Improving local search in a minimum vertex cover solver for classes of networks

Markus Wagner[1], Tobias Friedrich[2] and Marius Lindauer[3]

***Abstract*— For the minimum vertex cover problem, a wide range of solvers has been proposed over the years. Most classical exact approaches are encountering run time issues on massive graphs that are considered nowadays. A straightforward alternative approach is then to use heuristics, which make assumptions about the structure of the studied graphs. These assumptions are typically hard-coded and are hoped to work well for a wide range of networks—which is in conflict with the nature of broad benchmark sets.**

With this article, we contribute in two ways. First, we identify a component in an existing solver that influences its performance depending on the class of graphs, and we then customize instances of this solver for different classes of graphs. Second, we create the first algorithm portfolio for the minimum vertex cover to further improve the performance of a single integrated approach to the minimum vertex cover problem.

I. INTRODUCTION

Randomized search heuristics and local search algorithms are commonly used for difficult optimization problems, where no efficient exact algorithms are available. As our testbed in this research we use the classical MINIMUM VERTEX COVER problem, which is the problem of finding a minimal subset of vertices of the input graph such that every edge in this graph is incident to at least one node in this subset. The problem is NP-complete and only algorithms with exponential worst case running time are known. In fact, under the assumption of the Exponential Time Hypothesis [21] there cannot be any algorithm solving it with subexponential worst case running time.

Depending on the network structure, some exact solvers for MINIMUM VERTEX COVER can perform very efficiently on sparse real-world networks [2], but the problem remains theoretically hard in the worst-case and practically hard for other common benchmark instances. We therefore only focus on local search approaches and try to achieve a best possible approximation in a fixed time budget.

Popular local search approaches for tackling MVC include PLS [29], NuMVC [13], TwMVC [11], COVER [31] and FastVC [9]. The two currently most popular solvers are NuMVC and FastVC (see Section II). We compare them on 123 test instances (see Appendix) from the Network Repository Benchmark Suite [32] and allow a time limit of 1000 seconds for each instance and solver. The straightforward quality measure of a vertex cover solver is then the sum of the sizes of the discovered vertex covers of all test instances. In this measure, as expected, the more recent FastVC algorithm improves upon NUMVC by finding in total vertex covers with 3,966 fewer nodes on the test set.[1]

Our aim is to find a new variant of FastVC that performs even better. In this research, we determine a set of FastVC variants that are tuned using automatic algorithm configuration [19], namely the configurators PARAMILS [19], ROAR [18], and SMAC [18]. Most new variants outperform the classical FastVC on classes of instances. Depending on the chosen parameter set, we find vertex covers that are in total up to 4306 nodes smaller than the ones found by NuMVC (see Section III). Compared to NuMVC, our variants are therefore about 9% better than the original FastVC on the test set.

In a second step in this article, we study which FastVC variant performs best for which classes of networks. For graph classes with joint features like social or web graphs, we observe that even smaller vertex covers can be found within the same time limit. For different network classes we propose different FastVC variants.

After developing these specialized solvers for specific network families, we build an algorithm selector [30] which decides on the used variant depending on 14 instance features for a given instance (see Section IV). We observe that the portfolio again further improves upon the single variants and discuss limitations.

II. PRELIMINARIES

Minimum Vertex Cover Problem: Finding a MINIMUM VERTEX COVER of a graph is a classical NP-hard problem. Given an unweighted, undirected graph $G = (V, E)$, a vertex cover is defined as a subset of the vertices $S \subseteq V$, such that every edge of G has an endpoint in S, i.e., for all edges $\{u, v\} \in E$, $u \in S$ or $v \in S$. The decision problem k-vertex cover decides whether a vertex cover of size k exists. We consider the optimization variant to find a vertex cover of minimum size.

