hand, must always be satisfied. Illegal phenotypes are not permitted to exist (although their corresponding genotypes may be, as will be shown).

This paper identifies eleven methods to enforce constraints on phenotypes during various stages of evolutionary algorithms. Five methods are experimented on a run-time error constraint in a Genetic Programming (GP) system. The results are compared and analyzed.

The paper is structured as follows: section 2 provides related work; section 3 classifies and describes the constraint handling methods; section 4 presents the experiments; section 5 analyzes the results and section 6 concludes.

2. Related Work

Genetic Algorithms: Michalewicz and Schoenauer provide perhaps the most comprehensive reviews of implementations of constraint handling in genetic algorithms (GAs) (Michalewicz 1995b, Michalewicz & Schoenauer 1996). They identify and discuss eleven different types of system. However, upon examination it is clear that his classification is based upon differences in implementation, and perhaps because of confusion of various multiobjective techniques, it fails to group constraint handling methods which employ similar underlying concepts. Nevertheless, the work of Michalewicz and colleagues provides some of the key investigations in this area. For example, Michalewicz (1995a) describes the application of five methods (three based on penalizing illegal phenotypes) to five test functions. Michalewicz et. al. (1996) describe the use of ‘behavioral memory’ and other penalty-based approaches in GAs to evolve different engineering designs. Schoenauer and Michalewicz (1997) describe the use of a repair method in a GA to evolve legal phenotypes.

Evolution Strategies & Evolutionary Programming: In their original implementations, both ES and EP performed constraint handling during the creation of the initial populations. Schwefel's ES algorithm also used a 'legal mutant' constraint handling method, where the creation of an individual is simply repeated as long as the individual violates one or more constraints (Bäck, 1996). Standard EP, on the other hand, typically does not enforce constraints during the generation of new offspring. More recent research on constrained optimization problems in EP is described in (McDonnell et al., 1995) and (Fogel et al., 1996).

Genetic Programming: The traditional GP paradigm (Koza, 1992) does not distinguish genotypes from phenotypes, i.e. the search space is regarded as being the same as the solution space. An individual is represented as a program parse tree. This parse tree represents both the genotype and phenotype of an individual as it is modified by the genetic operators and it is evaluated by the fitness function. Consequently, constraints in traditional GP are perceived as being applied to phenotypes and genotypes.

For example, program parse trees in GP are restricted by syntactic constraints: they must satisfy the syntax of the underlying language. Various other forms of syntactic constraints have been proposed (Gruau, 1996; Janikow, 1996). Yu and Clack (1998) applied both syntactic constraints and type constraints in their GP system.

Banzhaf (1994) proposed an alternative paradigm for GP, where the search space is separated from the solution space. A mapping scheme is used to transform genotypes into legal phenotypes (Keller & Banzhaf, 1996).

3. Constraints in Evolutionary Algorithms

Just as evolution requires selection pressure to generate phenotypes that satisfy the objective function, evolution can have a second selection pressure placed upon it in order to generate phenotypes that do not conflict the constraints. However, using *pressure* in evolutionary search to evolve legal solutions is no guarantee that all of the solutions will always be legal (i.e., they are soft constraints).

Constraints can also be handled in two other ways: solutions that do not satisfy the constraints can be *prevented* from being created, or they can be *corrected*. Such methods can have significant drawbacks such as loss of diversity and premature convergence. Nevertheless, these two types of constraint handling ensure that all solutions are always legal (i.e., they are hard constraints). The following section identifies eleven methods which enforce 'hard constraints' or 'soft constraints'. These methods also fall within the three conceptual categories: Prevention, Correction, and Pressure, see Table 1. (Note that this categorization encompasses the Pro-Life, Pro-Choice categorization of Michalewicz and Michalewicz (1995). It is felt that the use of more neutral terminology is more appropriate for such technical classifications.)

Prevention	HARD	C1, C2, C3, C10
Correction	HARD	C4, C5
Pressure	SOFT	C6, C7, C8, C9, C11

Table 1. Classification of constraint handling methods.

