# Evolutionary Algorithms and an Efficient Rocket Design

Cooper Davies[1]

Dept. of Computer Science, Faculty of Science, University of Calgary, 2500 University Drive N.W., Calgary, Alberta, Canada T2N 1N4
ctdavies@ucalgary.ca

**Abstract.** This paper focuses on the application of a genetic algorithms in respect to a 3-Dimensional, physical-space example constrained by user input. By implementing different approaches on how to incorporate scoring, and mutation of a generation, an aerodynamically, cost-efficient rocket is discovered. The multiple evolution-like techniques used allow for great customization, and tailoring to allow for dynamic, and widely diverse conditions for rocket creation.

## 1 Introduction

One of the largest, eclipsing problems of space ight currently is the complexity involved in escaping earths gravitational eld i.e. achieving escape velocity. This escape velocity is calculated using the equation $v = \sqrt{\frac{q2GM}{r}}$ [1], where G is the universal gravitational constant, M the mass of the earth, and r is the distance the rocket is from the earth. Due to having M on the numerator of this equation, the velocity v has to be extraordinarily large for planets. Earths escape velocity is $11.19km/s$ [2]. This brings to light another diculty though; the Tsiolkovsky rocket equation [3].Essentially what this means for the purpose of this study is that by adding more propellant, more weight is also added.

This extra weight means the need for more propellant, which again brings with it weight. This entails that the volume/weight of the fuel required is substantially larger than initially thought. When the cost of putting one kilogram of matter into space is between \$10000 to \$19000 USD [4], the desire for an ecient model is important. Is it possible then, to use an evolutionary algorithm to design a rocket which is eciently able to maximize the ability of getting to space in the presence of drag? By manipulating the physical 3-Dimensional rocket shape, can a possible blueprint for rockets be achieved in which a rocket is able to hold the required fuel yet also minimize drag?

What distinguishes this work from others is the ability to define a rigid, immutable object from which to build the rocket around. By removing the optimization of the payload of the rocket, the optimization of a rocket stemming from the algorithm will be very dependent on the payload given as initial parameters. The idea is to determine the viability of genetic algorithms for optimizing the container around an unoptimized payload.

## 2 Related Work

Among many works relating to specifically Aerospace Design [5], ones specifically relating to flight trajectories were considered. This consisted of design choices for the rocket [6] [7], (specifically shape-optimization [8]), which would directly effect the flight-path/trajectory of the rocket [9]. Although much work has been done on other aspects of aerospace design such as Ignition optimization for fuel [10] as well as the physical hull material [11], this paper focuses only on design optimization, and as such, will dismiss the latter papers.

One thing of importance is the fact that related work suggests that it is entirely unfeasible to perform an ecient launch using a single stage rocket [12]. Due to time limits and the complexity of such an endeavour, this project is only meant to model single stage rockets, thus it is likely that my model will be incredibly inecient. However, this intent of this model is not to be the most ecient, but to simply lay the ground work for an approximation of how an evolutionary algorithm for such a task might be created.

## 3 Conditions

In order to calculate the aerodynamics of an object, two standard approaches were considered.

– Physical Model
– Functional Model

In a physical model, the object is placed in a wind-tunnel, and various forces are measured during run-time [13]. This can be reproduced via programming collisions between the object and air particles, however in order to get an accurate result, a large amount of CPU power is required simply to perform individual time steps. A Functional model is a model in which the object is represented as a function, and various integrals/functions/heuristics are used to calculate the forces which would likely be produced [14]. The number of calculations are rather large, and require not only very complex mathematics, but a rather in-depth understanding of physics.

Because of the inherent flaws of the two approaches, a third approximation was designed (explained in 4.) , and implemented for a specific scenario in order to minimize computation time, whilst still allowing for a relatively accurate result.

## 4 Environment/Model

Two factors were maintained throughout the design process of the model. Firstly, a user must be able to produce any shape/size of exterior model they choose, and secondly that the algorithm is able to perform at the very least, 5 generations of search space. In order to allow these constraints, a Voxel-based[15] engine was developed. As shown in figure 1, a rocket is made up of individual blocks, which are all held together by a physical x,y,z location relative to one another. Due to the nature of voxels, a user could define how an individual piece(Block) of the rocket looks and behaves, and as long as it adheres to the defined format; input it for use in the program (figure 2). The result is a seemingly infinite number of hull possibilities, tailored to the needs of the user.

