Evolutionary Approaches for Mechanical Regression

Adeola Bannis
Electrical & Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
abannis@andrew.cmu.edu

Mark Whiting
Department of Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
mwhiting@andrew.cmu.edu

Abstract

Generating specific, arbitrary output from a mechanical system is a challenging task. Similar problems, such as symbolic regression, have been effectively solved with evolutionary strategies but they tend to be more easily abstracted. We introduce an approach to the mechanical regression problem using a network aware evolutionary algorithm based on NEAT. We evaluate this approach by conducting several controlled experiments against a naive genetic algorithm and comparing a typical NEAT algorithm with one designed to accommodate spacial representation. The results show that our approach is substantially more effective than the naive algorithm and recommend steps for moving forward in the development of more general mechanical regression problems.

1 Introduction

Our initial focus was to demonstrate the use of Evolutionary Algorithms (EA) in evolving complex systems, with a specific focus on developing complex and unexpected behaviour. Historically, EAs are often used to optimize a known system within a complex optimization space, or design solutions with a number of specific requirements in mind. In practice, this form of design also appears much like an optimization problem as it is generally modeled as an array of identifiable fitness functions and possibly some way to deal with compromises between them.

The prospect of evolving unanticipated behaviour is attractive because in the advent of increasingly complex design problems, its common that the outcome of a design process is unpredicted by the needs that motivated its start, a symptom of Wicked Problems and other classifications of real world complex problem solving. To avoid a philosophical argument on the meaning of unpredicted goals we took the standpoint that EAs could be used to develop emergent behaviour and set the scope in mechanical systems, a context in which stochasticity is particularly easy to model, and almost entirely unavoidable in practice.

To set up an experiment of this kind we adjusted our project to focus on solving tough optimization problems, as opposed to ones with a dynamic or fuzzy fitness function, which are more difficult to represent. Some output constraints are easy to represent symbolically, but hard to represent mechanically (e.g. sin(x), log(x), 1/x). EAs fit well with this kind of problem as the design space is very large and provides almost no information about better designs (so typical numerical methods do not work whatsoever). Further, although mechanically reproducing something like a sin wave is not impossible for a human designer to do, it requires an alternative view of the problem, one that was not effectively dealt with by industry until the start of the industrial revolution, controlled reciprocal motion. In this way, solving a problem like this requires the system to be generative and derive emergent behaviours, (i.e. situations in which components act outside of their expected roles).

In the rest of the paper we motivate a general approach to this mechanical regression problem, evaluate a solution for an indicative, tractable but tough test case within the limitations of a mechanical

simulator with two different algorithm concepts, and finally discuss ways to move forward such as approaches to getting faster convergence on problems of this type.

2 Background

Developing arbitrary mechanical regression exists at the intersection of two thriving research areas: symbolic regression, and computational mechanical design. In each of these, other researchers have identified useful approaches that provide background to solve our problem.

2.1 Graph Based and Symbolic Regression

Regression of arbitrary data is a quintessential goal in many science and engineering communities. In 2009, Schmidt and Lipson [3,4] successfully demonstrated algorithmic symbolic regression by using an Evolutionary Strategy (ES) based approach. Their method involved calculating partial derivatives for every pair of variables in a data set, evolving equations to fit the data and derivatives. They rewarded simplicity and are now able to provide fundamental relationships in arbitrary data. Their evolution approach used data driven variables and a limited set of symbols (+, , , and), meaning that the interactions each element of the evolution strategy could have were predictable and did not require much simulation. Mechanical regression requires a way to deal with unexpected interactions so this approach has some shortcomings.

In an earlier paper, Schmidt and Lipson [2] considered the impacts of different representation schemes for these kinds of problems, and found that graph representations have better performance than trees (as typically used with evolutionary strategy approaches), when complexity exceeds 5 symbolic components. Stanley & Miikkulainen [5] introduced, NeuroEvolution of Augmenting Topologies (NEAT) which allows EA and ES style augmentation in graph representations of neural networks. Though the neural network focus is not directly suitable for modeling a mechanical system, the mechanism of NEAT provides a framework to build off for evolving complex systems.

