Claremont
GRADUATE UNIVERSITY

Final Report for
Los Alamos National Laboratory

Machine Learning Algorithms for Graph-Based Representations of Fracture Networks

Fall 2016 – Spring 2017

Team Members
Adrian Cantu
Zhengyang (Michael) Guo
Priscilla Kelly
Sean Matz
Manuel Valera (Project Manager)

Advisor
Allon Percus

Liaisons
Jeffrey Hyman
Gowri Srinivasan
Hari Viswanathan

Abstract

Microstructural information, such as fracture size, geometry, aperture, and material properties, plays a key role in modeling the dominant physics for flow propagation through fractured rock. Discrete fracture network (DFN) computational suites, such as DFNWORKS, have recently been developed to simulate flow and transport in such media. These methods allow for particle tracking, revealing a small backbone of fractures through which most transport occurs, and therefore providing a significant reduction in the effective size of the fracture network. However, the simulations needed for particle tracking are computationally intensive, and may not be scalable to large systems.

In this Mathematics Clinic project, conducted at Claremont Graduate University under the sponsorship of Los Alamos National Laboratory, we introduce a machine learning approach to characterizing transport in DFNs. We consider a graph representation where nodes signify fractures and edges denote their intersections. Using supervised learning techniques that train on particle-tracking backbone paths found by DFNWORKS, we predict whether or not fractures conduct significant flow, based primarily on node centrality features in the graph. Our methods run in negligible time compared to particle-tracking simulations. We find that our predicted backbone can reduce the network to approximately 20% of its original size, while still generating breakthrough curves in close agreement with those of the full network. Finally, we present a modeling framework for the dynamic problem of fracture propagation, ultimately intended to provide rapid predictions of when and where material failure occurs.

Contents

Chapter 1

Introduction

1.1 Sponsor

Our CGU Math Clinic team has been working under the sponsorship of Los Alamos National Laboratory (LANL), a United States Department of Energy national laboratory located in Los Alamos, New Mexico. LANL was established in 1943 to develop nuclear weapons during World War II. A current effort at LANL involves developing a method of modeling fractures which can exist in shale gas reserves and after nuclear underground explosions. In order to make the process more computationally efficient, they aim to develop machine learning algorithms to characterize primary flow subnetworks within fractured subsurface materials, and to model the development and propagation of fractures under load.

1.2 Background

Discrete fracture networks (DFN) have been a topic of interest since 1990 when researchers began applying network theory to fracture networks [1]. High value applications, such as water quality monitoring [2], accessing natural gas [3, 4], and the monitoring of gas emissions [5] have motivated the development of high-fidelity models which could be used to predict flow.

Large rock structures contain connected networks of fractures that conduct fluid. The DFN models a fracture network as a random set of intersecting fracture planes, whose statistical properties match those of the original material. These statistical properties can include a multitude of hydrological and geometric quantities associated with each plane. Computational suites such as DFNWORKS [6], developed at LANL, use these properties to simulate flow and transport through the fracture. Such simulations allow more accurate prediction of fluid mechanics than is typically possible using continuum models [7, 8]. On the other hand, a subsurface fracture network can typically contain millions of fractures, with thousands of volume elements needed to describe each fracture. Resolving flow in such a structure is an enormous computational undertaking.

Fortunately, fracture systems often exhibit flow channeling. These channels form a primary flow subnetwork within the DFN, where much of the flow and transport takes place [9, 10, 11, 12, 13, 14]. Restricting simulations to this subnetwork, or "backbone," offers considerable computational savings. Methods such as particle flow simulations have been developed to determine which fractures comprise the backbone [15, 16]. An example is given in Figure 1.1, showing (a) the full DFN and (b) the backbone extracted using particle trajectories. However, these methods require computing transport through the entire fracture network. Furthermore, since a given DFN is only one random realization of a fractured material, the process must be repeated numerous times, to generate sufficient data to reduce the statistical error in the quantities measured.

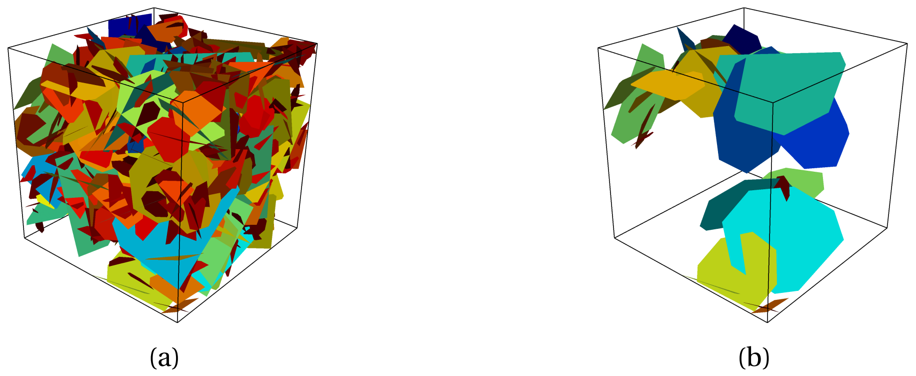


Figure 1.1: (a) Full DFN composed of 499 fractures. (b) Backbone extracted from (a) using particles trajectories, with particles flowing from inlet plane on front left to outlet plane on rear right. Figure by Jeffrey Hyman [17].

A separate challenge is understanding how fractures propagate over time, notably to predict how and when such propagation leads to material failure. High-resolution models such as the hybrid optimization software suite (HOSS) developed at LANL can simulate crack propagation in brittle materials [18]. However, the computational barrier is even more significant here, with millions of degrees of freedom needed *at each time step* to describe the dynamics. Unless a different approach is used, modeling interactions of fractures over time can quickly become intractable.

1.3 Project Goals

It has been observed that flow through sparse fracture networks is governed more by the network topology, namely which fractures connect to which others, than by the hydrological details of the fractures [19]. This raises the question of whether it is possible to identify the backbone of the network purely from topological characteristics. Doing so would solve the computational bottleneck of having to simulate flow and transport explicitly. The difficulty is determining exactly how to combine topological properties for this purpose.

In recent years, there has been increased interest in the use of machine learning in the geosciences. A range of different regression and classification methods have been applied to a model of landslide susceptibility, demonstrating their predictive value [20]. Community detection methods have been used in fractured rock samples to identify regions expected to have high flow conductivity [21]. Clustering analysis has been used in subsurface systems to construct more accurate flow inversion algorithms [22]. **The main goal of the 2016–17 CGU Math Clinic project is to use machine learning techniques to reduce DFNs to subnetworks that carry most of the network's flow. By treating topological properties as features that describe fractures in the network, we develop fast classification methods that learn to characterize the backbone in the feature space.**

Python code for all of our classification methods is available on our project GitHub site. The code is organized into jupyter notebooks, which are described in Appendix A.

1.3.1 Graph representation

Our approach is based on representing a given DFN by a graph, where a node in the graph represents a fracture in the network, and an edge connecting two nodes in the graph represents an intersection between two fractures. Under this construction, the graph retains topological information about the network as node-based properties or features. An example of representing a DFN by a graph is given in Figure 1.2 for a two-dimensional six-fracture network.

Such a graph representation was proposed by Ghaffari, et al. [23] and independently by Andresen, et al. [24], in order to study the network topology of both two- and three-dimensional fracture systems. The graph mapping allowed for a characterization of the topology of fracture networks, and moreover enabled

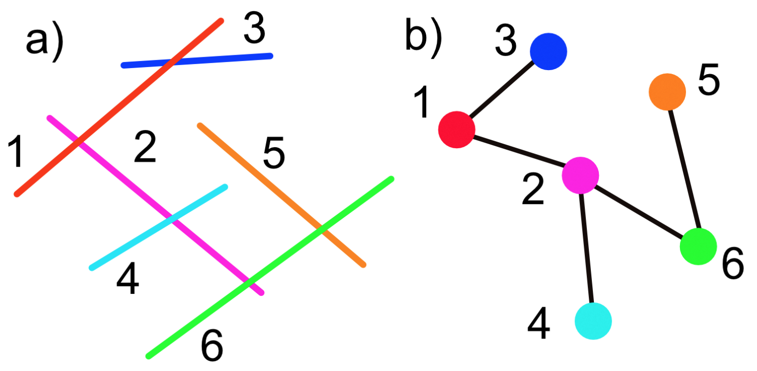


Figure 1.2: Graph representation of a six-fracture Discrete Fracture Network in two-dimensions. a) fractures in the 2D network are translated to nodes and their intersections become edges in b).

quantitative comparisons between real fracture networks and models generating synthetic networks. Vevatne, et al. [25] and Hope, et al. [26] have used this graph construction for analyzing fracture growth and propagation, showing how topological properties of the network such as assortativity relate to the growth mechanism. Santiago, et al. [27, 21, 28] have proposed a method of topological analysis using a related graph representation of fracture networks. By measuring centrality properties of nodes in the graph, which describe characteristics such as the number of shortest paths through a given node, they developed a method intended to predict regions of high flow conductivity in the network.

1.3.2 Machine learning approaches

In our work, we adopt the approach of using different forms of node centrality as predictors of flow. We consider four centrality measures as node features: three of these are global topological measure, while one is a local topological measure. We supplement the centrality measures with two physical (geometric) features that describe the fracture in the DFN model. On the basis of these six features, we apply machine learning to identify systematically the network's backbone. Our two machine learning algorithms are random forest and support vector machines. Both are supervised learning methods: given training data consisting of DFNs whose backbones have already been determined through simulation, they learn how to reduce new DFNs to appropriate subnetworks.

A random forest is constructed by sampling the training set with replacement, so that some data points may be sampled multiple times and others not at all. Those data points that are sampled are used to generate a decision tree, which outputs a classification based on feature values. Those data points that are not sampled are run through the tree to determine its quality. The procedure is repeated so as to generate a large collection of trees. A test data point is then classified by having each decision tree "vote" on its class. This leads not only to a predicted classification, but also to a measure of certainty (the fraction of trees that voted for it) as well as to an estimate of the importance of each feature [29, 30, 31, 32, 33, 34, 35]. That final estimate is particularly useful when the features consist of quantities that measure different aspects of node centrality.

Support vector machines (SVM) separate high-dimensional data points into two classes by finding an appropriate hyperplane. Based on the generalized portrait algorithm [36] and subsequent developments in statistical learning theory [37], the current version of SVM [38] uses kernel methods [39] to generalize linear classifiers to nonlinear ones. SVMs have been shown to perform well in applications with highly correlated feature variables, in part because one can choose kernels or separator boundaries that are most appropriate to the feature space describing the data [40, 34].

