LLNL-TR-687141

# Developing a Learning Algorithm-Generated Empirical Relaxer

W. Mitchell, J. Kallman, A. Toreja, B. Gallagher, M. Jiang, D. Laney

March 30, 2016

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

# Developing a Learning Algorithm-Generated Empirical Relaxer

Wayne Mitchell[1], Josh Kallman[2], Allen Toreja[2], Brian Gallagher[2], Ming Jiang[2], and Dan Laney[2]

[1]Dept. of Applied Math, University of Colorado, Boulder CO
[2]Lawrence Livermore National Laboratory, Livermore CA

**Abstract**

One of the main difficulties when running Arbitrary Lagrangian-Eulerian (ALE) simulations is determining how much to relax the mesh during the Eulerian step. This determination is currently made by the user on a simulation-by-simulation basis. We present a Learning Algorithm-Generated Empirical Relaxer (LAGER) which uses a regressive random forest algorithm to automate this decision process. We also demonstrate that LAGER successfully relaxes a variety of test problems, maintains simulation accuracy, and has the potential to significantly decrease both the person-hours and computational hours needed to run a successful ALE simulation.

## 1 Motivation and Background

Lagrangian hydrodynamics simulations (in which the mesh moves with the fluid) can cause distortions in the mesh due to fluid motion. Such distortions arise especially in problems which include turbulence, interfaces between materials, complex geometries, etc. Severe distortions of zones can cause unwanted non-physical calculations of certain values such as negative volumes or densities. When distorted zones begin to induce such non-physical calculations, this may cause the simulation to crash, and we say that zone has "tangled."

Avoiding tangled zones while retaining the accuracy benefits of a Lagrangian simulation over pure Eulerian is the goal of the Arbitrary Lagrangian-Eulerian (ALE) approach. Thus we seek a balance between performing too much relaxation (which can diminish accuracy) and performing too little relaxation (which may result in tangled zones). Finding such a balance is one of the main difficulties facing those who wish to perform ALE simulations. The current system for determining when a zone needs to be relaxed involves the manual creation and tuning of some relaxation criteria by the user. This "by-hand" approach to relaxation often results in a trial-and-error period in which the user runs several simulations of a given problem while varying these criteria until a satisfactory result is obtained. This process of setting the criteria for by-hand relaxers is highly subjective (it is up to the user to decide what the appropriate level of relaxation is), brittle (as a small change in initial conditions can break the ALE strategy), and highly expensive in terms of both user and computational time.

Our goal is to automate the relaxation of the mesh through the use of a machine learning algorithm. This algorithm will be trained on various simulations which produce tangled zones, learning what conditions may lead to tangling. After training, the algorithm is then deployed to perform real time predictions during a simulation and used as the new relaxation criteria. Thus we call this new approach to relaxation a Learning Algorithm-Generated Empirical Relaxer (LAGER). Our primary goal for LAGER is to reduce overall time spent generating by-hand relaxers while still accomplishing the goals of ALE: retaining a high level of accuracy while avoiding tangled zones. We also hope that LAGER will achieve these goals on a wide class of problems without the need for training on each particular problem.

The learning algorithm we chose to use for LAGER is a regressive random forest algorithm. Section 2 gives some general information on random forests and why they are a good choice for this application. Section 3 details the training process we employed as well as the implementation of the algorithm as a relaxer. We found that LAGER was capable of successfully relaxing a variety of test problems, retaining physical accuracy, and preventing zone tangling. Also, though we have not yet optimized LAGER in terms of computational efficiency, we found it to be competitive in terms of computational time with the by-hand relaxation strategies in place. These results, as well as some discussion of the effect of training on different data sets are presented in section 4.

## 2 Random forest algorithms

The learning algorithm employed by LAGER is a regressive random forest algorithm. A random forest algorithm is an ensemble method based on creating a set of decision trees. Each decision tree is based on a random subset drawn with replacement from the training data. The creation of splits when growing each decision tree is then based on a random subset of features. Thus the algorithm generates a diverse set of decision trees which all vote in order to make predictions on new data[1].

