Evo-Devo Optimization of Soft Robots

Dawson Cohen , Amr El-Azizi , Hod Lipson
Department Of Mechanical Engineering, Columbia University New York, USA

***Abstract*—Evolutionary development (evo-devo) has revolutionized our understanding of the remarkable diversity observed in the natural world, offering a framework to understand the impact of genetic mechanisms and environmental influences on the evolution of species. In this paper, we begin to explore how these environmental influences impact the evolution of soft robots. We form a framework that allows robots to evolve such that their morphology is influenced by their interaction with the environment. To facilitate this evolution, we introduce a massively parallel soft-body simulator that outperforms other simulation software for this task. Additionally, we define a novel robot morphology with the evolution and development operations needed to achieve this genetic and environmental interplay. Through these foundational contributions, we aim to provide a solid groundwork for future exploration in soft-robotic evolution.**

I. Goals and Motivation

Soft robotics offers incredible potential for safe human-robot interaction and for adaptability to complex environments. Automated design of soft robots may offer rapid prototyping, customization, and exploration of a high-dimensional design space. The goals of this research is to lay the groundwork for future exploration of soft robot design by evolution.

The primary computational task in automated soft robot design lies in soft body simulation. Existing GPU-accelerated soft body simulators are built for arbitrarily large elements. As a result, they are unable to fully utilize the GPU's functionality such as on-chip data caches. Thus, the first goal of this paper is to introduce a custom GPU-accelerated simulator that surpasses comparable simulation software in performance, leveraging the full potential of GPU architecture.

The second goal of this paper is to lay the groundwork for evolution and development operations suitable for evo-devo optimization of soft robots. While exploration of the performance of these operations is anticipated in future work, the initial focus is to define a robot encoding and explore random solutions in this encoding's search space.

II. Background

This paper builds on the existing work on Evolution of Soft Body Robots. Evolutionary Algorithms is a field of Machine Learning that attempts to create novel solutions to problems. It is inspired by the way nature creates life, and takes its core operations from that process. Life constantly evolves and develops through a combination of reproduction and random mutation, with the ultimate goal of ensuring survival. Evolutionary Algorithms replicates this process by creating a variety of random solutions, then modifying them to optimize some fitness function. This modification is done using crossover, which combines aspects from two existing solutions, and mutation, which randomly varies one of the parameters of a solution. In 1994, Karl Sims released research on Evolved Virtual Creatures utilizing a rigid body simulation [1]. The work defined bodies utilizing a tree morphology, where the vertices represented the rigid body components, and the edges described the connections between them, as well as the neural circuitry. The follow up work by Cheney et al. examined the state of the field as of 2014, and provided two hypothesis for the lack of progress [2]. First, they believed the rigid elements limited the design space. Second, they believed the regular encoding prevented complexities. To create more fluid and complex morphologies, they implemented evolution on soft-body robotics, and used a compositional pattern-producing network (CPPN) as their encoding. Additionally, they introduced the concept of multiple materials, where each individual voxel of the soft-body could either be passive (bone or tissue) or active (muscle), with two different forms of active muscle. In our previous semester, we replicated this process by implementing our own Spring-Mass simulator and robot encoding. We utilized both a direct encoding where each spring held its own material, and a voxel encoding where a cube of springs shared a material. The initial version of the simulator was built for CPU but ported over to CUDA.

However, one aspect missing both from our work and the field was the introduction of development. Development is the natural process that most biological organisms undergo during their lifetime. As we utilize or neglect muscles, they can either grow or regress. Our robots however are static across the course of their lifetime. Combining this process with Evolutionary Algorithms creates the combined Evolution-Development (Evo-Devo) process.

III. Literature review

The most directly relevant work is two papers by Kriegman et. al. published in 2017 [3] [4]. The research explores what they name Ballistic Evo-Devo. Rather than evolving a static Volume for each Voxel, they evolve a start and end volume, and the robot linearly increases or decreases the volume of each voxel towards the end point over the course of a simulation. Their results found that even this minimal Evo-Devo model outperformed their pure Evo model. Additionally, they found that the difference between the start and end volumes decreased as the robot converged on a solution. However, this development is entirely deterministic and controlled by the evolution, rather than being development based on the interplay with the environment, which is where we focus our work.