Applications arise in numerous areas such as network security, scheduling and VLSI design [17]. The vertex cover problem is also closely related to the problem of finding a maximum clique. This has a range of applications in bioinformatics and biology, such as identifying related protein

[1]Optimisation and Logistics Group, School of Computer Science, The University of Adelaide, Australia, markus.wagner@adelaide.edu.au

[2]Algorithm Engineering Group, Hasso Plattner Institute, Germany, friedrich@hpi.de

[3]Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Germany, lindauer@cs.uni-freiburg.de

[1]Here: not on all 123 instances, but on the 112 instances on which NUMVC produces an output in the alloted time.

978-1-5090-4601-0/17/$31.00

sequences [1] and for finding genetic polymorphism in the human genome [24].

Numerous algorithms have been proposed for solving the vertex cover problem. As the target of our investigations, we choose FASTVC [9], [10] over the popular NUMVC [12] as a solver for the minimum vertex cover problem as it works better for massive graphs. FASTVC is based on two low-complexity heuristics, one for initial construction of a vertex cover, and one for choosing the vertex to be removed in each exchanging step, which involves random draws from a set of candidates.

Minimum Vertex Cover Solver: For our experimental investigations, we select all 86 instances used by Cai [9] and Friedrich et al. [16]. Furthermore, we add 37 instances to further reduce noise in our experiments and provide statistically more stable results. In the combined instance set the number of vertices ranges from about 400 to over 6 million, and the number of edges ranges from about 1000 to over 56 million. All 123 instances are available online [32].

The instances vary widely in their origin. For example, we are including 14 collaboration networks (ca-*, from various sources such as Citeseer and also Hollywood productions), 14 web graphs (web-*, showing the state of various subsets of the internet at particular points in time), five infrastructure networks (inf-*), six interaction networks (ia-*, e.g. about email exchange), 21 general social networks (soc-*, e.g., Delicious, LastFM, Youtube), and 44 subnets of Facebook (socfb-*, mostly from different American universities).

Many solvers work in the way that they create a first cover, and then repeatedly attempt to drop a vertex from it, which in return might require the addition of other vertices. A lot of work on local search has focused on criteria for filtering the set of candidate vertices and on the function for comparing elements. Once this is done, simply the best candidate is selected. If the approach aims to select the very best of the candidates, then this is not feasible for the massive graphs that we consider due to the computational complexity involved.

One of the central contributions of FASTVC is its cost-effective heuristic *Best from Multiple Selections* (BMS) for picking a good-enough candidate. For a set S, the BMS heuristic works as follows [9]: *Choose k elements randomly with replacement from the set S, and then return the best one (w.r.t. some comparison function f), where k is a parameter.*

In the original article, k was set to 50 to strike a balance between the time complexity and the quality of the selected vertex. As a consequence of this choice, the probability that the BMS heuristic chooses a vertex whose so-called *loss* value is not greater than 90% vertices in the current vertex cover is greater than $1 - 0.9^{50} > 0.9948$. This means that the BMS heuristic returns a vertex of very good quality with a very high probability—we omit further details on the heuristic as these are outside the scope of this article.

III. PARAMETER TUNING

Although, $k = 50$ is justified by this theoretical insight, so far a study is missing whether the empirical performance of FASTVC can be improved by using a different value of k. Since manual tuning of parameters is often an error-prone and tedious task, we can use algorithm configuration procedures, such as PARAMILS [19], GGA [4], [3], ROAR [18], SMAC [18] or IRACE [27], to automatically determine a well-performing parameter setting of k. Here, we used the algorithm configuration (AC) methods PARAMILS, ROAR and SMAC. ROAR is a random search, PARAMILS is based on local search in the space of possible parameter settings and SMAC uses Bayesian optimization [8] as a global optimization strategy. All three methods use an aggressive racing strategy [19] to discard poor configurations early with performance evaluations on only few instances.

In our setting, all three AC methods optimize FASTVC's internal parameter k in the range $k \in [0, 100]$ using $1,000$ evaluations per tuning and a time budget of $1,000$ seconds per iteration. For the sake of efficient experiments, we evaluated FASTVC with all 101 settings of k on all instances with 9 different random seeds and used the median-performance across the random seeds of this pre-evaluated grid in the AC experiments. The 123 instances are split into a training and test set; using 61 training instances for optimizing k and 62 test instances to verify that an optimized k generalizes well to new instances. To ensure that all instances classes are represented in both instance sets, we use stratified sampling with strati being instance classes.

