

Title: Choosing Branch Based or Branch Avoiding Techniques in Triangle Counting Algorithms for Applications to Social Network Graphs

Problem Statement

Graphs are a natural representation for modeling relationships between entities, whether in web traffic, financial transactions, computer networks, or society. For basic terminology, let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E the set of undirected edges with $n = |V|$ and $m = |E|$, the number of vertices and edges respectively. Define an edge $e = (u, v)$ connecting vertices u and v . Graphs can be used to model relationships between entities, where the vertices v in V can be thought of as players or users, and the edges e in E represent the relationships between these players. Assume the graph G is given in adjacency list format, i.e., for every vertex v , we have an array of its neighbors, denoted as "list(v)". Note that therefore $\text{degree}(v) = \text{size}(\text{list}(v))$. Since we are dealing with undirected graphs, each edge $e = (u, v)$ in the graph will appear twice in the adjacency list representation, once in list(u) and once in list(v).

Specifically, we focus on those graphs modeling social networks. Much research of social networks focuses on identifying important players in a graph, using some centrality metric. One such metric is clustering coefficients, which is useful for finding key players in a network based on their local connectivity. This property is calculated using the number of triangles a vertex belongs to, making triangle counting an essential computation for social network analysis.

A very basic algorithm for triangle counting utilizes list intersections. Suppose we are given an edge $e = (u, v)$. If vertices u and v have any common neighbors (denote the common neighbor as w), they will be found using a list intersection of list(u) and list(v). The triangle is then the one formed by vertices u , v , and w . We are presented with two algorithms for computing list intersections: a branch-based approach, and a branch-avoiding approach (as we will explain further below). The algorithm for triangle counting is then to iterate through every edge in the adjacency list, compute list intersection for the two vertices in question, and increase the triangle count for any vertex found in the intersection. Therefore, we will need to compute $2m$ list intersections, one for each edge as it appears in the adjacency list (note again the factor of 2 because each edge appears twice).

In many algorithms dealing with massive amounts of data, branch prediction is an important consideration when concerned with performance of these algorithms. "Branching" in an algorithm refers to the algorithm making a choice to do one of two or more things; the simplest and most common programming example is the "if" statement. From a performance perspective, the presence of a conditional

branch in an algorithm will interrupt the flow of instructions; if the algorithm does not know beforehand which branch to take, the processor will not know which instruction to fetch next, which will stall the pipeline, impeding fast runtimes. Therefore, previous research has explored branch-avoiding algorithms as alternates to a typically branch-based approach to many graph centric problems.

For our purposes, we are given two algorithms to calculate list intersections as mentioned above. For the scope of this project, we take the two algorithms as given and analyze the data resulting from execution times as explained below. Denote the branch-based approach as BB and the branch-avoiding approach as BA. Previous research shows that BB tends to require less total time than BA for list intersections, where “total” refers to the time for all $2m$ list intersections for a graph. However, BA can outperform BB on execution times for individual edges. This suggests that there may be a hybrid approach of both BA and BB that achieves an optimal runtime for counting triangles. This gives rise to the question that we seek to answer: for an edge in the graph, can we predict which approach to take (BA or BB) to calculate the list intersection of the two corresponding vertices of the edge, given some parameters of the edge? For example, for an edge $e = (u,v)$, possible parameters include $\text{size}(\text{list}(u))$ and $\text{size}(\text{list}(v))$. After compiling a list of these parameters for each edge, we wish to identify if we can predict which approach (BB or BA) will outperform the other and which approach would be ideal for a hybrid approach.

Data Source

We used undirected graphs from the DIMACS 10 Graph Challenge, which are widely used in social network analysis research. As a result of current research projects in the HPC lab in CSE, pre-existing code was used to generate this data. Since we assume the graph is in adjacency list format and each edge appears twice to represent bidirectional connection, we have execution times for both BA and BB for $2m$ edges, where m is the number of edges in the graph.

As an example, the data may look like the following:

Edge	Branch Avoiding (s)	Branch Based (s)
e1=(u1,v1)	0.00123	0.00456
e2=(u2,v2)	0.00223	0.00111
...
Total	0.0953	0.0843

We parsed this data and composed a file with the first and second elements in some i th row being the respective indices of two vertices, u_i and v_i , which are connected by the edge e_i in the graph. For the third and fourth entries in this row, we inserted $\text{degree}(u)$ and $\text{degree}(v)$, respectively. For the fifth and sixth elements we inserted the times for BB and BA to execute list intersection on the adjacency lists of u and v , respectively.

In some cases we see that the individual execution time for BA is less than the corresponding one for BB. However, the overall runtime of BB is faster than BA on all observed graphs.

As mentioned earlier in the Problem Statement section, we consider a set of features for each edge $e = (u,v)$; namely: (1) $\text{degree}(u)$, and (2) $\text{degree}(v)$. In our modified data, these correspond to the third and fourth columns, respectively.

Although we analyzed several other graphs with our classifiers, we chose to discuss two, which we will call "PGP" and "in2004." Although both graphs could be considered large, we opted to choose one graph which was small and one that was large by "big data" standards. PGP has 48,632 edges and 10,679 vertices. in2004 has 27,182,946 edges and 1,382,905 vertices.

We also wanted to discuss the impact of the frequency of BB outperforming BA, which can vary greatly from graph to graph, although on average, BB usually outperforms BA 70% of the time. PGP and in2004 also satisfied this variation. In PGP, BB is faster than BA for 22,024 edges (45% of all edges). In in2004, BB is faster than BA for 25,922,740 edges (95% of all edges). Although there are graphs with different combinations of size, BB/BA performances, and other characteristics, we leave further exploration to future work.