Other work has been done with evo-devo for engineering. In 1997, Sipper et. al. explored the potential for bio-inspired hardwaware that developed much like bio-organisms do and

presented theoretical paths to pursue it [5]. They focus on the distinction between phylogenesis, the evolution of a species over generations, and ontogenesis, the development of a specific organism within its lifetime and the interplay that has with evolution. Kodjabachian et. al. combined evolutionary algorithms with the training of Neural Network to control simulated insects [6]. They found that the introduction of evolution to the traditional self-developed control networks provided a marked improvement. Guo et. al. combines evo-devo with multi-robotics to replicate cellular organisms [7]. They allow the robots to learn and adapt to a variety of tasks after evolving them using a traditional multi-optimization evolutionary algorithm. However, both these methods are developmental approaches that added evolution to them to help create better starting points for the evolution. We are taking a problem traditionally approached by evolutionary algorithms and introducing development, meaning we will need to define a development paradigm rather than use an existing one. The main thing to learn is the developmental operations are inspired directly by the bio-organisms being replicated.

IV. Method/Technical approach

A. Robot Encoding

The robots evolved are represented soft bodies defined by two primary units:

1) Point masses that hold mass and location information
2) Springs that connect two masses and hold material information

Robot movement is made possible by expansion and contraction of active "muscle" springs, supplemented by the passive actuation of "bone" and "tissue" springs. Each material is defined by a spring constant k, the maximum relative change in length ΔL_0, an activation frequency ω, and an activation phase ϕ. Spring activation is defined as follows:

$$L = L_0(1 + \Delta L_0 \sin(\omega t + \phi))$$

Each spring material is built from the following base materials inspired by Cheney et al. [2]:

	k	ΔL_0	ω	ϕ
Muscle	5000	0.14	1.0	π
Tissue	4000	0	0	0
Muscle2	5000	0.14	1.0	π
Bone	10000	0	0	0
Air	0	0	0	0

TABLE I
Base Material

1) Morphology: The robot morphology is defined by the neural network shown in Fig 1. A fixed list of random points are sampled from a unit sphere. Each point represents one mass in the robot. The neural network's role is to map each mass's location to a new location within a unit cube and assign a material to it. The neural network architecture includes a size 3 input layer, two size 25 hidden layers (each with ReLU activation), and a size 8 output layer. The input layer takes the original x-y-z location of the point. The output layer defines the mapped location in the first 3 outputs with a tanh activation. The assigned material is defined by the maximum activation of the bottom 5 neurons with a softmax activation.

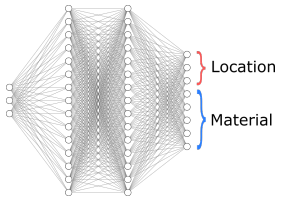

Fig. 1. Neural Network Encoding

Springs are allocated between each mass's 25 nearest neighbors using a GPU accelerated K-nearest neighbors algorithm. Spring materials are assigned as an interpolation between the materials assigned to the connected masses by the neural network.

B. Evolution Operations

Evolution is conducted by two major operations: mutation and crossover. Mutation makes small changes to an individual solution and is conducted by setting a random weight in the neural network encoding to a random value between -1 and 1. Crossover swaps genetic material between individuals. Crossover is conducted by taking a random neuron in the neural network encoding and swapping the connected weights between the individuals.

C. Massively Parallel Soft Body Simulation

The primary bottleneck in evolving robots lies in the execution time of the soft body simulation. A custom soft body simulator has been implemented that takes full advantage of massively parallel computation of GPUs by simultaneous simulation of multiple robots.

