Automatic Tuning of the SkewedKruskal Algorithm

Mattia Lecchi and Giovanni Righini

Uncorrected Proof

1 Introduction

We consider the minimum cost spanning tree problem, that is, the problem of finding a minimum cost spanning tree (MST) of a connected weighted graph. A classical algorithm for this problem is due to Kruskal [4].

In its most basic implementation, Kruskal algorithm starts by fully sorting the edge list by nondecreasing edge weights. Then, starting from an empty set, a forest F is grown by scanning the edge list and inserting one edge at a time in F, discarding edges that produce cycles, until a spanning tree is obtained. The resulting time complexity is $O(m \log n)$, where m is the number of edges and n is the number of vertices.

Janson et al. [3] proved that the largest weight edge in an MST of a random weighted graph is expected to lie within the $\frac{1}{2}n \log n$ smallest weight edges. This suggests to save computing time, sorting only the relevant part of the edge list.

Following this idea, Paredes and Navarro [7] proposed the QuickKruskal algorithm, where edge sorting is interleaved with edge selection. For this purpose, edge sorting is done as in QuickSort, i.e., partitioning the unsorted edge list in two lists by comparing all weights with a suitably chosen pivot element; if the MST is found within the edges of the first list, then the edges in the second list are not sorted. This procedure is recursively executed for each list. From a theoretical viewpoint, the average time complexity of QuickKruskal is $O(m + n \log^2 n)$ for random graphs with randomly generated weights (Paredes and Navarro, [7]). From an experimental viewpoint, QuickKruskal is substantially faster than Kruskal algorithm.

M. Lecchi · G. Righini (✉)
Department of Computer Science, University of Milan, Milan, Italy
e-mail: mattia.lecchi@studenti.unimi.it; giovanni.righini@unimi.it

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

B. Goldengorin (ed.), *Theory, Algorithms and Experiments in Applied Optimization*, Springer Optimization and Its Applications 226,
https://doi.org/10.1007/978-3-031-91357-0_9

A further improvement, named FilterKruskal, was proposed by Osipov et al. [6]. When an edge list is recursively partitioned into two parts as in QuickKruskal, after processing the first list and before processing the second one, all edges that would close a cycle in the current forest are filtered out from the second list, thus reducing its size and the computing time needed to sort it. As shown in [6], the average time complexity of the FilterKruskal algorithm for random graphs with randomly generated edge weights is $O(m + n \log n \log \frac{m}{n})$.

In this context, the recursive partition of the edge lists should better be done in an unbalanced way, contrary to QuickSort that achieves the best performance by splitting each list into two parts of equal size. The unbalanced partition is especially useful in the first recursive call, when the whole edge list is partitioned the first time. Following this observation, Righini and Righini [8] presented a variation of the algorithm, called SkewedKruskal, where a random sample is extracted from the complete edge list and the pivot element is suitably chosen among the samples. The size of the sample is $\lceil \sqrt{m} \rceil$, and the pivot is selected in position $\lceil \frac{n \log n}{2\sqrt{m}} \rceil$ in the sorted list of samples. This corresponds to the expected position of the *critical edge*, i.e., the largest weight edge in the MST according to the result of Janson et al., which refers to complete graphs with random weights generated according to a uniform distribution of probability.

However, for different types of graphs, different choices of the sample size and the pivot position can lead to better results. The purpose of this study is to optimize the time performance of the SkewedKruskal algorithm by estimating how to partition the sorted edge list on the basis of some graph characteristics, so that the MST is likely to be completely contained in one of the two initial edge lists and the size of such a list is minimized.

Paper Outline In Sect. 2, we describe two classes of randomly generated weighted graphs we studied, namely, random graphs and KNN graphs. Section 3 describes the main steps that can be followed to estimate the position or the weight of the critical edge starting from an automatic classification of the graph and the estimation of its parameters. Section 4 describes an alternative and more general technique, based on graph density, that does not require to classify the graph. In Sect. 5, we illustrate the results of computational tests and comparisons between different versions of the SkewedKruskal algorithm.

2 Randomly Generated Graphs

As a preliminary step, in this study, we initially consider two classes of randomly generated graphs, namely, random graphs and k-nearest neighbor (KNN in the remainder) graphs. Hereafter, we describe the details of the graph generation procedure.

Uncorrected Proof

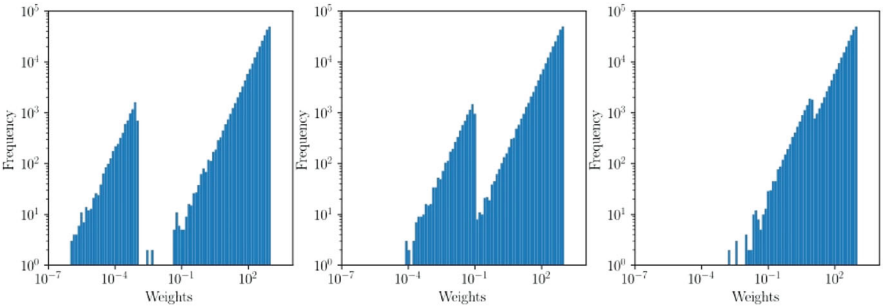


Fig. 1 Distribution of the edge weights in a random graph as a function of the clustering index γ (logarithmic scale)

Uncorrected Proof

Random Graphs In our tests, random graphs are generated according to the Erdős–Rényi model [2]: it allows to generate graphs $G_{n,\rho}$, where n is the number of vertices and $0 < \rho \leq 1$ is the graph density, i.e., the probability of including any edge (self-loops are excluded). The edge weights are generated according to a uniform probability distribution.

Besides size and density, we also consider a third parameter, that is, the clustering degree, indicated by γ. To generate random graphs with different clustering degree, we partition the vertex set into $\lceil\sqrt{n}\rceil$ disjoint subsets $C_1, C_2, ...C_{\lceil\sqrt{n}\rceil}$, such that $\lfloor\sqrt{n}\rfloor \leq |C_i| \leq \lceil\sqrt{n}\rceil \ \forall i = 1, \ldots, \lceil\sqrt{n}\rceil$. We define a parameter $0 < \gamma \leq 1$ representing the clustering degree. Then, the weight of all edges whose endpoints belong to the same cluster is multiplied by γ, while the weight of the other edges is divided by γ. For small values of γ, the resulting graph is strongly clustered.

According to this method to produce clustered random graphs, the number of intra-cluster edges grows as $\rho n\sqrt{n}$, while the number of inter-cluster edges grows as ρn^2. Figure 1 shows the distribution of the edge weights as a function of γ in random graphs $G_{1000,0.5}$ using a logarithmic scale, where the frequency of small weights can be better appreciated although they are relatively few.

KNN Graphs A KNN graph is obtained by generating its vertices at random as points in a circle of radius R. Each vertex is adjacent to its k-nearest neighbors, and the edge weights are the Euclidean distances between the endpoints. In our tests, these graphs were generated using $K - D$-trees and ball trees by the algorithms illustrated in [1, 5]) and implemented in the Python module `sklearn.neighbours`.

By construction, KNN graphs are clustered, with a clustering degree depending on k. Figure 2 shows the edge weight distribution of KNN graphs with $n = 1000$ for different values of k.

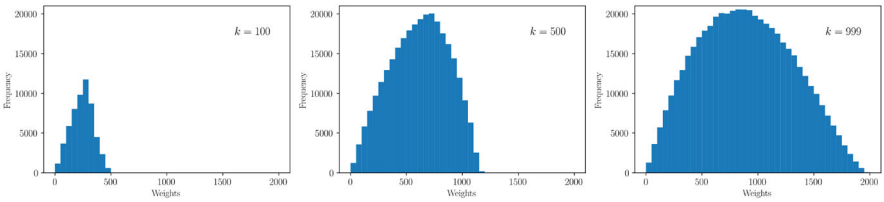


Fig. 2 Edge weight distribution in KNN graphs for $n = 1000$ and different values of k

3 Graph Classification

In this section, we describe a technique to estimate the position or the weight of the critical edge, starting from an automatic classification of the graph and an estimation of its relevant parameters. In Sect. 3.1, we describe an automatic classifier; in Sect. 3.2, we describe how the relevant parameters can be estimated once the graph has been classified; in Sect. 3.3, we analyze the relationship between the graph parameters and the position or weight of the critical edge; in Sect. 3.4, we consider the problem of suitably selecting a sample size from the edge list to obtain a reliable estimate of the position or the weight of the critical edge; in Sect. 3.5, we consider the selection of the pivot, i.e., the edge weight value that is used to partition the edge list into two parts.

3.1 Automatic Classification

We set up an automatic classifier to distinguish the two classes of randomly generated graphs shown above. The two classes are characterized by different distributions of edge weights. Therefore, the mean value and the variance of (a sample of) the edge weights are good candidates for an effective classifier.