Random forest algorithms are favored among many applications due to their demonstrated robustness and accuracy, and they are well suited to our task of predicting zone tangling. The ensemble of decision trees built using different features automatically gives more weight to the most discriminating features (i.e. the features most important for prediction). It is unclear a priori what features are most important when determining whether a zone may tangle in a given situation, so the fact that the random forest algorithm can determine the importance of features on its own is a very attractive attribute. Also random forests are robust to large and somewhat noisy training data. This is also important for us since our process for generating training data is potentially noisy.

Random forests may be used as classifiers or as regressive algorithms. A regressive algorithm gives us the ability to adjust the amount of relaxation performed on a given zone. This is in contrast to the current by-hand relaxers which simply flag zones for relaxation (i.e. relaxation is “all or nothing” for a given zone). Thus we hope the choice of a regressive algorithm will lead to more efficient relaxation by allowing us continuous control over the relaxation process.

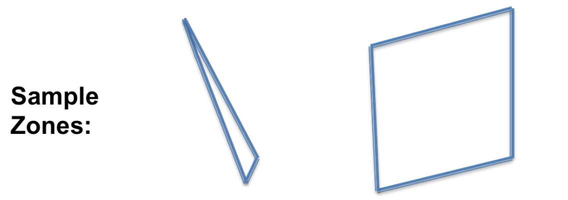
## 3 Implementation of LAGER

We now detail the implementation of LAGER in KULL, a radiation and hydrodynamics code under development at Lawrence Livermore National Laboratory[8]. Though we may refer occasionally to specific features of KULL, all of the procedures outlines below should be generalizable to any other ALE code already in use. We used an open source regressive random forrest algorithm distributed as part of the Scikit-Learn module for Python[7]. Section 3.1 will deal with the generation of data used to train the random forest algorithm, and section 3.2 will explain how predictions made by a trained algorithm are used to perform relaxation during the Eulerian step of ALE.

### 3.1 Data generation

In order to train the random forest algorithm, we first had to generate a training data set. Our goal was to obtain geometric feature data (the verdict metrics in KULL) on zones with various risks of tangling. Figure 1 displays some example zones with associated features. In order to generate a data set including zones with various levels of risk for tangling, we repeatedly ran simulations with no global criteria for relaxation until tangling occurred. The simulation was then restarted, now with zones which had previously tangled marked for constant relaxation (the idea being that now the simulation would proceed a bit farther until some other zone tangled). The geometric feature data was stored at each simulation cycle for each zone, giving us a large data set from which to select the information we wished to train on.

We chose two classic hydrodynamic problems to run repeatedly during data generation. Both involved disparate density gas systems subjected to shocks: the first was a helium bubble in air subjected to a planar shock, and the second was a perturbed air-SF6 interface also driven by a planar shock, generating a Richtmeyer-Meshkov instability. To generate the data, we used a coarse mesh for each problem: the bubble shock problem used a 2-dimensional, polar-coordinate mesh with approximately 2,800 zones concentrated in and around the helium bubble (for a problem size of 65cm$\times$9cm); the RM problem used a 3-dimensional slice (a single zone thick) with 3,220 zones concentrated near the instability interface (for a problem size of 218cm$\times$6cm).



| Geometric Features: | | |
|---|---|---|
| Largest Angle | 177.7° | 104.0° |
| Smallest Angle | 9.5° | 75.6° |
| Scaled Jacobian | 0.04 | 0.97 |
| Skew | 0.90 | 0.23 |
| Condition Number | 1.0 | 47.1 |
| Assigned $\varphi$ | 1.0 | 0.0 |

Figure 1: Some example zones included in the training set. We list a few of the geometric quantities included as feature data for each zone (note that this is not a complete list of the geometric features used). Also displayed is the associated $\phi$ value we assign as a measure of risk for tangling.