2.2 Computational Mechanical Design

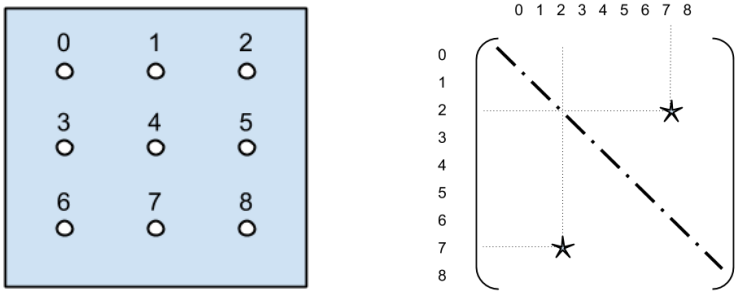
Though many real world cases for mechanical regression exist in design problems such as timing mechanisms, automation systems, and materials handling, a specific case was shown by Coros et Al [1] when using computational design to build animatronics with realistic motion patterns. Their work focused on finding mechanisms that could produce a specified 2D path of motion, although difficult to do by hand, this problem is well constrained and all mechanisms can be made from the same set of parts, two drive shafts and two leavers. Because of the well defined setting, their method involved choosing from a set of known path shapes, matching them, then using gradient based optimization methods to do final refinements toward a goal path. This approach offers a high level example of a very specified mechanical regression problem.

3 System Structure and Representation

The mechanical systems created with these algorithms are modelled after a block-building schema: there are a number of physical parts which are attached to a grid, and any two parts may be connected with spring, belt, hinge or rod connections. Each part had a height and length (length is ignored for circles), and each attachment had several parameters:

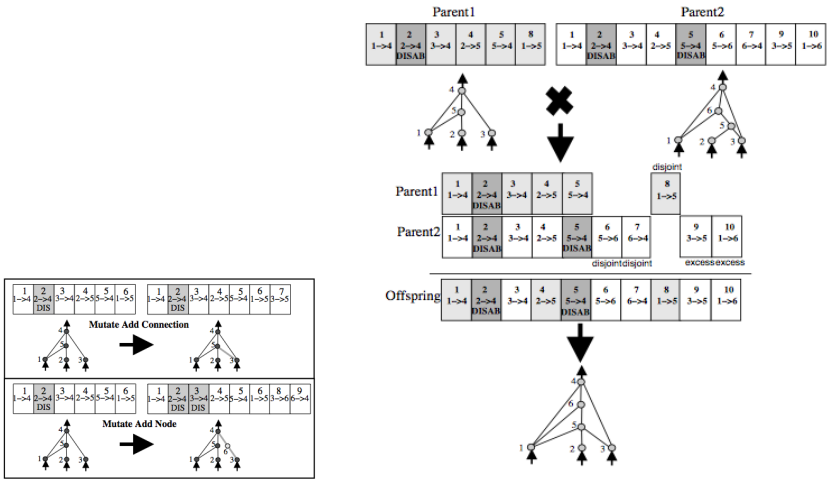
- The points of attachment on each part, represented as points between (-1,-1) and (1,1), where (-1, -1) corresponds to the bottom left corner of a part.
- The type of connection: spring, gear/belt, rod.
- Special parameters for each type. For example, rest length, spring constant and damping for spring; gear ratio and phase for gear/belt; pivot point for hinges.

The machine structure was represented as a graph, in which the nodes were solid rectangular or circular parts, and the edges were attachments between two parts. The nodes were kept in a list and numbered according to their spatial position: numbers increased left to right and from top to



(a) Numbering of parts (b) Adjacency matrix

Figure 1: System representation



(a) Mutation (b) Recombination

Figure 2: NEAT genome representation

bottom. This enabled the attachments to be maintained as an adjacency matrix. The diagonal of this matrix was reserved for the attachments of the parts themselves to the wall. For example, the 3x3 grid in Figure 1(a) would have its 9 pegs labelled as shown. The adjacency matrix would be 9x9 as shown in Figure 1(b), and an attachment between parts 2 and 7 is shown as a star; the attachment information is repeated as the attachments are undirected edges. Each machine also had an input and output position, chosen at initialization. The input part received some signal, such as rotation, and the response of the output part was measured. Care was taken to ensure that mutation and recombination did not cause the input and output parts to become disconnected.

