

# On the Possibilities of an Analytic Synthesis System Using Construct Based Genetic Algorithms

**Michael J. Prerau**  
**Columbia University**  
**4782 Lerner Hall**  
**New York, New York 10027-8346**  
**mjp61@columbia.edu**  
**<http://www.columbia.edu/~mjp61/>**

This paper explores the possibility of the use of *Construct-Based Genetic Algorithms* as a means of implementing an *Analytic Synthesis* system through the development of a music generation system. The overall objective of this system is to analyze several user-selected musical works and create a new piece of music that will retain desired characteristics present within the input compositions. This paper describes the important theoretical aspects and issues surrounding the initial design of the system, created for use with simple monophonic music.

## **Introduction**

*Construct-Based Genetic Algorithms* can be considered as a means for implementing an *Analytic Synthesis* system. In this paper, this concept is explored through the development of a music generation system. The overall objective of this on-going research is to analyze several user-selected musical works and create a new piece of music that will retain desired characteristics present within the input compositions.

## **Analytic Synthesis**

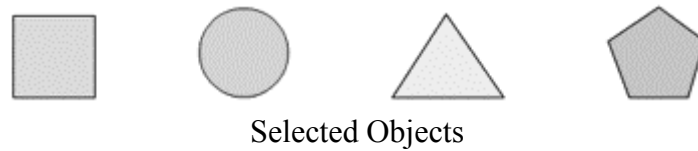
The process, which shall be called *Analytic Synthesis*, is used to create new objects or systems that will possess desired characteristics from preexisting entities. Preexisting working examples are analyzed to acquire the building blocks that will then be used to automatically create new configuration possibilities. The purpose of this system is to take musical pieces that are deemed desirable in different ways by a user and to use Analytic Synthesis to create a new piece that should be considered. The goal of the initial implementation is to serve as proof of concept for this method and for the utilization of genetic algorithms in its implementation, as well as to pave the way for wider-reaching subsequent applications.

Analytic Synthesis is implemented by a three step process:

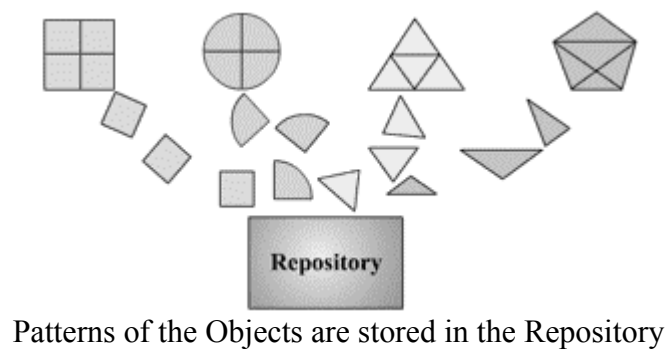
- Pattern Recognition
- Analysis Integration
- Synthesis

Before Analytic Synthesis can begin, preexisting objects, from which the new object will be produced, must be chosen. Each object should possess desirable qualities that are wished to be carried through to the new object. In many cases all that might be known is simply that each object works in some way. The objects are therefore included in the input set with the

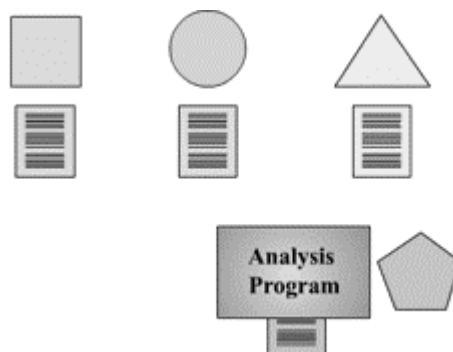
hopes that a new working object will be produced. Once these objects are chosen, Analytic Synthesis can begin.



In the *Pattern Recognition* step, a set of user-selected objects is run through a pattern recognition program that isolates relative or context-free patterns within each object. The patterns are then stored within a repository for later use in the genetic algorithm

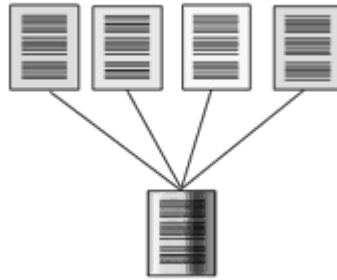


In *Analysis Integration*, each of the input objects is run through an analysis program. This program produces an analysis file for each object that encapsulates its important qualities. Deciding exactly what these qualities are and how to isolate them is a problem crucial to the process, for the analysis program is the most important factor in determining the desirability of the output.