Methodology

Our methodology mainly consists of implementing various classifiers and evaluating which model best fits our data. Before training the data to get the model, we divide the data into two sets - the training set, and the test set. For each classifier, we analyze the fit based on various amounts of training data, from the first 20% of the data to the first 80% in increments of 10%.

To assess how well our classifiers fit our data, we decided to analyze the percentage accuracy of each classifier at predicting which of BB or BA is faster, and what percentage of time hybrid selection under each classifier saves over the pure BB approach. While we considered also analyzing standard statistical significance values, our initial experimentation revealed that often, predictors that have higher

accuracy rates or save more time than other predictors can also be less statistically significant. For simplicity, we decided to only analyze accuracy and time savings, and leave statistical significance for future work.

In our initial experimentation, we noticed that our original data files appear to be ordered in ascending order by the first column. In some cases, there exist patterns of BB outperforming BA or vice versa, which cause our training data to not be a representative sample. Therefore, we compare the performance of our classifiers on our files after first randomly sorting their rows.

We also analyze the effect of first sorting our data by the two parameters used, and then training on this new data as described previously. Again, these parameters, for some row concerning edge $e = (u,v)$, are $\text{degree}(u)$ and $\text{degree}(v)$. We will sort according to these parameters in six ways. Let "1asc" indicate sorting in ascending order by the third column in our data, which contains the degrees of the first vertices in the edges. Similarly, let "2asc" indicate sorting in ascending order by the fourth column in our data, which contains the degrees of the second vertices in the edges. Let "sum_asc" indicate sorting by the sums of the entries in each row's third and fourth columns. Let "1desc," "2desc," and "sum_desc" be defined in the same way, except in descending order.

Linear regression

Let y_1 be the execution time for BB and y_2 be the execution time for BA. Based on the features, $x_1 = \text{degree}(u)$ and $x_2 = \text{degree}(v)$ for each edge $e = (u,v)$, we can get the regression model

$$y_1 = a_0 + a_1*x_1 + a_2*x_2$$

$$y_2 = b_0 + b_1*x_1 + b_2*x_2$$

Then, if $y_1 > y_2$, we predict that BA is faster than BB; if $y_1 = y_2$, we predict that neither BB nor BA is faster, and choose BB as it is more widely used than BA; if $y_1 < y_2$, we predict that BB is faster than BA.

Logistic regression

Let "z = 0" represent that BA is faster than BB, and let "z = 1" represent that BB is faster than BA, and assume that z follows Bernoulli(p). Based on our two parameters for each edge, we use the logistic regression model to estimate $p = h(c_0 + c_1*x_1 + c_2*x_2 + \dots + c_p*x_p)$, where $h(x) = 1/(1+\exp(-x))$.

If $p < 0.5$, we predict that the probability that "z = 0" is greater than the probability that "z=1", and predict that BA is faster than BB. If $p=0.5$, we choose BB as it the more widely used than BA. Otherwise, we predict that BB is faster than BA.

LDA/QDA

Assume that the data are from two classes, "BB" and "BA". If some edge's BB time is less than its BA time, then it belongs to the "BB" class; otherwise, it belongs to the "BA" class. Each class follows the normal distribution. In LDA, the variances of the two normal distributions are the same; In QDA, the variances of the two normal distributions are different, which results in the difference of the decision boundaries for LDA and QDA. For LDA, the decision boundary is linear; For QDA, the decision boundary is quadratic.

LDA: $P(x|Class = BB) \sim N(\mu_{BB}, \Sigma)$

$P(x|Class = BA) \sim N(\mu_{BA}, \Sigma)$

QDA: $P(x|Class = BB) \sim N(\mu_{BB}, \Sigma_{BB})$

$P(x|Class = BA) \sim N(\mu_{BA}, \Sigma_{BA})$

By maximum likelihood estimation, we get the estimation for μ_{BB} , μ_{BA} , and Σ for LDA or (Σ_{BB} , Σ_{BA} for QDA), and then get the decision boundary for the data. Using the decision boundary, we get the classification for each edge. Note that there maybe an overfitting problem in QDA, which might result in poor performance for the training set, as the decision boundary of QDA is quadratic, which has more terms than that of LDA.

Evaluation and Results

Data

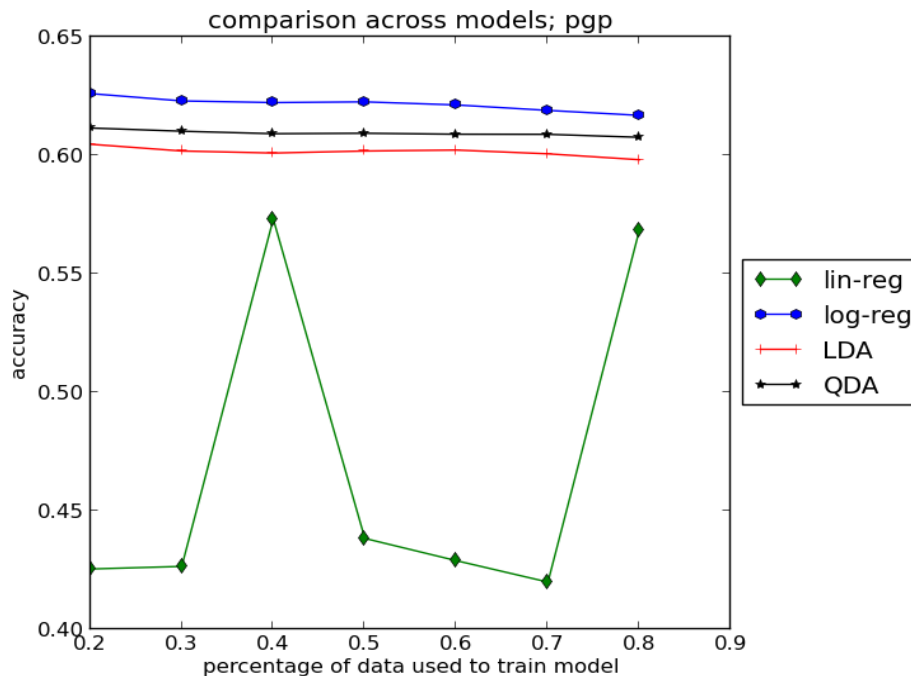
The following table shows the five graphs we ran our experiments on and the number of edges in each network.