1) Soft Body Simulation: At each time step the simulator applies environmental forces (gravity, collisions, and drag) to each mass where collisions are simulated as a spring force with a high spring coefficient. The simulator then applies body forces from each spring to its connected masses using Hooke's Law. Finally, the resultant forces are integrated to find the position and velocity for each mass using Euler integration:

$$\mathbf{F} = k(x_1 - x_2) \quad \ddot{x} = m\sum \mathbf{F}$$
$$\Delta\dot{x} = \zeta\ddot{x}\Delta t \quad \Delta x = \dot{x}\Delta t + \ddot{x}(\Delta t)^2$$

The specific simulation parameters are described in Table II

Variable	Description	Value
μ	coefficient of friction	0.5
k_{floor}	floor stiffness	1000000
ζ	damping factor	0.99
g	gravity	9.81
Δt	time step	0.0005

TABLE II
Simulation Parameters

2) GPU Parallelization: To take full advantage of the computational power of the GPU, efficient GPU algorithms should have the following properties:

1) Computations can be run simultaneously
2) Data transfers from host to device are minimized
3) Coalesced reads from and writes to device memory, or in other words, there is contiguous and predictable access to memory

This method of simulation is particularly well suited to GPU computation. All mass and spring calculations can be performed independently by separate threads and data can live on the GPU for the entirety of the simulation. This allows easy use of parallelism.

Coalesced reads and writes, however, are not guaranteed by this method of simulation. Environmental forces and integration updates are coalesced as each thread can be assigned a mass by the thread ID. However, spring forces require near-random access to the mass buffer as there is no guarantee of correlation between the memory location of a spring and the memory location of the connected masses. This is a significant performance bottleneck that exists in soft-body simulators such as Titan [8].

To address this slowdown, mass information is brought from global memory to shared memory (i.e. L1 cache) with coalesced reads. Forces are calculated and added in this L1 cache and the integrated mass position and velocity are written back to global memory in a coalesced manner. This method , but shared memory drastically reduces the performance slowdown with much faster read and write speeds. As a consequence of this strategy, a robot's masses must fit within the 48 kB or the maximum amount of shared memory in a thread block. This also requires that a thread block only operates on robots that fit within this shared memory limit.

3) Data Architecture: The simulator combines all robots' information into a mass buffer and a spring buffer. Under the assumption that each soft body holds a maximum mass and spring count, an individual robot's mass and spring information can be uniquely identified. This allows the simulator to load a specific robot's mass information into shared memory for a thread block.

D. Development Operations

The primary differentiation in our work from previous soft-robotics evolution is the implementation of dynamic development. First, we created a spring-stress calculation metric to identify which springs would be subject to devo operations. Next, we define two primary devo operations, one of which works on the spring level, and the other on the mass level.

1) Spring Stress Calculation: To identify valuable springs for devo operations, we need to quantify which springs are doing the most and least amount of work. We do this by calculating the force applied by every spring at each time-step, henceforth the spring stress. However, keeping a running sum of the atomic total stress for each spring introduces a massive amount of global-memory reads, causing previously described slowdowns. With the atomic stress calculation, the simulation runs at 50% of the previous benchmark.

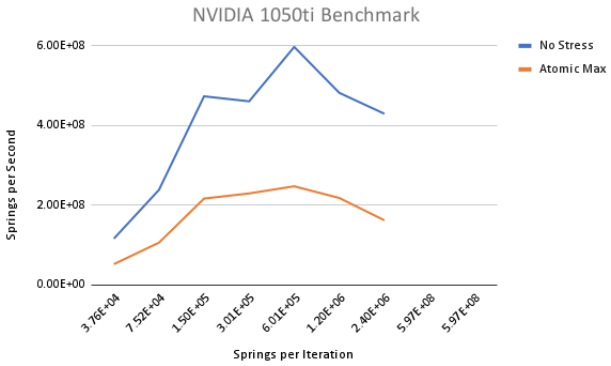

Fig. 2. A graph showing the 50% slowdown that introducing an Atomic Mass adds. Performance measured on NVIDIA 1050ti Laptop GPU.

Our first solution was to keep a global count of the most and least stressed springs at each time-step. This would reduce the number of global writes only to the springs incrementing their counter, rather than for every spring. However, this introduced a loss of parallelism when calculating the absolute max and min stresses across multiple threads, creating a slowdown that negated the gains from reduced read and writes.

We settled on our Skip-Shift solution. Instead of calculating the spring with the global max and min stress, each thread runs the calculation on only the springs it processes. Then, it adds a counter to a global array for its most and least stressed spring. The springs consistently applying high or low stresses accumulate the highest counter. To introduce variability and approximate the atomic max, we shuffle the max and min springs between threads every time a counter is incremented. We found that doing this once every 100 time-steps identified the most and least stressed springs with relative accuracy compared to the atomic method, while introducing only a negligible slowdown.

