On Clustering Simulation Traces

Daniel M. Gordon, Peter Kemper
Department of Computer Science
College of William and Mary
Williamsburg VA 23187, USA
kemper@cs.wm.edu

Abstract

Discrete event simulation is widely used in performance and dependability analysis of systems, often in a way that many replica of a model are simulated. Those replica may reveal substantially different dynamic behavior of the simulation model. Recognizing if this is the case and what different classes of behavior are present can be of relevance to gain more insight in the system under study but also to recognize errors that may be present in a simulation model. Trace analysis is a classic technique to figure out what happens in a single simulation run. In this paper, we discuss work in progress on clustering a set of simulation traces. The outcome helps a modeler to select traces that suggest themselves for an individual and detailed analysis either as being representative for a whole class of traces or being an outstanding extraordinary case.

I. Introduction

Discrete event simulation of stochastic models is a common technique in the performance and dependability assessment of systems. A number of usage scenarios lead to the situation where a model is parameterized and is simulated for a number of parameter settings. We highlight three example settings. 1) Replicated simulation runs are performed for the same model but with different seeds of random number generators. Those seeds can be seen as parameter settings for that simulation model. 2) A parameter study requires us to conduct a series of simulation runs, where parameters of a model are varied in a specified manner. 3) In optimization simulation, we use automated search techniques to identify parameter settings that yield an optimal value of a given objective function that is defined on the outcome (performance or dependability measure) of a simulation of a parameterized model. The bottom line is that we run a number of simulation runs that may result in very similar or very diverse behavior. For a modeler, it is interesting to have some support to organize this rich set of data in a way that makes it easy to identify those parameter settings resp. simulation runs that should be considered in further detail. Candidates are statistical outliers, i.e. simulation runs that are of much difference to other simulation runs since those are likely to show erroneous behavior of a simulation model, as well as representatives of large groups of simulation runs that are very similar. Conceptually, this task falls into the area of clustering which is extremely rich in techniques and applications and has a substantial research history.

II. Clustering

According to Jain et al [1] "clustering is the unsupervised classification of patterns (observations, data items or feature vectors) into groups (clusters)". In our context, we need to group data from simulation runs. A very detailed representation of a simulation run is a trace. We consider a trace as a sequence $\sigma = s_0 e_1 s_1 \dots e_n s_n$ of states $s_0, \dots, s_n \in S$ and events $e_1, \dots, e_n \in E$ over some (finite or infinite) sets S, E for an arbitrary but fixed $n \in \mathbb{N}$, where events carry also timing information. In the following, we assume that the set of traces we want to cluster consists of traces over common sets S and E. A trace can be considered as the base for more aggregated statistical measures like frequency counts of events but also throughput, waiting times, queue lengths, and availability. For the context of simulation traces, we assume that each simulation run documents itself in a trace σ and that information is reduced to a feature vector $x = (x_1, \dots, x_d)$. We focus on the frequency of events, i.e., let $d = |E|$ and x_i be the number of times event $e_i \in E$ is present in σ normalized by n, the length of σ. Other pieces of information could be taken into account as well. We consider the frequency of events with the understanding that this formalizes "what happens in a trace". In this way, we obtain feature vectors of same dimension and $\sum |x_i| = 1$ that give a simple estimate of the probability to find a certain event e at some position j in σ. In addition, we consider the progress p_σ of a trace σ which is the length of σ^*, the smallest possible trace that is obtained from σ by removing cycles of σ. For a formal derivation, algorithm and applications see [2, 3]. We consider $p_\sigma(i)$ for all prefixes $\sigma_i = s_0 e_1 s_1 \ldots e_i s_i$ of σ and use characteristics like mean, variance and max of $\{p_\sigma(i), i = 0, \ldots, n\}$ to compare traces, i.e., as entries of a feature vector.

A. *Distance Measures*

With a feature vector for each data point (trace), we need to define a distance measure (difference or dissimilarity measure) that quantifies to what extent two traces are similar or not. Of course, the selection of a distance is crucial for the outcome of our analysis. Many classes of distance measures have been defined in the literature, some are discussed in [1], see also Rubner et al for an overview [4]. Minkowski-Form distances are frequently named as commonly used ones.

$$d_{L_r}(x, y) = (\sum_i |x_i - y_i|^r)^{1/r}$$

Other examples are histogram distance, Kullback-Leibler Divergence and Jeffrey Divergence, χ^2 Statistics and Kolmogorov Smirnov distance. The latter two are familiar in the simulation context and apply here to compare two empirical distributions, the feature vectors we computed from traces. We focus on the Euclidean distance, namely D_{l_r} with $r = 2$.