3.1 Detailed Classification

Whilst previous work in classifying constraint handling methods within evolutionary search has identified implementation differences of existing systems, to date there has not been a general classification of constraint handling based on the underlying concepts of evolutionary algorithms.

Such a classification can be achieved, not only by examining the existing work of others, but also by examining the significant stages within evolutionary algorithms and identifying where it is *possible* to incorporate constraints. This allows all existing constraint handling methods to be clearly categorized and understood, and also identifies new, previously unexamined ways of tackling constraints in evolutionary search. Figure 1 shows the most significant and commonly used stages within current GAs and GP. (It should be noted that most algorithms contain a subset of these stages.)

After some careful consideration of these stages, it becomes clear that constraints can be incorporated at eleven different places within the design and execution of evolutionary algorithms (as shown on the right hand side of Figure 1). These eleven methods should not be confused with Michalewicz's (1995b) list of different researchers' implementations (which coincidentally also contains eleven elements). The methods shown in Figure 1 are categorized solely on their placement within the evolutionary algorithm, and can be used in combination or separately of each other. There follows a description of each method and its potential advantages and disadvantages:

C1: LEGAL SEARCHSPACE ***Design genotype representation.***
During the design of the evolutionary system, create a genotype representation that is only capable of representing legal solutions. Evolutionary search is then forced to consider only the space of legal solutions, where all constraints are satisfied. This

method is frequently used, although designers who use it are often unaware that they are performing constraint handling of any kind. For example, in GAs, if the range of a problem parameter must be between 0 and 255, most designers would automatically use a binary gene consisting of eight bits - and this genotype representation would then ensure that the 0-255 range constraint was always satisfied.

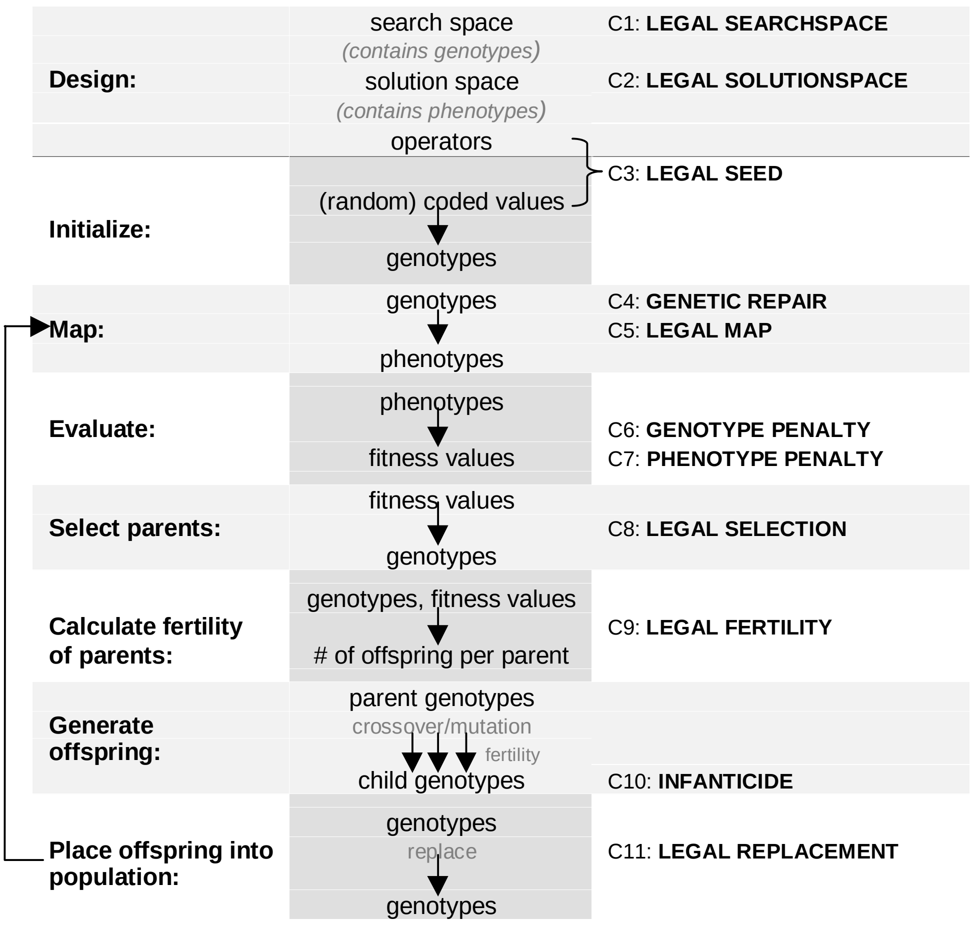

Fig. 1. Constraint placement within significant stages of evolutionary algorithms.

