https://ntrs.nasa.gov/search.jsp?R=20110023844 2019-08-30T18:18:29+00:00Z

Tool for Rapid Analysis of Monte Carlo Simulations

Carolina Restrepo* and Kurt E. McCall†
NASA Johnson Space Center, Houston, TX.
and John E. Hurtado‡
Texas A&M University, College Station, TX.

Designing a spacecraft, or any other complex engineering system, requires extensive simulation and analysis work. Oftentimes, the large amounts of simulation data generated are very difficult and time consuming to analyze, with the added risk of overlooking potentially critical problems in the design. The authors have developed a generic data analysis tool that can quickly sort through large data sets and point an analyst to the areas in the data set that cause specific types of failures. The Tool for Rapid Analysis of Monte Carlo simulations (TRAM) has been used in recent design and analysis work for the Orion vehicle, greatly decreasing the time it takes to evaluate performance requirements. A previous version of this tool was developed to automatically identify driving design variables in Monte Carlo data sets. This paper describes a new, parallel version, of TRAM implemented on a graphical processing unit, and presents analysis results for NASA's Orion Monte Carlo data to demonstrate its capabilities.

I. Introduction

A Monte Carlo approach that varies a wide range of physical parameters is typically used to generate thousands of simulated scenarios of flight design. The results provide a realistic representation of a vehicle's performance and eventually help with flight certification. NASA's Orion vehicle is a current example of the importance and benefits of the Monte Carlo design approach because several of its performance requirements are written in terms of Monte Carlo statistics.[1]

Identifying variables that can cause system failures is crucial. It is very difficult and time consuming to identify single variables, and even more so, combinations of variables that can cause problems in a design. Engineers spend much time and effort sorting through large data sets to find the causes of current problems, and often must solely rely on their system expertise. The main disadvantage to this approach, however, is not the time that it takes, but the fact that a manual search for problems does not guarantee that an analyst can find all of them.

To the authors' knowledge, there were no other general methods that could be used to identify individual variables, or their critical interactions in a reliable and timely manner for a flight dynamics problem before the development of this analysis tool in 2010.[2,3] The current version of this tool is now named TRAM, which stands for Tool for Rapid Analysis of Monte Carlo simulations. There are two main features that make this tool applicable to almost any Monte Carlo data set. The first is that the analyst using this tool is not required to write any problem-specific analysis scripts to run it. The second is that the analyst does not need to provide multiple Monte Carlo data sets. A single statistically significant set is enough for TRAM to run an analysis. Of course, if the single data set provided does not contain enough simulation runs or enough dispersions, the results will be only as informative as the data set provided.

The previous paper[3] written on this topic had the objective of explaining the algorithms and prove that TRAM worked well with a simple example that had an analytical solution. The goal of this paper is to demonstrate TRAM's new analysis capabilities for a much larger and complex data set from recent design and analysis work on the Orion vehicle. Additionally, this paper shows computation times for the parallel version

*Aerospace Engineer, Integrated GN&C Analysis Branch, AIAA member, *carolina.i.restrepo@nasa.gov*
†Aerospace Engineer, Integrated GN&C Analysis Branch, *kurt.e.mccall@nasa.gov*
‡Associate Professor, Aerospace Engineering Department, AIAA member, *jehurtado@tamu.edu*

of TRAM, which has been programmed on a graphical processing unit (GPU). The paper is organized as follows. Section II is a brief overview of TRAM. Section III explains the GPU code implementation. Section IV provides the background on the Orion simulation cases used as examples, and presents TRAM's analysis results for a specific system failure. Finally, Section V summarizes the benefits and capabilities of TRAM.

II. TRAM: Tool for Rapid Analysis of Monte Carlo Simulations

TRAM is a tool that can automatically rank individual design variables and combinations of variables according to how useful they are in differentiating the simulation runs that meet requirements from those that do not. To produce these rankings, the separability of good data points from bad data points is used to find regions in the input and output parameter spaces that highlight the differences between the successful and failed simulation runs in a given Monte Carlo set. The authors strived to develop this tool from the perspective of an aerospace engineer that has a solid flight dynamics background but who is not necessarily an expert in the fields of statistics of pattern recognition. The constraints listed below were published previously,[3] but are listed here once again because they were paramount in the selection of the algorithms for TRAM.