B. *Methods*

Numerous methods are known for clustering. Jain et al provide a taxonomy in [1], that distinguishes hierarchical from partitional approaches. A hierarchical approach yields a nested grouping of partitions and similarity levels at which groupings change, which is graphically shown in a tree-type structure called dendrogram. Important subclasses in the hierarchical category are the single-link and complete-link algorithms. The partitional approaches can be refined in squared-error clustering methods (including the popular k-means clustering algorithm), in graph theoretic, mixture resolving and mode seeking methods.

In our context, we decided to employ the classical ISODATA approach [5]. This algorithm shares ideas with the k-means algorithm, but does not enforce a clustering into exactly k clusters for a user given value of k. The ISODATA approach permits splitting and merging of the resulting clusters and belongs to the class of partitional algorithms. Partitional algorithms are iterative algorithms that generate a partition for a given set of data points. The most popular one is the k-means algorithm. In the k-means algorithm k is a positive integer, chosen a priori, and represents a desired number of clusters. K data points are selected at random and become the cluster centers or centroids of each partition. The algorithm then proceeds to assign each data point to the nearest cluster. Whenever a data point is added to a cluster, the centroid changes, which implies that data points may need to move to other clusters. The algorithm iterates towards a fix point or terminates when reaching an upper bound for the number of iterations. A disadvantage of this algorithm is that the clusters created depend heavily on the initial k values chosen. Since we are only interested in finding the natural clusters of the data, so we do not have an ideal number of final clusters k to begin with. The ISODATA algorithm is a variation of the k- means algorithm which allows the splitting and merging of clusters based on several heuristic values. These heuristic values typically include thresholds for standard deviation and maximum diameter of a cluster. We use the following variant of ISODATA.

a) ISODATA Algorithm: Parameters of the ISODATA algorith are the initial number of clusters k, a threshold γ for the maximum tolerable value of the standard deviation of distances between centroid r_i and data points s_i in a cluster C_i, a threshold δ for the smallest tolerable value of distance between centroids r_i and r_j of two clusters C_i and C_j, and finally an upper bound θ for the maximum number of iterations. Each cluster C_i has a centroid r_i of that is computed as the average of all data points in the cluster $r_i = \sum_{s_i \in C_i}(s_i)/|C_i|$.

The ISODATA algorithm is build upon 4 key steps:

Create a set of k clusters for given k centroids by assigning each data point to the cluster with the nearest centroid which also implies an update of the location of the centroid for that cluster.

Evaluate clusters by computing the mean and standard deviation of distances between data points and centroid within each single cluster, by computing the overall average distance of data points to the centroids of their cluster and by computing all distances between centroids.

Split if the cluster with the maximal value for the standard deviation of distances exceeds threshold γ and the mean distance exceeds the overall average distance then the cluster is split into two clusters by selecting two centroids.

Merge if the centroids of two clusters are less than δ apart, those two clusters are merged and the new centroid is determined.

The overall algorithm starts from an initial random selection of k data points as centroids and then iterates through *create*, *evaluate*, *split* and *merge* clusters till it either reaches a fix point or exceeds an upper limit of iterations.

We need to specify the way new centroids are selected in the *split* operation. We decided to use the old centroid and the data point in the cluster that has the maximal distance to the centroid as the other new centroid. The assumption is that we may have data sets that are rather homogenous with a few exceptional outliers that justify the splitting.

b) Adjusting parameter settings for ISODATA.: The influence of the initial k data points is much less pronounced with the ISODATA algorithm than for the k-means algorithm. We ran the ISODATA algorithm against several groups of example traces and found that it consistently performed well for suitable parameter settings of γ and δ but failed when poor values of the parameters were chosen. For a user, it is not necessarily trivial to make a sophisticated guess for a good parameter value. So we make use of a simple hierarchical algorithm applied to a randomly chosen subset of data points to compute a data-dependent "sophisticated" guess for values of γ and δ. This approach is based on [6, 7].