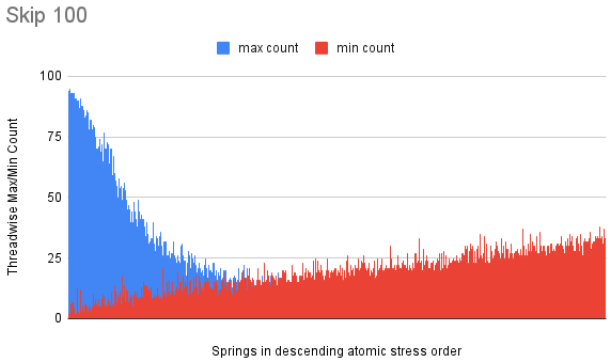

Fig. 3. A visualization of the amount of max and min counts each spring receieved over 10 seconds. The x-axis is every spring sorted in descending order of max stress, meaning the first spring is atomically the most stressed spring and the last spring is the least stressed spring. The tally count curves approximately align with the ground truth.

2) Spring Devo Operations: To replicate the process of growth and regression of utilized and neglected muscles, we decided to remove a chunk of the least stress springs and

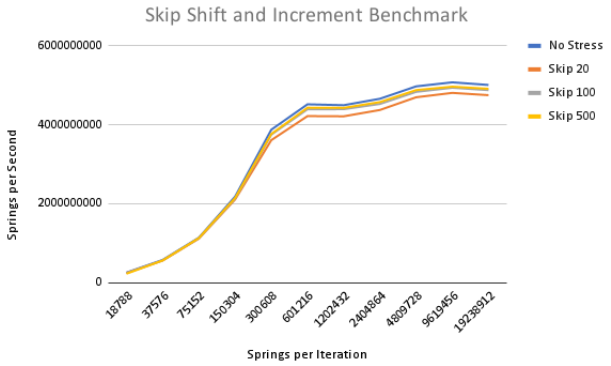

Fig. 4. A graph showing the performance decrease for implementing the skip-shift algorithm At Skip 100, the performances are within an acceptable range without the loss of quality from Skip 500.

introduce as many new springs randomly. The choice of random springs was made favoring simplicity, however we discuss plans for integrating the most stress springs into devo in our Future Work.

We do this Spring Devo operation once every devo cycle, and run our Devo process for a large amount of cycles. These are all parameters to be tuned, but starting estimates are provided below.

3) Mass Devo Operations: If a mass ever drops below a certain number of springs, the remaining springs would be primarily dedicated to moving that mass. To that end, we simply remove masses that fall below a threshold and add a new mass with the base amount of springs used during KNN. The mass is added randomly in bounded 3D space, with a minimum delta distance from the nearest mass enforced to prevent an infinite spring force. The springs will be connected to the new mass and randomly selected existing masses from the body. This operation is currently partially implemented.

4) Parameters to Vary: The parameters used for evo-devo will be set to the following starting values. Through trial and error we will vary them to find the optimal values for our solution:

- Number of Springs to Remove: 125
- KNN for initial spring allocation: 25
- Threshold for Mass Removal: 20
- Final Evaluation Time: 10 Seconds
- Devo Cycle Time: 1 Second
- Number of Devo Cycles to Run: 100

E. Fitness Function

The fitness function for evaluating the performance of the soft body robot is defined as the distance a robot can travel in a given direction over a 10-second period, measured in terms of the number of body-lengths covered. To avoid favoring strategies that fall in the desired direction, the fitness function incorporates a settling period that waits 3 seconds before measuring the distance traveled. It is worth noting that this settling period is only necessary for terrestrial walking robots. In future works, we expect to explore submarine robots, in which case this settling time will not be necessary.

F. Pareto Front

The Pareto front is a Multi-Objective Optimization concept utilized in both engineering and evolutionary algorithms. In Multi-Objective Optimization, solutions to a problem are evaluated with multiple criteria for success. Solutions that are Pareto Efficient are solutions that exceed every other solution in at least one of the objectives. In other words, they cannot be fully replaced by another solution without some trade off. The Pareto Front is the space of all Pareto Efficient solutions in a population. In Evolutionary Algorithms, rather than selecting which solutions to keep based on pure-fitness elitism, all solutions on the Pareto Front are kept. An Age-Fitness Pareto Front is often used in the absence of specific multiple objectives.

