Dynamic Hill Climbing: Overcoming the limitations of optimization techniques

Deniz Yuret and Michael de la Maza
Numinous Noetics Group, Artificial Intelligence Laboratory,
Massachusetts Institute of Technology, Cambridge, MA 02139, USA.
email: deniz@ai.mit.edu, mdlm@ai.mit.edu

Abstract

This paper describes a novel search algorithm, called dynamic hill climbing, that borrows ideas from genetic algorithms and hill climbing techniques. Unlike both genetic and hill climbing algorithms, dynamic hill climbing has the ability to dynamically change its coordinate frame during the course of an optimization. Furthermore, the algorithm moves from a coarse-grained search to a fine-grained search of the function space by changing its mutation rate and uses a diversity-based distance metric to ensure that it searches new regions of the space. Dynamic hill climbing is empirically compared to a traditional genetic algorithm using De Jong's well-known five function test suite [4] and is shown to vastly surpass the performance of the genetic algorithm, often finding better solutions using only 1% as many function evaluations.

1 Introduction

Genetic algorithms [13, 10] have been applied to a wide variety of optimization problems ranging from the traveling salesman problem [12, 16] to protein structure [14, 6] to horse racing [5]. Likewise, hill climbing techniques have been used to partition graphs [1], to learn behaviors in robots [3], and to construct Steiner systems [9]. In this paper we combine the strengths of these two algorithms into a single system that outperforms the traditional genetic algorithm on De Jong's test suite [4]. This algorithm introduces two new ideas, dynamic coordinate change and exploitation of local minima, that, to the best of our knowledge, have not been used in other search algorithms.

Genetic algorithms are patterned after biological systems. They maintain a population of individuals that undergo crossover and mutation. Individuals that are significantly less fit than the rest of the population are gradually eliminated, while stronger individuals are propagated. In this way, the genetic algorithm evolves a population of individuals of increasingly higher fitness. A typical genetic algorithm is implemented as follows:

```
t=0
Initialize P(t)
Until done
    Assign fitnesses to the individuals in P(t)
    Select P(t+1) from P(t)
    Apply mutation and crossover operators to P(t+1)
    t = t + 1
```

The genetic algorithm typically terminates after a certain number of function evaluations or after one of the individuals in the population is at the global optimum.

Hill climbing techniques, such as gradient descent, use local information about a search space to find optima. For unimodal functions, hill climbing techniques easily outperform most other methods, but for more complicated functions hill climbing techniques are apt to get stuck in local optima. Ackley [1] combines genetic algorithms with hill climbing to produce an algorithm called Stochastic Iterated Genetic Hillclimbing (SIGH). Although the motivations behind our algorithm and SIGH are similar, the algorithms themselves are quite different. Unlike our technique, SIGH is unable to change its coordinate frame and does not use a diversity-based distance metric. Also, SIGH employs ideas from simulated annealing and connectionism which are not used in our algorithm.

Section 2 of this paper describes the new algorithm in detail, Section 3 discusses related results, and Section 4 concludes the paper.

2 Algorithm description

2.1 Basic hill climber

At the core of dynamic hill climbing is a simple but efficient hill climbing technique. The process is initialized by selecting a random individual in the search space. The population is expanded by applying mutations to this single individual. The parent and the offspring are considered as candidates for the next generation and the fittest one is selected.

After each selection, the algorithm keeps track of the age of the selected individual. Whenever a parent gets selected twice in a row, its age is increased. Whenever a new individual is selected, its age is inherited from its parent.

Each mutation is performed by adding a scalar to one coordinate of the parent. If the space is bounded then care should be taken not to exceed the bounds. In an N dimensional space, we apply $2N$ mutations, one for each possible direction in the space. The magnitude of the mutations decreases with increasing age. This ensures the adaptation of the search grain to the local structure of the space.

This algorithm is mathematically tractable in unimodal functions, i.e., functions where there is only one extremum and the gradient of the surrounding area points to it. Our particular implementation uses $D/2$ as the initial mutation magnitude, where D is the size of the space in one dimension. At each age increase, the magnitude is halved. In one dimension, it takes $O(\log(D))$ generations to reach the optimum. In N dimensions it will take N times as many, since we are adjusting one dimension at a time. At each generation dynamic hill climbing makes $2N$ new function evaluations so the total number of function evaluations is $O(N^2 \log(D))$.

2.2 Problems

Although the straight hill climber is efficient in unimodal spaces, in most real world functions, it is destined to get stuck. The main reasons for the failure of the hill climber are explained in [17]. The algorithm can get stuck in a local optimum (the foothill problem), it can get stuck in mostly flat surfaces with few sharp peaks (the plateau problem), or it can get stuck because the direction of ascent is not within our directions of motion (the ridge problem).