1. Algorithm Constraints
   (a) Algorithms are for post-processing data only and cannot rely on iteratively running several Monte Carlo sets.
   (b) Algorithms must make no assumptions about input probability density functions.
   (c) Algorithms must compare all types of parameters at once regardless of their units or relative magnitudes.
   (d) Algorithms must filter out variable correlations that obviously do not affect a particular failure.
2. Usability Constraints
   (a) The tool must be generic enough to address problems for any flight vehicle design.
   (b) The tool must be specific enough to capture subtleties buried in large data sets while ignoring obvious variable correlations that are not informative and do not affect system failures.
   (c) The tool must not require an analyst to modify existing code or write new pieces of problem-specific code.
   (d) The tool must be flexible enough to allow a system expert to introduce additional variables and performance metrics to the analysis.
   (e) The tool results must be tractable enough that an aerospace engineer can trust and understand without being an expert in the fields of statistics or pattern recognition.
   (f) The tool must be consistent enough to yield the same results each time it is used on a given data set (this implies that random algorithms are not appropriate).

Based on these constraints, the authors selected two well-known pattern recognition methods, kernel density estimation (KDE) and k-nearest neighbors (KNN), and combined them into a stand-alone analysis tool that can identify both single variables and combinations of variables that affect a system failure specified by an analyst. A detailed description of how TRAM uses these two methods is published in references.[2,3] Here, consider a simplified flowchart of TRAM's layout as shown in Figure 1. The two main codes are located on a GPU, and the graphical user interface (GUI) is a MATLAB program that allows the user to interface with the GPU in a seamless way, and subsequently plot and save the data in a simple format.

III. GPU Implementation

In previous versions, both the kernel density estimation and the k-nearest neighbors algorithms were MATLAB programs. The authors were able to show that the combined algorithms are useful in identifying influential variables,[3] but the MATLAB version of TRAM was relatively slow in processing large data sets. The current version has been programmed on a GPU and processing times have decreased greatly.

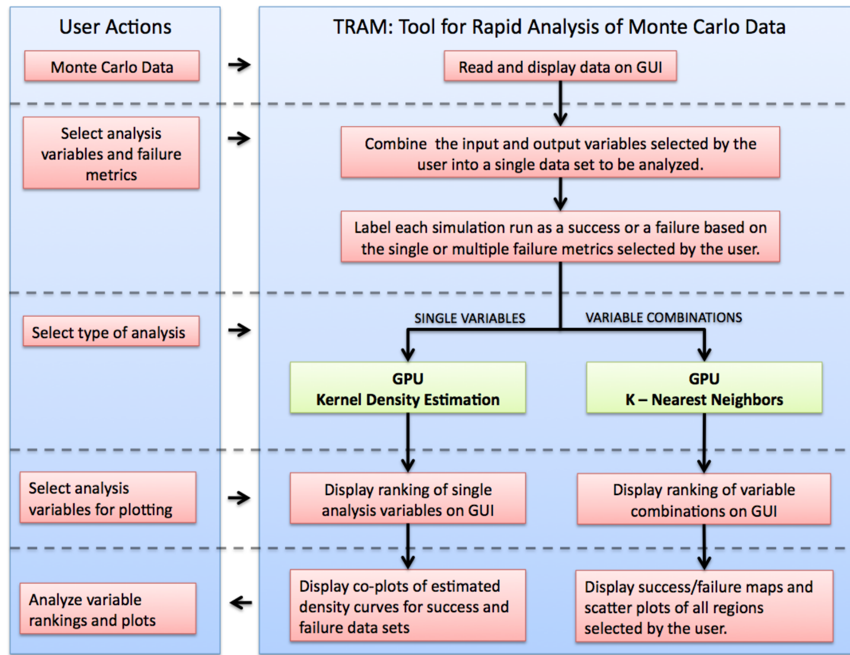

Figure 1: TRAM

CUDA, which stands for Compute Unified Device Architecture, is Nvidia's parallel computing platform and programming model for general-purpose computing on GPUs. A CUDA kernel is the program that runs on the GPU, while a thread is an instance of a kernel. Typically the GPU runs thousands of threads during each kernel invocation. Threads are grouped into 1-, 2- or 3-dimensional blocks, and threads within the same block can interact with each other using shared memory. In most cases, different blocks are executed independently of each other, with no inter-block communication. The blocks are organized into a 1-, 2- or 3-dimensional grid.