An Analysis file is produced for each Object

When an analysis file has been produced for each object, they are all combined to create a new file called the *Merged Analysis File*. This file will work as the blueprint from which the generated object is produced. The quality of the merging can be enhanced greatly if at the beginning the user can specify which aspects of each object they feel to be the most important. This way each aspect of the analysis can be weighted accordingly when the merger takes place.



A Merged Analysis File is produced

The *Synthesis* step utilizes the blueprint to reverse the process in order to yield a new object that fits the requirements set forth within the Merged Analysis File. The methodology utilized for this reverse creation is use of a *Construct-Based Genetic Algorithm* (CBGA). This uses the patterns from the Repository as elements in the chromosome to be combined to form a hybrid object. The object is then run through the analysis program used in Analysis Integration and an analysis file is produced. This file is then compared to the Merged Analysis File and is scored based on its closeness.



A CBGA creates a new Hybrid Object, which is scored against the Merged Analysis File

The result should be a new object that possesses important desirable aspects that are present within the set of input objects.

Work similar to several aspects of such a system as this has been seen before in the realm of music, most notably in the work of David Cope (Cope, 1992). He has created a system called EMI (Experiments in Musical Intelligence) that attempts to generate music in the style of a given composer by recombining phrases gathered by pattern recognition techniques using grammars similar to those used in natural language processing. An Analytic Synthesis system applied to music would differ from EMI in the respect that it would use a type of genetic algorithm for the actual generation of the pieces, whereas EMI relies on augmented transition networks. Another area in which the two systems would differ would be in that Analytic Synthesis is that it would base its comparison off of a holistic analysis of the input pieces rather than the statistical analysis used by EMI. In Analytic Synthesis, the analysis section of the system would ideally look at many different aspects of the pieces, from looking at the overall structural and thematic constructs to doing low-level statistical analysis of the notes as Cope did. These differences arise due to the fact that Analytic Synthesis is concerned with combing elements with different or unknown qualities, as opposed to discerning and replicating a single style.

### **Construct-Based Genetic Algorithms**

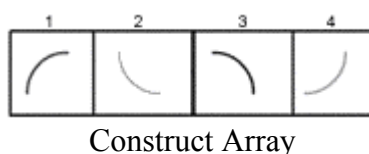
For the synthesis step, a species of the genetic algorithm, which will be called a *Construct-Based Genetic Algorithm*, is created. The purpose of a CBGA is to come up with optimal configurations from a set of "physical" building blocks with the aim of producing some larger macro-entity. Whereas standard genetic algorithms use each item in a given chromosome to represent a value, a CBGA uses each item within a chromosome to represent an object or group of objects. An entire chromosome therefore represents a certain configuration of these

objects or constructs. Thus, the aim of a CBGA is to come up with an optimal and novel configuration as opposed to optimal values.

The main elements of a CBGA are:

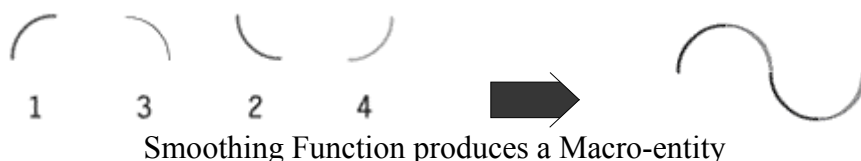
- The Construct Array
- The Smoothing Function
- The Fitness Function

The initial task in the implementation of a CBGA is the isolation of the constructs that are to be the foundation for the output. In the case of this project, the construct isolation will be achieved by the Pattern Recognition phase of Analytic Synthesis. Once the constructs to be used as building elements have been isolated, implementation begins by taking each construct and placing it within the Construct Array. This is the place where the constructs will be held until the actual generation begins.



The gene pool is created randomly or algorithmically, as would follow with a regular genetic algorithm. While a chromosome remains but a list of integers, each number within a given chromosome represents the index of the construct repository array.

Since a CBGA deals with the combination of discrete components, it must also contain something called a *Smoothing Function*, which will reassemble the parts into a macro-entity that will be tested by the fitness function. In this application, the smoothing function will take the fragments of music within each chromosome and join them into a whole piece of music. In a CBGA, the design of the Smoothing Function is just as important as deciding on a fitness function, because the more intelligently the macro-entity is constructed, the better chance there is of having a useful output object.



The Fitness Function of the CBGA acts on the macro-entity and scores it as would happen in any regular genetic algorithm.