A few requirements are needed though, in order for an accurate simulation. Each individual must have a given payload-type block, a thruster-type block, and a fuel-type block. The payload block is required for creating a payload which is then surrounded by the rocket. Thrusters are required in order to generate force, and fuel blocks are required to power them. These are required only for calculations, and the actual shape if these blocks is up to the user.

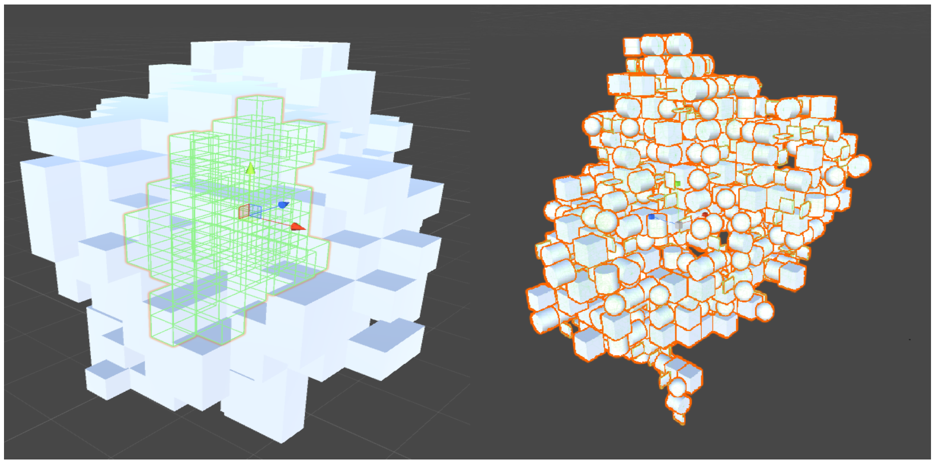


**Figure 1. On the left is a sample Rocket (Individual) with its payload defined in the centre outlined in green/orange. To the right is a larger, more sophisticated individual in which the blocks were predefined as input by a user**

## 5 Voxels

A model of voxels is then instantiated, and produced in physics-based environment using Unity [16]. The voxels are able to be manipulated within this environment, and calculations are performed to determine the orientation, position, velocity, etc. of the voxel. By containing the data of a rocket within voxels, the data representation of a rocket is able to be greatly minimized. A user must simply decide which types of blocks are allowed, and the factors associated with that block (i.e. cost, weight, name, and ID), and when an instantiated block is placed in the world space of the engine, it is given only the bare essentials for its existence. This is achieved by containing within the block structure, only location, rotation, orientation, and type values. Upon instantiating on the world, the program does a look-up of the ID for a block type, and imposes upon the instantiated block

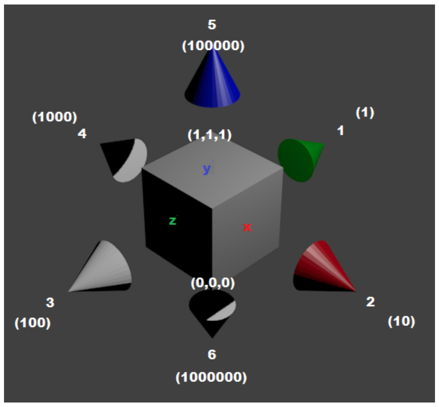


**Figure 2. The format defined for a block. It must be created with the following mapping. The positive Z axis must be the original facing component of the block.**

## 6 Functional Physics

As previously mentioned, the model we've created is an amalgamation of both the physical, and functional models. This is shown by the approximations used to calculate aerodynamics. In order to determine flight trajectory, a few minor shortcuts are made. One such shortcut is that we remove all blocks that do not have an exposed surface from the world space. However, doing this would greatly alter aerodynamics as the weight of the altered object would greatly affect drag. To compensate for this, each displayed block is parented to an invisible zero-point object located at the calculated centre of mass. The removed blocks' mass is then placed onto this point. Thus, we don't need to calculate the physics for the object as a whole, only the objects exposed to outside forces.