We implemented a Pareto-classify algorithm that assigns a Pareto layer to every solution in a given population. Each solution on a given layer is only dominated by solutions on higher layers, and dominates all solutions on lower layers. When comparing solutions for replacement during Crossover and Mutation, a solution is considered better if it belongs to a higher layer, or if belongs to the same layer with higher fitness. When creating new solutions, we assign new layers to all of our existing and newly-created solutions. During replacement, all solutions on the Pareto Front are exempt.

V. Results

A. Simulator Performance

Figure 5 illustrates the performance speedup of our new simulator utilizing this on-chip data cache over a similar simulator without this optimization. Performance benchmarks were conducted by simulating many 10x10x10 voxel-based cube soft-bodies over the course of 5 seconds of simulated time. The performance is measured by spring evaluations per second of real time. The results demonstrate a 175% performance speedups between the two simulators. By leveraging on-chip data caches, we achieved significant improvements to computational efficiency by trading for the maximum-allowable size of a simulated soft body. This improvement in simulation speed allows for more extensive exploration of the design space.

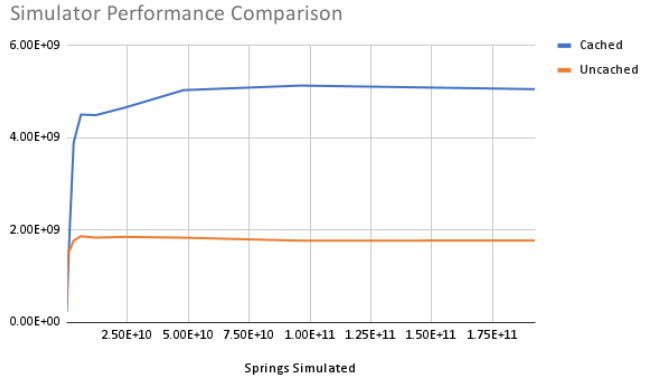

Fig. 5. Caption

B. Robot Encoding

Figure 6 showcases a selection of robots generated from random neural network encoding. This encoding produces a remarkably diverse set of robots that has the potential for unconventional soft robot design.

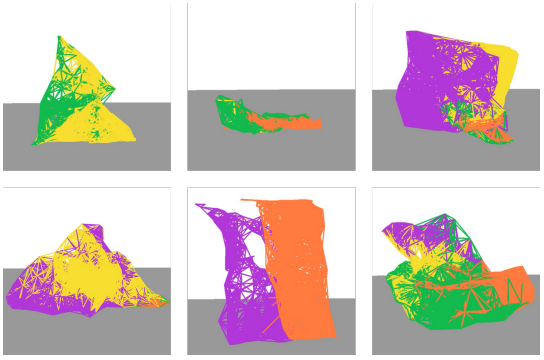

Fig. 6. Random Robot Solutions

VI. Conclusions

This paper made strides in establishing a foundation for future exploration in soft body robot design through evolutionary developmental optimization.

A major contribution of this research is the introduction of a state-of-the-art soft body simulator that outperforms current simulation software for the robots we aim to investigate. This custom simulator leverages the full capabilities of the GPU architecture to achieve remarkable speed and efficiency. By outperforming existing software, this simulator provides the ability for more efficient evolution and development of soft body robots.

Furthermore, we defined a novel robot encoding and devised associated evolution and development operations that facilitate interesting morphology evolution through the interplay of genetics and environmental interaction. This encoding scheme produces a diverse design set that can produce unconventional robot shapes and locomotion mechanisms. By incorporating environmental interactions, we expect to produce rapid environmental adaptions and greater phenotype diversity, even among solutions with similar genotypes. This integration of evolution and development may enable the exploration of more complex and adaptive solutions.

VII. Future Work

The major bottleneck to testing is the devo operations. We will debug the existing spring devo operations and finalize the mass removal and addition. Once complete, we will move forward with longer Evo-Devo runs. During these tests, the primary goal will be to tune the various parameters as discussed in the previous section.