One challenge in the DFN data is the imbalance between the size of the backbone ("positive") and non-backbone ("negative") class. In data from simulations, only about 7% of fractures are part of the backbone, reflecting the substantial network reduction afforded by the primary flow subnetwork. But this poses complications in validating our predictions. Simply trying to maximize the overall rate of correct classification could result in labeling much of the backbone as non-backbone (false negative), in which case the flow properties of the reduced network would not match those of the original. We therefore consider both precision (ratio of true positives to all positives) and recall (ratio of true positives to true positive plus false negatives). There is a trade-off between these two, controlled by how strict the classifier is in labeling a sample as positive. While our primary objective is recall, so as to minimize false negatives that can obstruct flow, we are also concerned with exploring the precision/recall space. We therefore use a grid search method that identifies classifier parameters critical to precision and recall. By modifying these parameters, we evaluate how far we are able to reduce the network without significantly affecting flow.

It is important to note that our objective is flow-maintaining network reduction, which is not identical to predicting the backbones determined from particle-based simulation. While we use these backbones as training data for our classifiers, the main validation of our results comes through considering the breakthrough curve (BTC). This curve shows the distribution of simulated particles passing through the network from source plane to target plane in a given interval of time. We would like the BTC for our reduced networks to match that of the full network in a number of respects, including peak breakthrough time, and the nature of the tail of the distribution. While the particle backbone is important for identifying where mass transports through the network, it is only one of many valid network reductions from the perspective of characterizing flow.

We find that, under different parameter choices for random forest and SVM, we are able to reduce DFNs on average to between 39% and 2.5% of their original number of fractures. The two extremes correspond to recall values of 96% and 20%. Reductions to as little as 21% (with recall of 75%) provide good BTC matches, with Kolmogorov-Smirnov statistic values of 0.26 or less. We also assess the importance of the different features used to characterize the data, finding that they cluster into three natural groups. The global topological quantities are the most significant ones, followed by the one local topological quantity we use. The physical quantities are the least significant ones, though still necessary for the performance of the classifier. A manuscript describing our work and the results above has been submitted for publication [17].

We consider a further classification approach that, based on initial tests, not only provides significant network reduction but also comes closer to reconstructing the particle backbone itself. This is a two-stage method, which first forms an initial classification of nodes next to the source and to the sink, and then attempts to propagate labels from nodes identified as backbone. The method was originally motivated by the observation that straightforward applications of random forest and SVM, as described above, can result in backbones that do not connect to both the source and sink. Clearly, such disconnected structures are not usable as a flow subnetwork. In reality, the problem only occurs at parameter choices giving very low recall. But even there, the two-stage method almost always gives a connected backbone. At parameter choices giving high recall, the two-stage method appears to provide outstanding reduction, yielding a subnetwork with only 17% of the original number of fractures while maintaining 90% recall. This method merits further study and development.

1.3.3 Dynamic problem

Finally, we investigate the dynamic problem of modeling fracture growth, and of predicting material failure using machine learning techniques. A number of recent studies have considered modeling crack propagation using a set of rules that act directly on the graph representing the fracture network [25, 26, 18, 41, 42]. To understand this process, consider a fracture with multiple intersections, modeled as a node in a graph with multiple neighbors. Fracture networks have been found to exhibit strong negative correlations in the degree of neighboring nodes; there is a tendency for nodes of high degree (many neighbors) to connect to nodes of low degree (few neighbors) and vice versa. Such networks are called disassortative. The strong correlations lend credence to the idea that new fractures depend heavily on the existing network, and that the growth process is similar to a preferential growth mechanism, where new nodes are more likely to couple to existing nodes of high degree. Note that node degree may or may not be the best metric to utilize for preferential growth; for example, Vevatne et al. [25] use the lengths of existing fractures as the growth metric. We present a preliminary study of fracture growth using a basic random graph model, and suggest improvements to this model. We also discuss promising methods for rapid prediction of material failure, using classification based on topological and geometric features.

Chapter 2

Classification Methods

Given a set of features and class assignment for some observations, supervised learning algorithms try to "learn" the underlying function that maps features to classes. Those observations are the training set. The learned function can then be used to classify new observations. In our study, we use as observations the nodes (fractures) from 80 graphs as a training set. We then test the function using nodes from 20 graphs as a test set.

One challenge we face is a significant imbalance in the number of observations in each class. For the 80 graphs in the training set, only about 7 percent of nodes are part of the backbone class. A classification algorithm could simply assign all nodes to the non-backbone class, and still achieve an overall classification accuracy of 93 percent. Since our aim is to find a meaningful backbone, careful attention must be given to identifying the parameters of our algorithms that maximize the recall, or fraction of backbone nodes in the training set that are correctly predicted.

In this chapter, we describe the features that we use to describe nodes, and our machine learning methods based on random forests and support vector machines. Both are general-purpose supervised learning methods that are suitable both geometric as well as non-geometric features. Furthermore, while both of these algorithms are highly tunable, they rarely require very extensive parameter tuning. We discuss our process for parameter selection, and how we use this to investigate the tradeoff between network reduction and accuracy of flow properties. Both algorithms are implemented using the *scikit-learn* machine learning package in python, with the functions *RandomForestClassifier* and *SVC*.

2.1 Node Features

It may seem that the most natural way to identify a flow subnetwork is to consider all possible source-to-sink paths, and to predict which of those are part of the backbone. This approach, however, is impractical due to the exponential proliferation of possible paths. Instead, we consider individual nodes, and investigate features describing these nodes that we expect to be insightful in predicting the backbone.

Recent studies suggest that graph-theoretic quantities associated with node centrality can help describe crucial topological characteristics of the network [24, 25] and identify regions important in conducting flow [21, 28]. Such quantities can be divided into two categories: global topological measures, which describe a node's place within a graph structure, and local topological measures, which describe a node's immediate neighborhood. We expect that both of these can play a role in predicting the flow properties of a fracture in a network. We therefore consider features from both categories, calculated using the NETWORKX graph software package [43]. We also supplement these with a third category of features, representing physical properties of the fractures. Figure 2.1 illustrates the six different features that we choose to describe nodes on a graph representing the DFN shown earlier in Figure 1.1. It shows how feature values relate to the graph structure and to the particle backbone.

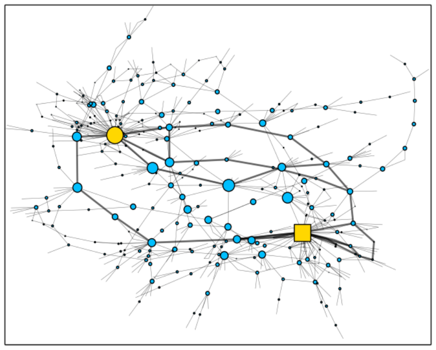
Code for generating the features described below is found on the GitHub site in the jupyter notebook *generate_features.ipynb* (see Appendix A).

2.1.1 Global topological features

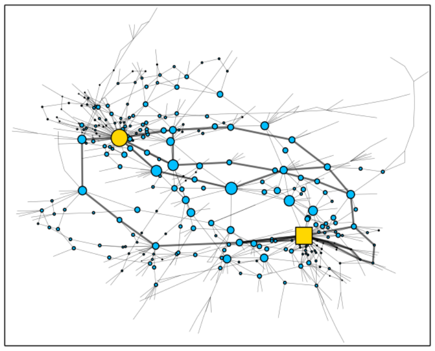
- The *betweenness centrality* [44, 45] of a node (Figure 2.1a) reflects the extent to which that node can control communication on a network. Consider a geodesic path (path with fewest possible edges) connecting a node u and a node v on a graph. In general, there may be more than one such path: let σ_{uv} denote the number of them. Furthermore, let $\sigma_{uv}(i)$ denote the number of such paths that pass through node i. We then define, for node i,

$$\text{Betweenness centrality } = \frac{1}{(N-1)(N-2)} \sum_{\substack{u,v=1 \\ u\neq i\neq v}}^{N} \frac{\sigma_{uv}(i)}{\sigma_{uv}}, \tag{2.1}$$

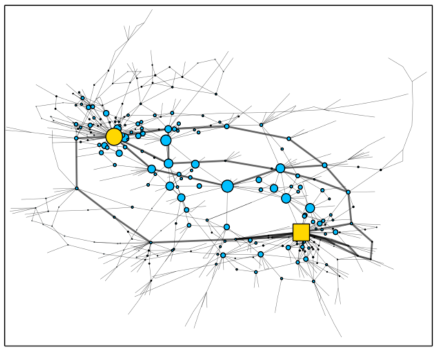
where the leading factor normalizes the quantity so that it can be compared across graphs of different size N. Nodes with high betweenness centrality might well be expected to have a large influence on transport across



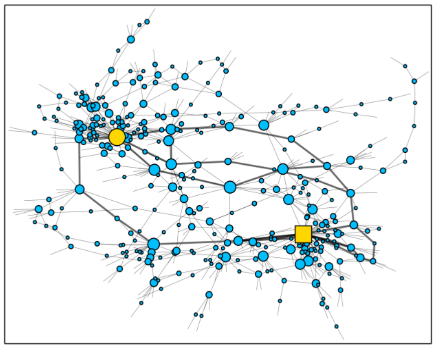
a) Betweenness centrality



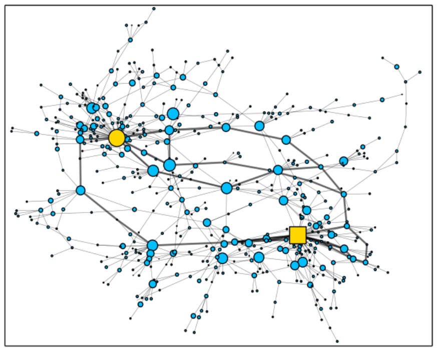
b) Source-to-sink current flow



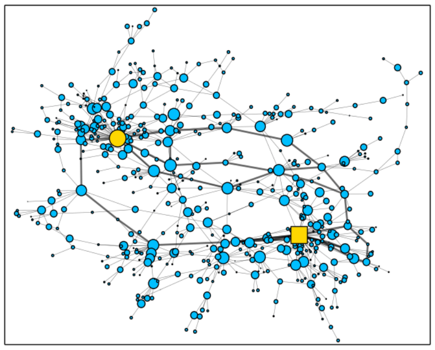
c) Source-to-sink simple paths



d) Degree centrality



e) Projected volume



f) Permeability

Figure 2.1: Visualization of a graph derived from a random DFN as shown in Figure 1.1. Blue circles represent normalized feature values using six different features, in panels a) through f). Yellow square denotes source, yellow circle denotes sink. Heavy lines represent particle backbone paths in the graphs. Note varying extent of correlation between particle backbone and associated feature strength.

the network, and thus be fundamental to the backbone. Figure 2.1a confirms that many backbone nodes do indeed have high betweenness values. At the same time, certain paths through the network that are not part of the backbone also show high values for this feature, reflecting the fact that betweenness centrality considers *all* paths and not only those from source to sink. It could therefore be an important feature for backbone classification, but only in conjunction with others.