	Edges
<i>hep-th</i>	31,502
<i>PGPgiantcompo</i>	48,632
<i>astro-ph</i>	242,502
<i>kron16</i>	4,912,142
<i>in2004</i>	27,182,946

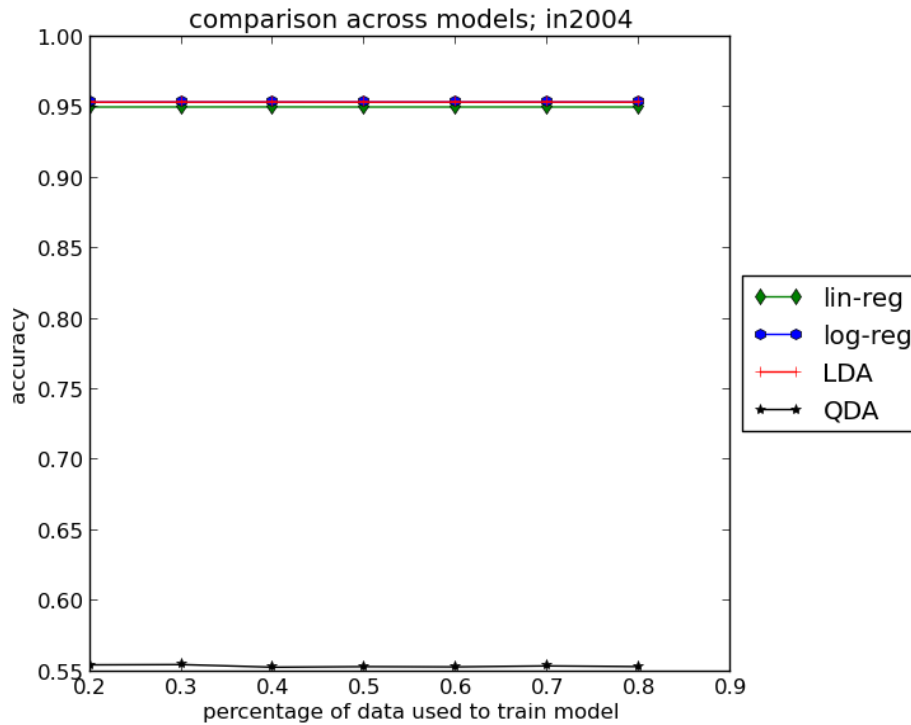
Plots and Tables

We first present a series of plots comparing various aspects of experiments conducted, such as randomizing input order for each graph and analyzing if sorted order of edges has any impact on overall accuracy. We also provide speedup calculations on the hybrid approach over a purely branch based approach. Note that we only present results for two specific graphs for brevity and to perform more in

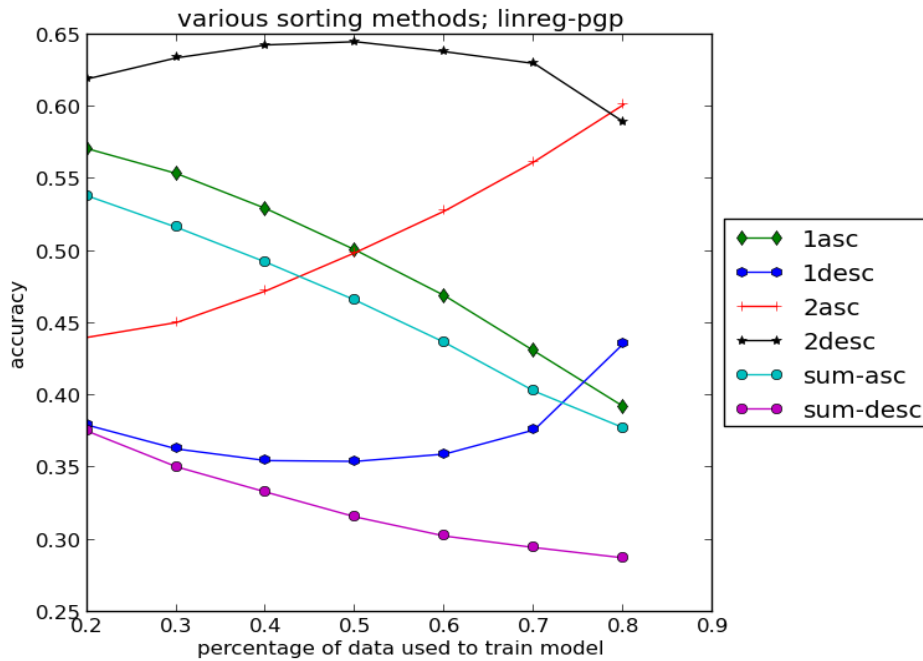
depth analyses of these two graphs, namely PGP and in2004, to show results on a medium sized network and a large sized network.