3.1 Algorithm

The first algorithm attempted was a naive implementation of a basic genetic algorithm. Initially, machines were generated with different numbers of parts (2 - 6) with random attachments between each part and the previous part, and then input and output positions were chosen. Recombination was done by swapping either a part or an attachment between the two machines, and mutation was done by adjusting one of the attachment parameters, or changing the type of an attachment.

The NEAT algorithm [5], was designed for evolving neural networks, which are representable as undirected, weighted graphs. This approach seemed like a more natural model for our machine system algorithm. An overview of the NEAT algorithm follows.

The genome is maintained as a list of edges, where each edge is represented as a pair of nodes (Figure 2). In addition to the mutations shown, it is possible to mutate the parameters of an attachment, which is equivalent to mutating a connection weight in a neural network, and does not directly affect the attachment genome. The value of the genome comes from being able to combine two different structures, as shown in Figure 2b. A counter of all new attachments created, called innovation number, is kept and incremented each time a new gene is introduced in a generation. This provides an ordering of genes in the attachment genomes for recombination. Alleles of the same gene are inherited at random from either parents, while disjoint (lying between matching genes) and excess (outside the range of matching genes) genes are inherited from the more fit parent.

Another feature of NEAT is speciation, which creates subpopulations with different structures, and helps keep new developments that are not immediately fitter around for mutation. Species each have a representative individual, kept from the previous generation, and are repopulated with individuals that fall within a certain distance of the representative individual. The distance between two individuals is calculated as a weighting of the proportion of disjoint and excess genes and the difference between shared genes. For our system, a difference between two matching genes, i.e. between two attachments, is a constant if they are of different types, else a sum of normalized differences between parameter values:

$$norm = \begin{cases} 0 & \text{if } v_1 = v_2 = 0 \\ \frac{v_1 - v_2}{v_1 + v_2} & \text{otherwise} \end{cases} \tag{1}$$

The full expression for difference between individuals (taken from [Stanley 2002]) is:

$$\delta = c_1 \cdot \frac{D}{N} + c_2 \cdot \frac{E}{N} + c_3 \cdot \bar{W} \tag{2}$$

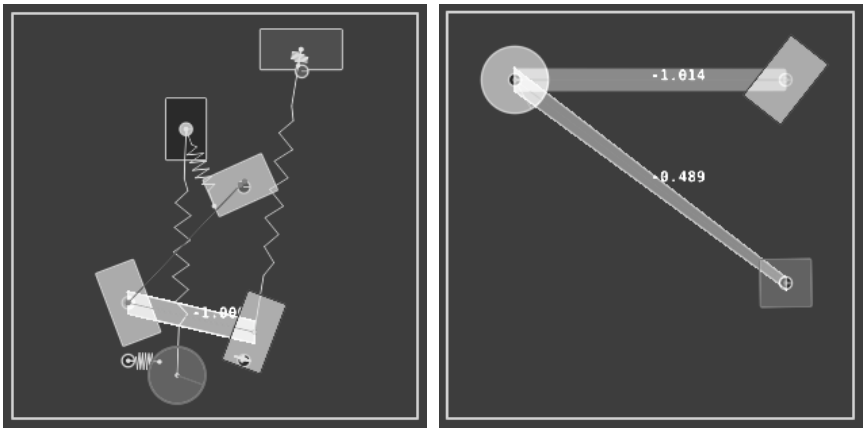
where N is the length of the longer genome, D is the number of disjoint genes, E is the number of excess genes, and $\bar{W}$ is the mean difference between matching genes.