Fitness Function scores the Macro-entity

Attached to each chromosome there is a *Mutation List*, which contains information about how

each “smoothed” macro-entity is altered if mutation occurs. These mutations are applied, if present, right after smoothing so that they can be taken into account when evaluated by the fitness function. As generations progress, pattern recognition is applied to long-surviving chromosomes and these mutated segments can be added into the construct repository. In that way, though original segments are initially used, over time the chromosomes will end up containing newly generated “working” constructs.

Breeding takes place in almost the same way as in a normal GA, except the Mutation Lists must be bred as well when two chromosomes reproduce.

### **Implementation Design**

Before development on any system can begin, a platform must be decided on. In respect to receiving input from music files, the system that was chosen was the GUIDO music notation format. This format was selected primarily because it allows music to be specified by quantized pitch and metric values in a simple text file. This saves a lot of work that would need to be done if one were dealing with a MIDI file. Secondly, Java was chosen as the implementation language for its portability and web accessibility.

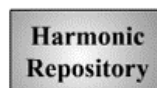
With the platform decided on, the implementation design process can continue. In the Pattern Recognition phase, a set of input music pieces in GUIDO format is run through a pattern recognition program that isolates common relative harmonic patterns as well as metric patterns. The patterns from all of the input pieces are then gathered in Repository Arrays—one for the harmonic patterns and one for the metric.

The system will first take the GUIDO file and convert it into an object. The system then will pick the size of the pattern that it is going to look for. It will go set by set and gather the harmonic patterns that are then placed in the Harmonic Repository. What size patterns work the best as building blocks is yet to be discovered, though it is conjectured that they will be multiples of the number of beats in a measure.

[ \clef<"g"> c d e c c d e c e f g/2 e/4 f g/2 g/8 a g f e/4 c g/8 a g f e/4 c ]



0-1-1



Harmonic Patterns are found and placed in the Harmonic Repository

All the harmonic patterns that are found are relative in the sense that the values of all their elements are relative to the first note of the pattern. The pattern is converted into numerical representation with the first item equaling zero, and the following elements represented by the distance from the first note.



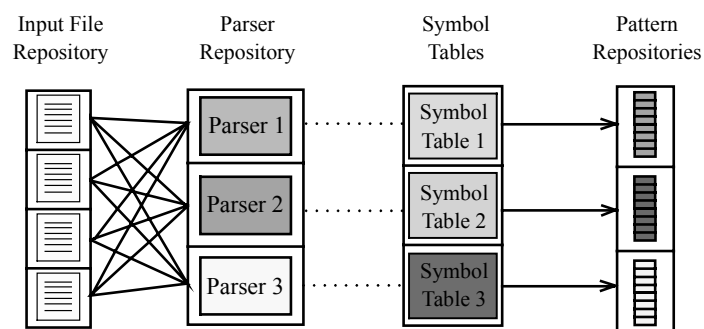
Conversion to Relative Harmonic Patterns

The metric patterns are collected as well and placed in the Metric repository. A quarter note is represented by 4, a half note by two, an eighth note by 8, etc. Notes of more complex values are represented in decimal form.