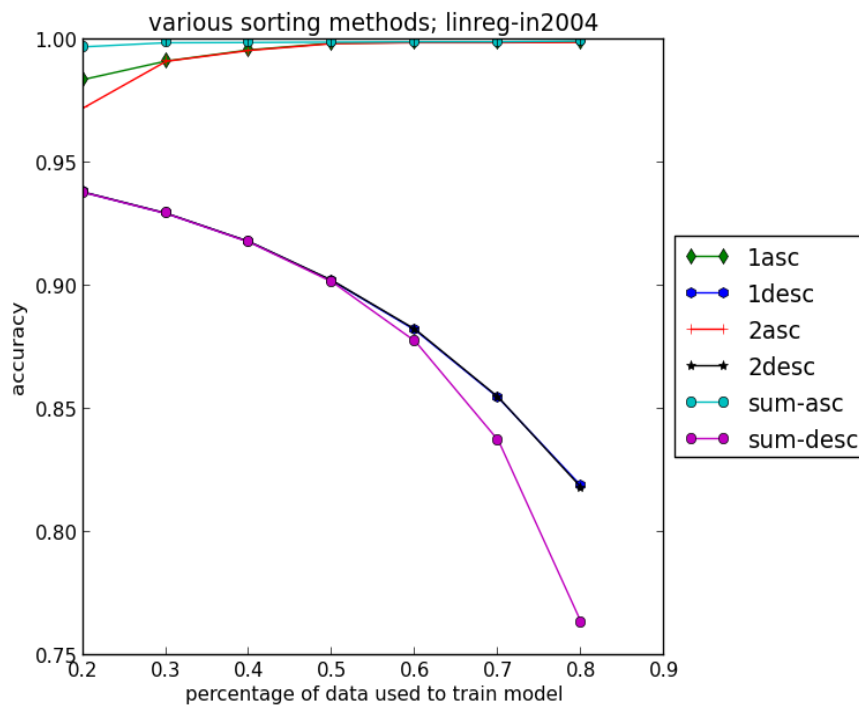
Above we see a comparison of all four models for the PGP graph, with accuracy of the model as a function of how much of the data was used as training data. In our randomized PGP file, we find that logistic regression, LDA, and QDA produce similar results, regardless of how much data used to train these models. From using the first 20% to 80% of data for training, each of these three classifiers chooses the fastest list intersection algorithm (BB or BA) with between 60% and 65% accuracy. Linear regression has much lower accuracy across training percentages, and this accuracy fluctuates dramatically compared to that of the other three classifiers.



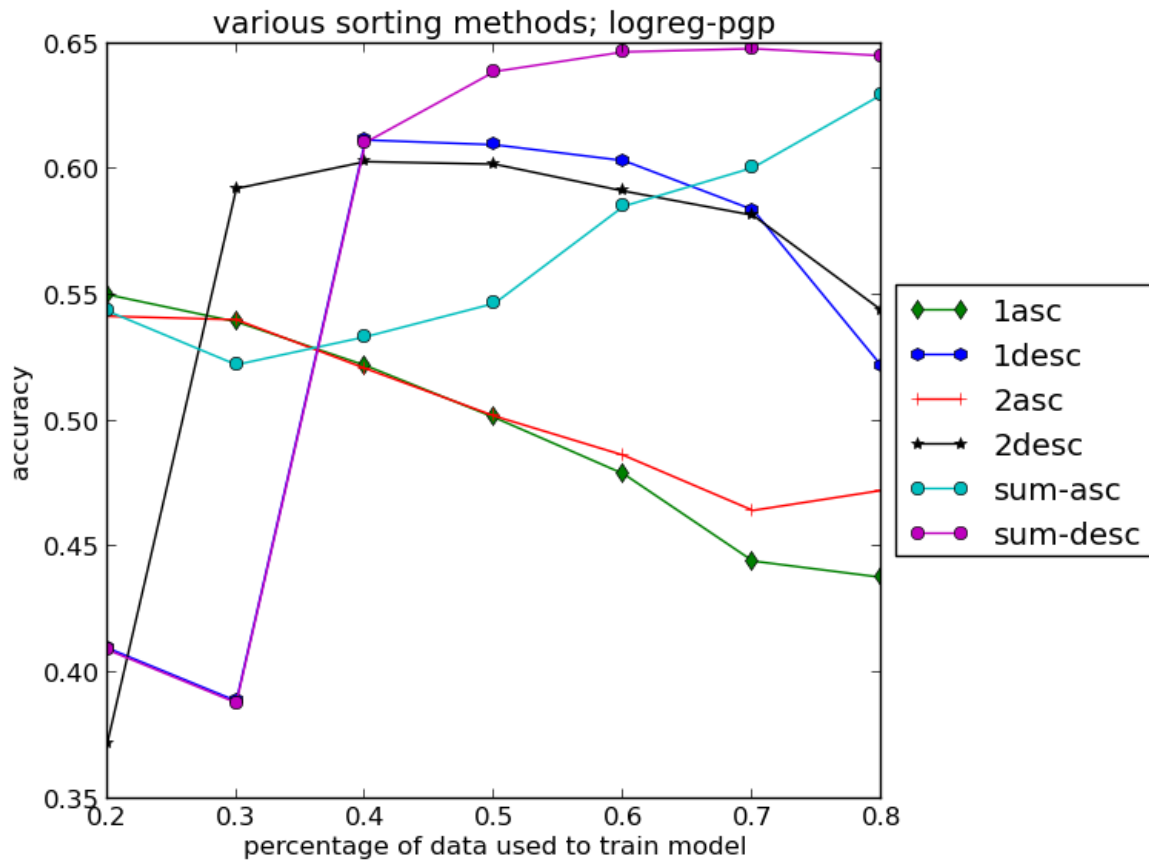
Above we see a comparison of all four models for the in2004 graph, with accuracy of the model as a function of how much of the data was used as training data. As we mentioned, the graph represented in in2004 is much larger than that in PGP. Also, BB is faster than BA for a much higher percentage of the edges in2004 than for those in PGP. Therefore, one would expect most of the results observed here, where our predictions are not only more consistent across training percentages, but also between classifiers. This effect is best observed by comparing the relative performance of our linear regression classifier between PGP and in2004. One surprising result is an apparent reduction in performance by QDA in in2004 and a disparity between its results and those of the other three classifiers. This reflects that there is an overfitting problem when applying QDA to the data.