To determine well-performing settings of k, we used two strategies:

1) Configuration on all 61 training instances;
2) Configuration on each of the instance classes.

Based on this setup, the best-performing k across all training instances is set to 20, see Table I. In comparison to the original default value 50 of k, FASTVC with $k = 20$ covers 332 nodes less on the training instances. However, the same setting covers only 8 more nodes on the test instances. This is to be expected, since all AC systems have the assumption that the used instance set is homogeneous (i.e., there is one well-performing parameter setting for the entire set), but here, we use a very heterogeneous instance set from different class. The heterogeneity is a particularly significant issue here because of the aggressive racing that can reject a parameter setting already if it performs poorly on a single instance which can be non-representative for the entire instance set here.

Looking at the instance subsets with respect to the classes, the AC systems find better settings of k on all of them except on class (ia-*) with respect to the training instances. For sc-*, tech-*, soc-*, socfb-*, the performance improvement also generalizes well to the test instances. In contrast, the performance on inf-* considerably suffers from the optimized k. Similarly, the performance does not improve on web-*, bio-* and ca-* on the test instances, on which different values of k have no impact on the performance of FASTVC. Although the instances in training and instances stem from the same class, FASTVC shows a substantially different behavior on instances from the same class; hence, even not all of the subsets seem to be not perfectly homogeneous.

	Best k on Training			Training Performance			Test Performance		
	ParamILS	ROAR	SMAC	ParamILS	ROAR	SMAC	ParamILS	ROAR	SMAC
ALL	33	23	20	234	309	332	52	−49	8
inf	17	2	2	15	342	342	−210	−2043	−2043
ia	50	50	50	0	0	0	0	0	0
web	17	5	5	227	410	410	0	0	0
bio	17	10	4	2	2	2	0	0	0
sc	17	8	8	99	397	397	19	92	92
tech	0	2	2	160	216	216	233	219	219
soc	17	3	3	9	33	33	100	401	401
ca	83	13	13	7	12	12	0	0	0
socfb	33	26	26	11	11	11	13	12	12

TABLE I: Comparing FastVC with parameter k optimized on by different algorithm configuration systems on different instance subsets. The performance is the number of nodes saved (=fewer covered) compared to the default setting of $k = 50$. ALL refers to all 61 training instances (for Training Performance) and to all 62 test instances (for Test Performance) respectively.

Last but not least, we observe that the setting of k substantially differs between the different subsets (with a trend to smaller values than the default of $k = 50$) and choosing the best k for each subset would cover 739 more nodes than FastVC with $k = 50$. This supports our observation that our instances are heterogeneous and k should be selected on a per-class or per-instance base.

Figure 1 shows an excerpt of our results on the testing set. The box-plots represent the distribution of results observed by different algorithms and algorithm configurations. NuMVC (green) is typically the worst-performing algorithm. FastVC50 (original, black) performs better, however, it is usually outperformed on socfb-* by FastVC26 (tuned on socfb-*). FastVC20 (best overall in training, red) performs better than FastVC20 on the large soc-* instances.

Note that NuMVC could not generate a solution on 11 of the 123 instances. In these cases, we set the solution quality to the number of vertices in the graph.

IV. Instance Analysis and Algorithm Portfolios

Overall, we notice that an optimized k for an instance class typically improves the performance of FastVC substantially. A well-known method to exploit this is algorithm selection [30]. The idea of algorithm selection is that given instance, an algorithm selector selects a well-performing algorithm from a (often small, finite) set of algorithms, the so-called algorithm portfolio.

In the following, we study whether we can also apply algorithm selection to FastVC to improve its performance further than with algorithm configuration only. Based on the results of algorithm on the training instances, we constructed a portfolio of settings of k, i.e., $k \in \{2, 3, 4, 5, 8, 13, 26, 50\}$, which we treat as an algorithm and try to predict which setting of k should be used for a new given instance. Additionally, we also add NuMVC to our portfolio as a possible algorithm to be selected.