Random Graphs By definition, the distribution of the edge weights is uniform when $\gamma = 1$. Hence, it is expected that the mean value and the variance of a sample subject to min-max scaling be close to the expected value $\mathbb{E}(X)$ and the variance $\mathrm{Var}(X)$ of a random variable X uniformly distributed between $a = 0$ and $b = 1$, i.e.,

$$\mathbb{E}(X) = \frac{a+b}{2} = \frac{1}{2}$$

and

$$\mathrm{Var}(X) = \frac{(b-a)^2}{12} = \frac{1}{12}.$$

Uncorrected Proof

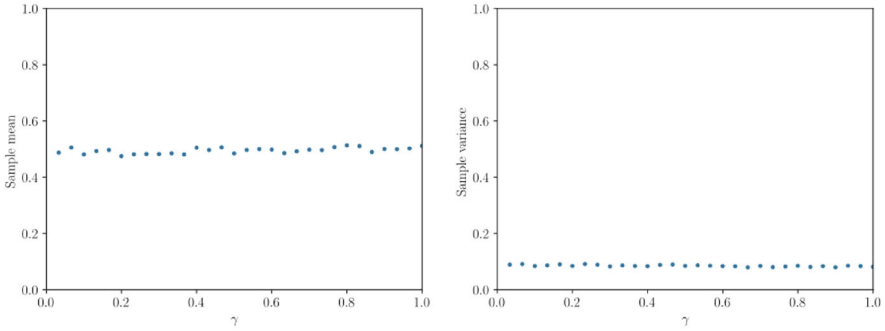


Fig. 3 Mean value and variance of edge weights in random graphs for different values of the clustering degree γ ($n = 2000$)

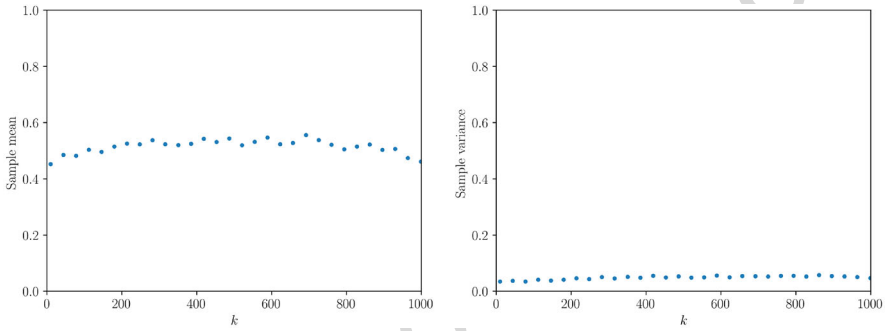


Fig. 4 Mean and average of the edge weights in KNN graphs for different values of k ($n = 1000$)

For clustered graphs, with low value of γ, deviations from these values are possible. However, the number of intra-cluster (small weight) edges is very small. Figure 3 shows the values of the mean and the variance for $n = 2000$ and different values of γ.

It is apparent that the value of the clustering degree does not significantly affect the mean and the variance.

KNN Graphs The mean of the edge weights in KNN graphs is very close to the mean of random graphs, when k is neither very small nor very large. However, the variance is slightly smaller, thus allowing to distinguish the two types of graphs. Figures 4 and 5 show the mean and the variance for $n = 1000$ and different values of k as well as for $k = 400$ and different values of n.

Similar to random graphs, the mean and the variance are almost independent on γ also for KNN graphs.

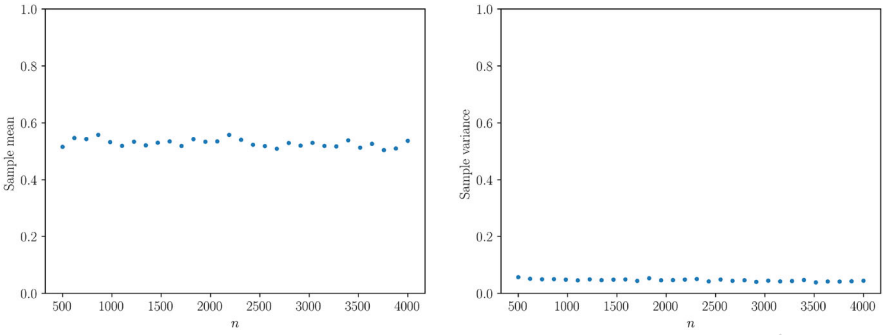
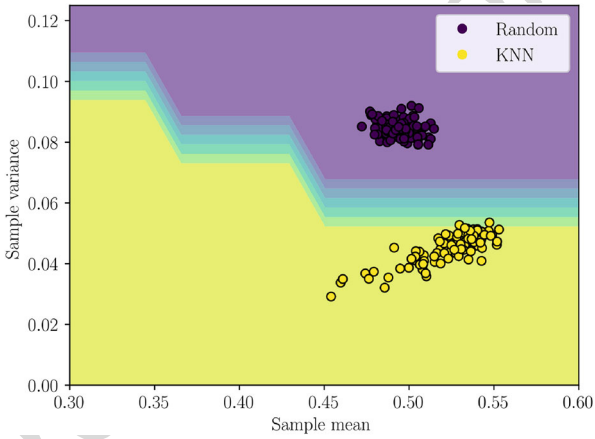


Fig. 5 Mean and average of the edge weights in KNN graphs for different values of n ($k = 400$)

Fig. 6 Borders of the trained classifier



A Classifier From the observations above, it is possible to construct, train, and evaluate an automatic graph classifier, which takes the mean and the variance of (a sample of) the edge weights in input and automatically determines the type of graph.

We used a k-nearest neighbor classifier with $K = 5$. The model was trained with 120 random graphs and 120 KNN graphs. Its reliability has been tested on 200 randomly generated graphs, 100 for each type, taking a sample of 1000 edges. Figure 6 shows the results: each area corresponds to a type of graph after training the classifier. Each point lies in the area of its own type, i.e., all classifications are correct.

Selection of the Sample Size To minimize the impact of the classification on the computing time, one wants to use a sample size s as small as possible. Figure 7 shows the degradation in the accuracy of the classifier when s decreases. The results have been obtained from 70 graphs for each type, with random size and parameters. Keeping $s = 1000$, no classification errors were observed. This is the reason why the value $s = 1000$ was used to train the classifier.

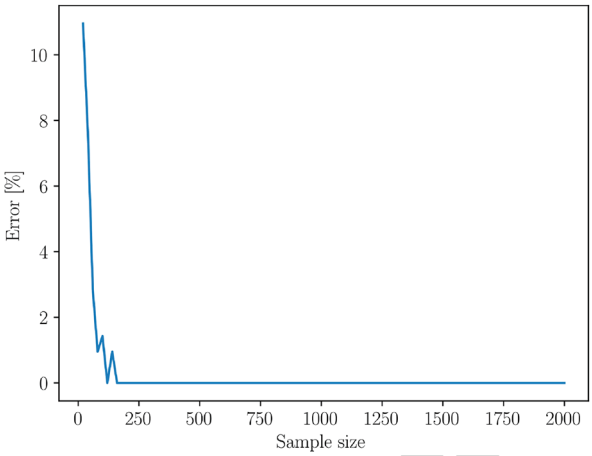


Fig. 7 Percentage error of the classifier for different values of the sample size

3.2 *Parameter Estimation*

Random Graphs In a random graph, the size of the graph and its density are immediately observable by counting the number n of vertices and the number m of edges. The clustering degree γ can be also estimated from the distribution of the edge weights. For this purpose, we analyzed the difference between theoretical quantiles and actual quantiles in the list of edge weights, and we studied its dependency on γ, n, and ρ. By a suitable regression on some randomly generated datasets, we could reliably estimate γ, especially for large values of n and ρ. However, further tests illustrated in Sect. 3.3 showed that γ is not a good predictor for the position of the critical edge in random graphs, and this is consistent with the results observed in Sect. 3.1. Therefore, we did not make further efforts to estimate γ.

KNN Graphs The parameter k in KNN graphs plays a similar role to the density ρ in random graphs, although the distribution of the edge weights in KNN graphs is not uniform as it is in random graphs. It is fair to assume a correlation between k and the number of edges m. This is confirmed by the analysis illustrated in Fig. 8 which shows an approximately linear correlation.

Hence, a simple way to estimate k is to take a fraction η of the ratio $\frac{m}{n}$. To calibrate η, we compute $\frac{nk}{m}$ for 50 graphs with n randomly selected between 500 and 5000 and k between 50 and $0.9n$. We obtain a mean value $\eta = 1.76258$ with a standard deviation of 0.04482. The average percentage error in the estimate of k is only 0.02455%.

Owing to the slight inflection of the curve when k tends to n, the estimate tends to be worse for large k. However, as observed in the remainder, a precise estimate is required only for small values of k.

Unconnected Pro

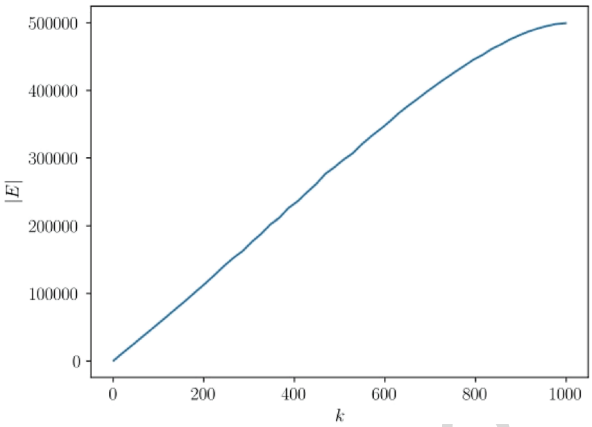


Fig. 8 Correlation between the parameter k and the number of edges in KNN graphs

3.3 Critical Edge and Graph Parameters

To optimize the time performance of the SkewedKruskal algorithm, the goal is to estimate the position or the weight of the critical edge. Let $\hat{w}$ be the critical edge weight, after a min-max scaling procedure which is executed to eliminate the dependency on the range of the weights. Alternatively, one can estimate the critical edge position in the sorted edge list: let $n - 1 \geq \hat{p} \leq m$ be such position in a sorted list of m edges. This estimate is more indirect, but it does not require any preprocessing to scale the weights.

In this section, we analyze the distribution of the values of $\hat{w}$ and $\hat{p}/m$ for different types of randomly generated graphs. This analysis allows to estimate a minimum size of the edge list fraction that is needed to include the MST.

Since the position and the weight of the critical edge depend on some parameters of the graphs, we search for the correlations that can be a base for a reliable prediction.

Random Graphs We generated random graphs with different values of size n, density ρ, and clustering degree γ, and we observed the values of $\hat{w}$ and $\hat{p}/m$.

Dependency on γ Figure 9 shows the results obtained from 200 random graphs with $n = 1000$, $\rho = 0.5$ and different values of γ. No significant correlation can be observed between $\hat{p}/m$ and the clustering degree. A weak correlation between $\hat{w}$ and the clustering degree is observed only for small values of γ.