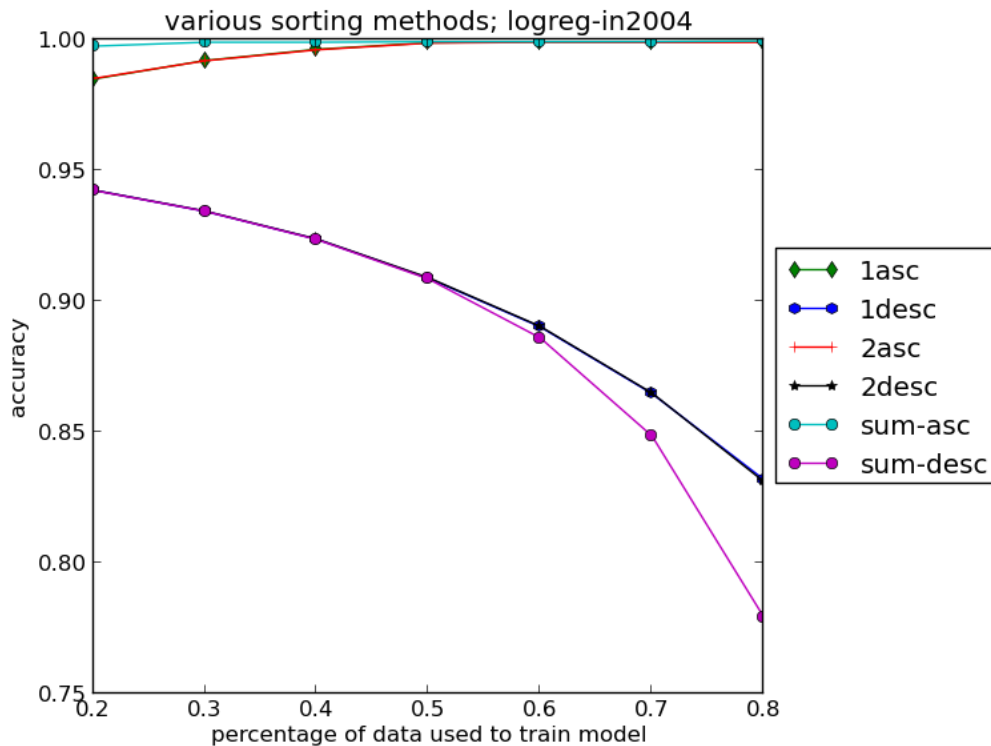
Above we see a comparison of different sorting methods for the PGP graph for linear regression, with accuracy of the model as a function of how much of the data was used as training data. We see that 1asc, sum-asc, and sum-desc decrease in accuracy as more training data is used whereas the other methods seem to increase in accuracy given more training data. Overall, the 2desc approach gives the best accuracy for linear regression on this particular graph, albeit nowhere as near as high as other models as explained further.



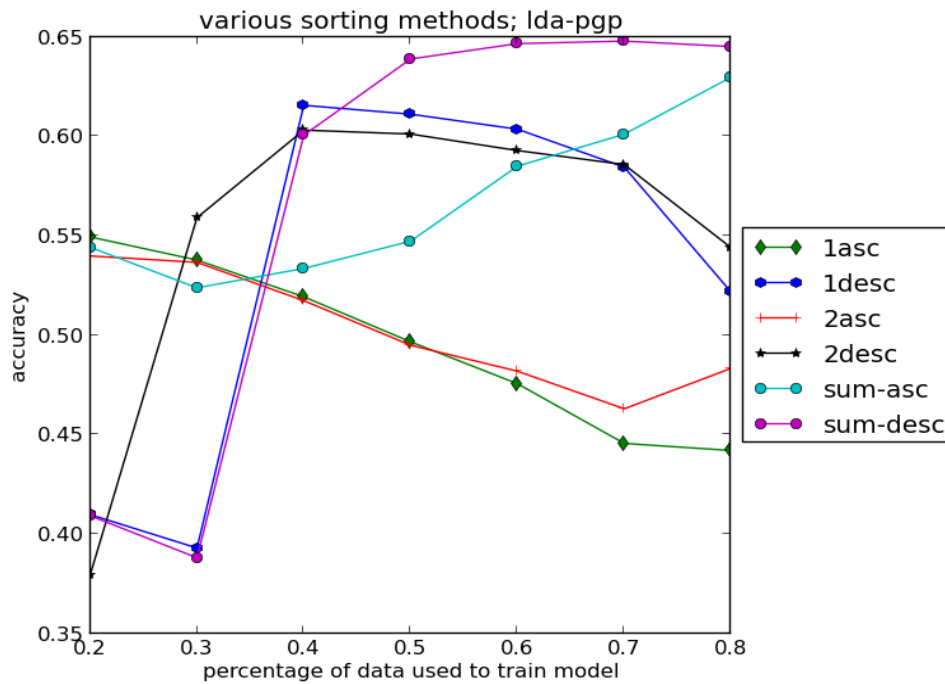
Above we see a comparison of different sorting methods for the in2004 graph for linear regression, with accuracy of the model as a function of how much of the data was used as training data. In this case, 2desc and sum-desc give decreasing accuracies as we increase the amount of training data while the other methods all get about 100% accuracy. Upon further investigation we see that linear regression has a tendency to pick the branch based approach every time regardless of the edge in question and over 95% of the time for this graph the branch based approach is faster, which is why the accuracy is so high but gives no real time speedup as we see later in the tables.