- *Source-to-sink current flow* (Figure 2.1b) is a centrality measure that is similar to betweenness, but uses an electrical current model to measure flow within network [46], and assumes a given source and sink. Imagine that one unit of current is injected into the network at the source, one unit is extracted at the sink, and every edge has unit resistance. Then, the current flow centrality is equal to the current passing through a given node. This is given by Kirchhoff's laws, or alternatively in terms of the graph Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{A}$ is the adjacency matrix for the graph and $\mathbf{D}$ is a diagonal matrix whose element $D_{ii} = \sum_j A_{ij}$ is the degree of the node i. If $\mathbf{L}^+$ represents the Moore-Penrose pseudoinverse of $\mathbf{L}$, then for node i, we define

  $$\text{Current flow} = \sum_{j=1}^{N} A_{ij} |(L^+_{is} - L^+_{js}) - (L^+_{it} - L^+_{jt})|, \tag{2.2}$$

  where s is the source and t is the sink. Current-flow centrality is also known as random-walk centrality [47] since the same quantity measures how often a random walk from s to t passes through i.

  Unlike betweenness centrality defined above, current-flow centrality excludes any nodes in the graph that branch off the central core. We therefore expect high current flow values to correlate with nodes that have large influence on source-to-sink transport.

- *Source-to-sink simple paths* (Figure 2.1c) is a straightforward centrality measure that counts non-backtracking paths that cross the graph from source to sink. This information would seem essential for predicting a backbone since such a path directly conducts flow through the material. Similarly to the formalism for betweenness centrality, π_{st} denotes the number of simple (non-backtracking) paths from source s to sink t, and $\pi_{st}(i)$ denotes the number of such paths that pass through node i. We then define, for node i

$$\text{Simple paths} = \frac{\pi_{st}(i)}{\pi_{st}}, \tag{2.3}$$

where normalization by π_{st} allows to compare values of simple path centrality across different graphs. Since the complexity of path enumeration can scale exponentially in the size of the graph, we limit our search to paths with 15 nodes or less. This restriction has physical justification, as direct paths tend to be more common in backbones. While these path lengths can vary considerably, we find empirically that if we increase the upper bound on path length beyond 15, the effect on the simple path values is negligible.

Figure 2.1c illustrates that nodes with high source-to-sink simple path centrality are more likely to lie on backbone paths than are nodes with high betweenness centrality in Figure 2.1a. However, simple path centrality also fails to identify one isolated backbone path that is disjoint from the others. We expect it to serve an important role in backbone classification, though again, only in conjunction with other features.

2.1.2 Local topological feature

- *Degree centrality* (Figure 2.1d) is a normalized measure of the number of edges touching a node. For node i,

$$\text{Degree centrality} = \frac{1}{N-1}\sum_{j=1}^{N} A_{ij}. \tag{2.4}$$

Nodes with high degree centrality tend to be concentrated in the core of the network. Conversely, nodes with low degree centrality are often in the periphery or on branches that cannot possibly conduct significant flow. Furthermore, since the degree centrality of a fracture is the number of other fractures that intersect with it, degree centrality is closely related to fracture volume.

2.1.3 Physical features

We supplement the four topological features with two features describing physical and geometric properties of fractures.

- *Projected volume* (Figure 2.1e) measures the component of a fracture's volume that is oriented along the direction of flow from inlet to outlet plane.

Fracture planes have different orientations in the DFN, and those that are oriented parallel to the main flow direction are more likely to conduct significant flow than those that are oriented perpendicular the normal flow direction. We therefore consider the projection of the volume onto the axis of flow. Let fracture i have volume V_i and orientation vector $\mathbf{O}_i$ (unit vector normal to the fracture plane). Taking the flow to be oriented along the x-axis, the projected volume is expressed in terms of the projection of $\mathbf{O}_i$ onto the yz-plane:

$$\text{Projected volume} = V_i \sqrt{(O_i)_y^2 + (O_i)_z^2}. \tag{2.5}$$

Figure 2.1e shows similarities between this feature and degree centrality, but also some fractures where one feature correlates more closely with the backbone than the other.

- *Permeability* (Figure 2.1f) measures how easily a porous medium allows flow. Given the aperture size b_i of fracture i, the permeability is expressed as

$$\text{Permeability} = \frac{b_i^2}{12} \tag{2.6}$$

The permeability of a fracture, which is nonlinearly related to its volume, is a further measure of its transport capacity. As illustrated in Figure 2.1, it displays similarities to both degree centrality and projected volume, with backbone fractures almost systematically having high permeability values (but the converse holding less consistently).

2.1.4 Correlation of feature values

As is seen in Figure 2.1, the feature values vary widely from one node to another in ways that we aim to manipulate in order to predict backbone paths. Figure 2.2 shows correlation coefficients for pairs that include the particle backbone and the six features that we have chosen. The fact that there are non-negligible correlations between the backbone and these features suggests that they are relevant ones for classification, although clearly no single feature is sufficient in itself. We also notice from the correlation coefficients that features tend to cluster naturally into the three categories above. The first three features, which are the global topological ones (betweenness, current flow, and simple paths), have significant mutual correlations among then. The same is true for the physical

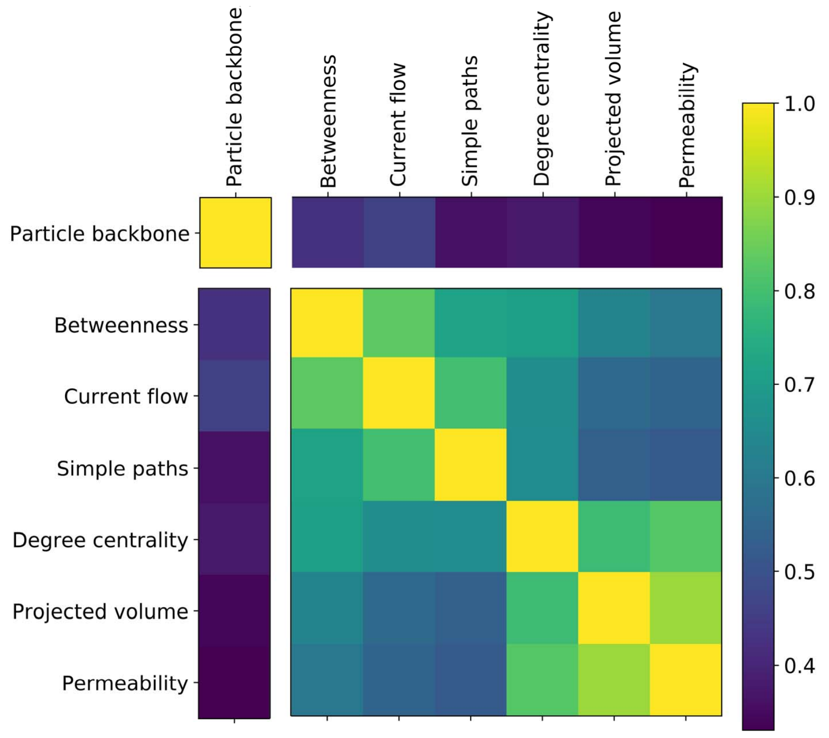


Figure 2.2: Heat map displaying correlations among the particle backbone and the six features used.

features (projected volume and permeability), which also exhibit some clustering with the related local topological feature (degree centrality). The latter correlations are consistent with our feature definitions above.

As a further illustration of both the relevance of the features chosen and the complexities associated with using them, consider the scatter plot in Figure 2.3. Here, we show values of source-to-sink current flow and source-to-sink simple paths, for all fractures in the example DFN that we have been considering so far. Fractures are colored according to whether or not they are in the particle backbone. On this two-dimensional plot, there is no clear boundary separating the two classes, but some differentiation of class densities is noticeable over different regions of the feature space. The plot confirms the effect, seen in Figure 2.2, that backbone membership is very weakly correlated with lying on many simple paths but somewhat more strongly correlated with high current flow. However, it also suggests that the density of backbone nodes is actually larger towards the

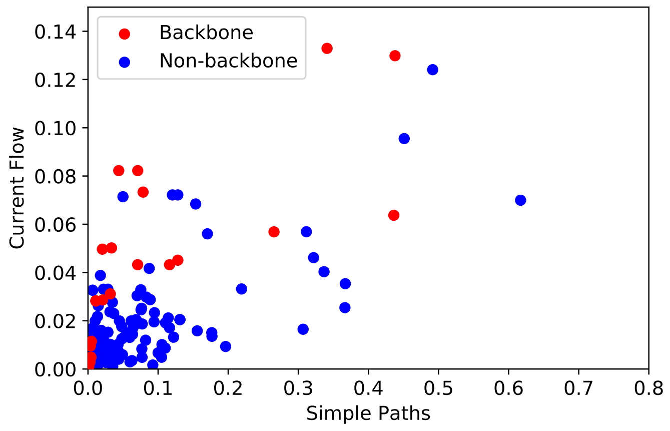


Figure 2.3: Scatter plot of feature values for source-to-sink current flow and source-to-sink simple paths, showing class identification for each fracture.

upper left of the plot, where current flow increases but simple path centrality *decreases*. This may reflect the fact that high current flow together with high simple path centrality signifies a bottleneck in the graph, with many source-to-sink paths needing to pass through a given fracture. The physical capacity limits of such a fracture could prevent it from conducting large flow, forcing particles to find alternate paths.

Code for generating the correlation plots above is found on the GitHub site, in the jupyter notebooks *main_classification_script.ipynb* and *2-D_plot_features .ipynb* (see Appendix A).

2.2 Random Forest

2.2.1 Algorithm

A decision tree [48] is a tree whose interior nodes represent binary tests on a feature (e.g., is $C_{SS}(i) > 0.25$) and whose leaves represent classifications (e.g., node i is part of the backbone). An effective way of constructing such a tree from training data is to measure how different tests, also called splits, separate the data. The *information gain* measure compares the entropy of the *parent* node to the weighted average entropy of the *child* nodes for each split. The splits with the greatest information gain are executed, and the procedure is repeated recursively for each child node until no more information is gained, or there are no more possible splits. A limitation of decision trees is that the topology is completely dependent on the training set, Variations in the training data can produce substantially different trees.

The random forest method [49, 50] addresses this problem by constructing a collection of trees using subsamples of the training data. These subsamples are generated with replacement (bootstrapping), so that some data points are sampled more than once and some not at all. The sampled “in-bag” data points are used to generate a decision tree. The “out-of-bag” observations (the ones not sampled) are then run through the tree to estimate its quality [29]. This procedure is repeated to generate a large number (hundreds or thousands) of random trees. To classify a test data point, each tree “votes” for a result. This provides not only a predicted classification, determined by majority rule, but also a measure of certainty, determined by the fraction of votes in favor. The use of bootstrapping effectively augments the data, allowing random forest to perform well using fewer features than other methods.

Additionally, random forest provides an estimate of how important each individual feature is for the class assignment. This is calculated by permuting the feature’s values, generating new trees, and measuring the “out-of-bag” classification errors on the new trees. If the feature is important for classification, these permutations will generate many errors. If the feature is not important, they will hardly affect the performance of the trees.

Code for running the random forest algorithm is found on the GitHub site in the jupyter notebook *main_classification_script.ipynb* (see Appendix A).