C2: LEGAL SOLUTIONSPACE ***Design phenotype representation.***
During the design of the evolutionary system, create a new phenotype representation, so that only legal phenotypes can be defined. All genotypes are then mapped onto these phenotypes, which by definition, must always satisfy the constraints.

Often great care can go into the design of suitable phenotypes. For example, practitioners of floor-planning problems have two important constraints: room-spaces should not overlap, and no space should be left unaccounted for. To ensure that the computer always evolves solutions that satisfy these constraints, designers of these systems use phenotype representations which define the location of rooms indirectly, by defining the location and number of dividing walls (Gero & Kazakov, 1998).

C3: LEGAL SEED ***Seed with non-conflicting solutions.***
The initial population is seeded with solutions that do not conflict with the constraints and the crossover and mutation operators are designed so that they cannot generate illegal solutions. Many constraints in GP are implemented using this method. For example, Gruau (1996) uses a context-free grammar to specify syntactic constraints of parse trees. Yu and Clack (1998) employ a type system to ensure that only type-correct programs are considered during evolution.

C4: GENETIC REPAIR ***Correct illegal genotypes.***
If a new individual conflicts with a constraint, correct the genes that are responsible for the conflict to make it satisfy that constraint. For algorithms such as GP which make no explicit distinction between genotypes and phenotypes, this method modifies the solution, *and the modification is inherited by its offspring*.

This ‘genetic engineering’ approach ensures that all solutions will satisfy all constraints, but may damage epistatic genotypes, discarding the result of careful evolution over many generations. In addition, the design of the repair procedure may be a non-trivial task for some problems.

C5: LEGAL MAP ***Correct illegal phenotypes.***
Map every genotype of an individual to a phenotype that satisfies the constraints using some form of simple embryology. This forces all solutions to satisfy all constraints, and also does not disrupt or constrain the genotypes in any way, allowing evolutionary search to continue unrestricted. For algorithms such as GP which make no distinction between genotypes and phenotypes, this method modifies the solution before fitness evaluation, *but the modification is not inherited by its offspring*. (Also note that although this method is often used in combination with C2, the use of a phenotype representation which can only represent legal solutions is not a prerequisite for the use of Legal Map.)

Using a mapping stage to generate legal phenotypes is a very common approach to perform simple constraint handling. Goldberg (1989) describes perhaps the simplest: mapping the range of a gene to a specified interval. This permits constraints on parameter range and precision to be satisfied without the need to redesign the genotype representation and coding. More recently mapping stages have become more intricate and deserving of the term 'artificial embryology'. Researchers in GP have also reported that the use of an explicit genotype and mapping stage for constraint handling can increase diversity in populations (Banzhaf, 1994).

Type constraints in GP can be implemented using this method as an alternative to the Legal Seed method. A simple example is to map a value with an illegal type of 'real' into a value with legal type 'integer'. However, for other more complex types such as list or array, a proper mapping scheme may be difficult to design. This kind of type-constraint handling is called 'dynamic typing' - in contrast to the 'strong typing' approach mentioned in the Legal Seed method.

C6: GENOTYPE PENALTY ***Penalize illegal genotypes.***
Identify alleles or gene fragments within genotypes that seem to increase the chances of a solution conflicting the constraints, and reduce the fitness of any individual containing these fragments of genetic code. Although the identification of ‘bad genes’ may discourage solutions from conflicting constraints, it will not guarantee that all

solutions satisfy all constraints. In addition, with epistatic genotypes, this approach may result in the discouragement of other, epistatically linked, useful features within solutions. To date, research has investigated the automatic identification of ’good genes’ during evolution to encourage the evolution of solutions with higher fitnesses (Gero & Kazakov, 1998). However, the authors of this paper are unaware of any work which identifies ‘bad genes’ for constraint handling.