We think that measures can be taken against some of these traps. In particular the ridge problem can be fought by adopting a dynamic coordinate frame, as explained by section 2.3, and the foothill problem can be fought by exploiting local optima as explained in section 2.4. The plateau problem is hopeless, since it means that the space is giving no information whatsoever about its structure. If that is the case any random search will perform as well as other heuristically informed methods.

2.3 Dynamic coordinate frame

The classical hill climbing techniques usually suffer from the fact that they have a rigid coordinate frame, and they cannot freely choose the direction of the motion.

Suppose we can move in four main directions: north, west, south and east. All these moves give us worse fitnesses. A narrow path in the northwest direction that leads to higher fitness might be hidden from our sight. One possible solution is to consider combinations of unit directions, but this approach faces the problem of exponential growth in the number of dimensions.

Dynamic hill climbing adopts a more feasible approach by using a dynamic coordinate frame. Once the basic hill climber gets stuck, it looks for a more appropriate coordinate frame which is more likely to point in the right direction. After every move, it stores the last two positions in a buffer. Then the Gram-Schmidt orthogonalization process is used to construct a set of new unit vectors based on the difference of these positions. The following mutations use this new set of vectors as their basis. This method is able to adjust the directions according to the particular structure of the space.

2.4 Exploitation of local optima

One possible measure against the foothill problem is considering the diversity of the individuals as an advantage in the selection procedure. This is an idea that has been explored by several researchers [17, 11].

In the standard genetic algorithm techniques, whenever one individual reaches a local optimum, its chance of survival surpasses the others, it stays alive too many generations, and eventually the rest of the population converges to that single point.

The goals of diversity based strategies are to protect the search from early convergence and to explore the space as homogeneously as possible. In the common practice, the selection procedure is responsible for taking into account both fitness and diversity. Fine tuning the balance between these two very different measures is not easy and usually leads to different kinds of traps. To achieve these goals we do not think mixing diversity and fitness is the right solution. They should be considered separately as explained in the following paragraphs.

The basic hill climbing technique explained in Section 2.1 can detect the positions of the local optima with great precision. When the age of an individual is beyond a certain limit, the mutation size gets small enough to ensure the absence of any nearby points with better fitness.

Dynamic hill climbing has a memory where it can store the positions of the local optima. This can be thought of as a separate population which we will call the Old. The individuals in the dynamic population freeze when they get beyond a certain age and they join the Old. The Old form a passive population which do not participate in reproduction or selection, but are only referred to when information about local optima is needed.

Whenever the search is stuck the new starting point is found by using the hill climber to maximize the diversity from the Old. At this stage no function evaluations are necessary, since we are using a fitness measure that is based on diversity. When this process converges, the age is reset to 0 and the search restarts. It is important to select the right diversity measure. Diverse should mean "most difficult to access using the particular search method." When dynamic hill climbing is used to optimize real-valued functions, the coordinates of an individual are typically represented using a bit vector, with a certain number of bits representing a real number for each dimension. The mutation magnitude is of the form 2^k, so the Euclidean distance metric is meaningless. A point that is 16 units away in one of the dimensions can be reached in one step whereas 15 would require four steps.

Dynamic hill climbing uses the Hamming distance which is the difference in the number of bits between two vectors. The diversity search maximizes the Hamming distance to the closest Old. This stimulates exploration of different regions of the space while lowering the probability of converging to the same local optima.

3 Related work

Figure 1 shows a comparison of dynamic hill climbing and a traditional genetic algorithm. Each experiment was repeated ten times. This section compares dynamic hill climbing to variants of the traditional genetic algorithm.

Eshelman *et al.*[8] fine-tune the population size, mutation rate, crossover rate, and crossover type of a genetic algorithm in order to maximize its performance on De Jong's five functions. Their best algorithm, which uses eight point traditional crossover, requires, on average, 1538, 9477, 1740, 4137, and 3004 function evaluations to find the global optimum of the sphere function, Rosenbrock's saddle function, the step function, the quadratic function, and Shekel's foxholes function, respectively. With the exception of Rosenbrock's saddle function, dynamic hill climbing requires only 4% to 22% as many function evaluations as this highly tuned genetic algorithm.

Whitley [15] describes GENITOR, a genetic algorithm that uses rank based selection and argues that it outperforms a traditional genetic algorithm. The Shekel's foxholes function is used to compare the GENITOR algorithm to the traditional genetic algorithm. After 1000 trials, the fitness of the standard genetic algorithms's best individual was on, average, approximately four, while the GENITOR algorithm's best individual achieved a fitness of slightly less than two. Figure 1 shows that dynamic hill climbing produces an individual with a fitness of approximately 1, which is the global optimum, in fewer than 1100 trials, hence outperforming the GENITOR algorithm.