Beyond further foundational work, we have a variety of potential modifications to examine:

- Implement two-point crossover with our Neural Network to allow for more variety in the evo operationsa
- Create the spring-mass pairings before passing the masses to the NN, rather than using KNN on the final morphology. This would allow the potential for robots to evolve with springs that do not connect to their nearest neighbor
- Add new-masses during devo on a timer rather than or in addition to when we remove a mass. This allows the robot to potentially grow more masses than it started with
- Reset the position of the robot to reset lengths after each devo cycle, potentially increasing system stability
- Modify the tension of highest stress springs rather than adding new springs
- Add new springs based on most stressed springs rather than randomly, making the devo operation more representative of effort done
- Weight the position of masses introduced in devo by the highest-stressed springs
- Use KNN for the new mass rather than random spring addition, which mimics the initial morphology creation
- Run tests to identify the minimum time before the top and bottom 100 springs are identified. We can then set our Devo Cycle Time to this value to ensure faster Devo Cycles while preserving data fidelity.

Once we have found our ideal setup, we will compare the full results of the Evo-Devo approach to the simple Evo approach to see if the desired improved results are found.

Further, we began to implement a computed-material concept for our spring calculations. Rather than reading precomputed material parameters from global memory to determine the spring activation functions, each spring material may be represented by a shorter encoding that can decoded calculated into the correct material on the GPU. This reduces the amount of data read from global memory, which should provide a significant speed-up. However, in our current implementation we see a mild slowdown instead. While this was put to the side to prioritize the evolution and development operations, we have the potential the investigate it further next semester.

References

[1] K. Sims, "Evolving virtual creatures," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '94*. ACM Press, 1994, pp. 15–22. [Online]. Available: http://portal.acm.org/citation.cfm?doid=192161.192167

[2] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson, "Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding," vol. 7, no. 1, pp. 11–23, 2014. [Online]. Available: https://dl.acm.org/doi/10.1145/2661735.2661737

[3] S. Kriegman, N. Cheney, F. Corucci, and J. C. Bongard, "A minimal developmental model can increase evolvability in soft robots," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2017, pp. 131–138. [Online]. Available: https://dl.acm.org/doi/10.1145/3071178.3071296

[4] S. Kriegman, N. Cheney, and J. Bongard, "How morphological development can guide evolution," vol. 8, no. 1, p. 13934, 2018. [Online]. Available: https://www.nature.com/articles/s41598-018-31868-7

[5] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Perez-Uribe, and A. Stauffer, "A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 83–97, 1997.

[6] J. Kodjabachian and J.-A. Meyer, "Evolution and development of neural controllers for locomotion, gradient-following, and obstacle-avoidance in artificial insects," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 796–812, 1998.

[7] H. Guo, Y. Meng, and Y. Jin, "A cellular mechanism for multi-robot construction via evolutionary multi-objective optimization of a gene regulatory network," *Bio Systems*, vol. 98, pp. 193–203, 06 2009.

[8] J. Austin, R. Corrales-Fatou, S. Wyetzner, and H. Lipson, "Titan: A Parallel Asynchronous Library for Multi-Agent and Soft-Body Robotics using NVIDIA CUDA," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 7754–7760.

VIII. Appendix - Supporting Files

A folder of supporting files can be accessed from the url listed below:

https://drive.google.com/drive/folders/1L_k5PiRujv0MFWD4s0UIfpNcAouqhU1q?usp=share_link

file	file type	description
Morphology Illustration	.mp4	An animation that shows a mapping of points in a sphere to the resulting location from a random robot encoding
Random Robots	.avi	A video showing short clips of random robots produced from random encodings.
Evolved Solutions	.avi	A video showing short clips of robot produced from evolved robot encodings over 10000 evaluations without the use of development operations.

TABLE III
Supporting Documents Table of Contents

IX. Appendix - Who did what

Task	Dawson	Amr
Robot Encoding	Programming	Debugging
Spring Stress Calculation	Programming	
Simulation Optimizations	Programming	Debugging
Development Operations		Programming
Evolutionary Optimizer	Initial Programming + Debugging	Programming
Literature Review	Yes	Yes

TABLE IV
Caption