Dependency on n Figure 10 shows the results obtained from 200 random graphs with $\gamma = 0.5$, $\rho = 0.5$ and different values of n from 50 to 3000. When n grows, both $\hat{p}/m$ and $\hat{w}$ decrease (both in average and in standard deviation).

Unconfected Proc

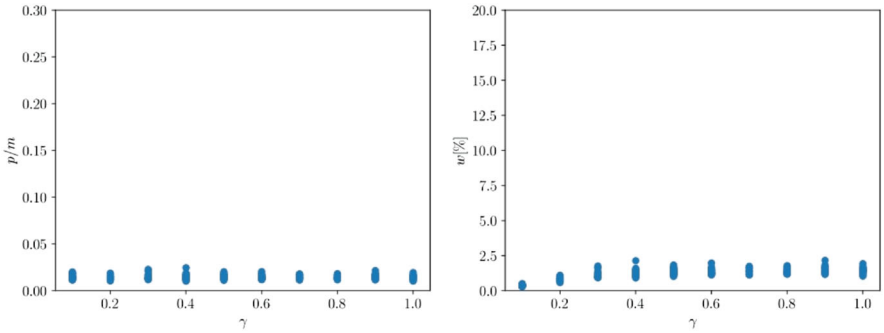


Fig. 9 Position $\hat{p}/m$ and weight $\hat{w}$ in random graphs for different values of the clustering degree γ

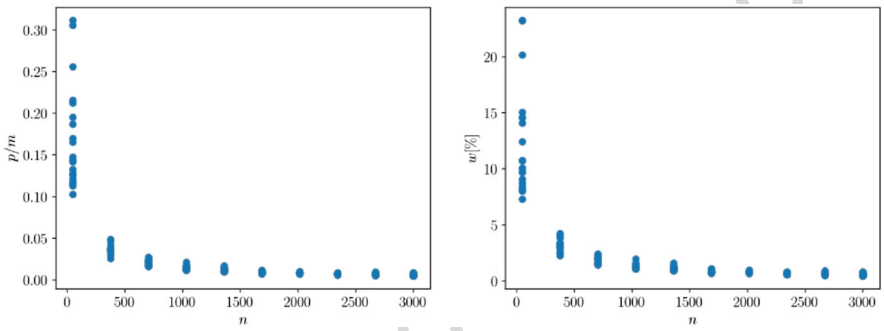


Fig. 10 Position $\hat{p}/m$ and weight $\hat{w}$ in random graphs for different values of the size n

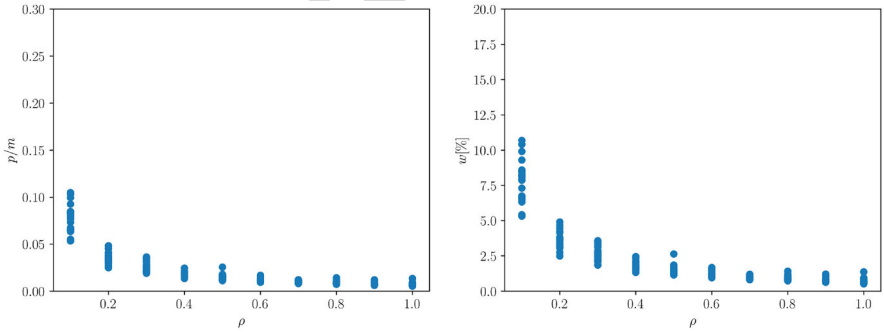


Fig. 11 Position $\hat{p}/m$ and weight $\hat{w}$ in random graphs for different values of the density ρ

Dependency on ρ Figure 11 shows the results obtained from 200 graphs with $n = 1000$, $\gamma = 0.5$ and different values of ρ. Both the average and the standard deviation of $\hat{p}/m$ decrease when ρ increases, and the same is observed for $\hat{w}$.

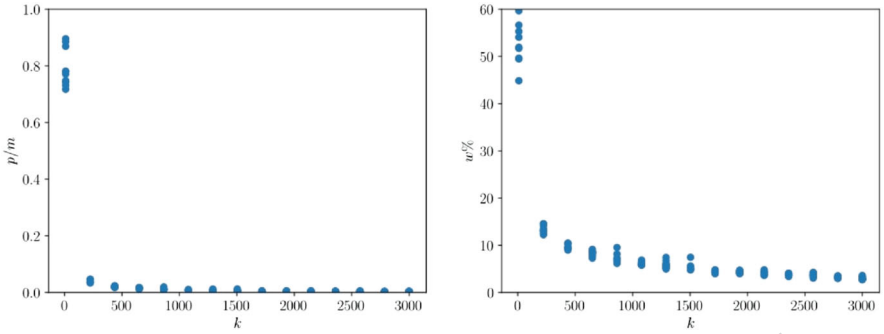


Fig. 12 Position $\hat{p}/m$ and weight $\hat{w}$ in random KNN graphs for different values of k

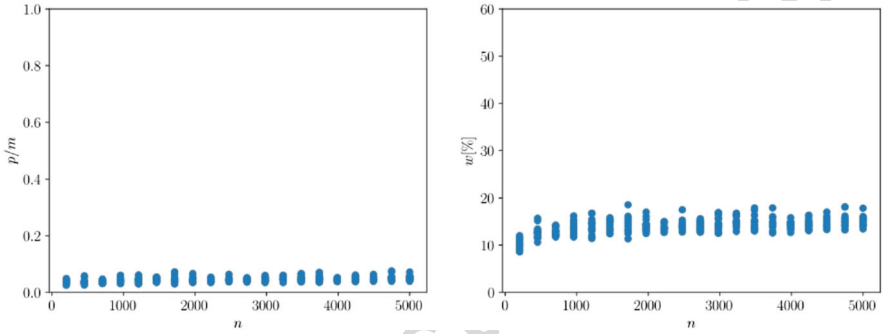


Fig. 13 Position $\hat{p}/m$ and weight $\hat{w}$ in random KNN graphs for different values of n.

KNN Graphs We generated KNN graphs with different values of n and k, and we observed the corresponding values of $\hat{w}$ and $\hat{p}/m$.

Dependency on k Figure 12 shows the results obtained from 400 graphs with $n = 3000$ and different values of k. A strong correlation is observed between $\hat{p}/m$ and k and between $\hat{w}$ and k. It should be noted that the connectivity of the graphs, and then the existence of a spanning tree, may be lost when k is too small.

Dependency on n Figure 13 shows the results obtained from 200 graphs, 10 for each size, with $k = 80$ and different values of n from 250 to 5000. Both $\hat{p}/m$ and $\hat{w}$ show a weak correlation with n, since they slightly increase when n increases.

Critical Edge Position From the analysis shown above, a possible conclusion is that a reliable estimate of $\hat{p}/m$ can be based on n and ρ for random graphs and n and k for KNN graphs. The type of graph at hand can be identified with the automatic classifier illustrated in Sect. 3.1.

We trained our models with 50 graphs for each type and different values of the relevant parameters. Then we observed the absolute errors in estimating $\hat{p}/m$ on 20

Uncorrected Proof

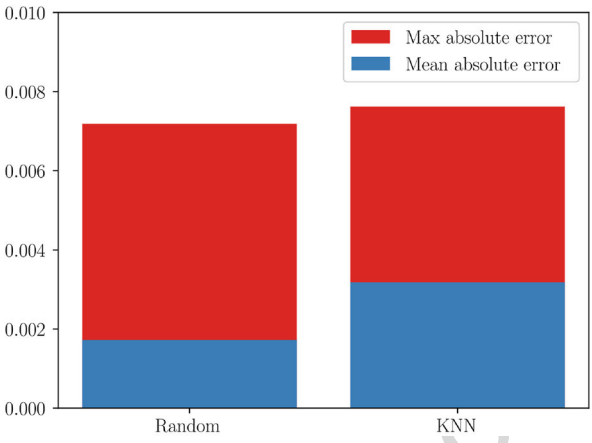


Fig. 14 Average and maximum absolute error in estimating $\hat{p}/m$ in different types of graphs

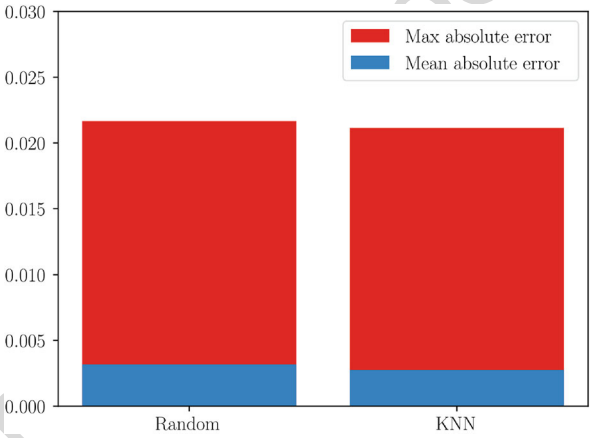


Fig. 15 Absolute error in estimating $\hat{p}/m$ from $\hat{w}$

graphs for each type with random parameters. The results illustrated in Fig. 14 show that $\hat{p}/m$ can be reliably estimated, since the error is kept below 1.5% of the number of edges in the graph. In this representation, we do not distinguish between errors in excess and in defect. In the next section, we differentiate, because the two types of error are likely to produce different effects on the SkewedKruskal algorithm.

Critical Edge Weight Estimating $\hat{w}$ would be even more useful for the purpose of optimizing the SkewedKruskal algorithm, since the value $\hat{w}$ could be directly used as a pivot value, to partition the edge list. However, the effect of a wrong choice of the pivot value does not depend on the value itself but rather on its position in the edge list, since its position affects the number of missing MST edges in the left part of the partition at the first level.

Figure 15 shows the error in estimating $\hat{p}/m$ from $\hat{w}$. The largest errors occur with random graphs, especially when γ is close to 0, owing to the existence of many edges with small weight.