Wright [18] studies genetic algorithms that use real parameters instead of binary strings. He runs the genetic algorithm for 1000 trials and reports the fitness of the best individual over 1000 runs. For the sphere, Rosenbrock's saddle, step, quartic, and Shekel's foxhole functions, the best individual has, respectively, a fitness of 0.00004, 0.0107, 0.573, -1.0906, and 9.7598. We repeated these experiments with our algorithm and in all the functions except quartic, dynamic hill climbing found the global optimum. In quartic the best individual had a fitness of -0.72559.

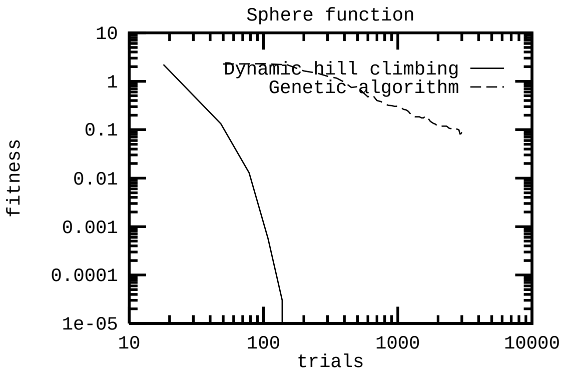
Eshelman [7] introduces a genetic algorithm, called CHC, that has a highly conservative selection scheme and a highly disruptive recombination operator. CHC, requires, on average, 1089, 9065, 1169, 1948, and 1396 function evaluations to find the global optimum of the sphere function, Rosenbrock's saddle function, the step function, the quadratic function, and Shekel's foxholes function, respectively. With the exception of Rosenbrocks' saddle function, dynamic hill climbing finds the global optimum of these functions using only 6% to 46% as many function evaluations.

These results demonstrate that dynamic hill climbing vastly outperforms a wide variety of genetic algorithms that have been specifically created to require few function evaluations on De Jong's test suite.

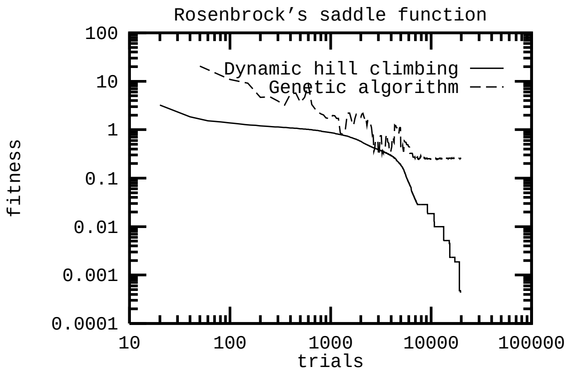
4 Conclusion

In this paper we have described dynamic hill climbing, a novel search algorithm that overcomes some of the limitations of genetic algorithms and hill climbing. A C program that implements dynamic hill climbing is available from the authors.

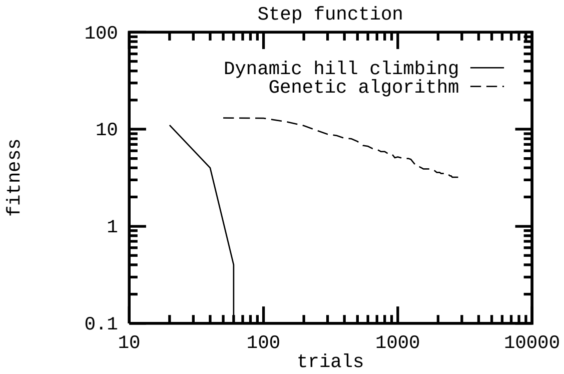
(A)



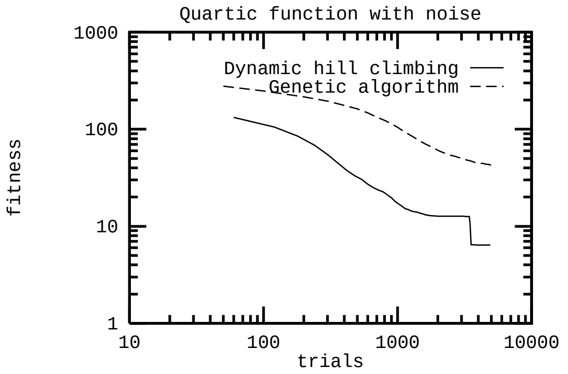
(B)