Since random forests require supervised learning, we then needed some way of assigning a "risk of tangling" value (which we will denote $\phi$) to each zone in the training set. All zones at the beginning of a simulation were added to the training set as zones with no risk of tangling, and we assigned the value $\phi = 0$ to each. Zones in a tangled state were assigned value $\phi = 1$ and added to the training set. To establish a smooth transition between the extreme states of $\phi = 1$ (tangled) and $\phi = 0$ (no risk of tangling), we then designed three different transition functions for $\phi$ in terms of the number of cycles before a zone tangled. Since we saved information on each zone at every cycle, we were able to extract feature data on a tangled zone any number of time steps before the zone actually tangled. We implemented three different transition functions in order to experiment with weighting the $\phi$ values differently according to how close a zone was to tangling. The transition functions we used can be seen in Figure 2. We also experimented with varying the length of the transition functions: the number of cycles before tangling for which the value of $\phi$ dropped to zero was set to 10, 20, and 50 cycles for each transition function.

In addition to data on tangled zones, we also gathered data on zones which caused a small simulation time step. During a simulation in KULL, the amount of time the simulation may advance on each cycle is limited by the CFL condition (source?) in the hydrodynamics solve, so very thin zones will cause very small simulation time steps. This can be a major problem if time steps become so small that it becomes infeasible to run simulations to a desired late time. Thus we added to the training set zones which caused a simulation time step lower than a set threshold. Again, the transition functions described

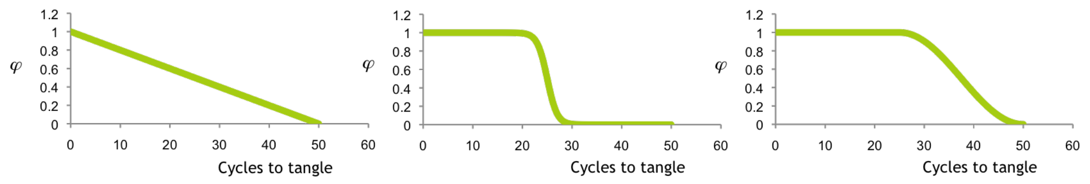


Figure 2: Different shapes of transition functions (functions of $\phi$ for a given zone vs. the number of cycles before that zone tangled): a) linear function b) logistic, c) piecewise constant/cosine (constant for half of the support, then a cosine decay to zero).

above were used to set values of $\phi$ for these zones in terms of the simulation time step (instead of in terms of cycles before zone tangling).

Varying the shape and length of the transition functions as well as selecting for which simulated problem we took data from allowed us to train several different random forest algorithms. In section 4, we discuss some of the observed differences produced by manipulating the training data in this way.

### 3.2 Relaxation via LAGER

We now describe how predictions from the trained algorithm are used to relax the mesh during a simulation. As we described above, LAGER is trained on some geometric features to determine a zone's risk of tangling. Relaxation occurs on a node by node basis, however. In order to make the transition from the zone-based predictions from the random forest algorithm to the node-based decision of how much to relax, we simply take an average over a given node's neighboring zones. We assign a value $\Phi = \frac{1}{n}\sum \phi_i$ to each node, where $n$ is the number of adjacent zones, and the $\phi_i$'s are the predictions from the algorithm for each of those adjacent zones.

We then take advantage of a functionality built into KULL which simply performs the Eulerian relaxation step for every node at every time step. Thus KULL calculates a relaxation vector $\vec{v}$ for each node. We then scale this vector for each node by the associated $\Phi$ value. Thus the relaxation for that node is scaled according to the predicted risk of its adjacent zones tangling. In this way, LAGER can dynamically and smoothly manipulate the amount of relaxation taking place at any specific point in the grid at each simulation cycle. A visualization of this process is displayed in Figure 3.

## 4 Results and discussion

We will discuss in the following subsections some results obtained running simulations with LAGER as the sole relaxer. We discuss LAGER's success relaxing a variety of test problems, the differences seen in the performance of LAGER when we train on different

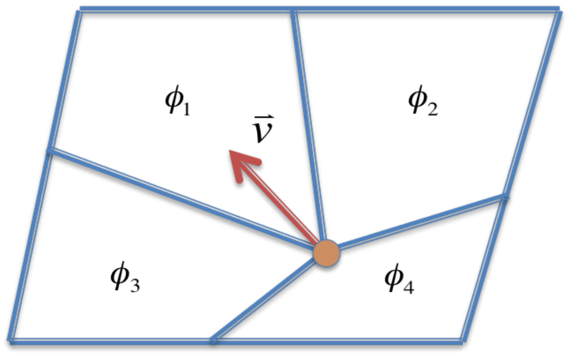


Figure 3: Relaxing via LAGER: for each node, we average predictions $\phi_i$ made for adjacent zones; we then use this average to scale the relaxation vector $\vec{v}$ calculated by KULL.

data sets, as well as it's computational cost.

