Proposal of a geometric algorithm for rapid data decomposition to write in non-volatile memory

*Christopher A Tucker*

Keywords: universal constructor, systolic array, flowware

Abstract:

This paper will introduce a model of an algorithm, its methodology of construction, and its effectiveness to increase the efficiency of high-frequency data flow problems such as the ADHD system.

**The Proposal**

Proposition I shows how to calculate the area under the parabola using an iterative carpet of triangles to approximate the area under the curve. The procedure is given by the following equation



for an arbitrary physical system of x complexity. The equation demonstrates that sufficiently smaller triangles can be created to increase the number of steps *h* and to decrease the spacing between the steps. The framework forms a closed ordered system that can be divided into as small components as necessary for the given problem.

An iterative process has tenets of self-organization outlined by W. Ross Ashby in that it is stable on each of its vertices; i.e., the system knows how the manifest form is generated by an examination of the pathways between the outer vertices and the center of mass thereby manipulating the generalized behavior of the object and making corrections in balance the strategy of how the data is addressed and stored in the spaces, in the spirit of geometric complexity theory within devised by Shamos. What analysis can be performed and what methods can be established by performing a geometric probing of the tenets of the Proposition I algorithm using a RAM machine?

Using the limits of this approach how can this rule-based approach illustrated in this method can be directly applied to modern problems? And most importantly, what type of algorithm can be constructed using this motif to create self-organizing automata to manage the task of data flow?

**Transitional Methodology and the Geometric Algorithm**

The inherent complexity in constructing a computational model is due to its need for modeling data based on its sensor inputs. By using auto-sequencing memory (ASM) address assignments and data counters—the number of times an agent traverses a vertex edge, an iterative loop first constructs the framework and by self-replication and addresses the transit data embedding it within the sub-frame. This is the model of a geometric data processor that raises the level of abstraction in algorithm description.

The model uses a geometric address generator—demarcated by a coordinate value of three vertices—a variant on the generic address generator (GAG) that is a generalization of direct memory access (DMA). In total, the form is a generalized systolic array with a space-time diagram on a generalized relativistic space-time manifold. Such a purpose is to start from a fundamental point that is not derived from a more fundamental sink. It is by this mark on a most fundamental point that the form is allowed its physicality by reference to its starting location on a Cartesian grid.

The hardwired data stream present in this machine needs two program sources to control it: 1) flowware, the sequence by which the data is input into the array—known by the constructor agents—and 2) configware, how the system knows where the data is located to be utilized by a high level programming language source to execute data streams at run time—communication of the three vertex coordinates of the agent with the application layer of the program. This is a model of a reconfigurable data-stream machine.

The agents create a co-compiler that has a partitioner which accepts input from a high level language source, such as, for instance a programming language, i.e., C with SQL, and automatically partitions the tasks into geometrically parallelizable parts suitable as a reconfigurable accelerator in how the locations are address by the vertices and sets the location for the memory rest for running on the microprocessor. Outwardly it requires a set of fundamental laws that define how it interacts with its environment, is able to remember, is able to learn, is able to adapt, and must survive. Representing this computationally, an artificial life system needs the following abilities:

1. A constructor that builds the shape on first iteration, traversing the shape only once,
2. Agents that possess the knowledge of their work for later search operations,
3. A limit to associated knowledge within the system so that the data is usable, i.e., doesn’t take long periods of time to assess data and make a decision based on it,
4. A method to pool data by the simplest addressing from sensor networks,
5. Present the refined data to a co-compiler.

At the onset, it is advantageous to have the algorithm to represent the problem at hand, a possible solution to the artificial life quandary by creating a system in the motif of an ancient cultural technological process, considering that culture already possessed technological expertise to construct automata. In this instance, the automata will be manifest in software. Some questions to consider are: how do we generate a self-replicating system that knows its framework and can carry the knowledge with it? Figure 2 represents a method by which agents perform a collective task of construction of a geometric object. Simultaneously, they construct an addressable “data-space” system that carries the knowledge to self-modify to accommodate incoming data from sensor arrays mapping its environment. It is covered in the next section.

**The Walker-Constructor Algorithm**

Consider Figure 3, a closed geospatial system consisting of a set of *n* vertices with a set of *m* edges built by a set of three identical agents who have the abilities:

1. walk,
2. mark,
3. and remember.

Here is the pseudocode for the walker agents, less the messy programmatic operations between the application layer and the database layer:

*Walker-Constructor Pseudocode*

--Task--

We observe the result of the agents as they perform their task of constructing the object.

Pick an arbitrary point on the Einsteinian space-time manifold, call it (0,0), and run a Cartesian grid in four directions with each ray trajectory 90 degrees from its left and right nearest neighbor in 2 dimensions. Label one axis 'x' and name the other axis firstly encountered in a counterclockwise rotation 'y'.