Since algorithm selection is often implemented using machine learning [34], [23], we need two preparation steps: (i) instance features [28] to characterize instances numerically and (ii) performance data of each algorithm on each training instance. Luckily, we already have the latter from our algorithm configuration experiments, and we need only to define some instance features for vertex cover instances.

A. Instance Features

Large networks come in many different flavors. Over the last decades, many ways have been devised to characterize different aspects of classes of networks. Examples are aspects of spatial networks [5], community structure in complex networks [25], and dynamics in small-world graphs [35]).

We use the following subset of 14 features as all feature values are available online [32]:

- V: number of nodes;
- E: number of edges;
- ρ: density, being the ratio of the number of edges to the number of possible edges $\rho = \frac{2E}{V(V-1)}$;
- d_{min}: maximum vertex degree, where the degree of node v is defined to be the number of nodes that are adjacent to v;
- d_{avg}: average vertex degree;
- d_{max}: maximum vertex degree;
- r: assortativity coefficient, which is the Pearson correlation coefficient of degree between pairs of linked nodes, $r > 0$ indicates a correlation between nodes of similar degree and $r < 0$ indicates relationships between nodes of different degree;
- T: number of triangles formed by three edges (3-cliques);
- T_{avg}: average triangles formed by an edge;
- T_{max}: maximum triangles formed by an edge;
- κ: global clustering coefficient is the ratio of triangles to the number of connected triplets of vertices, giving an indication of the clustering in the whole network;
- κ_{avg}: average local clustering coefficient, given by the proportion of links between the vertices within its neighborhood divided by the number of links that could possibly exist between them;
- K: maximum i-core number, where an i-core of a graph is a maximal induced subgraph and each vertex has degree at least i;
- ω_{lb}: lower bound on the size of the maximum clique.