2.2.2 Parameter Selection

In order to identify the parameters of random forest that affect our results most significantly, we use a grid search cross-validation method, implemented with the *GridSearchCV* function in *scikit-learn*. This method optimizes a classifier by exhaustive search within a given range of parameter values, recursively building and testing models. Given the class imbalance in our problem, we set the parameter ranges to aim for high recall. We find the greatest sensitivity to a parameter that sets the minimal number of samples in a leaf node. This parameter forces the algorithm to reject any candidate split that would result in a child node having fewer than a fixed number of observations. Adjusting that number can prevent overfitting, which in the context of unbalanced classes could cause practically none of the feature space to be assigned to the minority class.

Another important parameter choice involves class weights. When random forest classifies a new observation, and the observation ends up in a leaf node with training observations from more than one class, that particular tree will output a fractional vote for a class according to the number of training observations of that class in the leaf. One often sets all training observations to have equal weights. However, by instead assigning training observations a weight that is inversely proportional to class frequency, so that votes from the minority class count more, we can effectively move the decision boundary in favor of a backbone classification.

Additional code for random forest parameter selection is found on the GitHub site in the jupyter notebook *model_selection.ipynb* (see Appendix A).

2.3 Support Vector Machines

2.3.1 Algorithm

Support vector machines (SVM) use a *maximal margin classifier* to perform binary classification. Given training data described by p features, the method identifies boundary limits for each class in the p-dimensional feature space. These boundary limits, which are $(p-1)$-dimensional hyperplanes, are known as local classifiers, and the distance between the local classifiers is called the margin. SVM attempts to maximize this margin, making the data as separable as possible, and defines the classifier as a hyperplane in the middle that separates the data into two groups. The data points on the boundaries are called support vectors, since they "support" the limits and define the shape of the maximal margin classifier. An example for two features ($p = 2$) is shown in Figure 2.4.

Once SVM has generated a maximal margin classifier from training data, one uses this to predict the class of test data points. For example, given p features, the classifier for a linear kernel can be written as:

$$f(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} \beta_j x_j, \tag{2.7}$$

where $\mathbf{x}$ is a vector whose jth component x_j represents the value of the test point's jth feature, and the parameters β_j are derived from the training data. The point is assigned to one class if $f(\mathbf{x}) < 0$, and to the other if $f(\mathbf{x}) > 0$. In the $p = 2$ example of Figure 2.4, any test point lying on the left of the solid line would be assigned to the blue class and any test point lying on the right would be assigned to the red class.

SVM falls into the category of kernel methods, a theoretically powerful and computationally efficient means of generalizing linear classifiers to nonlinear ones. For instance, on a two-dimensional surface, instead of a straight line we can choose a polynomial curve (Figure 2.5a) or a radial loop (Figure 2.5b), by

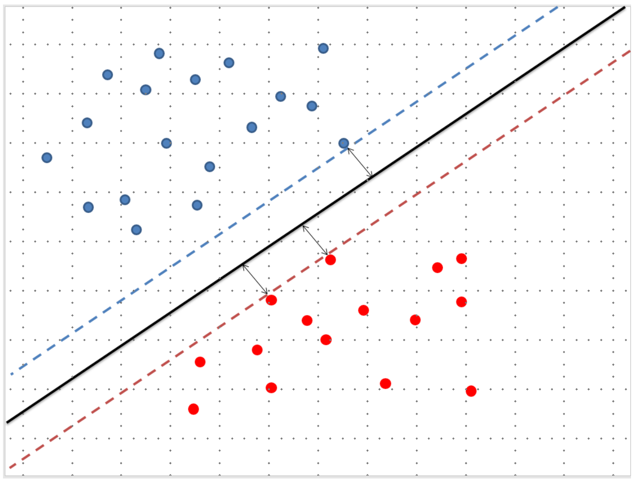


Figure 2.4: Maximal margin classifier (solid line) chosen to maximize distance between local classifiers (dashed lines). Support vectors are points on dashed lines.

specifying the appropriate kernel function. Given n training points $\mathbf{X}^{(1)},\ldots,\mathbf{X}^{(n)}$, the classifier $f(\mathbf{x})$ can be expressed in the general form

$$f(\mathbf{x}) = \alpha_0 + \sum_{i=1}^{n} \alpha_i K(\mathbf{x},\mathbf{X}^{(i)}), \tag{2.8}$$

where $K(\mathbf{x},\mathbf{X}^{(i)})$ is the kernel function chosen. Radial kernels often provide the best classification performance, but at higher computational costs.

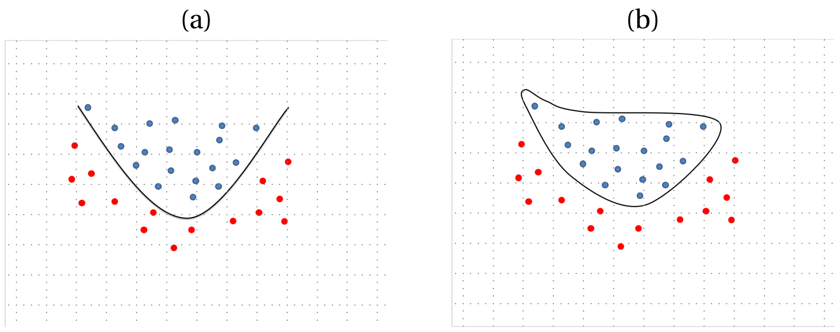


Figure 2.5: (a) Maximal margin classifier with a polynomial kernel. (b) Maximal margin classifier with a radial kernel.

There are a number of practical considerations in feature selection for SVM. First of all, we follow the usual practice of standardizing the data set as a preprocessing step. This involves rescaling each feature so that values have mean zero and variance one, thereby eliminating distortions that could bias the classifier in favor of a given feature. Second of all, it is rarely a disadvantage to increase the number of features used by SVM, even if those additional features are relatively unimportant to the class assignment. The maximal margin classifier changes very little if noisy features are added to the data, making SVM less prone to overfitting than many other methods. We therefore enhance our feature space for SVM in the following way. For each of the six features discussed in Section 2.1, we rank the values within a given graph: the lowest value within the graph has rank 1, and so on. Rankings for tied values are arbitrary. We then consider the raw and ranked values as separate features, doubling the size of the feature space for both training and prediction.

Code for running the SVM algorithm is found on the GitHub site in the jupyter notebook *main_classification_script.ipynb* (see Appendix A).

2.3.2 Parameter Selection

The most important parameter affecting the performance of SVM is the tolerance, known as C. In order to avoid overfitting, SVM often uses a "soft" margin rather than a hard one, allowing misclassification among the training data. When C is set to be large, tolerance is low: $C \to \infty$ is the limiting case of the hard margin described above, where local classifiers strictly bound data points from one class. When C is small, tolerance is high: a training point from one class may be found on either side of the local classifier.

As in the case of random forest, we use grid search cross-validation to identify and optimize crucial parameters such as tolerance. Due to the class imbalance in the problem, it is necessary to adjust the weights associated with the classes. Rather than setting the same tolerance value, C, for both classes, we assign a weight to the tolerance for each class. This allows the local classifier to more strictly bound the (minority) backbone nodes rather than the (majority) non-backbone nodes. In this way, we can prevent the classifier from overfitting the majority class while simultaneously preventing it from missing points in the minority class. Using different weights allows us to construct classifiers that are more likely or less likely to assign a node to the backbone class.

Additional code for SVM parameter selection is found on the GitHub site in the jupyter notebook *model_selection.ipynb* (see Appendix A).

2.4 Two-Stage Method

We have also developed another classification method that uses two successive implementations of machine learning algorithms. Our two-stage method is motivated by the observation that straightforward node classification using RF or SVM can yield a disconnected backbone. Since each node is classified independently, the algorithm cannot guarantee that the backbone will connect the source to the sink, even if in practice this does normally occur in solutions with higher recall. The two-stage method attempts to form source-to-sink paths, while also coming closer to reproducing the particle backbone.

In the first stage, a small subset of nodes are classified. The subset consists of the source node's immediate neighbors and the sink node's immediate neighbors (Figure 2.6a), i.e., fractures on the inlet and outlet planes. These nodes are either classified as backbone, or left as unclassified. There are two reasons for making an initial round of predictions in this way. First, there is a much smaller class imbalance next to the source and sink, with a significant fraction of backbone nodes in the training data. Second, if certain of these nodes can be iden-

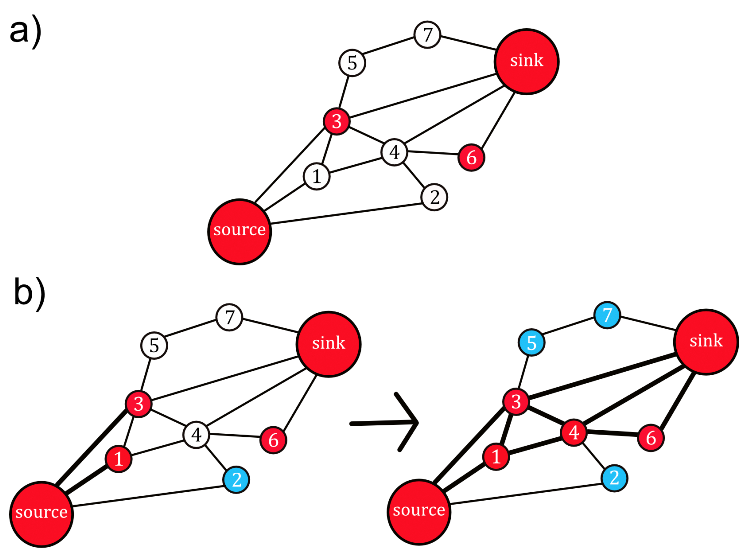


Figure 2.6: Depiction of the two-stage method. In the first stage (a), certain neighbors of the source and sink are identified as backbone (red). In the second stage (b), a classifier that has been trained on neighbors of backbone nodes successively identifies neighbors of red nodes as backbone (red) or non-backbone (blue), until all nodes have been classified.

tified as positive (backbone) with high accuracy, then it may be easier to obtain accurate predictions of how flow propagates further through the network.

The second stage addresses precisely the flow propagation problem, but uses a different classifier that is trained explicitly on *neighbors* of backbone nodes. The algorithm starts from the nodes identified as backbone in the first stage, and considers their neighbors, classifying them as backbone or non-backbone (Figure 2.6b, left). It then successively repeats this process on neighbors of nodes already classified as backbone, thereby generating backbone paths, until all nodes have been classified (Figure 2.6b, right). This almost always results in a connected source-to-sink backbone. If it does not, a connected backbone can be enforced by requiring that at each step, at least one previously unclassified neighbor is classified as backbone. Physically, this is similar to forcing flow to continue propagating from backbone nodes.

In order to lower the rate of false positives, the algorithm uses an enhanced space of ranked features. For the first stage, this consists of precisely the six raw and six ranked features used for the SVM implementation in Section 2.3. For the second stage, however, in addition to these twelve, the algorithm also consider ranked feature values where the ranking is determined only among neighbors of a given node. By doing so, it attempts to mimic the flow options available to particles at a given fracture, incorporating the simultaneous use of local and global information to predict how flow propagates.