3.4 Tuning the Edge Sample Size

Since the SkewedKruskal algorithm is meant as a tool to solve the MST problem for very large graphs, one wants to avoid scanning the whole edge list to select the pivot, especially at the first level of the recursion. Instead, a suitably sized random sample should be used: we indicate its size with s. Clearly, the accuracy in the pivot selection depends on s, generating a trade-off between the amount of computing time devoted to the pivot selection and the total amount of computing time required by the SkewedKruskal algorithm.

Starting from an estimate of the position $\hat{p}/m$, we consider two main techniques to select the pivot. One is the extraction of the edge in position $\lceil s\,\hat{p}/m \rceil$ in a sorted random sample of size s. Another is the use of the IntroSelect algorithm to compute the element of the sample in position $\lceil s\,\hat{p}/m \rceil$ without sorting it.

We remark that this analysis applies to the technique based on the estimate of the critical edge position; when the critical edge weight is estimated, instead, no additional operation is required, because the estimated value $\hat{w}$ can be directly used as a pivot.

Random Graphs The case of random graphs is the most favorable among the two graph types considered so far, because it is possible to discard a very large fraction of the edges. We considered random graphs with different values of n and ρ. We did not consider γ because the results shown in the previous sections suggest that it is not likely to affect the position of the critical edge significantly.

Figure 16 shows the results obtained with $four$ random graphs of different size n and density $\rho = 0.5$. Figure 17 shows the results obtained with $four$ random graphs of different density ρ and size $n = 4000$.

The estimates in Fig. 16 are clearly more accurate for larger n. It is interesting to note that accurate estimates on a graph with 2000 vertices require a larger sample than accurate estimates on a graph with 8000 vertices.

Density turns out to be relevant: estimates in sparse graphs are unstable, i.e., the variability in the estimate of $\hat{p}/m$ is larger than in dense graphs, as shown in Fig. 17.

KNN Graphs The critical edge position in KNN graphs heavily depends on parameter k. Here, we want to analyze its dependency on the sample size s.

Figure 18 shows the results obtained with random KNN graphs of different size and $k = 500$, while Fig. 19 shows the results obtained with random KNN graphs with different values of k and $n = 5000$.

The quality of estimates is robust with respect to the size n. When k is low (for instance, $k = 50$), the estimate is less accurate. This was expected, because for small k the variance of the critical edge position tends to be large.

Uncorrected Proof

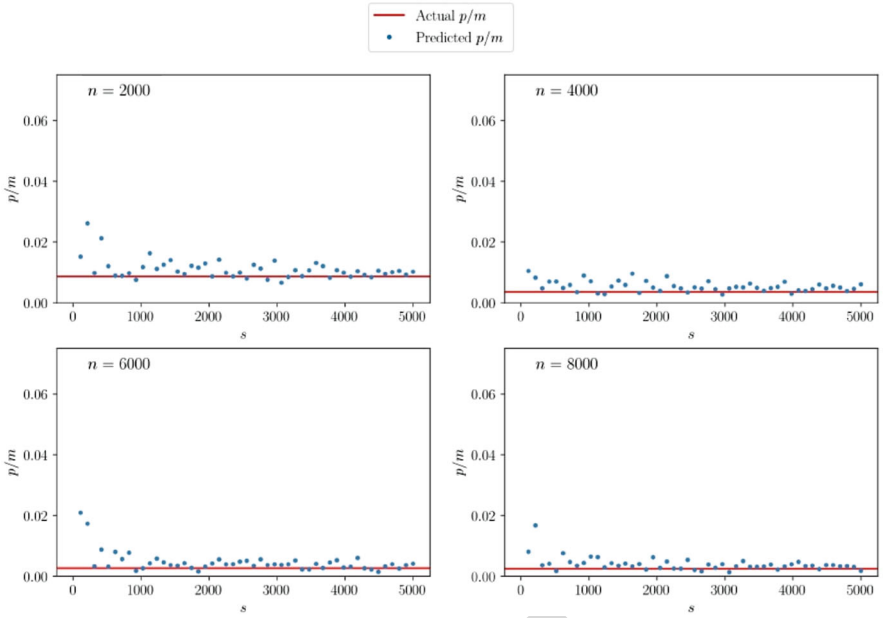


Fig. 16 Pivot selection on $four$ random graphs of different size. The horizontal line represents the position of the critical edge. The dots represent the estimated position of the critical edge as a function of the sample size s

Uncorrected

3.5 *Selection of the Pivot*

The outcome of the search for the critical edge position reported in Sect. 3.3 shows a significant standard deviation of $\hat{p}/m$ for all graph types. This suggests that the pivot selection should be done in a rather conservative way to increase the probability that the pivot is chosen in a position beyond that of the critical edge.

To search for such a pivot, we generated many graphs with the same parameters, and we observed the maximum values of $\hat{p}/m$. Then, with a nonlinear regression model, we tuned a threshold $\overline{p}/m$. To make this estimate even more reliable, we added 20% of the thresholds, artificially shifting the pivot to larger weight edges.

Once computed the threshold value, it is also necessary to revise the cardinality s of the sample. In the remainder, we show some graphical representations, where two horizontal straight lines indicate the threshold $\overline{p}/m$ obtained from the regression model, including the artificial shift described above, and the actual value of $\hat{p}/m$ for the graph, while the dots show the outcome of the estimate of $\hat{p}/m$ for different values of s.

Random Graphs Figure 20 shows that a reliable estimate for $\overline{p}/m$ requires a larger size of the random sample compared to the sample size needed to estimate $\hat{p}/m$.

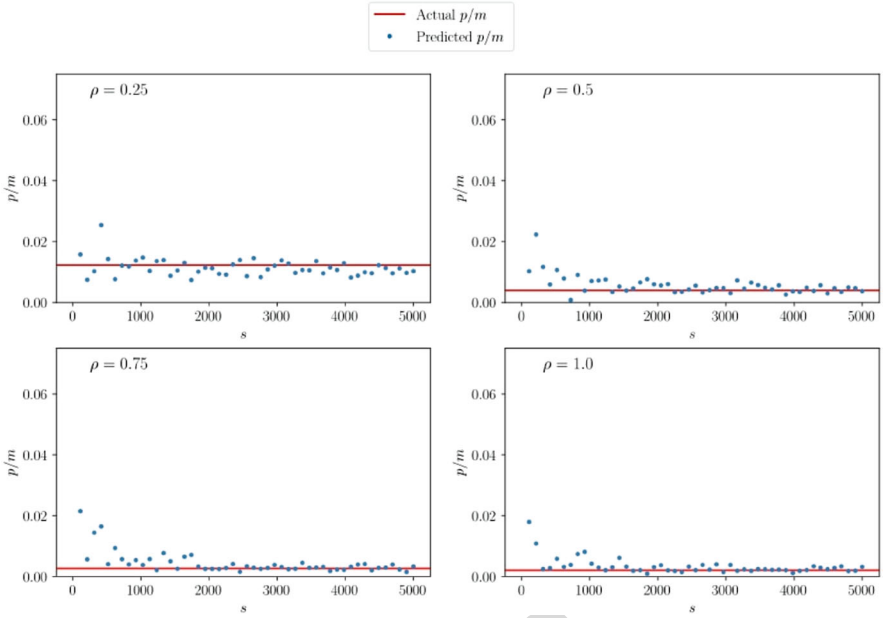


Fig. 17 Pivot selection on four random graphs of different density. The horizontal line represents the position of the critical edge. The dots represent the estimated position of the critical edge as a function of the sample size s

Uncorrected

Similarly, for random graphs of different density shown in Fig. 21, the precision of the threshold estimates turns out to be affected by the density. For low values of ρ, it is necessary to further increase the random sample size s.

In all cases, the estimated conservative threshold $\overline{p}/m$ always yields an effective selection of the pivot, since the selected pivot position is systematically beyond the position of the critical edge.

KNN Graphs Figure 22 shows the results obtained with KNN graphs with $k = 400$. As with random graphs, it is necessary to consider a larger sample to estimate the thresholds compared to the sample needed to estimate $\hat{p}/m$. Furthermore, for small n, the precision is more sensitive to s than it is for large n, as expected. Figure 23 shows similar results with KNN graphs with different values of k and $n = 4000$. The width of the estimated range is heavily dependent on k; the larger the value of k, the better the precision of the estimated thresholds.

4 A More General Technique

The classifier presented in Sect. 2 allows to identify the two types of randomly generated graphs considered so far; in turn, this allows to use suitable indicators to estimate $\hat{p}/m$ or $\hat{w}$. However, for the sake of general applicability, one wants to

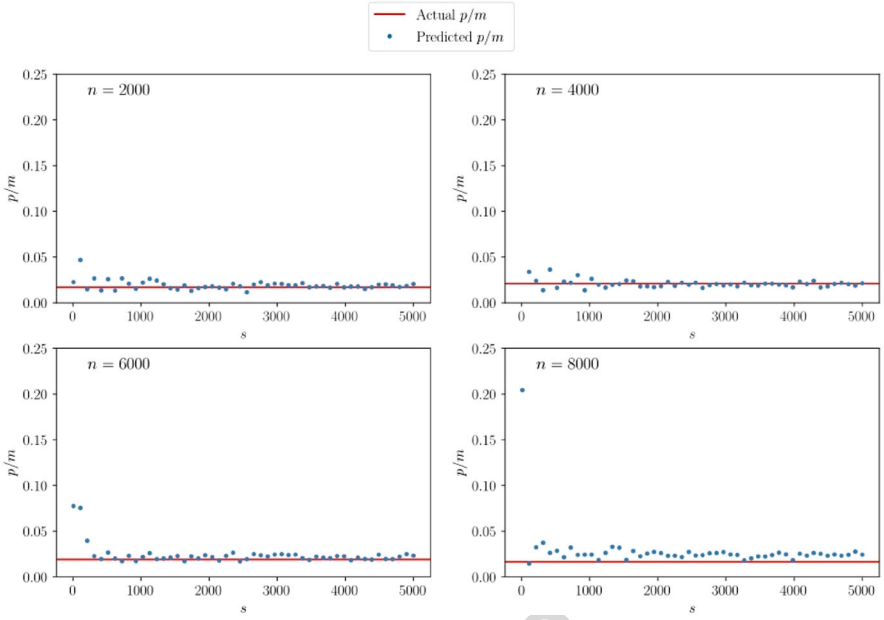


Fig. 18 Pivot selection in random KNN graphs of different size

Uncorrected

be able to estimate $\hat{p}/m$ or $\hat{w}$ for any type of graph and without any prior knowledge about it. For this purpose, we search for some significant correlation between $\hat{p}/m$ or $\hat{w}$ and some general characteristic of graphs, independent of their type.