### 4.1 Relaxing test problems

As mentioned in section 3.1, when training LAGER, we generated training data on simulations of a helium bubble shock tube problem and a Richtmyer-Meshkov instability problem. These problems also served as test problems for the trained algorithm. Though trained on coarsely meshed versions of each of these problems, LAGER successfully relaxed much more finely meshed versions of both the bubble shock and the Richtmyer-Meshkov instability problems (up to 36 times the coarse number of zones for the bubble shock problem, and up to 81 times coarse number of zones for the Richtmyer-Meshkov problem). Not only did the simulations run without encountering tangled zones, the simulation time step was kept appropriately large, allowing us to reach late simulation times in far fewer cycles than were required when using the by-hand relaxers (see Table 1).

In addition, we used LAGER to relax a simulation of the ablation of a hohlraum wall using lasers. Though these simulations incorporate much more complicated physics than the the simple hydrodynamics problems posed above, and despite the fact that LAGER received no training on data generated by running this problem, LAGER was able to successfully perform relaxation in this scenario. This result along with LAGER's success in relaxing the finely meshed hydrodynamics problems suggests that the LAGER does achieve a certain amount of robustness across multiple problems usually not achievable through the use of by-hand relaxers.

Apart from prevention of tangled zones, we also demand that we preserve accuracy when using LAGER (i.e. we wish to demonstrate that LAGER is not simply over-relaxing our simulations). To assess the accuracy of our simulations using LAGER, we compared a simulation of the Richtmyer-Meshkov instability problem with experimental results from Collins and Jacobs[3]. As can be seen in Figure 4, we were capable of replicating the

| **Bubble Shock Problem:** | Minimum simulation time step | Cycles to reach goal time |
|---|---|---|
| Using LAGER | $O(10^{-1})$ shakes | ∼71,000 |
| Using by-hand relaxers | $O(10^{-2})$ shakes | ∼252,000 |
| **R-T Shock Tube Problem:** | | |
| Using LAGER | $O(10)$ shakes | ∼35,700 |
| Using by-hand relaxers | $O(10^{-9})$ shakes | ∼57,600 |
| **Hohlraum Wall Problem:** | | |
| Using LAGER | $O(10^{-11})$ shakes | ∼26,100 |
| Using by-hand relaxers | $O(10^{-11})$ shakes | ∼35,300 |

Table 1: This table shows the minimum simulation time step encountered and the number of cycles required to achieve a late goal time for each of the test simulations (comparing simulations using LAGER to simulations using by-hand relaxers for each).

experimental results quite accurately using LAGER.

## 4.2 Training on different datasets

As mentioned in Section 3.2, we varied several different factors in the creation of the training data and produced several different random forest algorithms based on the resulting data sets. Namely, we varied: the shape of the transition function; the length of the transition function; and the simulated problem from which the data was obtained. Comparing different transition function shapes, we found the logistic function produced the least reliable results (occasionally resulting in tangling during a simulation). The piecewise constant/cosine and linear functions produced similar results and both reliably avoided tangling. Varying the lengths of each of these functions resulted only in very subtle changes in simulation results, and results from each was satisfactory. For simplicity, we therefore restrict the remainder of our discussion to the case of the piecewise constant/cosine function with support from 0 to 50 cycles before zone tangling.

The most interesting variations in performance of LAGER arose from choosing which simulation problem training data was generated on. We trained random forest algorithms on data sets coming exclusively from the bubble shock problem, exclusively from the Richtmyer-Meshkov problem, or a combination of both data sets. The different geometries and physical regimes of these two problems produce zones which tangle in different fashions for each problem (when run without relaxation), creating data sets with different characteristics. This translates into very different behavior of the random forest algorithms trained on the different data sets.