Using this representation gave better results, but further refinement could still be made. NEAT does not differentiate between nodes - the only difference between nodes is their time of creation and the connection on them. In our implementation, however, nodes are not interchangeable - they have a spatial location that clearly differentiates them from another node with the same attachments. Thus, a modification to NEAT was made that used the spatial location of attached nodes to replace the innovation number in the attachment genome - that is, no matter what generation an attachment was created in, the same gene would be created. Because attachments could be easily correlated throughout the run, there was no need to retain disabled alleles, so any deleted connections were simply removed from the genome. Since there are no disabled alleles, the concepts of disjoint and excess genes were collapsed into excess genes, resulting in a slightly different compatibility distance measure:

$$\delta = c_1 \cdot \frac{E}{N} + c_2 \cdot \bar{W} \tag{3}$$

3.2 Analysis

The naive algorithm did not reliably find a good solution; many times the final solution had low fitness and was a jumble of parts [figure]. When a good solution did emerge, it was typically very similar to one of the random starting solutions, indicating that the search space was too large, or that maxima were sparse. NEAT addresses at least the second concern, by maintaining several species at varying distances from the current best solution. The use of speciation allows solutions to spread out and ideally find new maxima.



(a) Cluttered system (b) Minimal system

Figure 3: Cluttered and minimal solutions to double rotation constraint

4 Experiments

4.1 Fitness Evaluation

Two functions for modelling were attempted: output rotation being half the input rotation, and constant input rotation giving sinusoidal output rotation. Note that in all experiments, fitness was being maximised. The fitness function in the rotation case integrated several desired factors:

- There should be a strong, positive correlation between the input(I) and output(O) signals. This was measured using the coefficient of correlation, c, giving a correlation term of $|c| + 0.5c$. This ensures that a strong negative correlation is still more fit than no correlation at all.
- The input and output variances should be above a certain threshold. This ensures that machines in which the input or output machines were jammed against other parts would be considered unfit. When the variances fell below the threshold, the system was simply given a tiny fitness value, typically 10^-8.
- The mean value of $\frac{dI}{dO}$ must be as close as possible to 2. This was represented as a distance, $|2.0 - \frac{dI}{dO}|$. The correlation term was divided by the square of this distance, to increase the impact of the term, with a small constant to prevent division by 0.

The final fitness function was thus:

$$\text{fitness} = \begin{cases} 10^{-8} & V_i < \epsilon \text{ or } V_o < \epsilon \\ \frac{|c|+0.5c}{(|2.0-\frac{dI}{dO}|+0.1)^2} & \text{otherwise} \end{cases} \tag{4}$$

where V_i and V_o are input and output variance, and I and O are input and output rotation, respectively.

In the sinusoidal output case, a mean squared error between the output and expected output signals was used. The expected output angle, O_E in terms of the input angle I was given by $O_E = 2\pi sinI$. In order to prevent jammed machines, the input and output variances were again taken into account, giving a final fitness function of

$$\text{fitness} = \frac{V_i \cdot V_o}{mse} \tag{5}$$

The positioning and physical interactions between the parts were simulated using the Chipmunk2D physics library [6], which also allowed us to see the structure of the best machine as the system evolved.