The most important advantage of estimating the critical edge weight is that it is not necessary to extract a random sample and to suitably tune its size s. When the graph is classified and hence some assumptions can be done on its edge weight distribution, one can estimate a range that is likely to contain the critical edge weight $\hat{w}$ instead of the critical edge position $\hat{p}/m$, as illustrated in Sect. 3. On the contrary, generic models like those illustrated in this section cannot predict $\hat{w}$, because the edge weight distribution is unknown. Therefore, in this section, we only consider the problem of estimating $\hat{p}/m$ and to select a pivot value from it.

Standard Deviation of Weight Distribution From the observation of Fig. 6, one can note that the standard deviation of the distribution of the edge weights in random graphs is larger than in KNN graphs. Hence, a search direction is for a possible correlation between such a standard deviation and $\hat{p}/m$. To test this hypothesis, we generated random graphs whose weights follow the normal distribution, with different values of the standard deviation. Unfortunately, from the results shown in Fig. 24, no correlation is visible.

Density Another potentially useful characteristic is density: in random graphs and KNN graphs, when ρ and k (respectively) increase, $\hat{p}/m$ decreases. To search for a possible correlation between $\hat{p}/m$ and the graph density independent of the weight

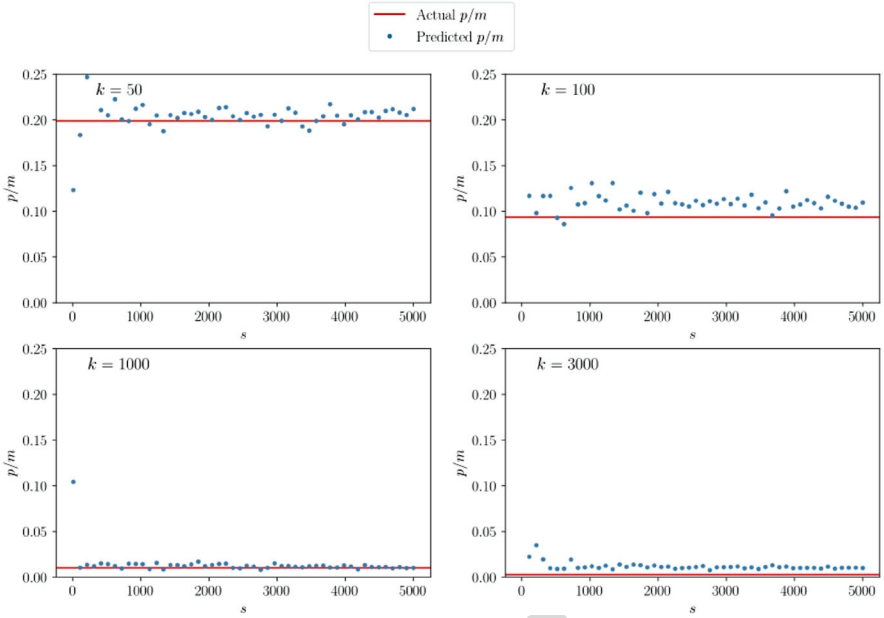


Fig. 19 Pivot selection in random KNN graphs with different values of k

Uncorrected

distribution, we generated random graphs with 8 different distribution models, including Gaussian, exponential, and gamma.

The results shown in Fig. 25 show a rather strong correlation between $\hat{p}/m$ and the graph density, while the weight distribution turns out to be unlikely to affect $\hat{p}/m$. This confirms the observations outlined in Sect. 3.3: the parameters that affect the weights, such as γ in random graphs, do not significantly affect $\hat{p}/m$, while the parameters that affect the graph structure, such as k in KNN graphs, also affect the position of the critical edge.

Therefore, we defined a nonlinear regression model, trained with 500 randomly generated graphs with different values of n and ρ and different weight distributions. We used 12 values of ρ to obtain a good approximation for low density: 6 values are equally spaced in a logarithmic scale between 0 (excluded) and 0.3 (included), while the other 6 values are equally spaced in a linear scale between 0.3 (excluded) and 1 (included). Figure 26 shows the quality of the estimate obtained with 20 graphs for each type, generated with random parameters.

Sample Size The technique based on the estimate of the critical edge position requires to extract a pivot element in a given position from the edge list. Since the IntroSelect procedure is time-consuming, in spite of its theoretically linear worst-case time complexity, it is advisable to extract the pivot from a random sample of suitable size s. Figure 27 shows the error in the estimate of $\hat{p}/m$ with different graph types of different size n as a function of the sample size s.

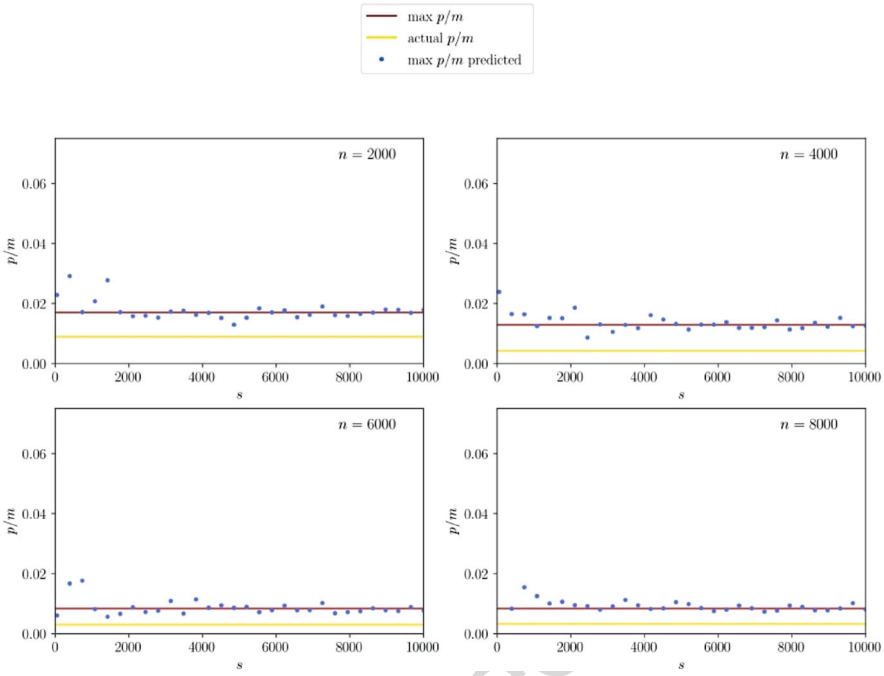


Fig. 20 Critical edge range for different values of the random sample size s (random graphs of four different size n) Each box represents a single instance of a graph

For all graph types, the method is reliable, and s can be tuned to a relatively small value, compared to the size of the whole edge list.

5 Computational Results

Computational Environment All computational tests reported in this section have been carried out on an AMD Ryzen 5 4500U processor. Algorithms have been coded in Python, and the learning algorithms were taken from the open-source library `Scikit-learn`. Input and output files are available from the authors upon request, as well as the code.

Versions of the SkewedKruskal Algorithm From the previous analysis, it is possible to design different techniques to drive the SkewedKruskal algorithm. We considered six of them, identified here as follows:

- Algorithm 1a: estimate the position of the critical edge, based on graph classification

Uncorrected

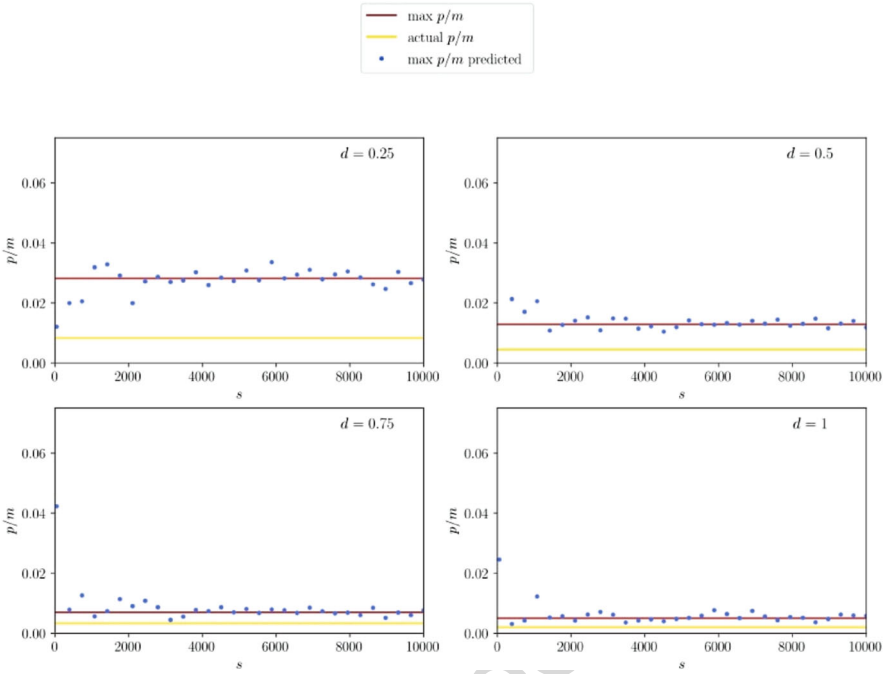


Fig. 21 Critical edge range for different values of the random sample size s (random graphs of four different density values ρ)

- Algorithm 1b: estimate the maximum position of the critical edge, based on graph classification
- Algorithm 2a: estimate the weight of the critical edge, based on graph classification
- Algorithm 2b: estimate the maximum weight of the critical edge, based on graph classification
- Algorithm 3a: estimate the position of the critical edge, based on graph density
- Algorithm 3b: estimate the maximum position of the critical edge, based on graph density

Optimal Size of the Random Sample The methods based on an estimate of $\hat{p}/m$ require a suitable sizing of a random sample of the edge set.