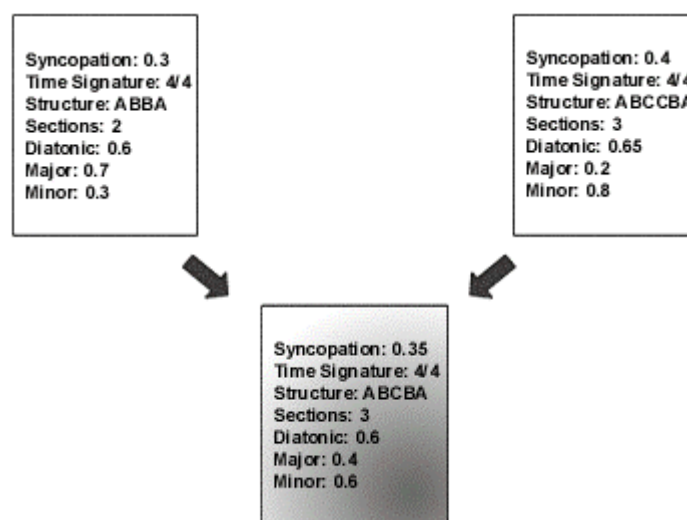
Conversion to Relative Metric Patterns

## Pattern Recognition



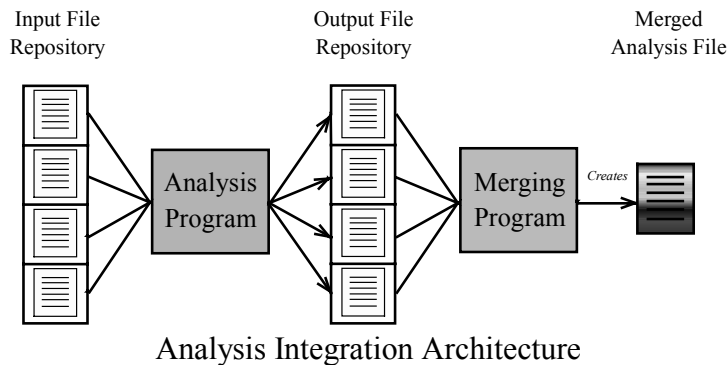
Pattern Recognition Architecture

In the Analysis Integration phase, each of the input pieces is run through a stylistic or compositional analysis program, the output of which is an analysis file that encapsulates characteristics present in the piece. The analysis files for all the pieces is then merged to create a new analysis file called the Merged Analysis File. This file will serve as the blueprint for the newly generated piece. As it turns out, the development of the merging program in itself might also be considered a problem best solved by Analytic Synthesis.

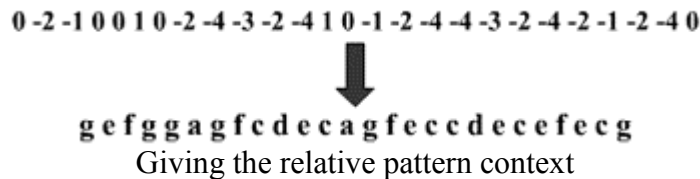


Creation of the Merged Analysis File

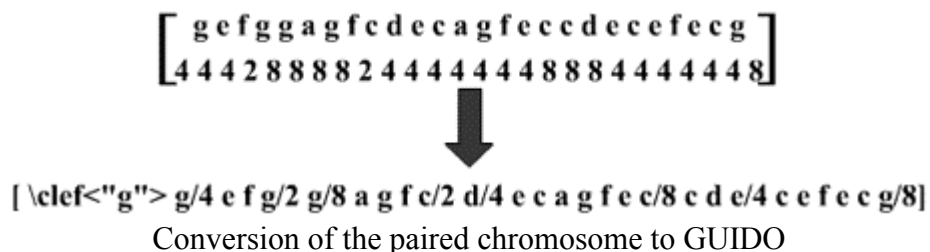
# Analysis Integration



The Synthesis phase utilizes the blueprint to reverse the process in order to yield a new piece of music that fits the requirements set forth within the Merged Analysis File. The system first takes all of the patterns in the Repository Arrays and randomly or grammatically combines them as pieces of music to be used as chromosomes in the CBGA. If done grammatically, this will insure at least a slightly more intelligent seed. Each harmonic chromosome is paired up with a metric chromosome of the same length. Once the genome is created in full, the smoothing function takes the construct numbers and converts them into their actual patterns. It then assigns a value to the first element in the harmonic chromosome of the pair, giving the piece tonal context.

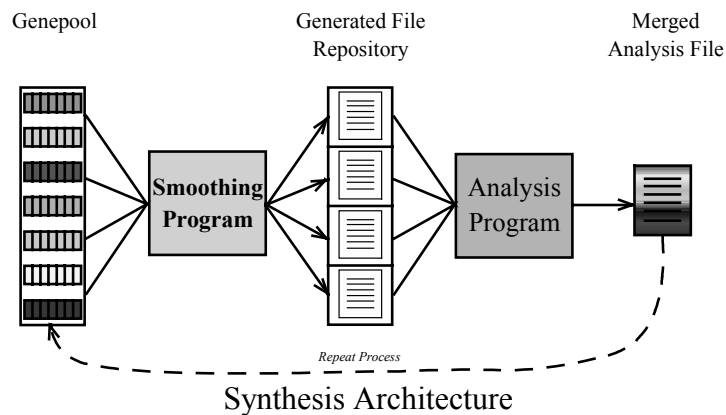


The paired harmonic and metric patterns are then laid over each other before finally being converted back to GUIDO. Each of the macro-chromosomes in the gene pool is run the analysis program, and the generated analysis files are compared to the Merged Analysis File. The CBGA is halted either after a desired number of generations or if a match is made with the blueprint within a given threshold.



The result should be a new piece of music that possesses the stylistic and compositional aspects of the input selections.