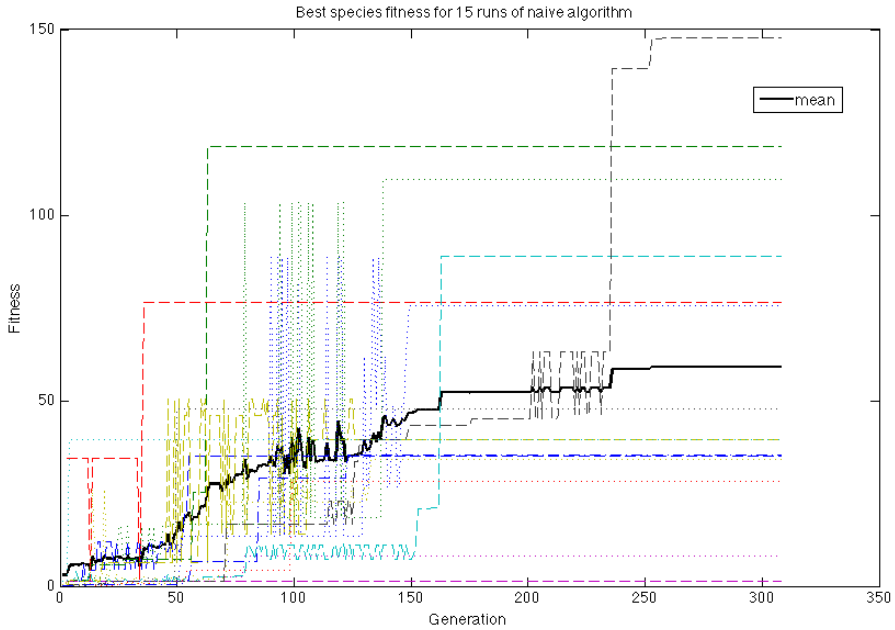


Figure 4: Naive algorithm performance on halved rotation constraint

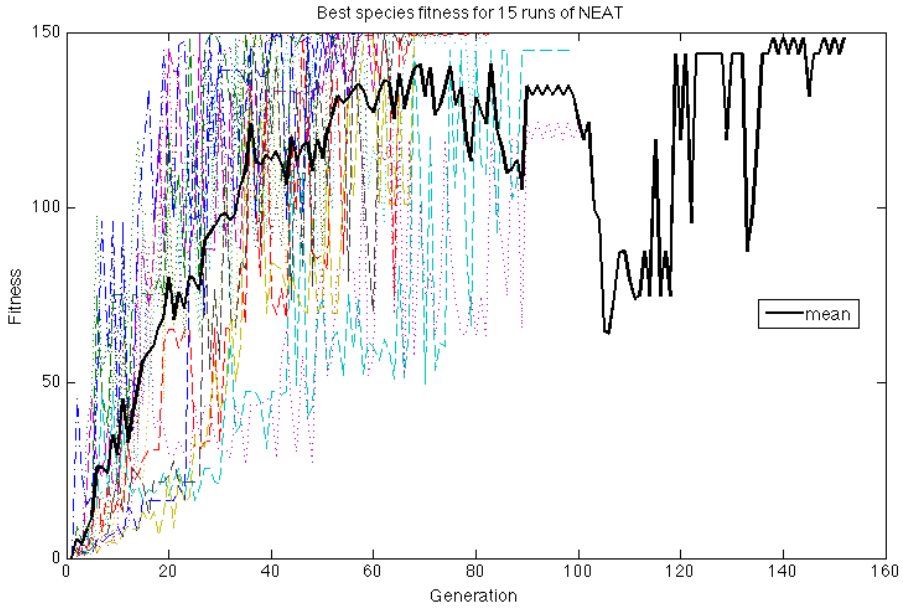


Figure 5: NEAT performance on halved rotation constraint

4.2 Results

4.2.1 Halved rotation

The rotation objective (input rotation = 2 * output rotation) was evaluated with the naive algorithm, the basic NEAT algorithm and the spatially aware variant. The results can be seen in Figs. 4-7.