In [8], it was suggested to select a random sample of size $s = \lceil\sqrt{m}\rceil$ from the edge list. However, in some of the techniques shown so far, the random sample is used not only for extracting the pivot but also for the purpose of classification and parameter estimation. Therefore, a larger value of s could work better. For this reason, we introduce a multiplicative factor α (to be suitably tuned), so that $s = \lceil\alpha\sqrt{m}\rceil$.

Uncorrected

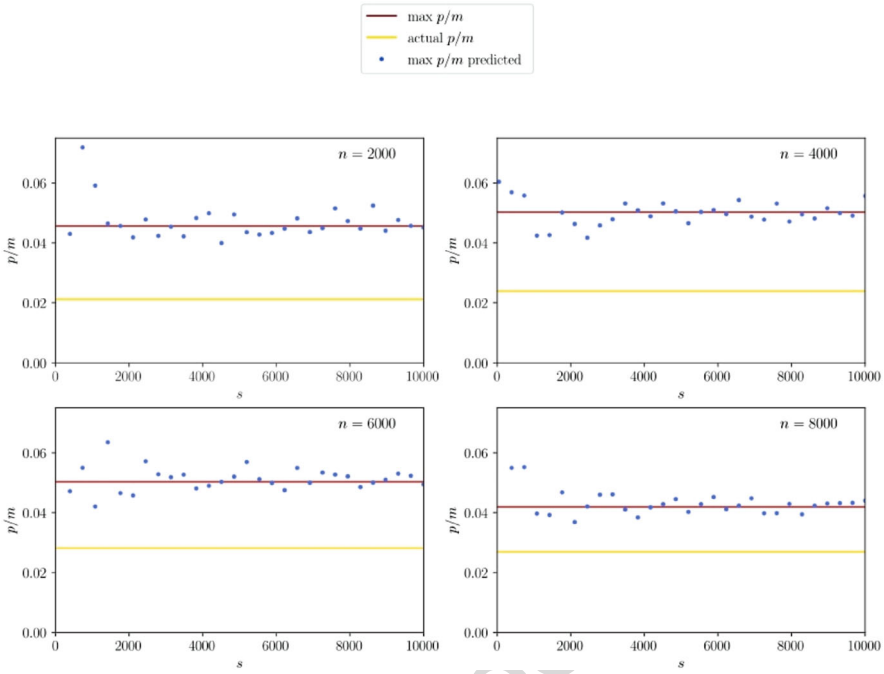


Fig. 22 Critical edge range for different values of the random sample size s (KNN graphs of four different size n)

Uncorrected

5.1 *Calibrating the Algorithm by Graph Classification*

Hereafter, we present the results of some computational tests in which graphs of different types (random graphs, KNN graphs) were generated, they were (correctly) classified, their critical edge position or critical edge maximum position or critical edge weight were estimated, and such an estimate was finally used to drive the SkewedKruskal algorithm, by suitably partitioning the edge list. Hence, this technique, corresponding to Algorithms 1a, 1b, 2a, and 2b, exploits the assumption that the input graphs belong to a restricted and known subset of randomly generated graph types.

Random Graphs Figure 28 shows how the computing time of SkewedKruskal depends on the size of the random sample, when the pivot element is selected after estimating its position $\hat{p}/m$ in the edge list (Algorithm 1a). The tests have been done on random graphs with density $\rho = 0.5$ and different size n. Each point is the average value from five graphs.

The computing time is highly variable: the illustration shows several peaks for some values of α. We also observed a rather high variance of the computing time for the same value of α.

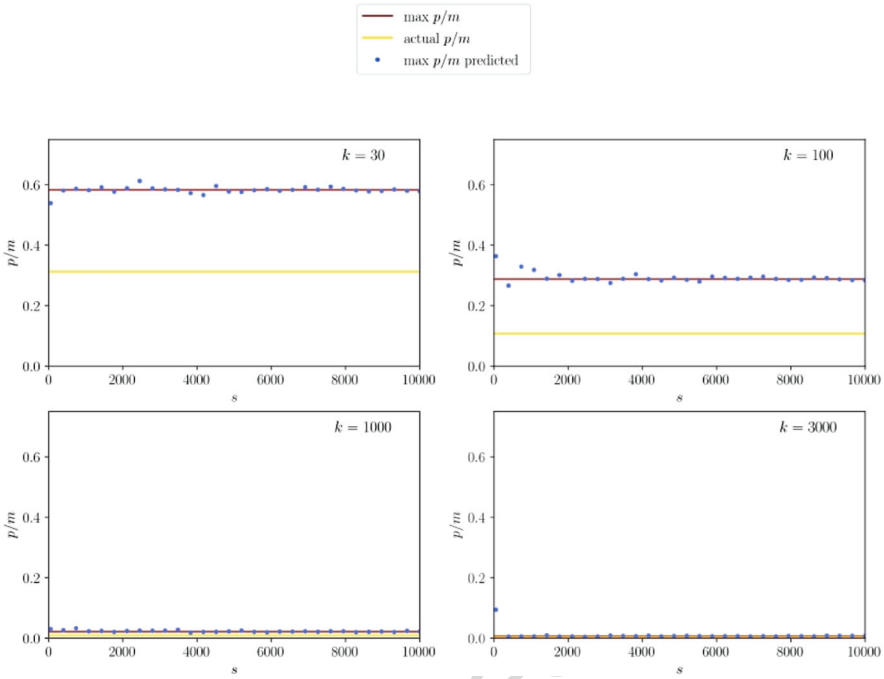
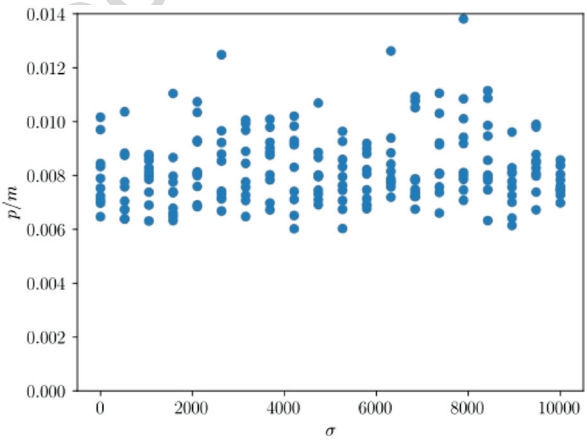


Fig. 23 Critical edge range for different values of the random sample size s (KNN graphs of four different values of k)

Fig. 24 Position of the critical edge as a function of the standard deviation of the weights distribution



When the pivot is selected in a more conservative way (Algorithm 1b), better results are achieved. Figure 29 shows the computing time of SkewedKruskal for the same graphs as in Fig. 28 when the pivot is selected in position $\overline{p}/m$, instead of $\hat{p}/m$.

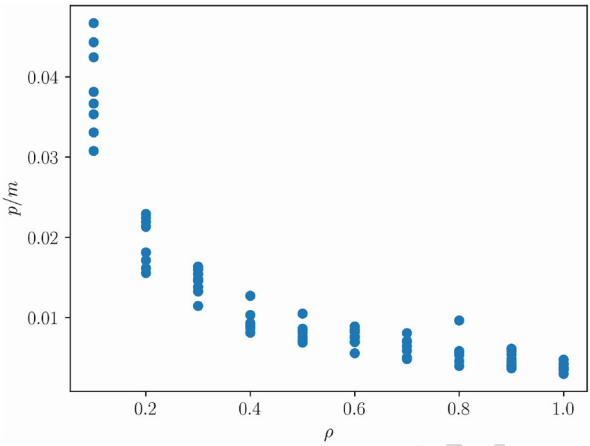


Fig. 25 Position of the critical edge as a function of graph density

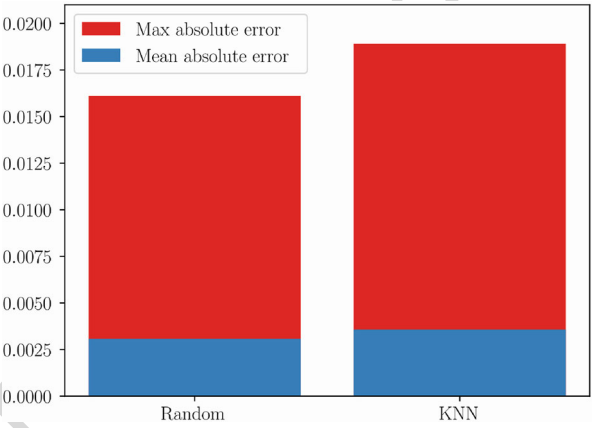


Fig. 26 Errors in estimating the position of the critical edge based on the density for different graph types