These algorithms were implemented under CUDA 4.0 in the Linux environment. They were tested on Nvidia GeForce 480 and 580 cards, which are of the Fermi class, and have not been tested on pre-Fermi cards. A network GPU server was created which can accommodate multiple users, providing efficient access to a limited number of GPUs. This server acted as the host for the implementations described below.

A. Kernel Density Estimation Algorithm Implementation

The GPU implementation of the KDE algorithms are divided into two parts: KDE for original variables and KDE for compound variables. An original variable comes directly from the given data set, and a compound variable is formed from the difference or the quotient of two original variables.[3] In each type of analysis the highest-ranked variables are those for which there is substantial separation between the passing and failing data points. These two types of analyses are performed separately as determined by the user, but the ranked results can be combined.

Let N be the number of Monte Carlo runs, and V be the number of original variables stored for each and every run. The CUDA kernels are designed for problems in which N is large and V is substantially smaller. For example, a typical problem might have 3000 runs but only 400 variables. For a given KDE analysis, run time is linear in the number of variables but logarithmic in the number of runs. In the KDE compound variable analysis every pair of original variables is analyzed, so the run time is in the order of $O(log_2 N)$ considering only the number of runs, but $O(V^2)$ considering only the number of variables.

Both KDE implementations use 1-dimensional CUDA blocks. Each thread is responsible for one Monte Carlo run or sample, which is read from global memory. An estimated probability density function, or the portion of it that is computed by each block, is stored in shared memory. The kernel program iterates over the points of the density, and each thread in parallel adds the contribution of its samples probability kernel

to each density point. At each iteration, distinct threads visit distinct density points to avoid shared memory conflicts. After this kernel finishes, a second kernel runs a reduction to sum up the portions of the densities computed by each block, creating the final densities. Other kernels rank the densities and sort the results.

B. K-Nearest Neighbors Algorithm Implementation

This type of analysis is concerned with two-dimensional graphs of failure regions within the data set. Each axis of the plane corresponds to either an original or a compound variable. Graphs for which there is separation between the passing and failing data points are ranked highest. Each point on the graph is classified as either an overall pass or an overall failure depending on the class of the majority of data points that fall within its local neighborhood.

This algorithm is an exhaustive search of all graphs formed by all combinations of original and compound variables. If V_{knn} is the number of original variables selected for this analysis, with $V_{knn} \leq V$, the number of compound variables that need to be generated is $O(V_{knn}^2)$. The graphs are exhaustively formed from all pair-wise combinations of the original and compound variables, and so the total number of graphs that must be evaluated is $O(V_{knn}^4)$. This is a very large number, so a user must specify a reasonable value for V_{knn}. For example, $V_{knn} = 40$ results in a runtime of approximately one hour on the machines described above when $N = 3000$.

The design of the nearest-neighbor CUDA kernel follows reference.[4] The graphs containing the failure regions are evaluated one at a time. The 2-dimensional CUDA blocks are 16 x 16, and the layout of the grid, which is also 2-dimensional, is shown in Figure 2.