There are additional steps one can take to reduce computational power however. Rather than simulate millions of collisions of air particles normally performed in wind-tunnels, an approximation of collections of particles is used instead. To calculate this approximation, each individual block within an individual is observed at each time step, and its velocity vector is calculated $(V_x, V_y, V_z)$. A ray cast is then projected towards the object from this point, and calculates the projection of $V- > N$ where N is the normal of the point of contact from the ray cast $(Cos(\theta) * V = (N_x, N_y, N_z)$ this new vector is the normal component of the drag force. This vector is then multiplied $F_d = N * V^2$ to generate a relatively quick approximation of drag force. The drag force is then applied to the normal of the point of contact from the raycast. A demonstration of this can be found in figure 3.

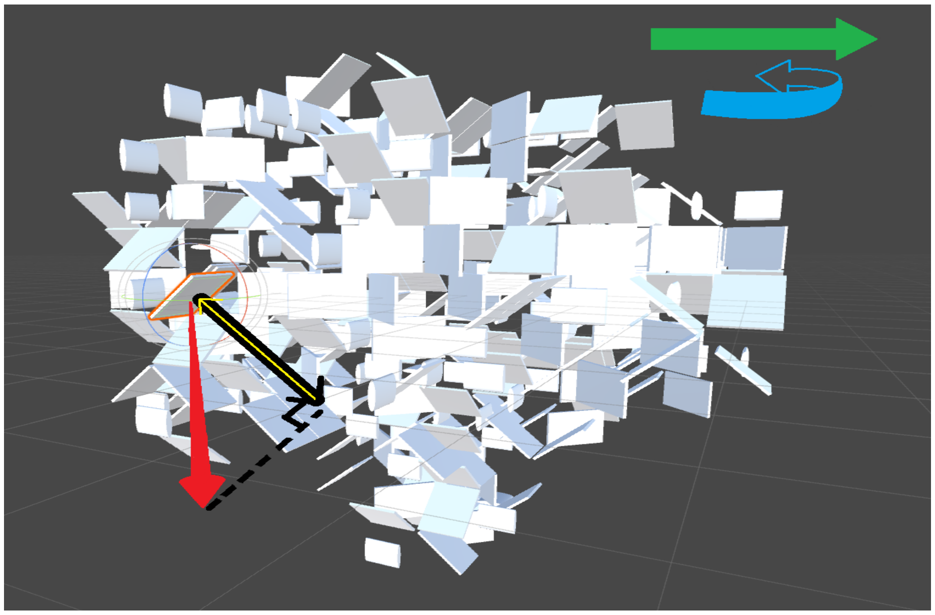


**Figure 3. The force of drag applied by the movement of the rocket. The green arrow indicates the velocity of the rocket, while the blue arrow indicates the angular velocity of the object. This causes the block highlighted in orange to have a local velocity proportional to the red arrow. Calculating the projection of local velocity to the black arrow (the normal of the raycast from red), the yellow arrow is generated, which is the initial force vector of drag. This vector then gets multiplied by the velocity squared to generate the drag on this object**

Thrust force $F_t$ of the individual is calculated quite similarly. Instead of calculating the drag on thruster objects, instead $F_t$ is calculated by taking the normal of the forward of the object (The normal of the direction the block is facing), and inverting the normal multiplying by the user defined force generated by the fuel source $F_t = -N * F_f$. At each time step, fuel is depleted at a rate defined by the user, and when all the fuel is consumed, force is no longer applied to the thrusters.

## 7 Scoring

To formalize the difference between a "good" individual and a "bad" individual, a fitness function or "scoring mechanism" must be defined. In this project, fitness was defined so as to maximize the height reached, while minimizing the cost, and weight of the rocket. The fitness function can calculated with the following formula $S = (((h+1) * N_{thrusters}) + t)/(w * c)) * 1000$, where h is the maximum height achieved by the rocket, $N_{thrusters}$ is the number of thrusters on the rocket, t is the time the rocket was in the air, w is the weight of the rocket, and c is the cost of the rocket. The factor of 1000 exists solely to make the number more readable.

A few considerations had to be made in order to make the scoring function better reflect reality. For example, h+1 was used instead of simply h because if the rocket only ever fell, then $h = 0$ which would cause for most of the first generation to have the relatively same score of 0. This would make the algorithm take longer to generate better results as each incremented generation would have minimal differences.