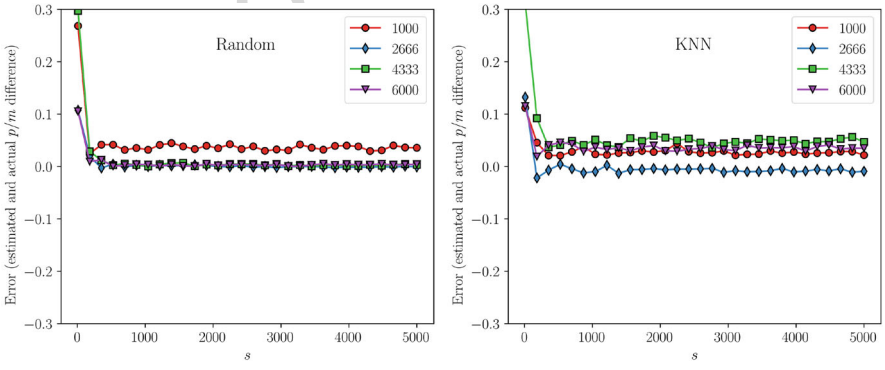


Fig. 27 Density-based estimate of the critical edge position

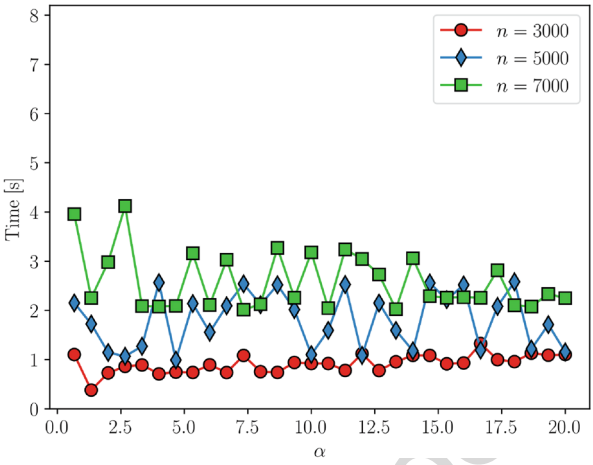


Fig. 28 Computing time of SkewedKruskal driven by the classification of the graph type and an estimate of $\hat{p}/m$ (random graphs with $\rho = 0.5$)

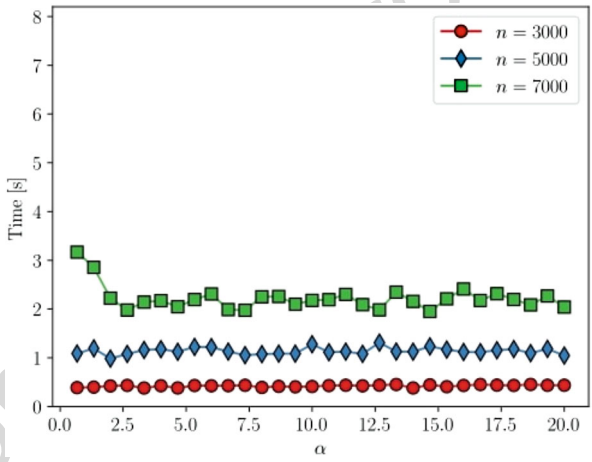


Fig. 29 Computing time of SkewedKruskal driven by the classification of the graph type and an estimate of $\overline{p}/m$ (random graphs with $\rho = 0.5$)

The observed computing time is smaller in average and also more stable. Concerning the optimal calibration of the sample size, values of α close to 2.5 yielded the best outcome.

Better results were obtained by estimating the maximum value of the critical edge weight instead of the critical edge position (Algorithms 2a and 2b), as illustrated in Fig. 30.

KNN Graphs Similar results were obtained with KNN random graphs. Figure 31 shows the computing time of SkewedKruskal when the pivot element is selected in position $\hat{p}/m$ (Algorithm 1a) in random KNN graphs with $k = n/10$.

The outcome is rather unstable, with some peaks due to graphs where the critical edge position was underestimated. This happened in 15% of the graphs.

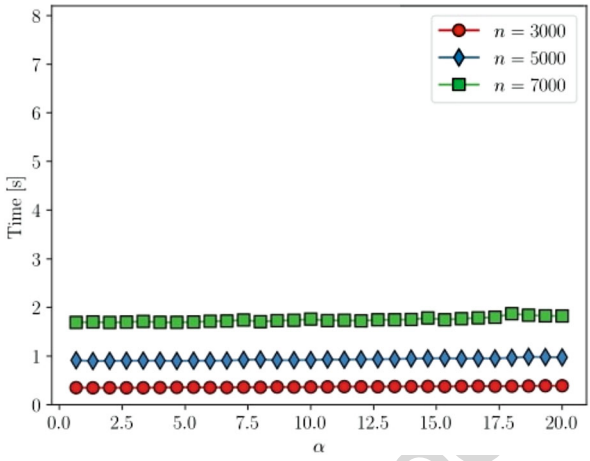


Fig. 30 Computing time of SkewedKruskal driven by the classification of the graph type and an estimate of $\hat{w}$ (random graphs with $\rho = 0.5$)

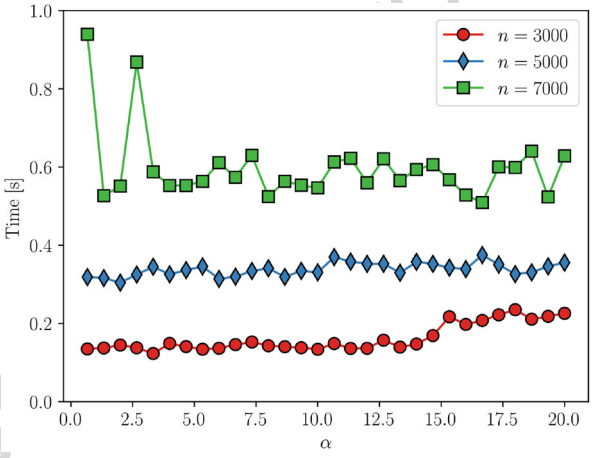


Fig. 31 Computing time of SkewedKruskal driven by the classification of the graph type and an estimate of $\hat{p}/m$ (KNN random graphs)

As shown in Fig. 32, more stable average computing time is observed when the pivot is selected in position $\overline{p}/m$ (Algorithm 1b). However, the average computing time is slightly worse in this case.

For this graph type, it is effective to estimate the critical edge weight. However, 20 underestimates were observed among 100 graphs when the pivot selection was done according to an estimate of $\hat{w}$ (Algorithm 2a). Better results were achieved when the pivot was selected according to an estimate of $\overline{w}$ (Algorithm 2b): the number of wrong partitions was almost reduced to zero, as shown in Fig. 33.

The computing time is similar to that observed for Algorithm 1b with $\alpha \approx 3$, but its dependency on s looks more predictable, as expected: in this case, the random sample of size s is used to classify the graph, not to compute $\overline{w}$.

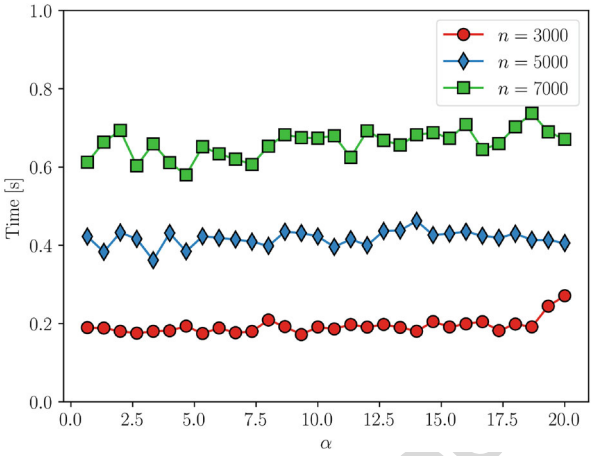


Fig. 32 Computing time of SkewedKruskal driven by the classification of the graph type and an estimate of $\overline{p}/m$ (KNN random graphs)

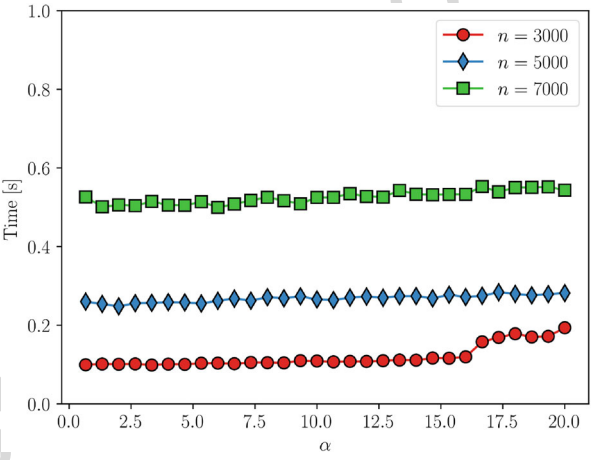


Fig. 33 Computing time of SkewedKruskal driven by the classification of the graph type and an estimate of $\overline{w}$ (KNN random graphs)

5.2 *Calibrating the Algorithm by Graph Density*

When the graph type is unknown and no assumption can be done about it, it is harder to estimate $\hat{p}/m$, since the distribution of the edges is unknown. For this reason, in Algorithm 3 that is meant to address this more general settings, we use a factor $c > 1$ to artificially increase the value of $\hat{p}/m$ estimated from the graph density. To tune c, we did some preliminary tests with random graphs of different type, size, and density. The size of the random sample was set to $s = 10\sqrt{m}$. Setting $c = 1.4$, we observed the best results; underestimates occurred in only 5% of the graphs.

Figures 34 and 35 show the computing time for random graphs ($\rho = 0.5$) and KNN graphs ($k = \frac{n}{10}$), respectively.