We first examine this difference in behaviors qualitatively. Figure 5 displays snapshots of the same simulation time from three different simulations running LAGER trained on each data set. These snapshots show the differences in how LAGER assigns $\phi$ values to each zone depending the training set used to construct the random forest. An algorithm

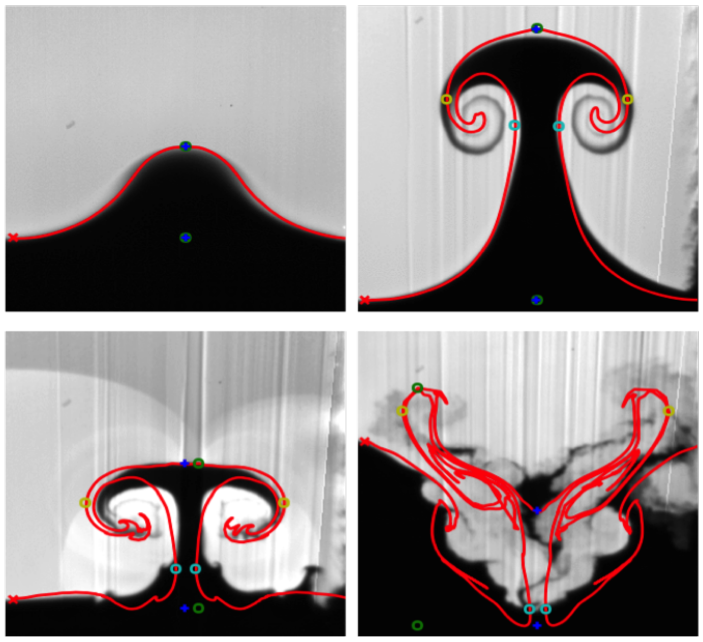


Figure 4: Simulation results using LAGER for relaxation (red outline defines the interface calculated in simulation) overlaid on experimental photos[3].

trained on the Richtmyer-Meshkov data appears to apply a small amount of relaxation to most zones in a large area around the interfaces in each simulation, while the other algorithms display more targeted relaxation, yielding larger $\phi$ values on fewer zones.

We may also obtain some quantitative measure of how the algorithms trained on each data set differ from one another. In Figure 6, we plot the total amount of relaxation occurring per cycle as well as cumulative relaxation over the course of the simulation vs. simulation time for each hydrodynamic test problem. For the bubble shock problem, we see that the by-hand relaxers end up performing significantly more relaxation than LAGER. In the Richmyer-Meshkov problem, however, we see the opposite effect: the by-hand relaxers actually perform less overall relaxation. In general, we associate over-relaxation with loss of physical accuracy. In the case of the Richtmyer-Meshkov problem, however, we have already verified in Section 4.1 that our simulation results agree very closely with experimental results. So the additional relaxation occurring due to the use of LAGER in the Richtmyer-Meshkov problem appears to not have a negative impact on accuracy in this qualitative comparison.

As to the difference between random forest algorithms trained on different data sets, the quantitative results agree with the trend we see in our qualitative analysis. For the bubble shock problem, we see that the algorithm trained on the Richtmyer-Meshkov data performs more relaxation overall than the algorithms trained on the other data sets. For