Consider the variable $t$ representing the time in the air. Say an individual flew in spirals downward, and only ever circled around the initial launch position, but fell slowly due to some minimal upward thrust. This individual can be said to be better than a competing individual in that it obviously has a larger propelling force, just that the propelling force is in the wrong direction. By simply orienting the thrusters properly, the spiralling rocket could drastically improve its height.

## 8 Generational Development

The freedom allowed to us from the quick process calculations allots us the ability to use large generations of individuals as mutation parameters. That is to say that initially a starting generation $g0$ is randomly produced, each with their own configuration of blocks around a given payload. $g_0$ is given a determinate size $g_0[n]$ where $n$ is the number of individuals in $g_0$. Then for each individual within $g_0$ say $i_0$ we perform the physics described above. After scoring, the individual with the best fitness is chosen. All other individuals are culled, and replaced by the best scoring individual. The replicated individuals are then mutated and the process repeats.

## 9 Mutations

The genotype of a rocket can be broken down as an expression of physical properties. That is to say that the phenotype of an individual arises from the combination and orientation of the blocks it is composed of as opposed to the different actions or sequences it could perform to arrive at a given goal. Due to the time independent display of the phenotype, the ordering of the genotype does not have an impact on the phenotype.

An interesting design implication that arose from this project is the disallowed recombination mutation operator [17]. Due to the structural 3D topology, a recombination event occurring with an individual of the same generation would require much too many constraints for what is to be considered an evolutionary approach. As an example, explore the possibility of an individual who is represented as a row of blocks extending entirely in the horizontal x axis. This individual is to get paired with an individual where its configuration is entirely vertical in the y axis. By randomly swapping individual pieces, the genotype would be perfectly normal, however, the phenotype that arises could be nonsensical. A visual representation of this is shown in figure 4.

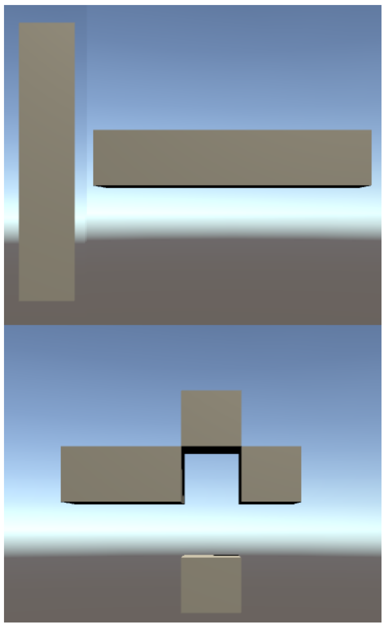


**Figure 4. An example of a nonsensical phenotype which arises from the recombination of two normal phenotypes**

Instead, we are left with the very basic mutation operators [17] of duplication, mutation, and isolation. The generation begins by simulating the physics described above, and being scored via the fitness function. After being scored, we select the individual with the best score (ties are handled via random selection), and delete or cull the remaining individuals from the generation. The remaining individual is then replicated to fit the size of the generation, and mutated based on given user parameters. The parameters for mutation are Addition Probability, Null Probability, Alteration Probability, and the Degree of Mutation.

To begin the mutation process, we iterate over every "open" space surrounding the individual. These are the spaces where a connecting block is allowed to be placed, but currently there does not exist a block in that spot. At each space, with probability $p_{add}$, a new block is added. The type and orientation of the block are completely randomized.

Following the addition of blocks, we then iterate over every block within the structure. This includes blocks that have been previously added, for if we did not consider them then we might yet again give rise to the formation of nonsensical phenotypes. For each block we decide if we would like to mutate that block with $p_{null}$, the Null probability defined. If we decide to mutate that block, we then look at the Alteration probability. With probability $p_{alter}$ we decide if we would like to either alter the block, or delete it $p_{delete}$. The probability of $p_{delete}$ can be defined by $p_{delete} = 1 - p_{alter}$. This is taken care of simultaneously with the Alteration probability where if it was decided not to alter the block, then it definitively gets removed. It would be redundant to first check if we would like to alter a block, then if we would like to delete it for deleting an already altered block wastes the computational power that went in to altering the block. The alteration of a block simply picks a random block and a random orientation, and replaces the current block with the newly random one. These newly mutated individuals are then simulated, and the process repeats. When a generation seems to have settled to a possible local maximum, in order to attempt to offset the locality, we iterate backwards through our list of rockets that were at one point the current best, and set an earlier version to be the current best, in hopes that a previous, less-fit individual is able to more likely mutate away from a local maximum.