For both the first and second stages, either RF or SVM may be used. The same method does not necessarily need to be used in both stages: different combinations may be tried. Our implementation uses RF in both stages, and the first stage employs the identical algorithm to that of section 2.2. However, the use of the two-stage method results in considerably improved precision and recall, as discussed below in section 3.5.

Code for running the two-stage algorithm is found on the GitHub site in the jupyter notebook *two_stage.ipynb* (see Appendix A).

2.5 Dynamic Graph

We briefly considered the separate problem of fracture growth and propagation, for predicting failure in brittle materials. The nature of the classification problem here is quite different from the static problem of backbone identification that was our primary focus.

The first challenge is simply to develop a set of basic rules that model fracture growth and material failure, understood as the moment when a failure plane develops in the network. We use a straightforward dynamic graph model, motivated by random geometric graphs, and similar in some respects to the recently proposed random neighborhood graph model [42]. We start with a collection of nodes placed uniformly at random in a unit square. These nodes represent "seeds" that subsequently grow into fractures. Initially, no edges exist. As the dynamic graph evolves, edges are added, representing the increasing intersection of planes due to fracture growth. Eventually, a connected path of edges spans the graph from one boundary to the opposite boundary, representing a failure plane. The time it takes for this plane to appear is called failure time, and the method used to attach edges in the graphs are a set of basic rules that emulate stress applied in the material, to make the fractures grow and coalesce. Different sets of rules have been applied in the literature [51, 25, 26].

For the details of our dynamic graph model, we adopt a scheme of growing ellipses. Each randomly placed seed (Figure 2.7a) is assumed to have a certain region of influence around it, represented by an ellipse of fixed eccentricity. Ellipses are all aligned with their principal axis in the horizontal direction, emulating a vertical stress field. As stress increases, ellipses grow as an increasing multiple of their original size. When an ellipse has grown large enough to include another node within it, the two nodes are connected by an edge (Figure 2.7b), representing fractures that have grown to intersect each other. Once a path forms from the left boundary to the right boundary (Figure 2.7c), material failure occurs and the process stops.

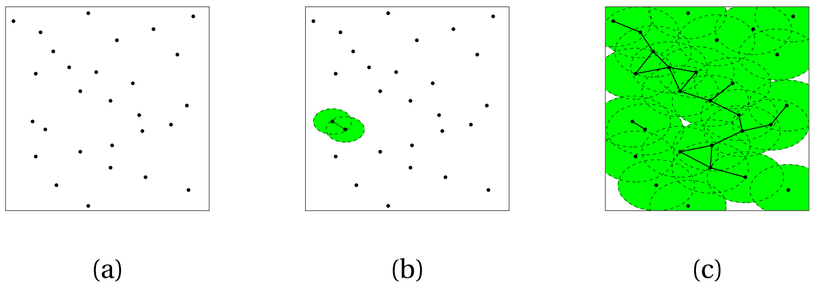


Figure 2.7: Simple dynamic case model using random geometric graphs with growing ellipses. (a) Random initial placement of nodes is given. (b) An ellipse is defined around every node with radius increasing over time, and edges added between the node and all others that are inside. (c) Failure plane develops when a path first spans the network.

In reality, initial crack seeds have an orientation. We assume that these orientations are uniformly random. However, we also assume that cracks that are perpendicular to the stress field will propagate the fastest, whereas cracks that are parallel to the stress field will not propagate at all. Therefore, we take a seed's original ellipse size to be proportional to the projection of the orientation vector (orthogonal to the crack) onto the stress axis.

Code for the dynamic graph simulation is found on the GitHub site in the jupyter notebook *dynamic_graph.ipynb* (see Appendix A).

The larger challenge is prediction and classification. While running this code is computationally trivial, running more realistic crack propagation simulations can exhaust computational resources. The intention is therefore to use our random model and related ones to form a testbed for predicting failure paths, using machine learning. Ideally, we want to identify fractures that lie on the failure path *without* having to run the simulation, and purely from features associated with the fracture topology and geometry. Identifying the failure path has similarities to identifying a flow backbone, but the relevant features are hardly straightforward. There is no clear definition of node centrality in the initial graph, because initially there are no edges. The distance from a node to its nearest neighbor may be a predictor for classification. More likely, however, one needs to consider distances to the kth neighbor, with $k > 1$, in order to capture the global information needed to predict failure.

Chapter 3

Results

We used a collection of 100 graphs, 80 of which were chosen as training data, and 20 of which were chosen as test data. We illustrate certain results, including breakthrough curves, on the DFN shown in Figure 1.1. Other results are based on the entire test set, which consists of a total of 9238 fractures, 651 of which (7.0%) are in the particle backbone and 8587 of which (93%) are not. The total computation time to train both RF and SVM was on the order of a minute, negligible compared to the time to extract the particle backbone needed for training. Once trained, the classifier ran on each test graph in seconds.

3.1 Performance Measures

We define a *positive* classification of a node as being an assignment to the backbone class, and a *negative* classification as being an assignment to the non-backbone class. True positives (TP) and true negatives (TN) represent nodes whose backbone or non-backbone assignment matches that of the labeled training data. False positives (FP) and false negatives (FN) represent nodes whose backbone or non-backbone assignment is opposite that of the labeled training data. Therefore, one straightforward measure of success is the TP rate. *Precision* and *recall* represent two kinds of TP rates:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.1}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.2}$$

Notice that precision is the number of true positives over the total number that we classify as positive, whereas recall is the number of true positives over the total number of actual positives. These values give an understanding of how reliable (precision) and complete (recall) our results are.

Note, however, that even though we train our classifier according to the particle backbone, our objective is not necessarily a perfect recovery of that structure. We aim to identify fractures that are a small subset of the full network, but nevertheless conduct significant flow and provide good agreement with the full network's breakthrough curve. Many, but not all, of the fractures in the particle backbone are essential for this purpose. Thus, high recall is needed, though not necessarily perfect recall. Precision is less essential: false positives will increase the size of our remaining network, but even low precision could allow for a significant reduction in the number of fractures. While we hope to reduce the network while still maintaining flow properties, we also wish to study the trade-off between these two goals.

3.2 Random Forest

We implemented random forest using the *RandomForestClassifier* function in *scikit-learn*, on the six features described in Section 2.1. Parameters were set to default values, except for the following: 250 trees were used (*n_estimators=250*), the number of features to consider for the best split was given by the binary logarithm of the number of trees (*max_features=log2*), and information gain was used to measure the quality of a split (*criterion='entropy'*). As noted in Section 2.2.2, we also used voting weights inversely proportional to class frequency (*class_weight='balanced_subsample'*), and then adjusted the minimum number of training samples in a leaf node (*min_samples_leaf*) in order to vary precision and recall. Table 3.1 gives the results from the classifiers generated with four different values of *min_samples_leaf*.

Model	Precision	Recall	Fractures remaining
RF(1400)	18%	90%	36%
RF(30)	26%	75%	21%
RF(15)	30%	65%	15%
RF(1)	58%	20%	2.5%

Table 3.1: Random forest models labeled by the value of the *min_samples_leaf* parameter specifying minimum number of samples requires to be at a leaf node. Percentages for precision, recall, and fractures remaining in network are calculated over all 20 graphs in the test set.

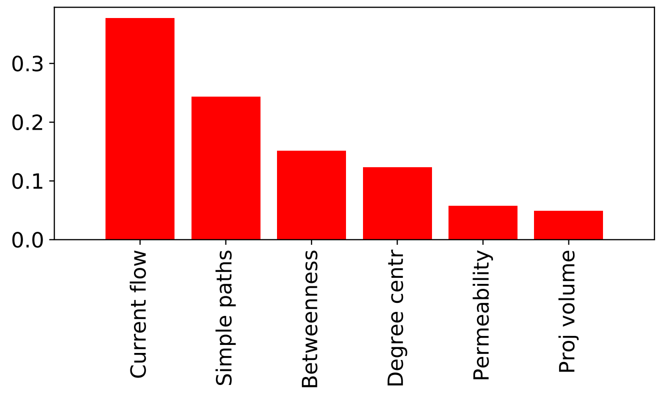


Figure 3.1: Importances of features based on training data for random forest.

Given that the full particle backbone accounts for only 7% of the fractures in the test set, even classifiers with relatively low precision can reduce the network significantly. That effect is seen in the results above.

Random forest also provides a quantitative estimate of the relative importance of each of the six features described in Section 2.1, based on how often a tree votes for it. Using the RF(30) model on our 80 training graphs, we find the feature importances shown in Figure 3.1. As can be seen, the source-to-sink current flow, source-to-sink simple paths, and betweenness centralities are the top importance features, followed by node degree, and followed finally by permeability and volume projection. Thus, as with the feature correlations shown in Figure 2.2, the feature importances cluster into three natural groups. Global topological features have the greatest importance, local topological features have significant but lower importance, and physical features play only a small role in classification. It is interesting to note that, by contrast with SVM, the performance of random forest does not benefit from using additional features, such as ranked values. The inherent bootstrapping of random forest enables strong classification performance even with a relatively limited number of features.

Given the large relative importance of current flow, it is reasonable to ask whether results would change significantly if we were to use this as our *only* feature. In that case, the simplest straightforward classification approach might be to threshold according to current flow, labeling all fractures with values above the threshold as backbone. When the threshold is set at zero, meaning that all fractures with nonzero current flow are labeled as backbone, the result is similar to dead-end fracture chain removal common in the hydrology literature, but more extreme in that it eliminates all dead-end subnetworks. This gives perfect (100%) recall, since all fractures in the particle backbone necessarily have nonzero current flow, and 15% precision, reducing the network to 50% of its original size. However, difficulties occur when increasing the threshold. While the precision and recall results (discussed below and in Figure 3.2) are in some cases competitive with RF, the backbone itself stops being a connected structure and loses its physical relevance. By contrast, with the exception of the low-recall case of RF(1), our multiple-feature classification methods always maintain a connected backbone in spite of the fact that they classify fractures rather than paths.

Just as one can vary the current-flow threshold to produce different precision/recall outcomes, one can adjust a given classifier. Note that this is not the same as generating *different* classifiers from the training data, as we do in Table 3.1. Instead, we take a trained classifier, and modify it to give more or fewer positive assignments. By default, random forest assigns a node to the class receiving at least 50% of the (weighted) tree votes. Changing this threshold will change the number of positive assignments. A low voting threshold will result in high recall (few false negatives) but low precision (many false positives). A high threshold will result in low recall (many false negatives) but high precision (few false positives). In this way, by varying an adjustable parameter we can travel along a precision/recall curve that has perfect recall as one extreme, and perfect precision as the other.