Create three agents who are programmed identically and whose task is to walk, mark, and remember. Between them, they hold two elastic strings. Start each at (0,0): one agent (1) walks five units in the positive y direction; two agents walk five units in the negative y direction and walk 90 degrees from their path in opposite directions: (2) in the negative x direction 5 units and (3) in the x direction 5 units.

Remember your location on the grid.

Draw tight the first elastic string and pin to the edge points (1,2,3). This is the first polygon of three vertices.

Scribe circles with a diameter of 2 units filled with quantity 'theta' centered on vertices 2 and 3.

1: cast a ray that intersects point 'a' continuing to twice the distance walked and mark the edge point '4'. Save the distance (d4).

2: Walk in the +y direction along the string to the grid coordinate (2.5,0) and mark the point 'c'.

3: Walk in the +y direction along the string to the grid coordinate (-2.5,0) and mark the point 'b'.

2: cast a ray that intersects the point 'c' continuing the ray to twice the distance traveled and mark the edge point '5'. Save the distance (d5).

3: cast a ray that intersects the point 'b' continuing the ray to twice the distance traveled and mark the edge point '4'. Save the distance (d6).

Draw tight the second elastic string and pin to the edge points (4,5,6). This is the second polygon of three vertices.

Mark the intersection of the three rays as the center of mass of the triangle centroid as point 'm'.

This is the Uhoo Carpet.

The walker construction has three agents that perform all the work from the time of construction to the accessing of the framework by application operations. The collective behavior of the agents generates, in this example, a Hamiltonian cyclic graph. The agents pass through each vertex only once and step until the shape is constructed without repetition.

The action generates a finite space of data addresses generating twelve on the first iteration. This paper will only discuss the first iteration of the carpet but the algorithm can further divide the spaces into smaller ones by manipulating the quantity of *θ* by its compass tool in tandem with its knowledge of the length of each edge.

The proposal carries the following features in its application layer:

1. Data-space addressing: stored as a location given by three coordinates (*i, j, k*). Since the system knows how the triangles were constructed, it can retrace edges to find each of the vertices,
2. Data-space resolution (how small data-block grids are), and increasing address storage (number of spaces and sequential numbering). Further iterations (the addition of vertex edges between vertices) can be constructed in Θ(*di*) time where *di* is the degree of the *i*th vertex,
3. Expense of access operations by assigning access point *p* as an input-output controller,
4. It describes the state values of any operation in terms of its common geospatial model,
5. Is a closed and well-defined system with a hub represented by *m* that it interacts with all points as the center of mass of the object and carries a unitary matrix of value 1.

*Algorithm Complexity*

Table 1 is a computation of the efficiency of the algorithm when it needs to be accessed at any point in the system: read, written, or searched. Complexity can increase very quickly if the system were to subdivide its spaces, so it is important to select the right balance in the data structure around a center of mass, since the weight of operations on the vertices and edges as agents are traversing the strings greatly impacts the performance of the algorithm. Using an adjacency matrix, *n x n,* *M* where *M* =[*i, j*], to represent the vertices while needing to know how many edges can be represented; if we keep edges defined by the limit points of the set, it should take Θ(1) time to test when an edge (*i, j*) is represented by the adjacency matrix, the system needs to read the appropriate bit. The simplicity of this solution is what makes this geometric algorithm appealing for artificial life systems. We can add as many vertices as the number of sub-triangles inside the numbered edges since at least one edge is known each time the area is subdivided to create more addressable space.

Currently, the algorithm is O(*n2*) in an undiscovered state; however, the construction procedure insures each vertex is completely explored, that is all incident edges have been visited. Searches, either breadth-first or depth-first, performed on the algorithm are done in O(*n + m*) time. As data-spaces become filled, the form can expand into *n* = 3 dimensions to accommodate before subdivision. However, the higher degree of order adds complexity as the as the number of edges increase. This algorithm is NP-complete.[[1]](#footnote-1)

Proof?

*Conclusion*

This method will result in robust geometric software for high-volume data applications of data storage and pre-parsing such that when control lines RW follow WR for non-volitile RAM, the classification of a “governing algorithm” governing algorithm for how future iterations of the processing system are constructed; it contains its own template (the constructor), the knowledge of how to construct it (data-flow), and by design, can address and organize many types of ADHD data into a meaningful format.

*References*

Ashby, W. Ross. Introduction to Cybernetics, ultra-stability in physical system. Here is the argument for its valid physicality of the model. Because an agent constructor on a grid that the system knows the entirety of the layout and is connected by the pathways created by the agent constructors and their ability to remember. I.E., data-tape to constructor.

1. Skiena, Steven. *The Algorithm Design Manual*. New York: Springer-Verlag, 1998. pp. 84, 229, 347. [↑](#footnote-ref-1)