## 10 Discussion

The single greatest limiting factor in this project is the reminder that it is entirely unfeasible to perform an ecient launch using a single stage rocket Vlacic:2010. When running the simulation using realistic fuel models (a fuel force that burns proportionately, and with similar force to that of real world fuels [4]), the rockets never make it further than simply falling forever. True to the initial assumption that the square-cube law [18] is indeed impossible to overcome, rockets that grew larger in size were too heavy to lift, which reduced their score. Inversely, rockets that were too small were unable to generate the force required, and also had a decreased score upon mutating. Thus, individuals were lucky if they progressed past the first few initial generations. This was done assuming that fuel burned at a rate of $5000T/s$ and produced $8400000Lbs/s$ of force [19]

In order to truly test the algorithm, an unrealistic assumption was made; The fuel burns so slowly that it is nigh infinite. This allows the individuals to fly regardless of fuel amount, while still having their weight act on the aerodynamics. When decreasing the burn rate, generations improved until approximately generation 600. The improvements were minimal and sporadic, but indicative of continued improvement.

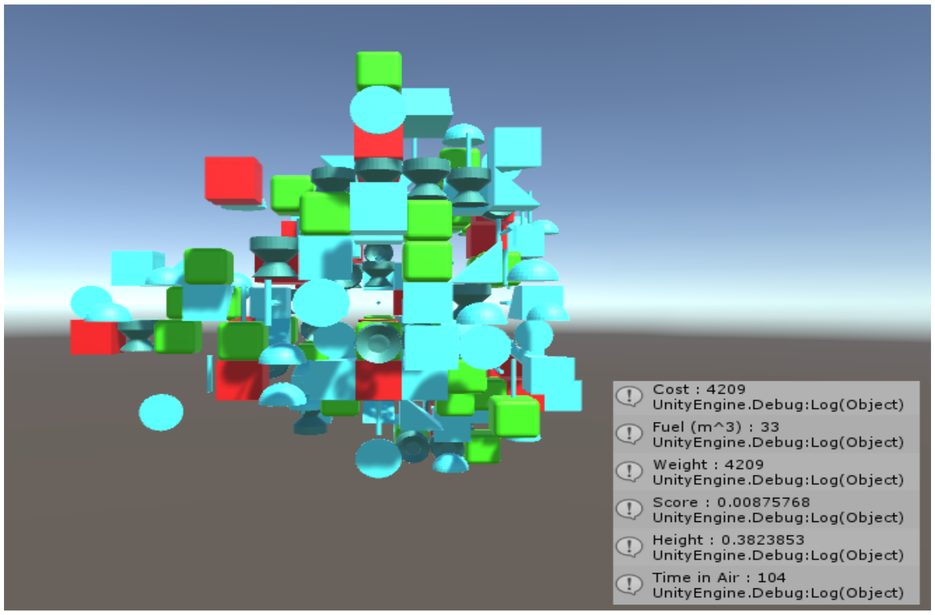
Another failing of the algorithm is the creation of unrealistic rockets. In its current form, the algorithm outputs what appears to be a conglomeration of pieces as opposed to a singular streamlined object. This is due in part to the inability of enforcing a system which disallows certain configurations of the building pieces of the rocket. For example, imagine two thrusters side-by-side. Their orientation is such that each thruster propels their force directly toward the other. In reality, this would cause for both thrusters to destroy each other, however in the current model, this is a perfectly acceptable configuration.

When simulating generations it is important to remember that evolution does not result in "better" individuals, only individuals more suited to the current task [20]. Originally the simulations were initialized with smaller individuals, requiring the need to grow in order to achieve a better fitness score. However, as previously mentioned, this made the fitness scores incapable of increasing. Instead, individuals started large, and were slowly "shaved" off until the smallest possible rocket is found which produces the greatest relative lift. Opposite adding pieces, when removing pieces, the weight decreased, which allows for the fuel source provided to generate more lift. This cause for very minimal increases in fitness. However, due to these minimal differences, the ability to constantly "evolve" is continuous. Figure 5 shows the final rocket of a long simulation

## 11 Future Work

As much work as there was done on this project, there is just as much left to be done, if not more so.