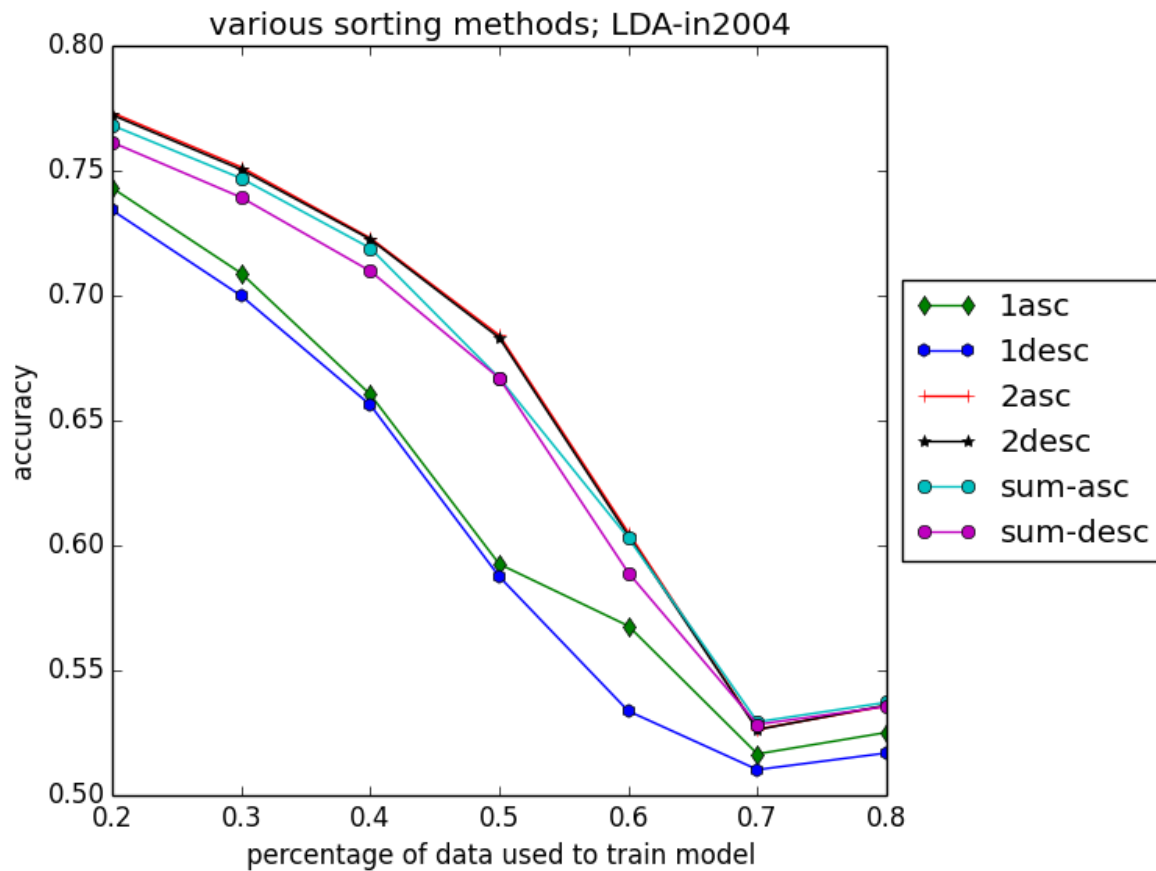
Above we see a comparison of different sorting methods for the PGP graph for logistic regression, with accuracy of the model as a function of how much of the data was used as training data. In the PGP graph, our logistic classifier seems to perform best in most sortings of the graph data when training on at least the first 40% of the data. While objective trends are difficult to extract from this data, it appears that sorting by sum yields better results than sorting by individual parameter, and that sorting in descending order tends to be better than sorting in ascending order. Certainly, it appears that sum-desc is the best sort for our logistic classifier in PGP. More interestingly, these results share a surprising similarity to those of LDA on PGP.