Hierarchical algorithms find clusters based on previously formed clusters. This type of algorithm can be either agglomerative (bottom-up) or divisive (top-down). An advantage of hierarchical algorithms is that they not necessarily need any *a priori* parameters to perform. The agglomerative algorithm we implemented is a hierarchical algorithm for single link clustering. The agglomerative algorithm commences by initiating each data point as its own cluster. It finds the two nearest clusters and merges them. Most agglomerative algorithms continue to iterate and create successively larger clusters until only a few high density clusters remain. Our goal for the algorithm is to gain a good idea for the heuristic parameters in the ISODATA algorithm, so our algorithm runs until every data point is in one cluster. The algorithm is fairly simple and our implementation is as follows:

We hand this algorithm only a sampled subset of data points from our original list. We then use the average distance and average standard deviation derived in the algorithm as starting values for the threshold parameters γ and δ in the ISODATA algorithm. With the combination of the agglomerative and the ISODATA algorithm, we have tried to take the advantages of each algorithm while discarding their respective limitations. We run the agglomerative algorithm on only a few traces thus limiting the computation effort for this step. Even though the agglomerative algorithm

```
AGGLOMERATIVE ALGORITHM
0  while the number of Clusters $n > 1$
1      Find the two closest clusters $C_i$and $C_j$
2      Merge the two clusters to form $C_t$
3      $r_t = (1/|C_t|) \sum_{s_i \in C_t} s_i$
4      get mean & std deviation of $d_{L_2}(r_t, s_i)$ for $s_i \in C_t$ ;
5  get averages of mean & std deviation over all steps
```

Fig. 1. Algorithm to obtain parameters γ and δ for ISODATA

is run with a few traces it experienced fairly accurate values for the average standard deviation and average distance of a cluster. We have found through initial testing that these average values work quite well as the parameters for the ISODATA algorithm. Thus we have eliminated the drawback of the ISODATA algorithm of struggling to determine favorable initial values for its parameters, while keeping a good time complexity and clustering formation of the algorithm.

III. TOOL ENVIRONMENT AND EXPERIMENTAL RESULTS

We integrated the clustering technique into Traviando [8], a software tool for the analysis of simulation traces. Traviando imports traces in an XML format generated by several modeling tools, which includes in Möbius [9] and the APNN toolbox [10]. The APNN toolbox is supports modeling with generalized stochastic Petri nets and provides functionality for corresponding qualitative and quantitative analysis. Möbius is a multi-paradigm multi-solution framework that is employed for performance and dependability modeling but finds applications also in biology and other areas where stochastic models are of relevance. Möbius is able to run simulation experiments in a distributed setting, for experiment series and optimization such that find this an interesting environment to use our clustering techniques in.

We discuss two sets of traces. One set of traces is obtained from a series of 25 simulation runs performed for different parameter settings of a Möbius model of a LEO satellite network. The model originates from Athanasopoulou et al [11]. We use this model to illustrate how clustering helps to identify a number of representatives for a set of traces. The other set of traces is obtained from an stochastic Petri net model of the Courier protocol model of Li and Woodside [12] where we generated 8 traces of different length and for some variants of the models where we injected errors by varying the incidence functions of certain transitions. We use this model to illustrate how clustering helps to separate ordinary from erroneous behavior in simulation runs.

We also vary the feature vector obtained from each trace for our experiments. We consider a feature vector $f = (f_1, \ldots, f_{|E|})$ with the frequencies of events $e \in E$, a feature vector $p_{mv} = (E[p_\sigma()], V[p_\sigma()])$ with mean and variance of progress values observed for $p_\sigma(i), 0 \leq i \leq n$, similarly p_q with 0.25, 0.5 and 0.75 percentiles of the empirical distribution of $p_\sigma(i)$, and finally p_{max} that carries just the maximum value observed for $p_\sigma(i)$.