C7: PHENOTYPE PENALTY ***Penalize illegal phenotypes.***
When a phenotype conflicts a constraint, reduce its fitness. This ‘soft constraint’ discourages all phenotypes that conflict the constraints, but does not force evolutionary search to generate legal solutions. In effect, the use of a penalty value becomes an additional criterion to be considered by the evolutionary algorithm, and multiobjective techniques should be used to ensure that all criteria are considered separately (otherwise one or more criteria may dominate the others) (Bentley & Wakefield, 1997). This is one of the most commonly used methods for constraint handling in evolutionary algorithms. (Indeed, it is the only one explicitly mentioned in Goldberg’s book.)

C8: LEGAL SELECTION ***Select only legal parents for reproduction.***
During reproduction, only select parent solutions which satisfy the constraints. This method should be used with a fitness-based replacement method to ensure that evolution is guided to evolve fit solutions in addition to legal solutions. (If all solutions are illegal, parents which violate the fewest constraints to the least extent should be selected.) However, the exclusion of potential parents may discard beneficial genetic material and so could be harmful to evolution. Other than the work described in this paper, only one recent investigation has been made on this method (Hinterding & Michalewicz, 1998).

C9: LEGAL FERTILITY ***Increase the no. of offspring for legal parents.***
Having selected the parent genotypes (based on their fitnesses) this method allocates a larger *fertility* to parents which better satisfy the constraints. This method can be thought of as an implicit multiobjective method, allowing independent selection pressure to be exerted for high fitness and legal solutions. Being a 'soft constraint', there are no guarantees that all solutions will always satisfy the constraints. In addition, if legal parents are favoured excessively, it is possible that the diversity of the population could be reduced. To the authors' knowledge, this idea has not been previously used for constraint handling.

C10: INFANTICIDE ***Stop illegal offspring from being born.***
If a new solution conflicts a constraint, discard it, and try generating another solution using the same parents. This brute-force method, which is sometimes used in GAs (Michalewicz, 1995b), forces all solutions to satisfy the constraints, but may discard useful genetic material (and may also be prohibitively slow).

C11: ILLEGAL REPLACEMENT ***Replace illegal solutions with legal offspring.***
When replacing individuals with new offspring in the population, always replace the solutions that conflict constraints. (If all solutions satisfy the constraints, either replace randomly or replace the least fit.) This method should be used with a fitness-based selection method to ensure that evolution is guided to evolve fit solutions in addition to legal solutions. However, the replacement of potential parents discards potentially beneficial genetic material and so may be harmful to evolution. This method requires the use of a steady-state GA (Syswerda, 1989).

4. Experiments with a Run-Time Constraint in GP

This section describes experiments conducted to compare five of the constraint handling methods described above in a GP system. The experiments are focused on one particular kind of constraint in GP: the *run-time error constraint*.

GP evolves computer programs as problem solutions. Thus, in most cases the genetic material is in some sense executable. When run-time errors occur during the execution of a program, its behaviour is undefined. A fundamental constraint is therefore imposed on GP: no programs can contain run-time errors.

Unlike other types of constraint, the run-time error constraint has a special property: when it occurs the fitness cannot be calculated. (When the behaviour of the program is undefined, the evaluation of its fitness cannot be performed.) Illegal phenotypes are therefore not allowed to exist. This means that soft constraint methods (where illegal phenotypes can exist as second-class) can only be used in conjunction with a phenotype correction method - they cannot be used on their own. In the experiments, the Legal Map method is used to serve this purpose.

A constraint can be handled using many different methods, yet some are more suitable than others. For the run-time error constraint, its prevention (in methods C1, C2, C3 and C10) is extremely hard because these errors are only evident during program execution. In addition, genetic repair (method C4) requires the corrected material to follow the genotype syntax (so that it can be inherited) which is not appropriate (or easy to implement) for this constraint. Consequently, none of the hard constraint methods are suitable for this problem except for the Legal Map method (C5), which corrects illegal phenotypes (and the corrections are not inherited by offspring).