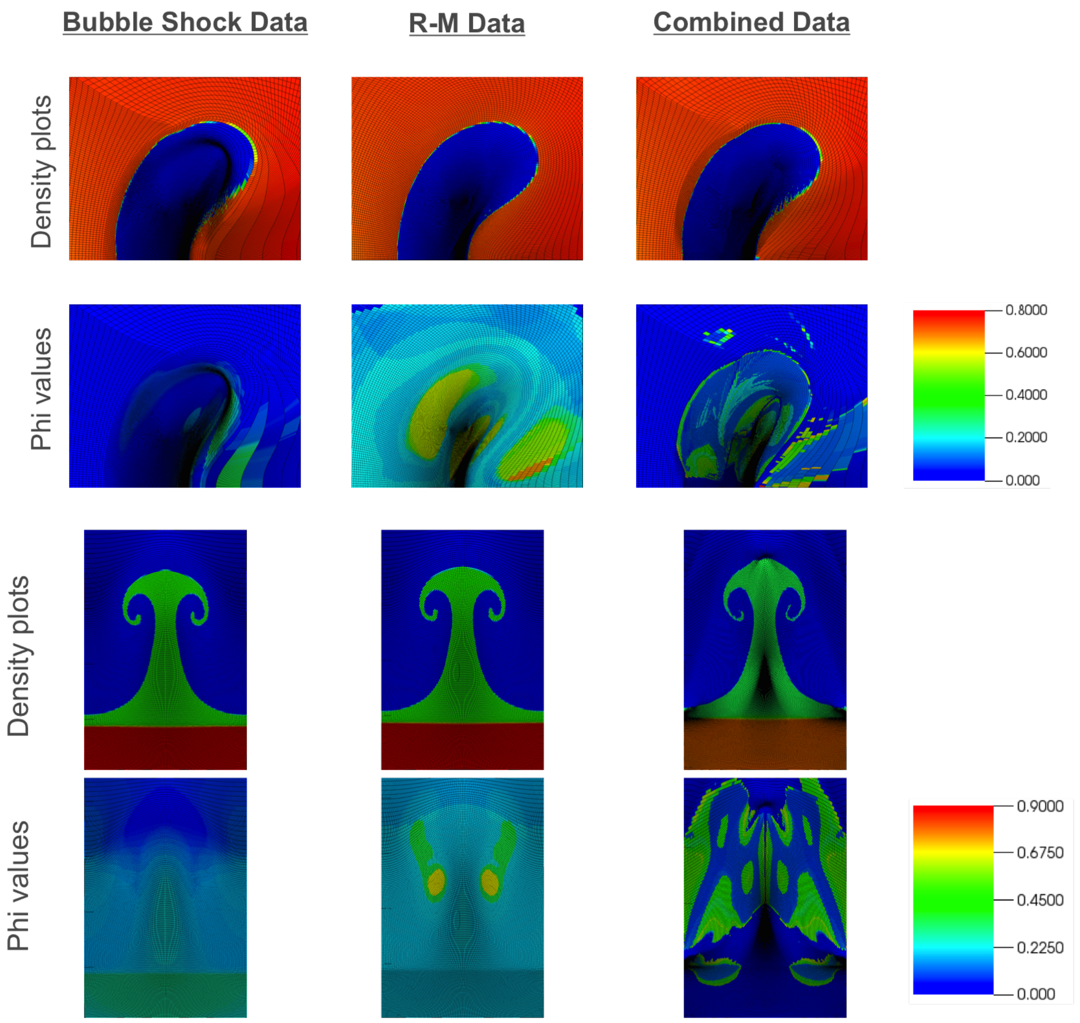


Figure 5: Snapshots from the same simulation time during simulations using LAGER trained on data coming from the bubble shock problem, the Richtmyer-Meshkov problem, or a combination of those two data sets. The top snapshots show a density plot (to more clearly show the state of the bubble or instability), while the bottom snapshots show the $\phi$ values assigned by LAGER to each zone.

the Richtmyer-Meshkov problem, again both the algorithms trained on individual data sets are relaxing more than the algorithm trained on the combined data sets. This indicates that the combination of both data sets yields the most strict relaxation criteria (yielding the most closely Lagrangian simulations). Again, there is some trade off here between amount of relaxation and the size of simulation time step: the combined data set yields a stricter algorithm which may benefit accuracy in some cases but may also produce longer simulations due to less relaxation of those zones causing small time steps.