Figure 3.2 shows precision/recall curves for the four RF classifiers, each one with a marker indicating the precision and recall values corresponding to the default 50% threshold used in Table 3.1. For comparison, we also show the precision/recall curve resulting from thresholding on current flow alone, as discussed above. The four RF curves are similar to another, and the marker values appear to represent nearly the best precision/recall pairs obtainable using random forest with our six features. Furthermore, the RF results lie appreciably to the right of the current-flow results, except at the low-recall value of RF(1) where the backbone is not systematically connected. Note that a perfect recovery of training data would push the precision/recall curve to the far right of the figure: 100% precision and 100% recall. However, we again stress that our ultimate objective is not reconstructing training data, but rather reducing the network size while maintaining crucial flow properties. These properties are measured by the breakthrough curve (BTC), which we consider in the validation of our results, in Section 3.4 below.

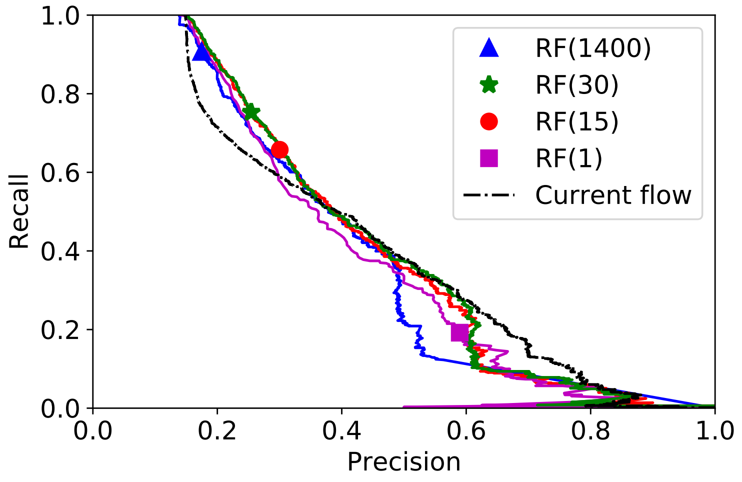


Figure 3.2: Precision/recall curve for the four different random forest classifiers in Table 3.1. Markers indicate default 50% classification threshold for the respective model. For comparison, precision/recall curve is also shown based on thresholding on current flow alone.

3.3 Support Vector Machine

We implemented SVM using the *SVC* function in *scikit-learn*, on twelve features made of up the six raw features described in Section 2.1 as well as their ranked counterparts. Parameters were set to their default values, which include a radial kernel, except for those noted in Section 2.3.2. We chose an overall tolerance of $C = 0.01$, weighted by additional coefficients (*class_weight*) for each class that we adjusted in order to vary precision and recall. This yields a pair of tolerances for the backbone and non-backbone classes. Table 3.2 gives the results from the classifiers generated with four different pairs of tolerances.

As with random forest, we can further adjust each one of these four classifiers to travel along a precision/recall curve. In SVM, the distance from a positively classified point to the decision boundary serves as a classification threshold. By default, this is zero: a point anywhere on one side of the maximal margin classifier (defined as positive distance) is assigned to the corresponding class. But by setting a nonzero threshold distance (negative or positive), a given classifier can generate more or fewer backbone assignments. Figure 3.3 shows the resulting precision/recall curves for the four SVM classifiers, each one with a marker indicating the precision and recall values corresponding to the default

Model	Precision	Recall	Fractures remaining
SVM(0.90,0.054)	17%	96%	39%
SVM(0.90,0.063)	19%	90%	34%
SVM(0.70,0.070)	23%	78%	24%
SVM(0.70,0.190)	44%	46%	7.3%

Table 3.2: SVM models labeled by the tolerance for the backbone class and the non-backbone class. The tolerance for a class is equal to *C* multiplied by its class weight. Percentages for precision, recall, and fractures remaining in network are calculated over all 20 graphs in the test set.

zero threshold that we use for Table 3.2. We again include the results of thresholding on current flow alone, for comparison. Unlike in the case of RF, the four SVM curves are noticeably different. However, notice that for all four of the markers, the model we have chosen is the one providing both the highest precision and the highest recall values. This suggests that there are unlikely to be parameter choices with significantly stronger classification performance than the ones we use here.

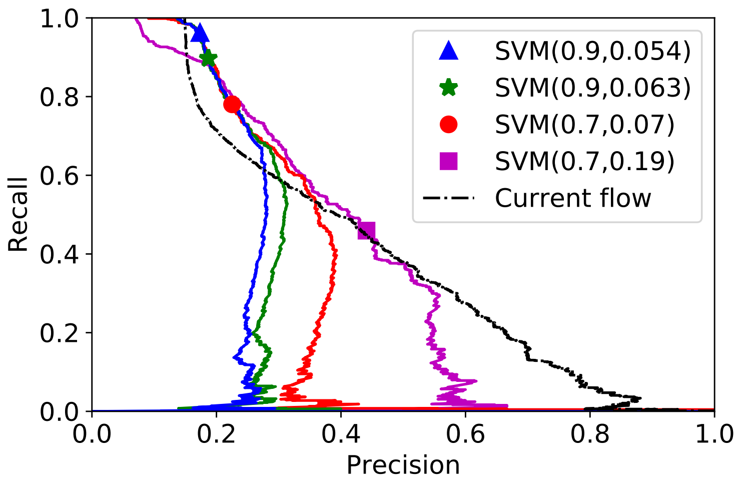


Figure 3.3: Precision/recall curve for the four different SVM classifiers in Table 3.2. Markers indicate classification with default threshold distance of zero for the respective model. For comparison, precision/recall curve is also shown based on thresholding on current flow alone.

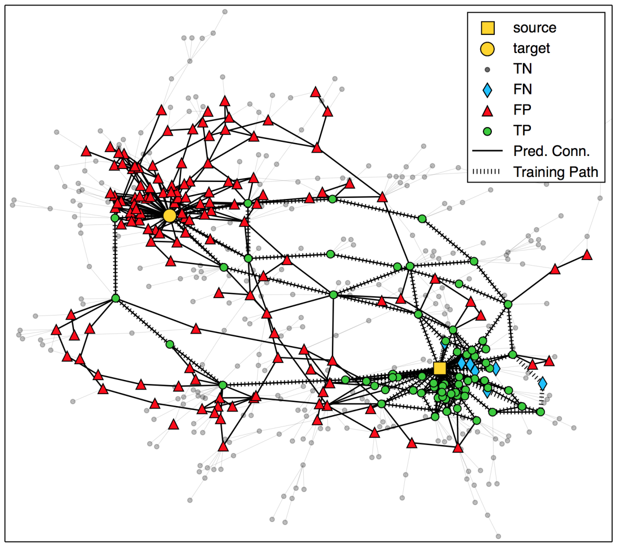
3.4 Validation

In order to evaluate the quality of our classification results, we illustrate two cases on the DFN from Figure 1.1. In Figure 3.4a, we visualize the result of our model with the highest recall and lowest precision, SVM(0.90,0.054). Here, most of the nodes in the particle backbone are classified as positive. The few false negatives (FN) are near the source, and are primarily fractures intersecting the source plane where high particle concentrations accumulate. False positives (FP) are far more prevalent, forming many connected source-to-sink paths that are not in the particle backbone. In spite of these, the reduced network identified by the classifier contains only 40% of the original fractures.

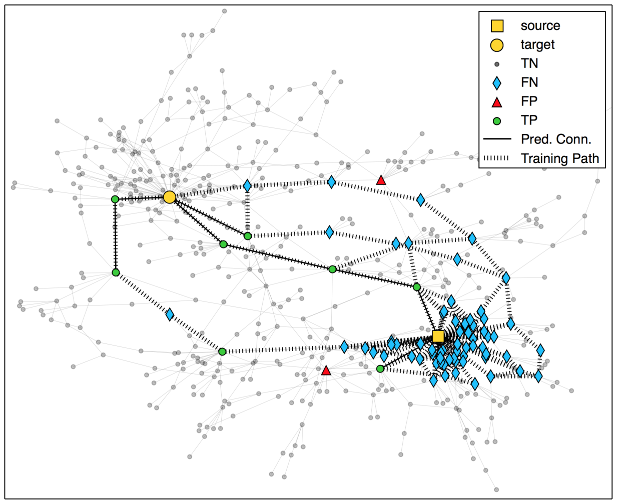
In Figure 3.4b, we visualize the result of our model with the highest precision and lowest recall, RF(1). We see almost no false positives (FP), but most of the nodes in the particle backbone are missed. While the false negatives (FN) near the source are not necessarily of great concern, as these simply represent the inlet plane, it is notable that only one connected path exists between source and sink. On other networks in the test set, the classifier does not even generate a connected source-to-sink path at all, leading to the same disconnected backbone problem that occurs in current-flow thresholding. The drastic reduction of network size, to 2% of the original fractures, results in too much loss of physical relevance.

Considering these two extreme cases is instructive, but the more meaningful results lie in between. Figure 3.5 shows breakthrough curves on this network for a representative sample of four of our classifiers, along with the curve for both the full network and the particle backbone. Broadly speaking, high recall results in good breakthrough curve agreement, whereas high precision results in good network reduction. Two of the four models (RF(30) and the high recall SVM(0.90,0.054) seen above) correctly predict the peak of the curve, and both match the tail closely. This suggests that both yield reduced networks with flow properties close to those of the original network. The reduced network resulting from RF(30) is 21% of its original size.

Finally, in order to quantify the tradeoff between breakthrough curve agreement and network reduction, we calculate the Kolmogorov-Smirnov (KS) statistic, giving a measure of “distance” between two probability distributions. The KS statistic is independent of binning, and most sensitive to discrepancies close to the medians of the distributions, making it particularly suitable for comparing peaks of breakthrough curves. The results are summarized in Table 3.3. They confirm that the model with highest recall, SVM(0.90,0.054), which reduces the network to 40% of its original size, has a breakthrough curve close to that of the full network (KS statistic 0.10).

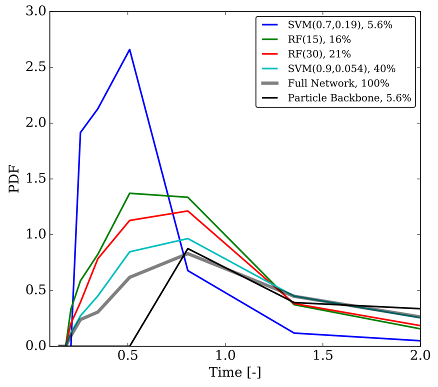


(a)



(b)

Figure 3.4: Extreme cases of classification results: (a) SVM(0.90,0.054) with high recall and low precision (40% of network remaining), showing many false positives (FP) and relatively few false negatives (FN), and (b) RF(1) with high precision and low recall (2% of network remaining), showing many false negatives (FN) and relatively few false positives (FP). Solid lines show predicted connected paths from source to sink. Dashed lines show particle backbone.