Soft constraint approaches, on the other hand, are appropriate for this problem. The experiments investigate four of these methods (C7, C8, C9, and C11). (Method C6 which penalizes illegal genotypes by identifying ‘bad genes’ was not investigated because of the substantial time required for its implementation).

Objective:	Find the symbolic function $x^4 - x^3 + x^2 - x$ using 9 pairs of sample points.
Terminal Set:	x
Function Set:	+, -, *, /
Fitness Cases:	9 data points (x_i,y_i) where x_i is the input value between -1.0 and 1.0 and y_i is the desired output value.
Fitness:	9/(9+*total_error*), where *total_error* is $\sum_{i=1}^{9} \lvert y_i - R_i \rvert$ and R_i is the result of phenotype execution given input x_i
Hits:	$\sum_{i=1}^{9} p_i \left(p_i = \begin{cases} 1, if \lvert y_i - R_i \rvert \le 0.01 \\ 0, otherwise \end{cases} \right)$
Parameters:	PopSize = 500, MaxTest = 25500, TreeSize = 25, Xover = 60%, Mutation = 4%, Copy = 36%, Runs = 20
Success predicate:	9 hits

Table 2. Tableau of the simple symbolic regression problem

In summary, the experiments investigate one hard constraint-handling method (Legal Map) and four soft constraint methods (Phenotype Penalty, Legal Selection, Legal Fertility, and Illegal Replacement) to enforce the zero-division run-time error constraint. The zero-division constraint was chosen as it is the most frequently observed run-time error, potentially occurring in any numerical problem tackled by GP.

The experiments use GP to solve a symbolic regression problem, which involves finding a function, in symbolic form (with numeric coefficients) that fits a given finite sample of data. It is "data-to-function" regression. The goal is to find the target function of x^4-x^3+x^2-x, given a data sample of nine pairs (x_i, y_i), where x_i is a value of the independent variable and y_i is the associated value of the dependent variable. Table 2 summarizes the features of this problem.

4.1 Implementation of Constraints

To allow the use of the Illegal Replacement method, the GP system uses a steady-state replacement scheme (Syswerda, 1989) where a population with a constant number of individuals is maintained. Unless otherwise stated, parents are selected using fitness proportionate selection, and offspring replace individuals with the worst fitness in the population. The five constraint handling methods were implemented as follows:

C5: Legal Map. When a run-time error occurs during the execution of a phenotype, the value ‘1’ is returned and the execution continues. For example, if the phenotype is 5+x/x and x = 0.0, *Legal Map* changes the phenotype to: 5+1. Corrected phenotypes are marked with a run-time error flag to allow this method to be used in conjunction with the following four.

C7 & C5: Phenotype Penalty with Legal Map. Phenotypes that have to be corrected are penalized by multiplying their *total_error* values by 2. Legal phenotypes that do not have to be corrected are not penalized.

C8 & C5: Legal Selection with Legal Map. During the selection of parents for reproduction, only programs without run-time errors are selected (randomly).

C9 & C5: Legal Fertility with Legal Map. If both parents are legal, three offspring are generated from them. If one parent is legal, two offspring are generated, and if neither of the parents is legal, only one offspring is generated from them.

C11 & C5: Illegal Replacement with Legal Map. One offspring (legal/illegal) is generated to replace a randomly selected illegal individual. If there is no illegal individual left in the population, the normal replacement scheme is used.

4.2 Results

Twenty runs were performed for each constraint handling method. Each run was terminated when a program which produced nine hits was found (i.e., when the evolved function produced output sufficiently close to the desired output for all nine data points) or when 25,500 programs had been processed. If the former occurs, the run is termed *successful*. Table 3 summarizes the experiment results.

The experiments show that Legal Map, Phenotype Penalty with Legal Map and Legal Fertility with Legal Map find a phenotype with nine hits in most of the runs (18/20 and 20/20). For the successful runs, the average number of programs tested is around 4,000. In contrast, Legal Selection with Legal Map and Illegal Replacement with Legal Map methods do not perform well. Most of the runs are unsuccessful and in the small number of successful runs, they have to test a larger number of phenotypes to find one with nine hits.