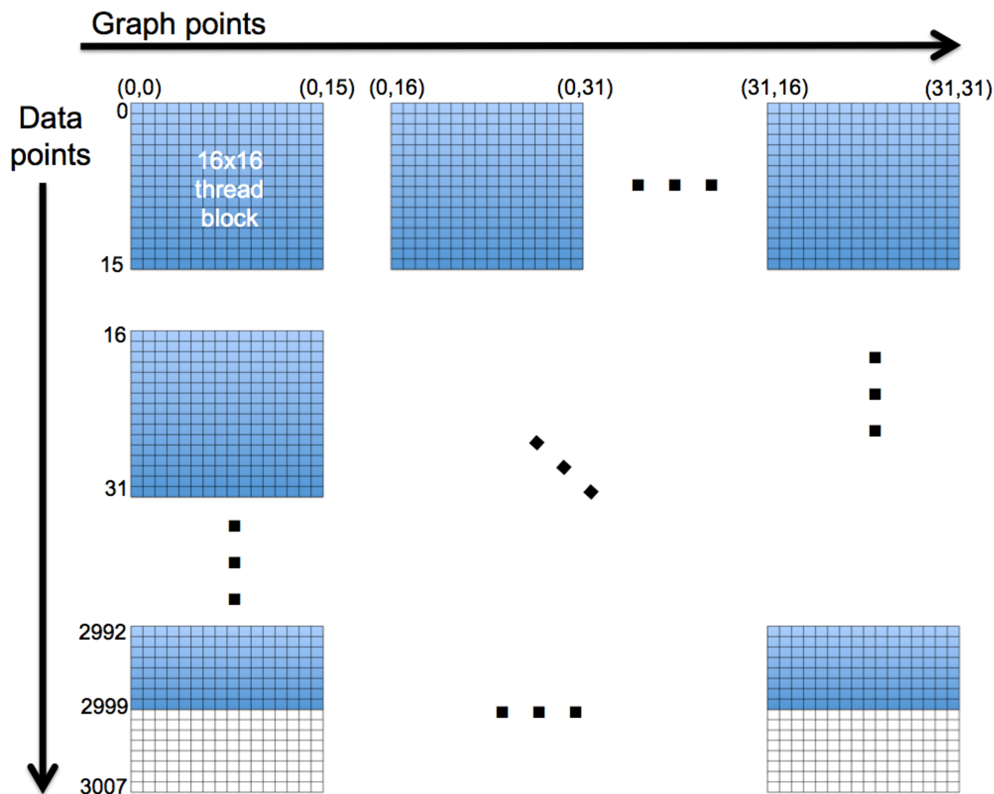

Figure 2: KNN Grid Layout for a Monte Carlo Set with 3000 Runs and 1024 Graph Points

Each point on the Y-axis of the grid corresponds to a single simulation run (data point), and each point on the X-axis of the grid corresponding to a point on the graph that displays the failure regions. Each thread is responsible for computing the distance between a single data point and a single point on the graph, so that distances for all possible pairings of data points and graph points are computed. The first kernel computes all distances, and then counts the number of successful and failed data points in the neighborhood of each

graph point on a per-block basis. A second kernel takes these partial counts and sums them up to form the unique success and failure counts for the neighborhood of each graph point on a failure region. Several other kernels compute the ranking of the graphs and sort the results. This implementation deviated from the original algorithm in[3] in that all data points that fall within the local neighborhood of each graph point are counted. This eliminated the need for the neighbors to be sorted according to their distance from a particular graph point. In the previous version of TRAM,[3] only a fixed number of the very nearest samples were used in determining the class of the grid point.

As an example of the speedup of TRAM's new parallel code compared to its original MATLAB version, a small data set with 500 simulation runs and only 6 dispersed design variables that yielded 630 failure regions was analyzed. In MATLAB, the analysis took approximately two hours whereas on the GPU, it took less than two seconds. Several different size data sets will be analyzed in the near future in MATLAB, a C++ serial code, and in CUDA to benchmark the speedup.

IV. Example

This current example demonstrates the use of the tool in analyzing a fully integrated spacecraft. All dispersed variables are treated the same way, meaning there is no need to categorize them into mass properties, aerodynamics, environment, etc. Monte Carlo simulation data from NASA's Orion vehicle is used here to show TRAM's ability to find the significant design parameters out of the hundreds that should be analyzed in more detail.