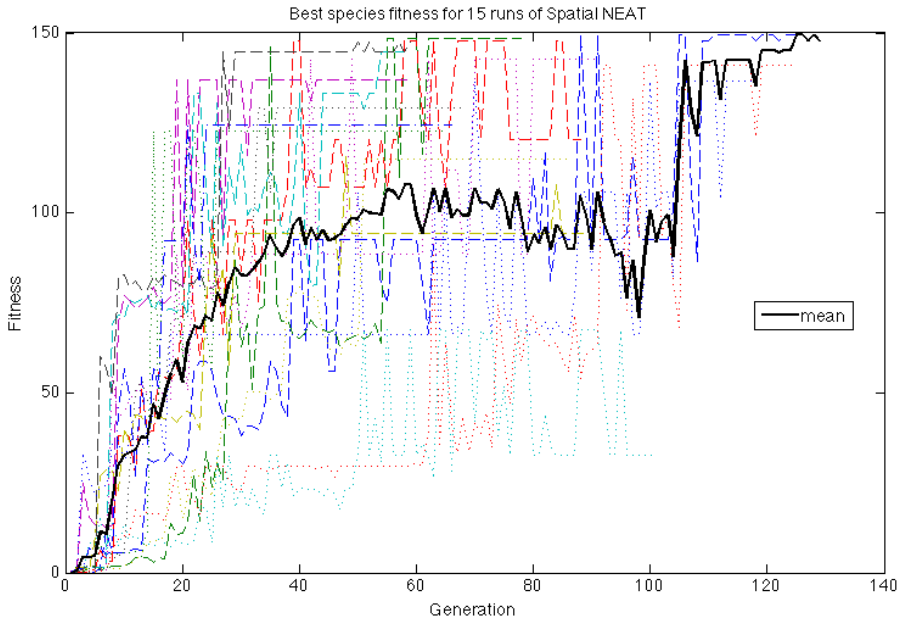


Figure 6: Spatial NEAT variant performance on halved rotation constraint

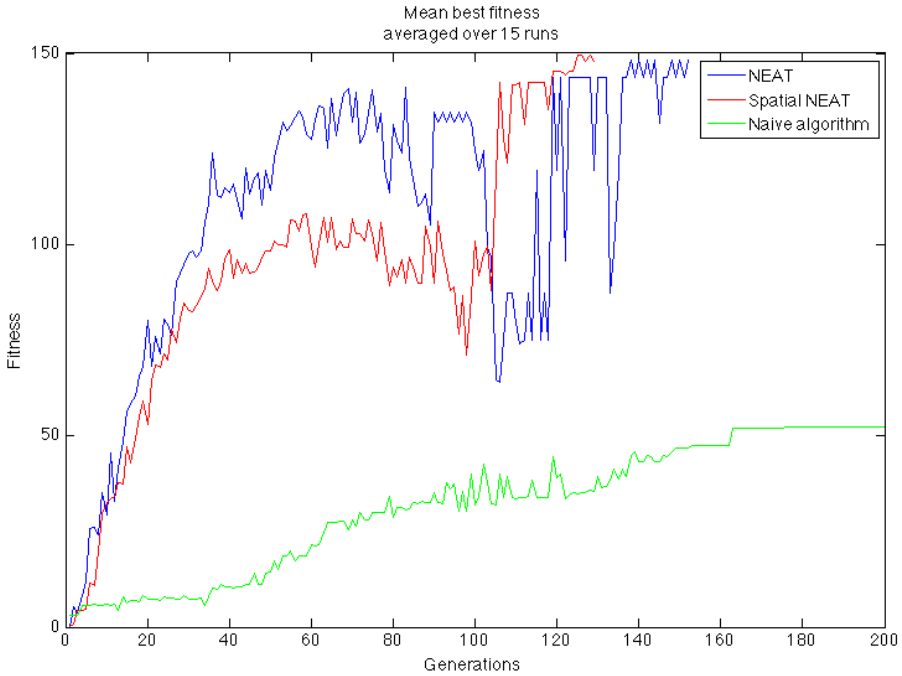


Figure 7: Comparison of all three algorithms on halved rotation constraint

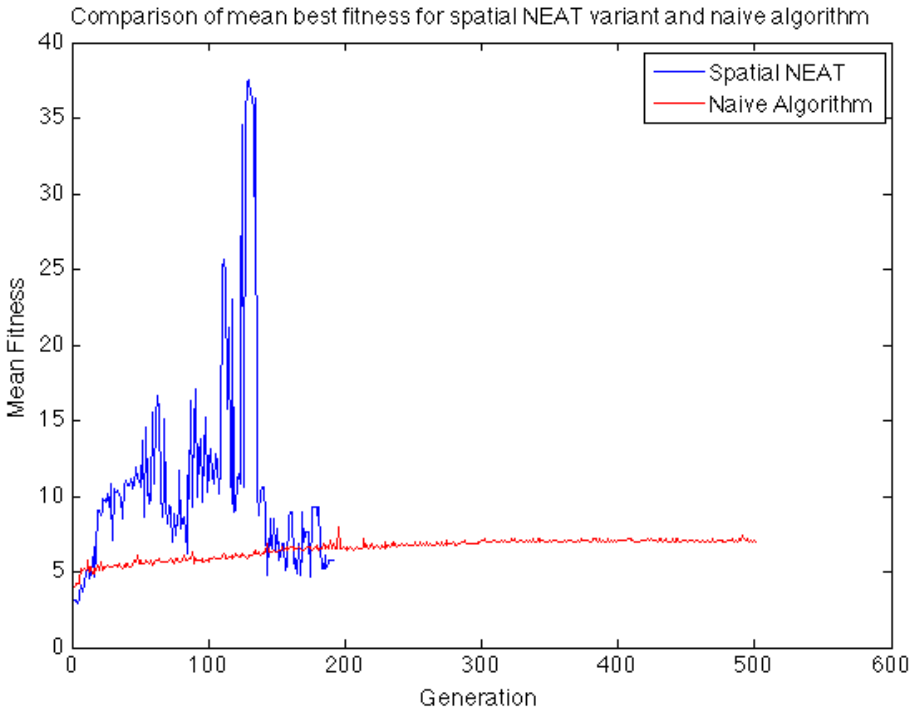


Figure 8: Comparison of naive and spatial NEAT algorithms on sinusoidal rotation constraint

The naive algorithm could not reliably find good solutions; an optimal solution was only found in 1 out of 15 runs, with fitness otherwise stagnating at very low values. In contrast, NEAT and the spatially aware variant performed about equally, finding much better solutions on average.

4.2.2 Sinusodal rotation

A comparison of the best solutions for the naive algorithm and the spatial NEAT variant (Figure 8) outlines the difficulty of finding optimal solutions for a more complex algorithm, and while NEAT did find better solutions, they were often lost. The algorithm would probably benefit from forcing the best ever solution to be included in each generation.

5 Conclusion

In this work we introduce Mechanical Regression as a challenging generative design problem and provide groundwork for general solutions in a planar mechanical environment. We show results from several experiments with 3 different evolutionary algorithms, a baseline GA, NEAT, and a Spatially aware version of NEAT, to show that the spatially aware form of NEAT converges most rapidly and provides robust solutions.