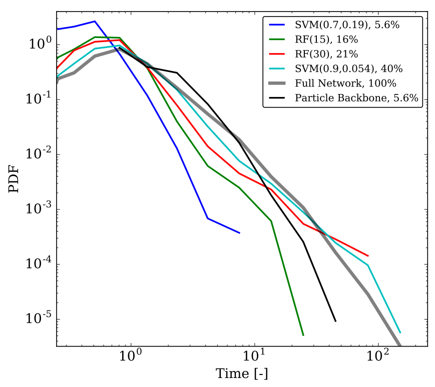


Figure 3.5: Predictions for the DFN from Figure 1.1, visualized as breakthrough curves produced by DFNWORKS. Tail plot on right is displayed on log-log axes for clarity. Representative results from four models are given, together with full network and particle backbone. Legend shows model parameters and remaining fractures in network. The last two classifiers provide good agreement while reducing network to 21% and 40% of original size.

Model	Fractures remaining	KS
SVM(0.90,0.054)	40%	0.10
RF(1400)	38%	0.12
SVM(0.90,0.063)	35%	0.12
SVM(0.70,0.070)	22%	0.25
RF(30)	21%	0.26
RF(15)	16%	0.35
SVM(0.70,0.190)	5.6%	0.59
RF(1)	2.0%	0.68

Table 3.3: Results of applying four RF and four SVM models to the DFN from Figure 1.1. Fractures remaining in network are those identified as backbone by classifier. Values differ slightly from results over entire test set, due to graph-to-graph fluctuations. KS statistic represents distance between breakthrough curve on reduced network and on full network. Note that final model RF(1) does not yield valid breakthrough curve on all graphs in test set, as it sometimes disconnects source from sink.

3.5 Two-Stage Method

Results from the two-stage method are still preliminary. We have not included these as part of our validation, and save a more complete study of the method for future work. In our preliminary tests, we implemented the algorithm using random forest for both stages, although one could also use SVM for either stage.

For the first stage, we used the identical RF parameters as in Section 3.2, but with 500 trees (*n_estimators=500*). For the second stage, class imbalance is practically not an issue, since we are classifying neighbors of backbone nodes. Here, we again used 500 trees (*n_estimators=500*), and set all other parameters to their default values.

Table 3.4 gives the results from the two-stage classifiers generated with two different values of *min_samples_leaf* in the first stage, to control precision and recall. We note that the method is able to achieve, simultaneously, high recall and sufficient precision for significant graph reduction. Our preliminary results suggest that this algorithm may provide a large improvement over the results in Sections 3.2 and 3.3. Research is ongoing, and fully exploring the potential of the two-stage method remains an open challenge.

Model	Precision	Recall	Fractures remaining
Two-Stage(1400)	37%	90%	17%
Two-Stage(1)	45%	82%	13%

Table 3.4: Two-Stage method using random forest in both stages, labeled by the value of *min_samples_leaf* used in the first stage. Percentages for precision, recall, and fractures remaining in network are calculated over all 20 graphs in the test set.

3.6 Dynamic Graph

The results of our dynamic graph simulation are shown in Figure 3.6. Initial ellipse sizes depend on the (random) orientations of the starting seeds, but the ellipses themselves are all oriented perpendicular to the stress field. We place a source node outside of the left boundary, connecting it to all ellipses crossing that boundary, and similarly with a sink node outside of the right boundary. Material failure occurs when a path first appears from source to sink.

These simulation results, while preliminary, motivate a possible improvement to the dynamic graph model. First of all, we see a number of clique-like

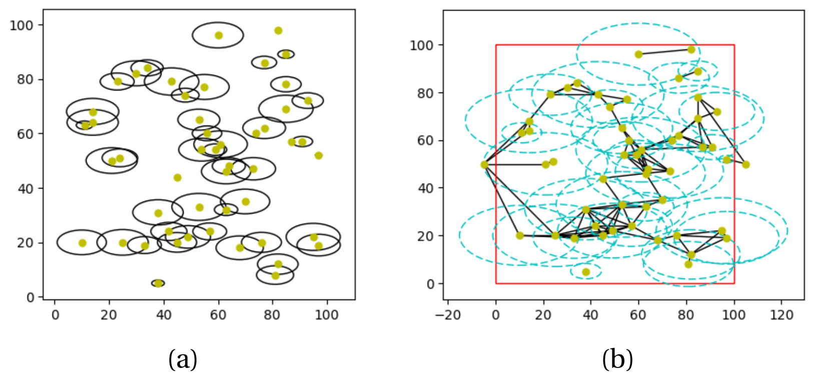


Figure 3.6: Results of dynamic graph simulation using growing ellipses. Major axes are oriented horizontally and are twice the length of minor axes. (a) Initial ellipses are placed randomly, and are of different sizes, depending on random orientations of associated crack seed. (b) Source node on left connects to all ellipses crossing left boundary; sink node on right connects to all ellipses crossing right boundary. Failure plane develops when path is created from source to sink.

structures in Figure 3.6b, with edge crossings and highly connected subgraphs. Fracture networks with this topology are unlikely to have straightforward physical interpretations. Second of all, the graph has very few terminal nodes, unlike the structure that we see in, for instance, the DFN from Figure 1.1. In order to solve these problems while better modeling the underlying physics of crack propagation, one might modify the dynamic graph model in the following way [52]. Make all edges directed, always oriented from the larger ellipse to the smaller ellipse. The direction of the edge therefore encodes the dynamics of which fracture propagates into which other fracture. Furthermore, since a fracture usually only grows in two direction, limit the number of outgoing directed edges from a node to at most two. If two edges already start from a given fracture, no further directed edges can come out from it, although new directed edges can come into it.

Continuing the dynamic graph modeling and implementing a classification approach to predicting the failure path remain to be explored as future problems.

Chapter 4

Conclusions

Simulating flow through fractured media is a major computational challenge. The use of discrete fracture networks (DFNs) allows flow to be modeled by particle tracking. Furthermore, one can realize significant computational savings by identifying a primary flow subnetwork within a DFN, and restricting simulation to that backbone subnetwork. However, identifying the backbone by means of particle simulations is itself too computationally intensive to be practical for large networks.

In this clinic project, we have presented a novel approach to finding a backbone subnetwork that does not require resolving flow in the network, and that takes minimal computational time. The method involves representing a DFN with an underlying graph whose nodes represent fractures, and applying machine learning techniques to rapidly predict which nodes are part of the backbone. We have used two supervised learning techniques: random forest and support vector machines. Once these algorithms have been trained on flow data from particle simulations, they successfully reduce new DFNs to subnetworks that preserve crucial flow properties. Our algorithms use topological features associated with nodes on the graph, as well as a small number of physical features describing a fracture's properties. We consider each node as a point in the multi-dimensional feature space, and classify it according to whether or not it belongs to the backbone.

By varying the parameters of our classifiers, we are able to obtain a wide range of precision and recall values. These yield backbones whose sizes range from 40% down to 2% of the original network. For reductions as small as 21% of the original size, the resulting breakthrough curve (BTC) displays good agreement with that of the original network. We therefore obtain subnetworks that are significantly smaller than the full network, useful for flow simulations, and generated in seconds. By comparison, the computation time needed to extract the particle backbone is on the order of hours.

In addition to the classification results, the random forest method gives a set of relative importances for the features used. These importances are determined by permuting the values of a given feature and observing the effect this has on classification performance. We have found that features based on global topological properties of the underlying graph were significantly more important than those based on geometry or physical properties of the fractures. This reinforces previous observations that network connectivity is more fundamental to determining flow than are geometric or hydraulic properties [19]. Quantitatively, the most important of our global topological features is source-to-sink current flow, which measures how much of a unit of current injected at the source (representing the inlet plane of the DFN) passes through a given node of the graph.

Some evidence suggests that if one could in fact generate backbone paths rather than backbone nodes, results would improve further. We have developed a two-stage classification method that initially labels fractures at the inlet and outlet planes, and then successively attempts to propagate fractures labeled as backbone through the network, thereby forming source-to-sink paths. The objective of this method is to generate subnetworks that are far closer to the particle backbone itself, with the training data used not merely to guide the classifier toward useful network reductions, but rather in the more conventional machine learning setting of providing ground truth to be reproduced. Our preliminary results suggest that such a method may considerably boost precision and recall simultaneously, generating subnetworks whose BTC closely matches the full network but whose size is not much larger than the particle backbone.

Finally, we have performed a preliminary study of the problem of predicting fracture propagation under a dynamic graph model. We implemented this using a basic random geometric graph with growing ellipses. As these ellipses grow and increasingly overlap with each other, edges get added to the graph. Once a path is generated through the graph, from a source node to a sink node, material failure is considered to have occurred. We hope that reduced models of this kind can help motivate classification methods that predict the failure path purely from initial geometry, without the need for simulations. This challenge remains open.

Appendix A

Code Description

All code developed for the 2016–17 Clinic has been placed on the project GitHub site. The code is written in Python, and is primarily organized into jupyter notebooks.

The following information is reproduced from the readme file for the code repository on GitHub.

A.1 Dependencies

Math clinic scripts require:

- *Python 2.7*
- *networkx 1.11*
- *numpy 1.12.1*
- *scikit-learn 0.18.1*
- *matplotlib 2.0.0*

All tests were performed using those versions, but older or newer version of the packages might still work.

A.2 Quick start

1. Put the files backbone.txt, connectivity.dat, source.txt, target.txt and volume.txt in transfer/x{i}/ with a different value of {i} for each network.

2. Put the file permeability.dat in perms/x{i}/ with a different value of {i} for each network.
3. Put the file {i}vol_proj.txt in vol_proj/ with a different value of {i} for each network.
4. Run generate_features.ipynb. Make sure to run the last cell twice, changing the value of "norm_rank", once for the raw features, and once for the ranked features.
5. Run main_classification_script.ipynb. Make sure to adjust the training/testing proportion in part 2, and the output folder and model in part 10.

A.3 Files

A.3.1 main_classification_script.ipynb

Script that loads all features, trains models on them and produces the confusion matrices and figures.

A.3.2 clinic_functions.py

Function to read and use the features.

A.3.3 generate_features.ipynb

Script to generate all features from the networks files.

A.3.4 PR_plots.ipynb

Generates the precision recall curves.

A.3.5 dynamic_graph.ipynb

Some code aimed to address the dynamic case.

A.3.6 model_selection.ipynb

Contains several model selection tests.

A.3.7 two_stage.ipynb

Two stage method that uses machine learning on nodes for the first stage and on edges for the second.

A.3.8 2-D_plot_features.ipynb

Generates a 2D plot of features from all graphs

A.3.9 3-D_plot_features.ipynb

Generates a 3D plot of features from all graphs

A.4 Folders

A.4.1 features/

All features for the two_core, current_flow_core and full_graph

A.4.2 norm_rank_features

All ranked features normalized by the number of nodes in its corresponding network.

A.4.3 final_results/

Results for all selected models.

A.4.4 transfer/

Raw data on the 100 networks.

A.4.5 figures/

Compendium of all figures done by model selection tests.

A.4.6 graphpositions/

Pickel objects that store the positions of nodes for consistent visualization.

A.4.7 perms/

Raw permeabilities. Needed to compute some features.

A.4.8 vol_proj/

Volume projection data.

A.4.9 corr_plots/

Correlation plots for BB-NB sets

A.4.10 archive/

Folder that contains old code. For archival purposes.