The Orion vehicle is required to provide full abort coverage throughout the ascent phase of flight.[5,6] This abort requirement has been a major design driver for the launch abort system, the rocket, and the vehicle itself. As shown in Figure 3 , the launch abort system (LAS) has three sets of motors up stream of the

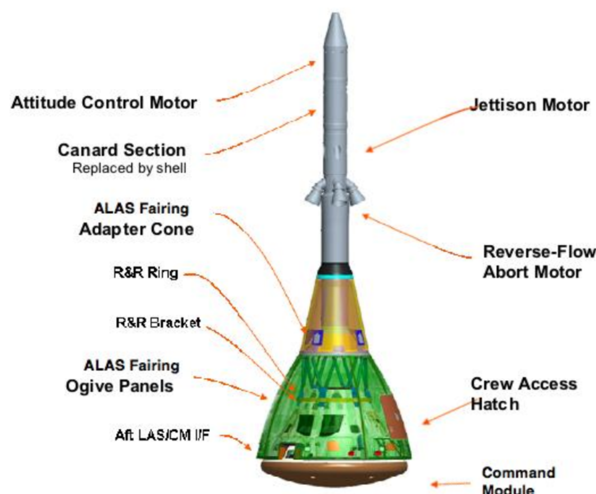

Figure 3: Orion Launch Abort System[5]

vehicle, labeled Command Module in this figure. At the current stage in the design, it is well known that the motor plumes are major drivers for GN&C algorithm design. One of the challenges has been to keep the vehicle under control during the reorientation maneuver that is performed along an abort trajectory (Figure 4). NASA flight dynamics engineers have already characterized the impact of certain influential aerodynamic variables on the performance of the vehicle along the abort trajectories through detailed analysis of several Monte Carlo simulation sets. This example shows that TRAM was able to identify and rank the same aerodynamic variables that affect performance during reorientation by analyzing a single Monte Carlo set in just a few seconds.

TRAM uses the kernel density estimation method to find and rank any individual variables that affect the reorientation maneuver. Figure 5 is a bar graph that displays the relative influence of each dispersed variable on the specific failure metric. Here, the cases that fail to perform a controlled reorientation maneuver are

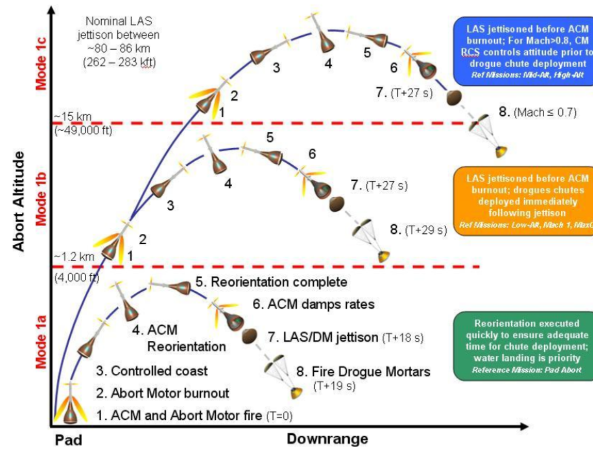

Figure 4: Orion Launch Abort Regimes[5]

labeled as failures. It is clear that, out of over 400 variables dispersed for an ascent abort Monte Carlo simulation, only a few (six) variables significantly affect the failure cases in comparison to the rest.

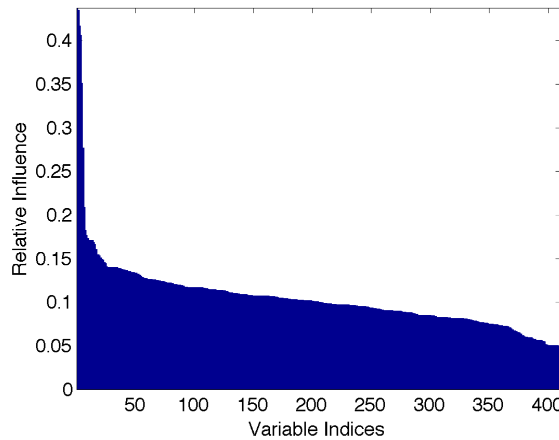

Figure 5: Orion Ascent Abort Performance Relative Effects of Dispersed Variables

The dispersed variables include aerodynamics, mass properties such as mass and inertias of the vehicle and motors, environment properties such as air density and wind magnitude and direction, and dispersed abort initiation conditions. The top variables that correspond to the highest bars in Figure 5 are precisely the aerodynamic variables that are already known to affect tumbling during reorientation. The next highest ranked variables are the LAS moments of inertia followed by the positions of the motor nozzles. Table 1 lists the top six influential variables in order. The analysis was done with a *single* Monte Carlo set. Due to ITAR restrictions on the data, the design variables used in this example are referred to by number only.