In future work, we see potential for an alternative approach to optimizing toward a fitness function, by using a function of cascading complexity in which more design requirements are enforced after certain levels of fitness are achieved. Other extensions of this work include considering networks of individual solutions, higher dimensional simulations, and demonstrating accurate, general solutions.

References

[1] Coros, Stelian, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel (2013), Computational design of mechanical characters, *ACM Transactions on Graphics (TOG)* **32**: 4

[2] Schmidt, M. & Lipson, H. (2007), Comparison of Tree and Graph Encodings as Function of Problem Complexity, *Genetic and Evolutionary Computation Conference (GECCO'07)* pp. 1674-1679.

[3] Schmidt, M. & Lipson, H. (2009) Distilling Free-Form Natural Laws from Experimental Data, *Science* **324**: 5923, pp. 81-85

[4] Schmidt, M. & Lipson, H. (2009) Symbolic Regression of Implicit Equations, *Genetic Programming Theory and Practice*, **6**

[5] Stanley, Kenneth O. & Miikkulainen, Risto (2002) Evolving Neural Networks through Augmenting Topologies, *Evolutionary Computation***10**:2, pp. 99-127

[6] Howling Moon Software (2013) Chipmunk2D Physics. Retrieved Dec 12, 2003 from http://chipmunk-physics.net/