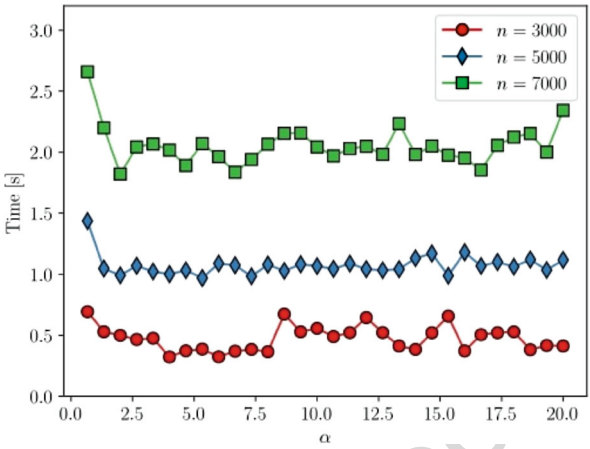


Fig. 34 Computing time of SkewedKruskal driven by graph density and an estimate of $c\overline{p}/m$ (random graphs)

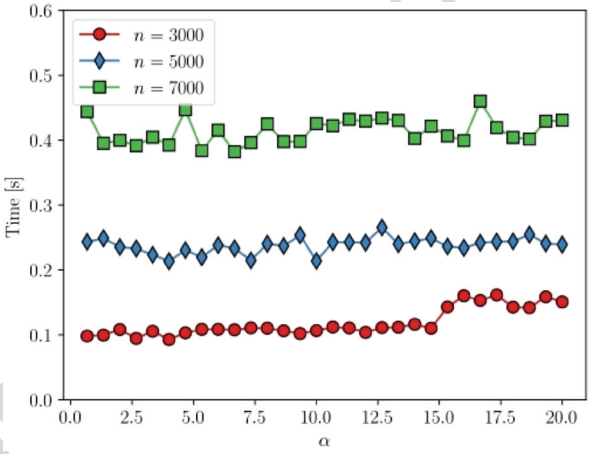


Fig. 35 Computing time of SkewedKruskal driven by graph density and an estimate of $c\overline{p}/m$ (KNN random graphs)

Values of α around $five$ allowed to minimize the computing time for all graph types, achieving results similar to those obtained with algorithms based on graph classification.

When applying this more general technique based on graph density, we cannot estimate the critical edge weight, because no assumption can be done on the edge weight distribution. Hence, the prediction is based only on the critical edge position.

5.3 Comparison Between Algorithms

To compare the versions of the SkewedKruskal algorithm, we observe the computing time for different values of the graph types and parameters. Following the results outlined in the previous subsections, the SkewedKruskal algorithm driven by graph

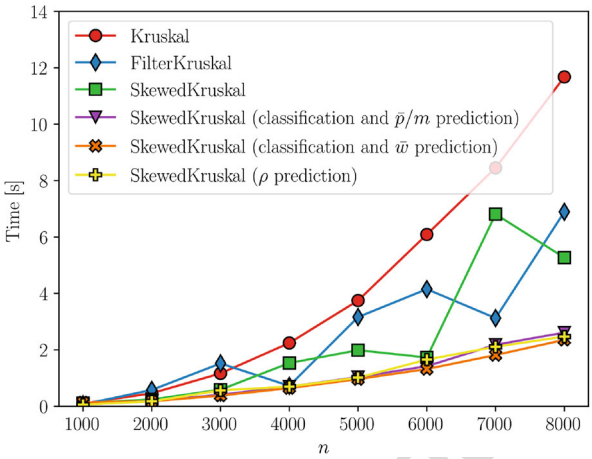


Fig. 36 Computing time with random graphs of different size n

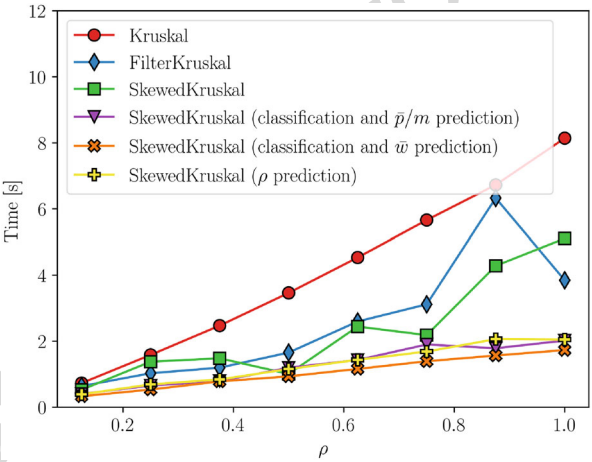


Fig. 37 Computing time with random graphs of different density ρ

classification uses the estimate of the maximum position of the critical edge $\overline{p}/m$ or the maximum weight of the critical edge $\overline{w}$ tuning the sample size with $\alpha = 3$, while the algorithm driven by graph density uses $\alpha = 5$.

Random Graphs Figure 36 shows the results with random graphs with $\rho = 0.5$, $\gamma = 1$, and some values of n.

All SkewedKruskal variants have definitely smaller computing time compared with Kruskal algorithm, and the gap grows with the size n. The variants based on classification and density are faster and more robust than the original Skewed-Kruskal algorithm, which is sometimes forced to make a recursive call on the second list at the first recursion level due to an underestimation of the critical arc weight.

The same observations hold when the size is fixed ($n = 5000$ in our tests), and computing time is measured on graphs with different density, as shown in Fig. 37. Varying the degree of clustering γ, for fixed size $n = 5000$ and density $\rho = 0.5$,

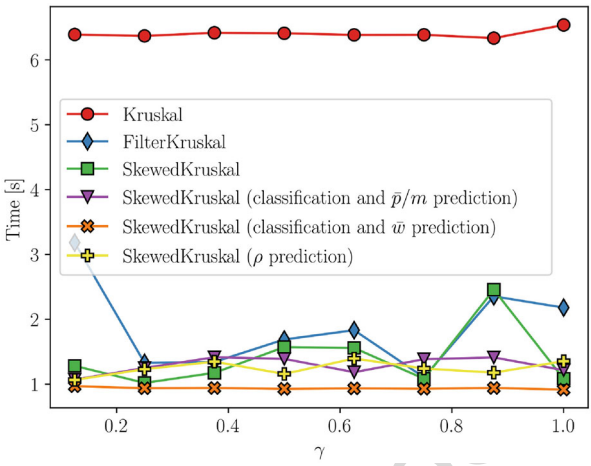


Fig. 38 Computing time with random graphs of different clustering degree γ

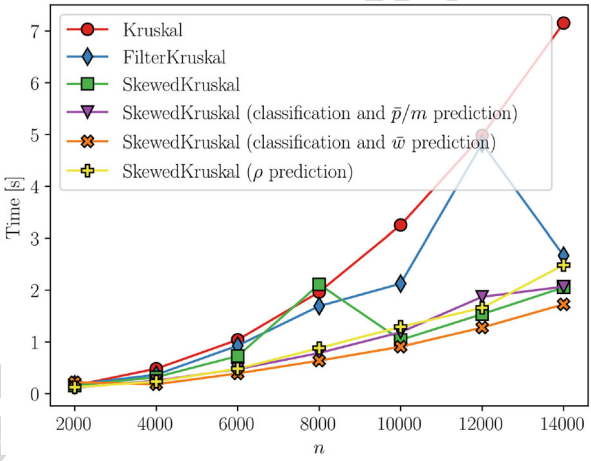


Fig. 39 Computing time with KNN graphs of different size

the computing time reported in Fig. 38 is observed. In general, the computing time turns out to be smaller for all variants of SkewedKruskal when γ is very close to 0, i.e., the degree of clustering is large. For the other values of γ, the algorithms based on classification and density have almost identical computing time.

KNN Graphs The comparison between the algorithms on KNN graphs shows even larger gaps between the computing time taken by SkewedKruskal and that of the new variants we have analyzed. Figure 39 shows the computing time for KNN graphs of different size n and $k = \frac{n}{10}$.

As with random graphs, the computing time growth with n looks also more predictable for the new versions compared with the original one. The same observation holds for the results shown in Fig. 40, obtained with KNN graphs with fixed size $n = 5000$ and different values of k.

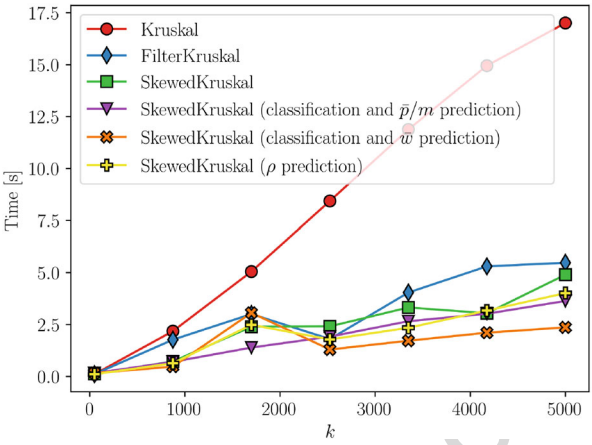


Fig. 40 Computing time with KNN graphs of different values of k

6 Conclusions

In this study, we have applied some techniques based on the analysis of the data of each specific instance of the minimum cost spannning tree problem to optimize the computing time taken by the SkewedKruskal algorithm, a variation of the classical Kruskal algorithm in which the underlying idea is to minimize the edge subset that needs to be examined and sorted to find a minimum spanning tree. This can be viewed as an application of (heuristic) learning algorithms to the performance improvement of (exact) optimization algorithms.

The results we have obtained show that the approach is promising; we have analyzed two techniques, one based on the classification of graphs in which one can use models suitably tuned for each specific graph type and the other based on graph density, which can be evaluated in any general case with no assumption on the structure of the graph.

Uncorrected Proof

References

1. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM **18**, 509–517 (1975)
2. Erdös, P., Rényi, A.: On random Graphs I. Publicationes Mathematicae Debrecen, Debrecen (1959)
3. Janson, S., Knuth, D.E., Łuczak, T., Pittel, B.: The birth of the giant component. Random Struct. Algorithms **4**(3), 233–358 (1993)
4. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. Am. Math. Soc. **7**(1), 48–50 (1956)
5. Omohundro, S.: Five balltree construction algorithms. ICSI Technical Report TR-89-063 (1989)
6. Osipov, V., Sanders, P., Singler, J.: The filter-kruskal minimum spanning tree algorithm. In: Proceedings of the Meeting on Algorithm Engineering & Experiments (2009)

7. Paredes, R., Navarro, G.: Optimal incremental sorting. In: Proceedings of the Meeting on Algorithm Engineering & Experiments (2006)
8. Righini, E., Righini, G.: The skewed kruskal algorithm. Learning and Intelligent Optimization, pp. 130–135 (2022)

Uncorrected Proof