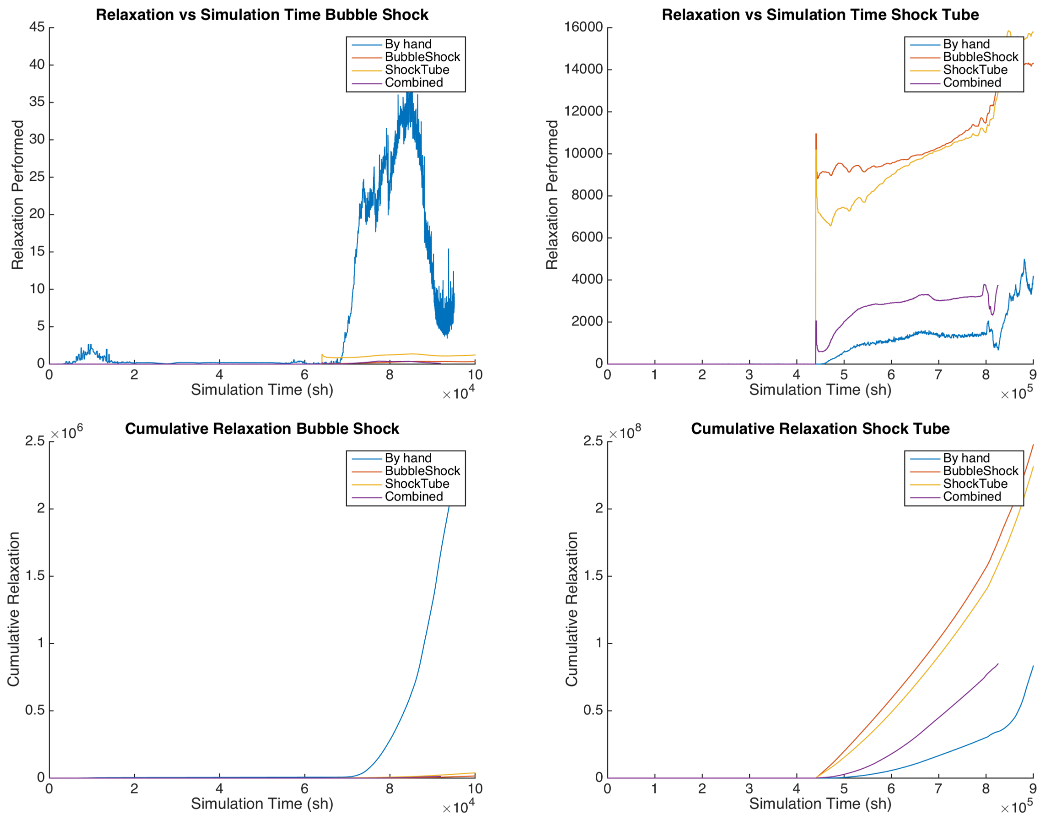


Figure 6: The top row shows the amount of relaxation (combined magnitude of all relaxation vectors) on each cycle vs. simulation time, comparing the by-hand relaxers and LAGER trained on each data set. The second row shows the cumulative relaxation done over the course of the entire simulation vs. the simulation time for by-hand relaxers vs. LAGER trained on each data set.

### 4.3 Computational costs

While we did not spend any time optimizing the incorporation of LAGER into KULL in terms of computational speed, we still desired to get some idea of the computational expense of using LAGER. In Figure 7, we show a comparison of computational time required to run the bubble shock and shock tube problems using by-hand relaxers vs. LAGER. For each problem, relaxation only begins to occur just before zone tangling starts to become an issue (at 65,000 sh for the bubble shock problem and 450,000 sh for the shock tube problem). For the bubble shock problem we see that LAGER trained on the bubble shock and shock tube data sets actually reaches late simulation times faster than the by-hand relaxers (while LAGER trained on the combined data set is slightly slower). This may seem surprising, since generating predictions with the random forest algorithm is much more expensive than simply deciding relaxation based on user defined criteria. Recall, however, that LAGER attempts to maintain an appropriately large simulation time step. Thus we observe that for the bubble shock problem, the by-hand relaxers require many more cycles than LAGER to reach late simulation times, resulting in overall greater wall clock time. For the shock tube problem, the by-hand relaxers outperform the LAGER relaxers in the sense of wall clock time spent vs simulation time achieved. This does not capture the fact, however, that constructing the by-hand relaxers requires a significant number of simulation runs to tune, which demands a significant amount of both computational and user time.