Method	**Success/Runs**	**Average Number of Programs Processed in Successful Runs**
Legal Map	18/20	3,983
Phenotype Penalty & Legal Map	18/20	4,841
Legal Selection & Legal Map	5/20	18,284
Legal Fertility & Legal Map	20/20	3,984
Legal Replacement & Legal Map	3/20	6,998

Table 3. Summary of experiment results

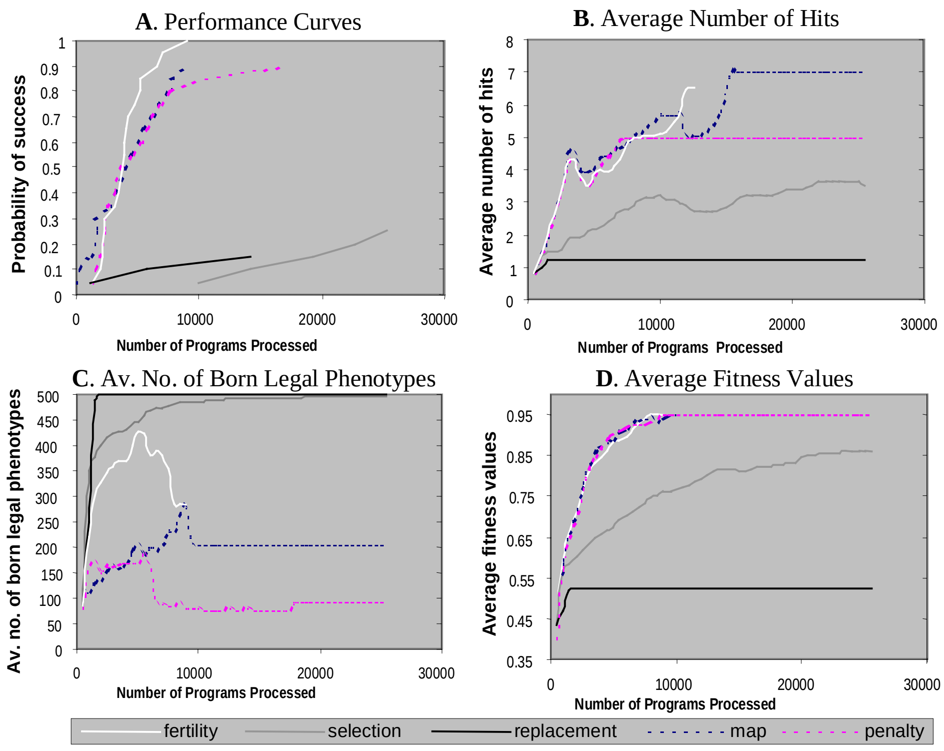

Fig. 2. Result summary charts.

Figure 2A provides the probability of success of each method based on the experiments. The Legal Map, Phenotype Penalty with Legal Map and Legal Fertility with Legal Map methods all perform comparably. Their success curves increase stability from the beginning. Most of the runs found a phenotype with nine hits before 10,000 phenotypes had been tested. However, Legal Selection with Legal Map did not achieve this. Its best success rate was 25% with a requirement of processing 25,000 phenotypes. The success probability of Illegal Replacement with Legal Map was also very low. Even when 14,000 phenotypes were processed, there was less than a 20% probability that this method would find a phenotype with nine hits.

It is clear that three of the methods provide good success rates in evolving phenotypes with nine hits, see figure 2B[1]. However, the results also show that these same methods were the worst at evolving phenotypes which satisfied the run-time error constraint. As shown in Figure 2C, the two methods with the lowest success rates: Legal Selection with Legal Map and Illegal Replacement with Legal Map were able to evolve considerably more legal phenotypes than the other methods. Only one method: Legal Fertility with Legal Map, had a high success rate and evolved larger numbers of legal phenotypes.