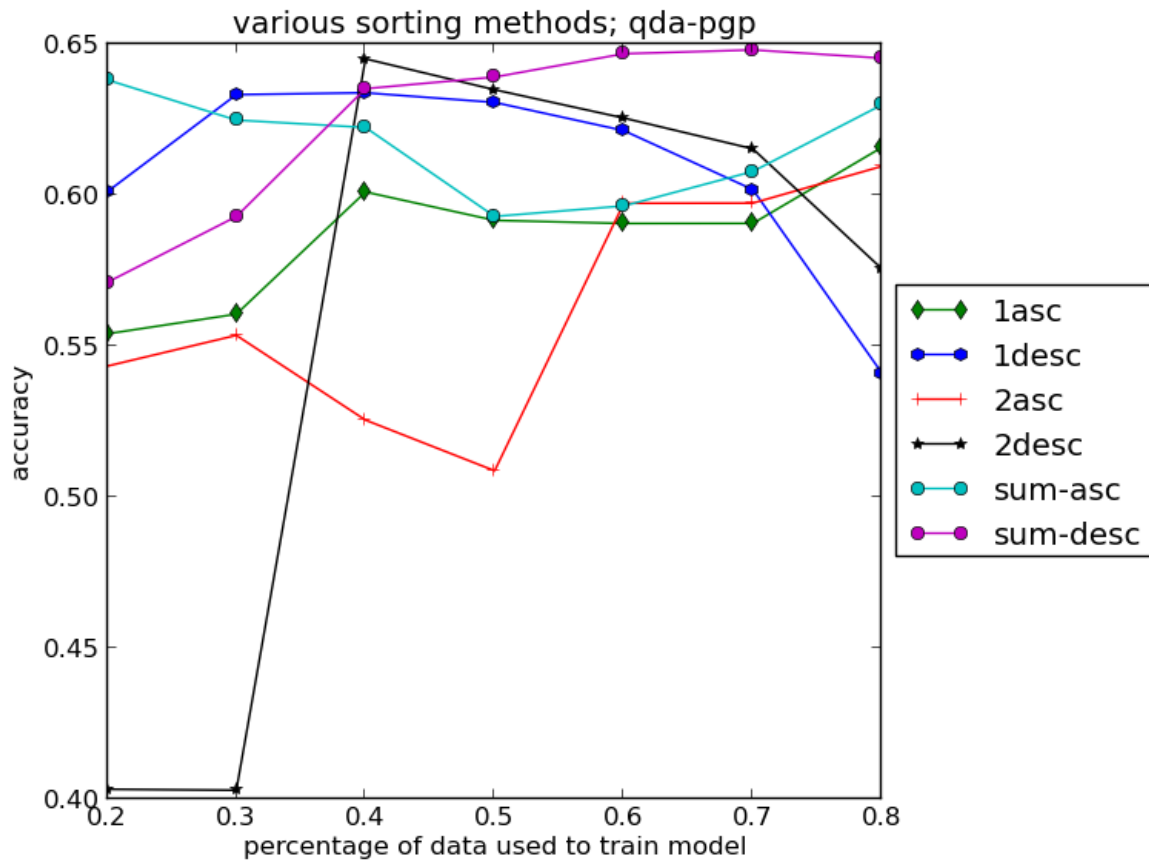
Above we see a comparison of different sorting methods for the in2004 graph, with accuracy of the model as a function of how much of the data was used as training data. In the in2004 graph, our logistic classifier performance seemed to have opposite results among our respective ascending and descending sortings. With increased training percentages, this classifier's performance among ascending sortings improved and appeared to converge, while its performance among descending sortings worsened and appeared to diverge. We did not expect such a consistent disparity. In fact, we expected an opposite result, wherein processing vertices with larger adjacency lists would provide our models with more information since these vertices would be more central in the graph than vertices with smaller adjacency lists. This led us to expect that descending sort orders would be more effective. This opposing result may be because the graph is mostly uniform, so central vertices may be outliers. However, we would then expect our model to improve with higher percentages of training data for descending-sorted data. The reason(s) for this disparity require further exploration.



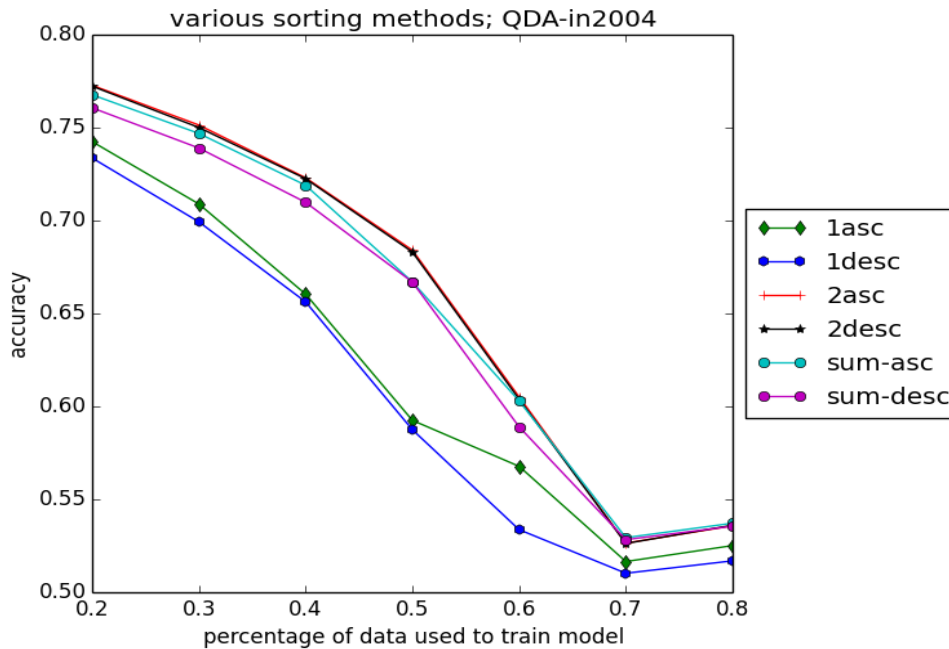
Above we see a comparison of different sorting methods for the PGP graph for LDA, with accuracy of the model as a function of how much of the data was used as training data. We can see that for LDA, 1asc and 2asc performs almost the same poorly. As the percentage of data used to train model increases, they performs even worse. 1desc and 2desc performs almost the same too. When the percentage is less than 40%, their accuracy is the worst among the six sorting orders, which is around 40%. But in 40%, there is a sharp increase of the accuracy. Sum-asc performs stably but not good as the sum-desc. Compared to other sorting orders, the sum-desc performs best when the percentage of data used to train model is larger than 50%, which is also better than the random sampling. So for pgp, the best way to implement LDA method is to use the sum-desc sorting data to train the model, and the best percentage for the training set is 60%.