Before TRAM, the same analysis had to be performed using several different Monte Carlo sets with modified input decks. Several analysts generated all the data and developed different scripts to compare the failure rates between sets with the goal of identifying which parameter dispersions were affecting the number of failures. Table 2 shows the result of this manual analysis process. The first column shows the ranking based on how much the failure rate is reduced when compared to the failure rate for a case with all variables dispersed. The second column of the table shows the name of the variable that was held constant. The third and fourth columns contain the number of tumbling cases and percentage of tumbling cases, respectively. When comparing the ranking in Table 1 to the ranking in Table 2, it is important to keep in mind that

Table 1: Ascent Abort Individual Variables

Rank	Variable No.	Type
1	90	aero
2	89	aero
3	92	aero
4	98	aero
5	97	aero
6	100	aero
⋮	⋮	⋮

some dispersions may actually improve the results so a "true" ranking may not always be explicit. However, the algorithm can still identify the handful of critical variables out of the hundreds of variables dispersed through the analysis of a single Monte Carlo set and do so without manipulating the simulation. This is a very significant improvement over a manual analysis technique. TRAM saves the analyst the time it takes to plan and run additional Monte Carlo sets, and it saves significant time sorting through large data sets. While this manual analysis typically takes a few days, TRAM can generate the same ranking of influential variables very quickly and the analysts can immediately start focusing on the variables that matter most.

Table 2: Ascent Abort Monte Carlo Results

Rank	Fixed Variable	# failures	% failures
	0 variables dispersed	0	0
1	variable 97	237	11.85
2	variable 92	238	11.9
3	variable 89	264	13.2
4	variable 100	301	15
5	variable 98	344	17.2
6	variable 90	367	18.35
	409 variables dispersed	391	19.55

Analyzing combinations of variables on TRAM's MATLAB version was prohibitively expensive in terms of computation times. The GPU version makes it possible to analyze a very large number of failure regions. For example, the analysis of seven dispersed variables that yield over one thousand combinations of variables took approximately two hours in MATLAB. The GPU version can analyze and rank the same number of combinations in approximately two seconds. A more formal set of benchmark runs will be tested to characterize the speedup of the new code in the future. Once again, due to ITAR restrictions, the failure regions containing Orion data are not displayed here.

TRAM can output a very large number of regions, some times making it difficult to display all of them. Future improvements to TRAM include a "plug-in" for analysis tasks that are specifically related to flight dynamics. In the case of Orion for example, there are variables that are linearly correlated, so the analysis of regions that contain two correlated variables may not be necessary. Narrowing down the ranked list through the use of basic flight dynamics equations will help engineers feel more confident about the combinations of variables that they should analyze in more detail while paying less attention to the ones that have been taken out from the list. This will be optional and will not affect the generic character of TRAM.

V. Summary

In general, TRAM is now a practical tool for the analysis of large Monte Carlo data sets. The GPU version has significantly improved computation times making it possible to analyze variable combinations in a timely manner. TRAM is currently a generic tool, and therefore applicable to non-aerospace data sets.

References

[1] "Crew Exploration Vehicle system requirements document," NASA CEV Document: CxP-72000, January 2007.

[2] C. Restrepo, "An analysis tool for flight dynamics monte carlo simulations," Ph.D. dissertation, Texas A&M University, August 2011.

[3] C. Restrepo and J. E. Hurtado, "Pattern recognition for a flight dynamics monte carlo simulation," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, Portland, Oregon, August 2011, number 2011-6590.

[4] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k-nearest neighbors search using GPU," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, Anchorage, AK, June 2008.

[5] Ryan W. Proud, John R. Bendle, Mark B. Tedesco, and Jeremy J. Hart, "Orion guidance and control ascent abort algorithm design and performance results," in *American Astronomical Society*, August 2009.

[6] John Davidson, Jennifer Madsen, Ryan Proud, Deborah Merrit, David Raney, Dean Sparks, Paul Kenyon, Richard Burt, and Mike McFarland, "Crew Exploration Vehicle ascent abort overview," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, August 2007, number 2007-6590 in AIAA.