Table I shows the results obtained by the automated combined clustering method, where the hierarchical clustering algorithm on a subset of sample traces is used to identify parameter settings for the ISODATA algorithm that is subsequently applied to obtain a clustering. The first column *Feat* indicates what aspects are considered in the feature vector of any data point, we consider frequencies *F* of events, mean and variance *E,V*, quartiles *Q1-3* and the maximum *Max* of progress values. Column *#C* denotes the resulting number of clusters. Columns three and four give the minimal and maximal distance between data point and centroid observed over all clusters, while columns five and six give the minimal and maximal distance observed between centroids. We see that for frequencies *F* we obtain 3 clusters for the Courier model which does not help to identify errors, since the type of errors does not manifest themselves in significant differences in the frequencies of events. For features based on progress p_σ the selection of characteristics *E,V*, *Q1-3* or *Max* all do well, which is based on the fact that p_σ gives for these type of errors a clear indication. p_sigma is either oscillating or a linear function. Fitting a linear regression yields a slope close to 0 for correct models or a slope close to 1 for faulty models. The result of two clusters, one for traces from correct models, one for traces from faulty models is appropriate. A value of 0 in the third columns shows that one element is used as the centroid. We decided to use the closest data point to the computed center as the centroid to always have a clear representative for each cluster. Note that values for the minimal and maximal distance between centroids are equal if only two clusters are present. For the set of traces from the LEO satellite dependability study we observed a diverse picture, the clustering results gives appropriate classes of traces of similar behavior whose further interpretation requires more details on the particular model.

Feat	#C	Inside C_i		Across C_i	
		Min	Max	Min	Max
Courier					
F	3	0	0.6	4.5	10.6
E,V	2	0	6713.2	8328100.5	8328100.5
Q1-3	2	0	44.8	9263.9	9263.9
Max	2	0	316.0	13508.3	13508.2
LEO					
F	8	0	0	0.006	0.063
E,V	7	0	4099.0	5359.0	77900.6
Q1-3	7	0	0	114.2	821.4
Max	7	0	1.6	49.0	605.0

TABLE I
CHARACTERISTICS OF CLUSTERS OBTAINED FOR EXAMPLES.

IV. CONCLUSION

We presented experiences in the use of clustering techniques to support trace-based debugging of simulation models. The approach has been implemented and applied within Traviando, a trace analyzer for the analysis of simulation models.

Acknowledgement We thankfully acknowledge that this research has been funded with a Chappell Undergraduate Research Fellowship by the Charles Center of the College of William and Mary.

REFERENCES

[1] A. K. Jain, M. N. Murty, and P. J. Flynn, “Data clustering: A review.” *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, 1999.

[2] P. Kemper, “A trace-based visual inspection technique to detect errors in simulation models,” in *Winter Simulation Conference*, 2007.

[3] P. Kemper and C. Tepper, “Automated analysis of simulation traces - separating progress from repetitive behavior,” in *Quantitative Evaluaiton of Systems (QEST 2006)*, 2007.

[4] Y. Rubner, C. Tomasi, and L. J. Guibas, “The earth mover’s distance as a metric for image retrieval.” *International Journal of Computer Vision*, vol. 40, no. 2, pp. 99–121, 2000.

[5] G. H. Ball and D. J. Hall, “Isodata, a novel method of data analysis and classification,” Stanford University, Stanford, CA, Tech. Rep., 1965.

[6] R. Duda, P. Hart, and D. Stork, *Pattern Classification*, 2nd ed. Wiley, 1998.

[7] A. Hardy, “On the number of clusters.” *Computational Statistics and Data Analysis*, vol. 23, pp. 83–96, 1996.

[8] P. Kemper and C. Tepper, “Traviando - debugging simulation traces with message sequence charts.” in *Proc. Quantitative Evaluaiton of Systems (QEST 2006).* IEEE Computer Society, 2006, pp. 135–136.

[9] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster, “The möbius framework and its implementation.” *IEEE Trans. Software Eng.*, vol. 28, no. 10, pp. 956–969, 2002.

[10] F. B. et al, “A toolbox for functional and quantitative analysis of DEDS.” in *Computer Performance Evaluation / TOOLS*, ser. Springer LNCS 1469, 1998, pp. 356–359.

[11] E. Athanasopoulou, P. Thakker, and W. H. Sanders, “Evaluating the dependability of a leo satellite network for scientific applications.” in *Proc. Quantitative Evaluation of Systems (QEST 2005).* IEEE Computer Society, 2005, pp. 95–104.

[12] C. M. Woodside and Y. Li, “Performance Petri net analysis of communications protocol software by delay-equivalent aggregation.” in *PNPM.* IEEE CS, 1991, pp. 64–73.