5. Analysis and Discussion

The experiments with the run-time error constraint demonstrate a common dilemma in all constrained optimization problems: both the objective and constraints need to be satisfied, and evolving phenotypes which fulfill one of them can sacrifice the evolution of phenotypes which fulfill the other. Using an evolutionary algorithm to find solutions for such problems is therefore difficult because evolutionary search is directed in different directions. The experiments investigated five different ways in which a GP system could be made to evolve both fit and legal programs. The results show, however, that each method exerted a different level of evolutionary pressure for the constraint and objective. It is clear that such different levels of pressure can effect the degree to which both criteria are met.

In the implementation described above, the Legal Map method (a hard constraint) is the neutral placement in the spectrum (i.e., the control method) as it *repairs* phenotypes without the addition of a second selection pressure for the constraint. The other four (soft constraint) methods use this same phenotype repair scheme with an added *pressure* to reduce the number of illegal programs evolved.

Figure 2C shows the average number of born-legal phenotypes in the population using these methods. Our control, the Legal Map method, enforces no pressure for the constraint and the average number of legal individuals remains around 200 throughout the runs. In contrast, the Illegal Replacement method shows that a very strong pressure is exerted on the GP system to evolve legal programs. After the processing of only 2,500 phenotypes, all illegal phenotypes have been replaced and the population contains only legal phenotypes. The Legal Selection method also exerts a strong pressure for the constraint. Since only legal phenotypes are selected (randomly) for reproduction, programs which satisfy the constraint are propagated quickly: after 15,000 phenotypes are processed, only legal phenotypes exist in the population. The Fertility method exerts pressure for constraints by generating more offspring for legal parents than for illegal parents. Compared to the control method, Legal Map, all twenty runs of this method show a consistent increase of legal phenotypes in the population. (The downward trend after 5,000 individuals have been processed, evident in figure 2C, is a distortion of the graph caused by a single run, and is not considered significant.)

Not all of the methods exert such consistent pressures for the constraint, however. The Penalty method generates a strong fitness-driven evolutionary process (illegal phenotypes have their *total_error* values doubled to reduce fitness values, so pressure for the constraint drops as individuals become fitter.). As figure 2C shows, this results in the number of legal phenotypes being gradually reduced to satisfy fitness (objec-

[1] Note that the data shown in figure 2B were generated in separate runs.

tive) requirement. It seems likely that the use of fixed penalty values might prevent this effect.

Figure 2D shows the average fitness in the population using these methods. Driven to satisfy only the fitness (objective), the Map method raises population fitness consistently through fitness proportionate selection. Similarly, the strong fitness-oriented pressure of the Penalty method and the Fertility method raises population fitness consistently. The Selection method also raises the average fitness as it replaces the worst individuals with newly created offspring. However, the average fitness stays below 0.87 because by only selecting legal phenotypes for reproduction, the genetic diversity is dramatically reduced. (Figure 2C shows that only 15% of initial population were legal). Because of this reduced diversity, combined with the strong pressure for constraints, the population tends to converge prematurely. This is why only 5 out of the 20 runs for the Selection method were successful. The same effect is evident for the Replacement method. Again, genetic diversity is lost as a large number of illegal phenotypes are replaced. Populations converged when around 2000 phenotypes had been processed. Only 3 out of the 20 runs were successful.

In summary, the combination of pressure for the run-time error constraint and fitness directs evolutionary search to find a legal phenotype which produces nine hits. While some of the results may be due to the type of constraint tackled and the implementation of the constraint handling methods, these experiments show that the Fertility method seems to provide the best balance of evolutionary pressure on both criteria. It raises the average fitness value and at the same time reduces the number of illegal phenotypes in the population. As a result, the average number of hits in the population is raised consistently (see Fig. 2B) and successful phenotypes are found in all 20 runs.

6. Conclusions

This paper presented a framework to allow the classification of constraint handling methods within various stages of evolutionary algorithms. Such methods impose either *hard* constraints or *soft* constraints, and all use *prevention, correction,* or *pressure* to enforce the constraints. Eleven methods were identified, including some which had not been explored previously.

Five of these eleven methods were tested on a run-time error constraint in a GP system. The results show that depending on the problem, the methods used and their implementation, the seesaw of evolutionary pressure can either favour constraints or objectives. Of the methods examined, the Legal Fertility method provided a good balance between these two criteria, and led GP to find phenotypes which satisfied both objective and constraints.