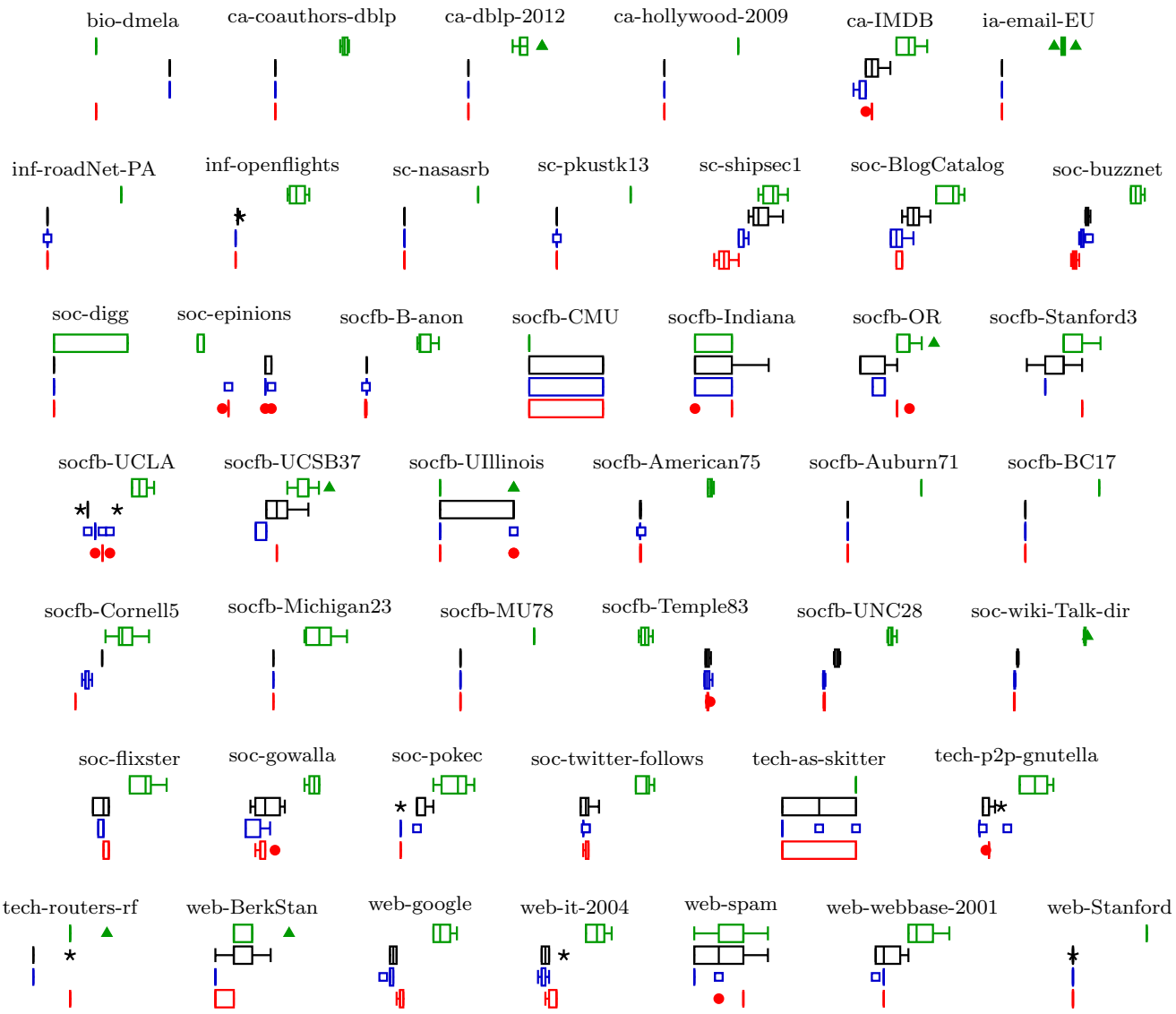

Fig. 1: Qualitative comparison across the testing set. The axes are omitted to facilitate the qualitative analysis; distributions further left (smaller) are better. For each instances, we show from top to bottom the performance of NuMVC (green), FastVC50 (original, black), FastVC26 (tuned on socfb-*), and FastVC20 (best overall in training, red). Shown are the results of nine independent runs of 1000s each. Omitted are instances where all algorithms always produced the same results.

We list all feature values in the appendix at the end of this article.

B. Algorithm Selection

Using the instance features from Section IV-A and the performance data from all relevant k from Section III on the training instances, we train an algorithm selector, which can predict with the help of machine learning models which k to choose for given instance features of a new instance. Here, we use the state-of-the-art AutoFolio [26] which implements a broad range of algorithm selection approaches and uses algorithm configuration (here SMAC [18]) to determine a well-performing one for the problem at hand. Our first interesting observation is that the default configuration of AutoFolio (which is quite similar to the well-known algorithm selector SATZilla [37], [38]) only slightly improves over selecting the best k on all training instances. Only after allowing AutoFolio a configuration budget of $1,000$ function evaluation (which needs roughly 10 minutes), AutoFolio optimizes its own parameter settings such that it performs better than single k.[2]

Table II shows the comparison of the optimized FastVC on our training instances ($k = 13$), the best FastVC on our test instances ($k = 26$), the optimized AutoFolio and a theoretical optimal algorithm selector, called virtual best

[2]We use the most recent version of AutoFolio available on `https://github.com/mlindauer/AutoFolio`. For our vertex cover training instances, this version chooses to use xgboot [14] instead of the default random forest as classification model [7] and adds a principle component analysis with at most 7 dimensions on top.

Solver	Diff. to FastVC $k = 50$
FastVC $k = 13$ (best on train)	-122
FastVC $k = 26$ (best on test)	2
AutoFolio (all features)	352
AutoFolio (cheap features, Sect. IV-D)	543
Virtual Best Solver	1568

TABLE II: On test instances, comparing algorithm selector AutoFolio, the theoretical best performing algorithm selector (called virtual best solver) and the single best solver from training and test instances. Performance measured by the difference to the default parameter setting of FastVC 50.

solver, i.e., always selecting the best performing solver for a given instance. We compare the performance of our systems to FastVC with the previously proposed $k = 50$ to show how much FastVC can profit from using methods such as algorithm configuration and algorithm selection.

