Evolvable Modeling: hierarchical evolution for 3–D jet plane models.

Thang C. Nguyen

Email: cthang@uirvld.csl.uiuc.edu

1 Introduction

This paper describes the functions and organization of a collection of Mathematica modules designed to evolve 3–D models of jet planes. The algorithm starts with some randomly-generated models from a given model-grammar with generic design rules and loose constraints, and the user interactively determines the "fitness" of planes in each generation, shaping the evolution path of the next population of models. For "historical reasons", the population size has been arbitrarily set at 4, with only 1 crossover per generation by deterministic selection. All of that should be fixed, and fairly straightforward, actually. Just that I haven't got that much time to play with these 4–population runs, really.

With respect to genetic operations (crossover and mutation), each of our jet plane models has 2 main levels:

- Structural level: with a production grammar and symbolic parameters. A model design expressed at this level has the form of a tree, with no numerical values associated. A crossover at this level is performed by exchanging 2 subtrees of the same functionality: e.g., 2 Wings subtrees.
- DNA–string level: for the numerical values of the parameters, all coded in binary strings. At this level, a crossover between 2 models can be preformed by crossing 2 binary strings, both of which refer to some common parameter, e.g. WingHalfSpan.

Furthermore, the parameters chosen for crossover are from a substructure sampled from the overall design structure. This has the effects of a hierarchical sampling for both structures and parameters: those lower-level structures/parameters are more frequently selected than higher-level ones. Since our jet plane models currently are fairly simple, having only two structural variations: *Engines* and *TailsMount*, the hierarchical sampling effects being subtle, we have chosen to set a constant threshold crossover probability of 0.9 each for *Engines* and *TailsMount*, just to get more chance of structural crossover. It would be simple to later allow also for true hierarchical sampling structurally, if desired.

Each model design has 19 "accessible" parameters (that can participate in crossover and mutation operations) and 13 "hidden" parameters that are kept the same for all and every model design. Most of the hidden parameters are actually algebraic relations or constraints between 2 or more parameters. In contrast, only a few "accessible" parameters are of this type. Mathematica allows very general relationships to be expressed into the substitution-rule form that we use for all parameters.

2 The main program flow

We use italics to denote Mathematica module variables, functions, or commands. The symbol **Mma>** denotes the Mathematica prompt; *<CR>* denotes carriage return, from user.

To run the software, the user sets up Mathematica, set the variables *SizeOfPopulation* and *FitnessGoal* to the desired values, loads all modules, then calls:

Mma> *EvolvePlanes["somestring"]; <CR>*

EvolvePlanes is the main module. It calls the random generation module *GenerateDesigns[sizeofPopulation]* for the first generation, then calls on *BreedNewGeneration[startPop, maxNumofPop, fitnessGoal]* to process the subsequent generations.

a. The first population of models consists entirely of randomly generated models from the model grammar, model primitives, parametric relations and constraints.
b. Every generation, each jetliner model is given a score, judged by the user. The two models with highest scores are then selected for crossover, or "mating". One randomly selected model (among the current 4) is then "mutated". And one new model will be randomly generated (in the manner of the first-generation models.) This process creates the new generation of 4 models.
c. The above loop for step (b) is continued until at least one model is given a score higher than a threshold of acceptability (say, any model to be accepted has score > 100.) Alternatively, a maximum number of generations can be set beforehand, and many runs can be tried.

During the interactive processing, after displaying a new generation of models, the main module will ask and wait for a list of evaluation scores from the user, as follows:

___just shown generation number (current generation number)

___enter list of 4 scores for the planes: (user enter a list of scores, for example, {25, 10, 40, 8}, then <CR>)

Currently, with determinsitic mating, the two models with the highest scores are always mated (crossover). In this example, planes numbered 1 and 3 would mate to give 2 offspring planes in the next generation (to be put at locations numbered 1 and 2, for ease of tracking). However, it is simple to allow for other elaborate, non-deterministic selection schemes here if desired.

The results of crossover and mutation (also a crossover, with a random un-recorded model) are also printed out for inspection. This could give some pretty long lists, and could be omitted to avoid a source of distraction and cluttering on the Mathematica screen.

To help keep a human-readable records, very screen printout during an *EvolvePlanes* session is also saved into a file named *runText-datestamp-...somestring* in the current directory.

When done with all generation processing, *EvolvePlanes* automatically saves all populations of models into a file named *runPlanes-datestamp-...somestring*. The list of all graphics renderings (*Graphics3D*) of the planes are not saved, as they could be reproduced by rendering modules, and typically the graphics size is large, more than 40 Kbytes for each plane. Currently, producing the 3–D geometric primitives and rendering each plane model (from the symbolic design and list of parameters) by *DrawFromDesign* takes about 40 seconds on a Sparc II.

When finished, *EvolvePlanes* returns a (big) list of 2 lists, one being a list of all designs (symbolic structures) and parameters, (grouped into populattions), the other being a list of all the 3–D graphics (*Graphics3D*) of the individual jet models, ready for display. Note that since graphical rendering is most time consuming (about 95% of all processing time), during the interactive processing of *EvolvePlanes*, only whole generation (of 4 planes each) are displayed for user evaluation, but no display of individual models, which can be done after *EvolvePlanes* finishes.

We keep a simple format for quick comprehension of the process. In each population (except the 0th generation), models at positions 1 and 2 are children of a crossover operation (sexual production) from the last generation; model at position 3 is a mutated children; and model at position 4 is a new randomly generated model. Note that we view a mutation as a crossover with a random unknown model, which is appropriate in our framework, and saves some coding.

3 Some notes

Functionally, the current Evolvable Modeling consists of 6 main groups of Mathematica modules: random grammar-based model generation, model structural sampling and crossovers, parameters manipulation, graphical model rendering/display, geometric primitive generators, and the main module (*EvolvePlanes*) for initialization, user interaction and data collection.

Please refer to the program listing for complete information. Later revision of this could include more details.

There are quite many significant improvements that could be made. Especially with speeding up the graphical rendering (possibly on a different software platform than Mathematica), and with interfacing the modules into a useful system for structural design, object recognition, etc.

Comments and improvements, bug reports and/or fixes are very welcome. Thank you.

References

[Calloway 87] Calloway, D., Using a genetic algorithm to design binary phase-only filters for pattern recognition, *2nd Intl. Conf. on Genetic Algorithms*, 1987, pp. 422–429.

[Goldberg 89] Goldberg, D. E., Genetic Algorithms in search, optimization and machine learning, *Addison-Wesley*, 1989.

[Hill 92] Hill, A., and Taylor, C. J., Model-based image interpretation using genetic algorithms, *Image and Vision Computing*, vol. 10, no. 5, June 1992, pp. 295–300.

[Ikeuchi 88] Ikeuchi, K., and Kanade, T., Automatic Generation of Object Recognition Programs, *Proc. of the IEEE*, vol. 76, no. 8, August 1988, pp. 1016–1035.

[Koza 92] Koza, J., Genetic Programming, on programming of computer by natural selection, *MIT Press*, 1992.

[Sims 91] Sims, K., Artificial Evolution for Computer Graphics, Computer Graphics, vol. 25, no. 4, July 1991, pp. 319–328.