Acknowledgements

Thanks to Bill Langdon for his Simple-GP on which our code is based and Tina’s supervisor, Chris Clack, for his support. Thanks also to the anonymous reviewers.

References

1. Bäck, T., *Evolutionary Algorithms in Theory and Practice*. Oxford Uni. Press, NY (1996).

2. Banzhaf, W. Genotype-phenotype-mapping and neutral variation - a case study in genetic programming. *Parallel Problem Solving From Nature,* 3. Y. Davidor, H-P Schwefel, and R. Mnner (eds.), Springer-Verlag, (1994) 322-332.
3. Bentley, P. J. & Wakefield, J. P., Finding acceptable solutions in the pareto-optimal range using multiobjective genetic algorithms. Chawdhry, P.K., Roy, R., & Pant, R.K. (eds) *Soft Computing in Engineering Design and Manufacturing*. Springer Verlag London Limited, Part 5, (1997), 231-240.
4. Fogel, L., Angeline, P. J., Bäck, T. *Evolutionary Programming V*, Porceedings of the 5th Annual Conference on Evolutionary Programming. MIT Press, Cambridge, MA (1996).
5. Gero, J. S. and Kazakov, V. A, Evolving design genes in space layout planning problems, *Artificial Intelligence in Engineering* (1998).
6. Goldberg, D. E., *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley (1989).
7. Gruau, F., On using syntactic constraints with genetic programming. *Advances in Genetic Programming II*, P.J. Angeline & K.E. Kinnear, Jr, (eds.), MIT Press, (1996) 377-394
8. Janikow, C., A methodology for processing problem constraints in genetic programming. *Computers and Mathematics with Application*, Vol. 32 No. 8, (1996) 97-113.
9. Keller, R. and Banzhaf, W. Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes. *Genetic Programming '96: Proc. of the 1st Annual Conf. on GP*., MIT Press, Cambridge, MA. (1996) 116-122.
10. Koza, J. R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA (1992).
11. McDonnell, J. R., Reynolds, R. G., Fogel, D. B. *Evolutionary Programming IV*, Proceedings of the 4th Annual Conference on Evolutionary Programming. MIT Press (1995).
12. Michalewicz, Z., Genetic algorithms, numerical optimization and constraints, *Proc. of the 6th Int. Conf. on Genetic Algorithms*, Pittsburgh, July 15-19, (1995a) 151--158.
13. Michalewicz, Z., A survey of constraint handling techniques in evolutionary computation methods *Proc. of the 4th Annual Conf. on Evolutionary Programming*, MIT Press, Cambridge, MA (1995b) 135--155.
14. Michalewicz, Z., Dasgupta, D., Le Riche, R.G., and Schoenauer, M., Evolutionary algorithms for constrained engineering problems, *Computers & Industrial Engineering Journal*, Vol.30, No.2, September (1996) 851--870.
15. Michalewicz, Z. and Michalewicz, M., "Pro-Life versus Pro-Choice Strategies in Evolutionary Computation Techniques", Ch. 10, *Evolutionary Computation*, IEEE Press (1995).
16. Michalewicz, Z., Schoenauer, M., Evolutionary Algorithms for Constrained Parameter Optimization Problems, *Evolutionary Computation 4* (1996) 1-32.
17. Hinterding, R. and Michalewicz, Z., Your brains and my beauty: parent matching for constrained optimisation, *Proc. of the 5th Int. Conf. on Evolutionary Computation*, Anchorage, Alaska, (1998) May, 4-9.
18. Schoenauer, M. and Michalewicz, Z., Boundary operators for constrained parameter optimization problems, *Proc. of the 7th Int. Conf. on Genetic Algorithms*, East Lansing, Michigan, July 19-23 (1997) 320-329.
19. Syswerda, G., Uniform crossover in genetic algorithms. In Schaffer, D. (ed.), *Proc. of the Third Int. Conf. on Genetic Algorithms*. Morgan Kaufmann Pub., (1989).
20. Yu, T. and Clack, C., PolyGP: A polymorphic genetic programming system in Haskell. *Genetic Programming '98: Proc. of the 3rd Annual Conf. Genetic Programming*, (1998).