Above we see a comparison of different sorting methods for the in2004 graph for LDA, with accuracy of the model as a function of how much of the data was used as training data. For all sorting methods we see a decrease in accuracy as we increase the amount of training data used to train the model. However, 2asc and sum-asc show the highest accuracy overall in this setting.



Above we see a comparison of different sorting methods for the PGP graph for QDA, with accuracy of the model as a function of how much of the data was used as training data. For QDA, the performance of the six sorting methods is much more steady than LDA. 2desc is the worst when the training set is less than 30%, but there is a sharp increase when the percentage comes to 40%. 1asc and sum-asc performs almost the same when the percentage is more than 50%. We can also see the same advantage of sum-desc as in LDA when percentage is bigger than 50%, which is also better than the random sampling way. Above all, for pgp, the best way to implement QDA method is to use the sum-desc sorting data to train the model, and the best percentage for the training set is 60%, which is the same as the conclusion for LDA.



Above we see a comparison of different sorting methods for the in2004 graph for QDA, with accuracy of the model as a function of how much of the data was used as training data. We see a similar behavior to LDA for in2004 wherein all sorting methods show a decrease in accuracy as amount of training data used increases, with the same conclusion of 2asc and sum-asc having the highest accuracy however.

	20%	30%	40%	50%	60%	70%	80%
Lin.Reg	.68	.67	.66	.66	.65	.65	.65
Log.Reg	.17	.18	.17	.18	.17	.17	.18
LDA	.18	.18	.18	.18	.17	.17	.18
QDA	.22	.18	.22	.22	.21	.22	.20

The table above shows time savings for each classifier for the randomized input PGP graph for various percentages of training data used to train each model respectively. Note we calculate time savings as $(BB_time - model_time) / (BB_time)$. We see that while linear regression gives the best hope in terms of time savings, recalling the earlier accuracy results show that it has the worst accuracy with respect to amount of training data used. On the other hand, logistic regression, LDA and QDA don't have as much time savings, but have the best

accuracy as earlier shown. In conclusion, taking into account the earlier accuracy results, we show an inverse relationship for the models in terms of accuracy and time savings with respect to amount training data used to train the model.

	20%	30%	40%	50%	60%	70%	80%
Lin.Reg	.00	.00	.00	.00	.00	.00	.00
Log.Reg	.00	.00	.00	.00	.00	.00	.00
LDA	.00	.00	.00	.00	.00	.00	.00
QDA	-.13	-.13	-.13	-.13	-.13	-.13	-.13

The table above shows time savings for each classifier for the randomized input in2004 graph for various percentages of training data used to train each model respectively. Unlike the PGP graph above, this much larger graph has almost no time savings and in fact takes longer with the hybrid approach according to our models (e.g. QDA). Upon further investigation, we see that linear and logistic regression chooses BB for all edges in in2004 across all training percentages. Since in this particular graph (in2004) we note that the edges are saturated with faster BB times for about 95% of the edges, it is not surprising that there is negligible time savings using the previously mentioned formula for these models.

Our results were more positive in PGP (we saved time) since neither BB nor BA was dominant, so a hybrid strategy should have advantage. In2004 gave us more consistent results, and a high accuracy. However, accuracy is a product of both classifier efficacy and the underlying graph. We believe that the graph contributed to most of our accuracy since we had very high accuracy in in2004 but no time savings. It seems that the saturation of BB being faster than BA across edges was so much that most of our classifiers just chose BB and had no advantage over the BB standard. All things being equal, we would expect % accuracy and % time savings to have a strong correlation, but different graph characteristics can derail this conclusion. Future work might include introducing new parameters that might help to deal with varying characteristics of graphs, so that % accuracy and % time savings align and more consistent classifier comparisons can be made.

Conclusions

We observe many varying results across graphs, training percentages, classifiers, and success criteria. Highlights include time savings in excess of 60% using linear regression in PGP, and roughly 95% accuracy using all classifiers except QDA in in2004. Lowlights include negligible or negative time savings for all classifiers in in2004 and only 55% accuracy using QDA in in2004. Although the inconsistencies

in our results make conclusions elusive, we are encouraged that with further exploration, means exist to make analyzing triangles in social network graphs more efficient.

One difference that we observed in the graphs that we analyzed was the relative dominance of BB and BA. When BA was faster than BB for a significant percentage of edges, as it was in PGP, our hybrid approach had more potential to save time over pure BB, but our models tended to have less predictive accuracy due to the increased variability of whether BB or BA was faster. When BA was faster than BB very infrequently, as it was in in2004, our hybrid approach had very little potential to save time over pure BB, but our models tended to have very high predictive accuracy. This went contrary to our initial expectations that such accuracy and time savings would share strong correlation.

We suspect that our models may have suffered in part from under-parametrization. Since accuracy is clearly a product of both classifier efficacy and the underlying graph, we suspect that parameters relating to graph structure may lessen our models' vulnerability to the relative dominance of BB or BA.

Further analysis of the correlation between the performance of BB and BA, the parameters that we used, and other graph characteristics may produce a hybrid approach that can save time over the standard BB approach, even in graphs such as in2004. In addition to work on parametrization, analysis of the statistical significance of these and other classifiers may improve results and inform which of these classifiers is optimal for certain graphs.