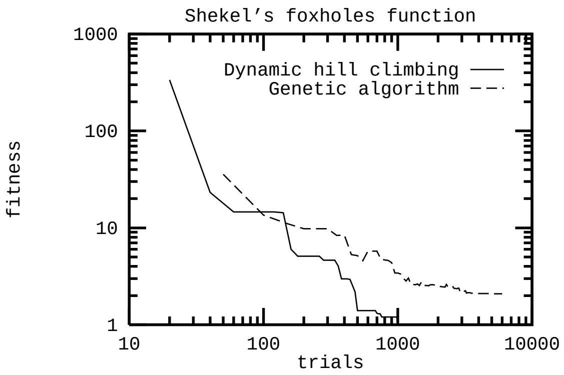
(C)



(D)



(E)



(F)

Function	Evaluations
Sphere	135
Rosenbrock	-
Step	70
Quartic	-
Foxholes	430

Figure 1: Comparison of traditional genetic algorithm and dynamic hill climbing on De Jong's test suite. Each graph shows the best individual averaged over ten runs. The global optimum of the first four functions is 0 and the global optimum of the fifth function is approximately 1. The GENEsYs 1.0 package [2] was used to implement the traditional genetic algorithm, while the dynamic hill climbing algorithm was implemented using a custom C program. (A) Sphere function (B) Rosenbrock's saddle function (C) Step function (D) Quartic function with noise (E) Shekel's foxholes function (F) Mean number of function evaluations for dynamic hill climbing to find the global optimum.

Acknowledgment

This paper describes research done at the Artificial Intelligence laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-91-J-4038.

References

[1] D. Ackley. *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic Publishers, Boston, 1987.

[2] T. Bäck and F. Hoffmeister. A user's guide to *GENEsYs 1.0.* Software package documentation, 1992.

[3] P. David and B. Kuipers. Learning hill-climbing functions as a strategy for generating behaviors in mobile robots. Technical Report AI90-137, University of Texas at Austin, 1990.

[4] K. De Jong. *Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Ann Arbor, 1975.

[5] M. de la Maza. A SEAGUL visits the race track. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 208–212, 1989.

[6] M. de la Maza and B. Tidor. Increased flexibility in genetic algorithms: The use of variable Boltzmann selective pressure to control propagation. In *Proceedings of the ORCA CSTS Conference: Computer Science and Operations Research: New Developments in Their Interfaces*, pages 425–440, 1992.

[7] L. Eshelman. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In *Foundations of Genetic Algorithms*, pages 265–283, 1991.

[8] L. J. Eshelman, R. A. Caruana, and J. D. Schaffer. Biases in the crossover landscape. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 10–19, 1989.

[9] P. Gibbons and R. Mathon. The use of hill-climbing to construct orthogonal Steiner triple systems. Technical Report 263/92, University of Toronto, 1992.

[10] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[11] D. E. Goldberg. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, 1989.

[12] J. Grefenstette, R. Gopal, B. Rosmaita, and D. van Gucht. Genetic algorithms for the traveling salesman problem. In *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pages 160–165, 1985.

[13] J.H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[14] R.S. Judson. Teaching polymers to fold. *Journal of Physical Chemistry*, 96:10102–10104, 1992.

[15] D. Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121, 1989.

[16] D. Whitley, T. Starkweather, and D'Ann Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 133–140, 1989.

[17] P.H. Winston. *Artificial Intelligence.* Addison-Wesley Publishing Company, Reading, MA, third edition, 1992.

[18] A. Wright. Genetic algorithms for real parameter optimization. In *Foundations of Genetic Algorithms*, pages 205–218, 1991.

Appendix: De Jong's Functions

$$f_1(\vec{x}) = \sum_{i=1}^{n} x_i^2 \quad (Sphere)$$

$$f_2(\vec{x}) = \sum_{i=1}^{n-1} (100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2) \quad (Rosenbrock's\ saddle)$$

$$f_3(\vec{x}) = 6 \cdot n + \sum_{i=1}^{n} \lfloor x_i \rfloor \quad (Step)$$

$$f_4(\vec{x}) = \sum_{i=1}^{n} i x_i^4 + \mathrm{gauss}(0, 1) \quad (Quartic)$$

$$\frac{1}{f_5(\vec{x})} = \frac{1}{K} + \sum_{j=1}^{25} \frac{1}{c_j + \sum_{i=1}^{2} (x_i - a_{ij})^6} \quad (Shekel's\ foxholes)$$

$$(a_{ij}) = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \cdots & 32 & 32 & 32 \end{pmatrix}$$

$$K = 500$$

Reprinted from: Goldberg [10]