Bibliography

[1] M. C. Cacas, E. Ledoux, G. de Marsily, B. Tillie, A. Barbreau, E. Durand, B. Feuga, and P. Peaudecerf, "Modeling fracture flow with a stochastic discrete fracture network: calibration and validation: 1. the flow model," *Water Resources Research*, vol. 26, no. 3, pp. 479–489, 1990. [Online]. Available: http://dx.doi.org/10.1029/WR026i003p00479

[2] National Research Council, *Rock fractures and fluid flow: contemporary understanding and applications*, U. C. on Fracture Characterization and F. Flow, Eds. National Academy Press, 1996.

[3] S. Karra, N. Makedonska, H. Viswanathan, S. Painter, and J. Hyman, "Effect of advective flow in fractures and matrix diffusion on natural gas production," *Water Resources Research*, vol. 51, pp. 1–12, 2014.

[4] J. Hyman, J. Jiménez-Martínez, H. Viswanathan, J. Carey, M. Porter, E. Rougier, S. Karra, Q. Kang, L. Frash, L. Chen, Z. Lei, D. O'Malley, and N. Makedonska, "Understanding hydraulic fracturing: a multi-scale problem," *Phil. Trans. R. Soc. A*, vol. 374, no. 2078, p. 20150426, 2016.

[5] C. Jenkins, A. Chadwick, and S. D. Hovorka, "The state of the art in monitoring and verification—ten years on," *Int. J. Greenh. Gas. Con.*, vol. 40, pp. 312–349, 2015.

[6] J. D. Hyman, S. Karra, N. Makedonska, C. W. Gable, S. L. Painter, and H. S. Viswanathan, "dfnWorks: A discrete fracture network framework for modeling subsurface flow and transport," *Computers and Geosciences*, vol. 84, pp. 10–19, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.cageo.2015.08.001

[7] S. Painter and V. Cvetkovic, "Upscaling discrete fracture network simulations: An alternative to continuum transport models," *Water Resour. Res.*, vol. 41, p. W02002, 2005.

[8] S. Painter, V. Cvetkovic, and J.-O. Selroos, "Power-law velocity distributions in fracture networks: Numerical evidence and implications for tracer transport," *Geophys. Res. Lett.*, vol. 29, no. 14, 2002.

[9] H. Abelin, I. Neretnieks, S. Tunbrant, and L. Moreno, *Final report of the migration in a single fracture: experimental results and evaluation.* Swedish Nuclear Fuel and Waste Management Company, 1985.

[10] H. Abelin, L. Birgersson, L. Moreno, H. Widén, T. Ågren, and I. Neretnieks, "A large-scale flow and tracer experiment in granite: 2. results and interpretation," *Water Resour. Res.*, vol. 27, no. 12, pp. 3119–3135, 1991.

[11] A. Rasmuson and I. Neretnieks, "Radionuclide transport in fast channels in crystalline rock," *Water Resour. Res.*, vol. 22, no. 8, pp. 1247–1256, 1986.

[12] J.-R. de Dreuzy, Y. Méheust, and G. Pichot, "Influence of fracture scale heterogeneity on the flow properties of three-dimensional discrete fracture networks," *J. Geophys. Res.-Sol. Ea.*, vol. 117, no. B11, 2012.

[13] A. Frampton and V. Cvetkovic, "Numerical and analytical modeling of advective travel times in realistic three-dimensional fracture networks," *Water Resour. Res.*, vol. 47, no. 2, 2011.

[14] J. D. Hyman, S. L. Painter, H. Viswanathan, N. Makedonska, and S. Karra, "Influence of injection mode on transport properties in kilometer-scale three-dimensional discrete fracture networks," *Water Resour. Res.*, vol. 51, no. 9, pp. 7289–7308, 2015.

[15] G. Aldrich, J. Hyman, S. Karra, C. Gable, N. Makedonska, H. Viswanathan, J. Woodring, and B. Hamann, "Analysis and visualization of discrete fracture networks using a flow topology graph," *IEEE T. Vis. Comput. Gr.*, 2016.

[16] J. Maillot, P. Davy, R. Le Goc, C. Darcel, and J.-R. De Dreuzy, "Connectivity, permeability, and channeling in randomly distributed and kinematically defined discrete fracture network models," *Water Resources Research*, vol. 52, no. 11, pp. 8526–8545, 2016.

[17] M. Valera, Z. Guo, P. Kelly, S. Matz, A. Cantu, A. G. Percus, J. D. Hyman, G. Srinivasan, and H. S. Viswanathan, "Machine learning for graph-based representations of three-dimensional discrete fracture networks," 2017, submitted.

[18] E. Rougier, E. E. Knight, S. T. Broome, A. J. Sussman, and A. Munjiza, "Validation of a three-dimensional finite-discrete element method using experimental results of the split Hopkinson pressure bar test," *International Journal of Rock Mechanics and Mining Sciences*, vol. 70, pp. 101–108, 2014.

[19] J. D. Hyman, G. Aldrich, H. Viswanathan, N. Makedonska, and S. Karra, "Fracture size and transmissivity correlations: Implications for transport simulations in sparse three-dimensional discrete fracture networks following a truncated power law distribution of fracture size," *Water Resources Research*, vol. 52, no. 8, pp. 6472–6489, 2016. [Online]. Available: http://dx.doi.org/10.1002/2016WR018806

[20] J. N. Goetz, A. Brenning, H. Petschko, and P. Leopold, "Evaluating machine learning and statistical prediction techniques for landslide susceptibility modeling," *Computers and Geosciences*, vol. 81, pp. 1–11, August 2015.

[21] E. Santiago, J. X. Velasco-Hernández, and M. Romero-Salcedo, "A methodology for the characterization of flow conductivity through the identification of communities in samples of fractured rocks," *Expert Systems With Applications*, vol. 41, no. 3, pp. 811–820, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2013.08.011

[22] M. K. Mudunuru, S. Karra, N. Makedonska, and T. Chen, "Joint geophysical and flow inversion to characterize fracture networks in subsurface systems," *arXiv e-prints*, no. 1606.04464, Jun. 2016.

[23] H. O. Ghaffari, M. H. B. Nasseri, and R. P. Young, "Fluid flow complexity in fracture networks: analysis with graph theory and LBM," *arXiv e-prints*, no. 1107.4918, Jul. 2011.

[24] C. A. Andresen, A. Hansen, R. Le Goc, P. Davy, and S. M. Hope, "Topology of fracture networks," *Frontiers in Physics*, vol. 1, no. August, pp. 1–5, 2013. [Online]. Available: http://www.frontiersin.org/Journal/10.3389/fphy.2013.00007/full

[25] J. N. Vevatne, E. Rimstad, S. M. Hope, R. Korsnes, and A. Hansen, "Fracture networks in sea ice," *Frontiers in Physics*, vol. 2, pp. 1–8, April 2014.

[26] S. M. Hope, P. Davy, J. Maillot, R. Le Goc, and A. Hansen, "Topological impact of constrained fracture growth," *Frontiers in Physics*, vol. 3, no. September, pp. 1–10, 2015. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fphy.2015.00075

[27] E. Santiago, M. Romero-Salcedo, J. X. Velasco-Hernández, L. G. Velasquillo, and J. A. Hernández, “An integrated strategy for analyzing flow conductivity of fractures in a naturally fractured reservoir using a complex network metric,” in *Advances in Computational Intelligence: 11th Mexican International Conference on Artificial Intelligence, MICAI 2012, San Luis Potosí, Mexico, October 27 – November 4, 2012. Revised Selected Papers, Part II*, I. Batyrshin and M. G. Mendoza, Eds. Berlin, Heidelberg: Springer, 2013, pp. 350–361. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37798-3_31

[28] E. Santiago, J. X. Velasco-Hernandez, and M. Romero-Salcedo, “A descriptive study of fracture networks in rocks using complex network metrics,” *Computers and Geosciences*, vol. 88, pp. 97–114, 2016.

[29] A. Liaw and M. Wiener, “Classification and regression by randomForest,” *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[30] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning (2nd edition)*. Springer, 2008.

[31] P. Guerts, D. Ernst, and L. Wehenkel, “Extremely randomized trees,” *Machine Learning*, vol. 63, pp. 3–42, 2006.

[32] A. Prinzie and D. Van den Poel, “Random forests for multiclass classification: random multinomial logit,” *Expert Systems with Applications*, vol. 34, no. 3, pp. 1721–1732, 2008.

[33] T. Dietterich, “An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization,” *Machine Learning*, pp. 139–157, 2000.

[34] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*. Springer, 2013.

[35] T. K. Ho, “A data complexity analysis of comparative advantages of decision forest constructors,” *Pattern Analysis and Applications*, pp. 102–112, 2002.

[36] V. Vapnik and A. Lerner, “Pattern recognition using generalized portrait method,” *Automation and Remote Control*, vol. 24, pp. 774–780, 1963.

[37] V. Vapnik and A. Y. Chervonenkis, “Theory of pattern recognition: statistical problems of learning [in russian],” 1974.

[38] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[39] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," *Proceedings of the fifth annual workshop on Computational learning theory COLT'92*, p. 144, 1992.

[40] A. Ben-Hur, D. Horn, H. Siegelmann, and V. Vapnik, "Support vector clustering," *Journal of Machine Learning Research*, vol. 2, pp. 125–137, 2001.

[41] LDRD-DR, "Dynamic graph representation of fracture propagation processes: basic principle," October 13, 2016, presentation slides.

[42] E. Estrada and M. Sheerin, "Random neighborhood graphs as models of fracture networks on rocks: structural and dynamical analysis," *arXiv e-prints*, no. 1607.06678, 2017.

[43] A. A. Hagberg, D. A. Schult, and P. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conferences (SciPy 2008)*, vol. 2008, 2008, pp. 11–16.

[44] J. Anthonisse, "The rush in a directed graph," *Stichting Mathematisch Centrum. Mathematische Besliskunde*, no. BN 9/71, pp. 1–10, jan 1971. [Online]. Available: https://www.narcis.nl/publication/RecordID/oai:cwi.nl:9791

[45] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977.

[46] U. Brandes and D. Fleischer, "Centrality measures based on current flow," *Proc. 22nd Symp. Theoretical Aspects of Computer Science*, vol. 3404, pp. 533–544, 2005.

[47] M. Newman, "A measure of betweenness centrality based on random walks," *Social Networks*, pp. 39–54, 2005.

[48] S. Russell, *Artificial intelligence : a modern approach.* Upper Saddle River, NJ: Prentice Hall, 2010.

[49] T. K. Ho, "Random decision forests," *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, vol. 14-16, pp. 278–282, August 1995.

[50] ——, “The random subspace method for constructing decision forests,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[51] G. Aldrich, J. D. Hyman, S. Karra, C. W. Gable, N. Makedonska, H. Viswanathan, J. Woodring, and B. Hamann, “Analysis and visualization of discrete fracture networks using a flow topology graph,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. X, pp. 1–15, 2016.

[52] D. O’Malley, personal communication, 2017.