Although there is a large amount that still needs to get done, for the sake of brevity, I will only list a few ideas which would be more important than other to implement



```
Cost : 4209
UnityEngine.Debug:Log(Object)
Fuel (m^3) : 33
UnityEngine.Debug:Log(Object)
Weight : 4209
UnityEngine.Debug:Log(Object)
Score : 0.00875768
UnityEngine.Debug:Log(Object)
Height : 0.3823853
UnityEngine.Debug:Log(Object)
Time in Air : 104
UnityEngine.Debug:Log(Object)
```

**Figure 5. The design of a rocket using realistic fuel parameters (140000 N/s force, 83 T/s consumption). Green blocks are payload blocks, Red is fuel, Darker teal is thrusters, and light blue is hull.**

- 1) A file saving structure.
  when using unity, it is a difficult process to serialize the models and their properties, and unfortunately time did not allow this to be implemented. This would allow for users to insert their own payloads/rockets rather than being forced to work with random ones
- 2) A more strict genotype.
  As previously mentioned, it is entirely possible for configurations which shouldn't be possible to be present in the phenotype. Removing this would require a large amount of coding, and so was removed from development in the earlier stages.
- 3) A Clades-like mode [21]
  I would like the ability for users to be able to decide their own fitness function simply based on visual appearance and selection. Currently, I am at a loss as to how to do this within unity as multiple cameras becomes very confusing very quickly.
- 4) A Game mode
  I think it would be interesting for users to attempt to build a rocket to lift a certain payload faster than the algorithm could generate one. However, this again falls into the category of "I don't know enough about unity"

## References

1. G., W.N.: Excel Preliminary Physics. PASCAL Press, PO Box 250, Glebe NSW 2037 (2000)
2. S., P.R.: Planets and pluto: Physical characteristics (2008)
3. D., P.: The tyranny of the rocket equation (2008)
4. Corporation, F.: Space transportation costs: Trends in price per pound to orbit 1990-2000. (2002)
5. B., A.M.: Genetic algorithms in aerospace design: Substantial progress, tremendous potential. (2003)
6. Bramlette, M., Cusic, R.: A comparative evaluation of search methods applied to the parametric design of aircraft. Proceedings of the Third International Conference on Genetic Algorithms (1989)
7. Kroo, I., T.M.: A quasi-procedure, knowledge-based system for aircraft design. Presented at the AIAA/AHS/ASEE Aircraft Design, Systems, and Operations Meeting (88-4428) (1988)
8. Sharatchandra, M.C., S.M., Gad-el Hak, M.: New approach to constrained shape optimization using genetic algorithms. AIAA Journal **36**(1) (1998)
9. Mondoloni, S.: A genetic algorithm for determining optimal flight trajectories. AIAA Guidance, Navigation, and Control Conference and Exhibit (98-4476) (1998)
10. Peretz, A., B.M.: Thrust profile optimization of solid-propellant two-pulse motors for range extension. Presented at the AIAA/SAE/ASME/ASEE Joint Propulsion Conference (92-3354) (1992)
11. Venter, G., Haftka, R.: A two species genetic algorithm for designing composite laminates subject to uncertainty. AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit (96-1535) (1996)
12. N., V.: Escape velocity. Undergraduate Journal of Mathematical Modeling: One + Two **3**(12) (2010)
13. L., N.: The methods of drag force measurement in wind tunnels. University of Gavle (2013)
14. Cebeci, T.: An Engineering Approach to the Calculation of Aerodynamic Flows. Horizons Publishing Inc., California State University (1999)
15. Frieder G., Gordon G., R.R.A.: Back to-front display of voxel based objects. IEEE Computer Graphics and Applications **5**(1) (1985) 52 – 60
16. (Unity)
17. D., G.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Professional (1989)
18. A., D.: How Mechanics Shaped the Modern World. Springer Science and Business Media (2013)
19. Aeronautics, N., Administration, S.: (Space launch system (sls) fun facts)
20. of California, U.: (”misconceptions about evolution”, understanding evolution)
21. Burt T., P.P.: Interactive evolution aby duplication and diversification of l-systems (2013)