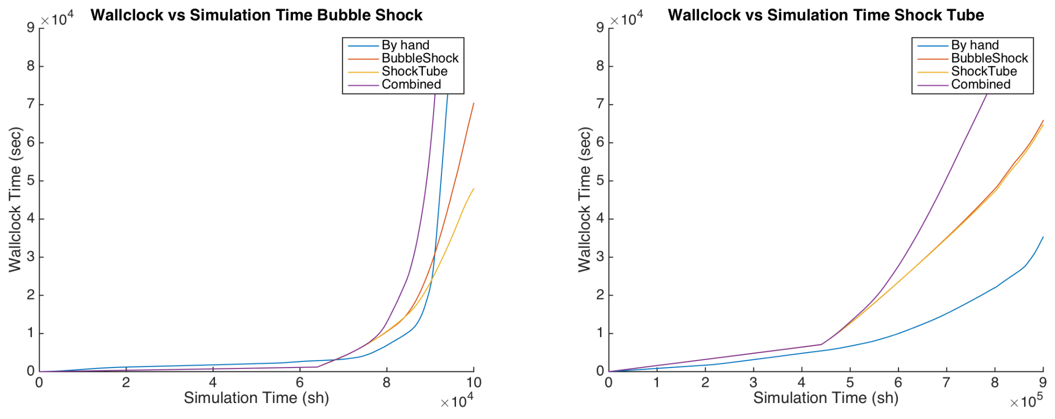


Figure 7: Computational cost of simulations using LAGER trained on the different data sets vs. by-hand relaxers.

## 5 Conclusion

We presented a machine learning approach to automating the relaxation process for Arbitrary Lagrangian-Eulerian simulations. Our Learning Algorithm-Generated Empirical Relaxer utilizes a random forest algorithm which makes predictions about a zone's likelihood of tangling during a simulation. We showed that training the algorithm on data generated by some simple, coarse meshed test problems is sufficient for LAGER to successfully relax more finely meshed problems as well as more complicated multi-physics problems such as the ablation of a hohlraum wall by lasers. We also demonstrated the physical accuracy of simulations using LAGER by comparing simulation results to experimental data. Though LAGER adds a significant amount of computational cost per cycle, its reduction in the number of cycles required to attain late simulation times makes it competitive with the current by-hand relaxation schemes in place. The especially attractive feature of LAGER is that it may potentially be used without modification for a wide variety of problems, eliminating the need for users to spend significant person-hours and computational time creating problem-specific relaxation criteria by hand.

In this paper we began exploring some of the factors which affect the performance of LAGER. This performance is wholly dependent upon the training data. We found that data sets generated on different simulation problems may yield significantly different results in terms of the overall relaxation strategy LAGER uses. A better understanding of how to generate good training data as well as a more detailed grasp of the effects of that training data on the performance of LAGER are therefore the most significant further topics of interest.

## 6 Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-TR-687141).

## References

[1] Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32.

[2] Caramana, E. J., D. E. Burton, M. J. Shashkov, and P. P. Whalen (1998). The Construction of Compatible Hydrodynamics Algorithms Utilizing Conservation of Total Energy. Journal of Computational Physics, 146, 227?262

[3] Collins, B. D. and Jacobs, J. W. (2002), PLIF Flow Visualization and Measurements of the Richtmyer-Meshkov Instability of an Air/SF6 Interface. Journal of Fluid Mechanics, 464:113-136.

[4] Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. Machine Learning, 40(2), 139-157.

[5] Hirt C. W., A. A. Amsden, and J. L. Cook (1974). ?An Arbitrary Lagrangian-Eulerian Computing Method for All Flow Speeds. Journal of Computational Physics, 14, 227-253.

[6] Liu, Wei, Sanjay Chawla, David A. Cieslak, and Nitesh V. Chawla (2010). A Robust Decision Tree Algorithm for Imbalanced Data Sets. Proceedings of the 2010 SIAM International Conference on Data Mining, 766-777.

[7] Pedregosa et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

[8] Rathkopf, J. A., et al. (2000). KULL: LLNL's ASCI Inertial Confinement Fusion Simulation Code. Physor 2000 American Nuclear Society Topical Meeting on Advances in Reactor Physics and Mathematics and Computation into the Next Millennium.