# Synthesis



## Discussion

Work is currently being done on the creation of a generic Analytic Synthesis toolkit in java which will be available on the project website. This toolkit is being designed in such a way that will allow for the greatest degree of inter-phase integration by demanding that all code conform to certain interfaces or extend certain classes that are designed to work regardless of phase-specific requirements.

As this is on-going research, statements on the implications of the system's performance are based on issues encountered thus far in development as well as areas of difficulty found in other similar projects. It seems that the strength of a system such as this will primarily in the quality of the analysis program. The better the analysis program is, the closer the output will be to "desirability." This begs the question of what exactly constitutes a good analysis program. There have been many music analysis systems created, notably Huron's Humdrum (Huron, 1999) and Cope's SARA (Cope, 1991). There have many approaches to creating such programs ranging from systems utilizing preference rules (Temperley & Sleator, 1999) to even systems using genetic algorithms to analyze scores (Taube, 1999). It is therefore only through experimentation that a suitable system can be created. It is possible, however, to conjecture that an optimal system must have a good balance of speed to analysis depth. A superficial analysis program will yield only the vaguest of notions of convergence, whereas a weighty analysis may yield an excruciatingly slow GA.

The other major factor in the creation of an effective system is the mechanism involved in the creation of the Merged Analysis File. A standard averaging of values may simply yield something that does not really sound like any of the input pieces at all. As the value of the output of the system may be as subjective as the input given to it, perhaps the best route would be to allow the user to rank different qualities themselves and a weighted average can be calculated. One might go so far as to say that the merging of the analysis files is a process well suited Analytic Synthesis itself. Regardless, it is absolutely essential to have the Merged Analysis file as richly defined as possible, for that is the overarching template and the goal toward which the entire system will strive. As this on-going work continues, the implementation of a system such as described in this paper will be completed, and these issues will be investigated.

In closing, it should be noted the an Analytic Synthesis system has possibilities as the basis for applications in addition to that mentioned in this paper. Such a system could be used to

create music based on a composed template rather than an explicit score that would always the same sort of song whenever it were played, though never identical to a previous listening. It could also be the basis of a search engine where one feeds in their favorite music and suggestions for similar artists to listen to would be produced that would hopefully be more insightful than a simple category comparison. Analytic Synthesis might even be extended beyond the music realm for applications in finding new chemical or biological substances similar to known samples.

### **Acknowledgements**

I would like to thank Professor Bradford Garton, Professor Andrew Kosoresow, and Dr. David S. Prerau for their continuing guidance throughout the course of this project.

### **References:**

1. Brown, Andrew R. " 4.1: An Introduction to Music Analysis with Computer " XArt Online Journal. 2000. <http://www.explodingart.com/XartJournal/XAOJ41.htm>
2. Cope, D. "Computer Modeling of Musical Intelligence in Experiments in Musical Intelligence." *Computer Music Journal* 16,2 (Summer, 1992): 69-83.
3. Cope, D. "Pattern Matching as an Engine for the Computer Simulation of Musical Style." *Proceedings of the 1990 International Computer Music Conference*. San Francisco: Computer Music Association, 1990.
4. Cope, D. "Computers and Music Style" Madison, WI: A-R Editions, 1991.
5. Henz, Martin, Lauer, Stefan, and Zimmerman, Detlev, "COMPOzE - Intention-based Music Composition through Constraint Programming." *Proceedings of the 8th IEEE International Conference on Tools with Artificial Intelligence*, November 16-19 1996, Toulouse, France.
6. Huron, D. "Music Research Using Humdrum: A User's Guide" Stanford, California: Center for Computer Assisted Research in the Humanities, (1999) 414 pages.
7. Hoos, H.H., K. A. Hamel, K. Renz, J. Kilian. "The GUIDO Music Notation Format - A Novel Approach for Adequately Representing Score-level Music" ICMC'98 Proceedings, p.451-454
8. Jacob, Bruce L., "Composing with genetic algorithms." *Proceedings of the 1995 International Computer Music Conference*, pp. 452-455. Banff Alberta, September 1995.
9. Oppenheim, D "Demonstrating Mmorph: A System for Morphing Music in Real-Time." *Proceedings of the 1995 International Computer Music Conference*, Banff, Canada, 1995.
10. Taube, H. "Automatic Tonal Analysis: Toward the Implementation of a Music Theory Workbench" *Computer Music Journal* 23, 4 (Winter, 1999): 18-32.
11. Temperley, D., Sleator, D. "Modeling Meter and Harmony: A Preference-Rule Approach" *Computer Music Journal* 23,1 (Spring, 1999): 10-27.