The on training optimized FastVC with $k = 13$ covers 122 nodes more than $k = 50$ on the test instances. This shows that it is not an easy task to determine a single well-performing k on the training instances that performs also well on the test instances.[3] A better choice would be $k = 26$ on the test instances, which also supports that the training instances do not perfectly resemble the test instances. We note that in practice, we can only make decisions on the training instances and do not know the performance on test instances. Therefore, studying FastVC with $k = 26$ shows only how valid our choice of $k = 13$ on the training instances is.

Our algorithm selector AutoFolio (when using all 14 instance features) performs substantially better than both static choices of k by covering 352 nodes less than $k = 50$. A perfect algorithm selector could even cover more nodes showing the importance of a per-instance selection of k and that a single k across all instances is a sub-optimal strategy. We believe that adding more instances to the training instance set and using more informative feature set can further improve the performance of our algorithm selector.

C. Portfolio Contribution

For our algorithm selection experiments, we simply use all relevant settings of k and also NuMVC to select a solver from. Since the algorithm selector automatically learns which solvers to select, it often does not hurt to add some more solvers than necessary to the portfolio. However, with this strategy, we still lack any insight about the importance of the individual solvers. Therefore, we also study the portfolio in two ways:

1) Sorting the solvers by the marginal contribution to the performance of the virtual best solver; [39]

[3]In contrast to the results in Table I, we choose $k = 13$ based on all training instances whereas the algorithm configuration systems make decisions based on a subset of instances.

Solver	Marg. Contr.	Iter. Marg. Contr.
FastVC $k = 13$	12	–
FastVC $k = 2$	462	732
FastVC $k = 8$	47	273
FastVC $k = 26$	17	104
FastVC $k = 4$	17	49
FastVC $k = 3$	8	8
NuMVC	0	2

TABLE III: Marginal contribution to the virtual best solver (Marg. Contr.) and marginal contribution by greedily adding a solver one at a time (Iter. Marg. Contr.). Both in number of nodes on training instances.

2) Iteratively, we greedily construct a portfolio of solvers by adding the most contributing solver in each iteration.

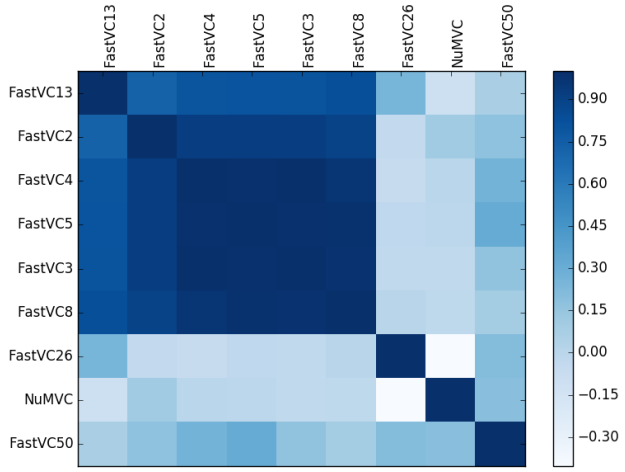

Fig. 2: Spearman rank coefficients in a heatmap (dark fields correspond to large correlation).

Table III shows the corresponding results sorted by the second criterion.

For the marginal contribution (Column "Marg. Contr."), only FastVC with $k = 2$ or $k = 8$ are important. However, this does not necessarily imply that all other solvers are not important. If two solvers are highly correlated, both solvers would have a small marginal contribution, since removing one solver would not decrease the performance of the virtual best solver dramatically as long the other solver is still in the portfolio. Figure 2 shows the correlation of all used solvers in a heatmap. FastVC with high values for k (26 and 50) and NuMVC are substantially different from the other settings with a small k. Surprisingly, FastVC with $k = 2$ still behaves differently than similar small values of k. Settings of k between 4 and 8 lead to quite similar results.

If we build our portfolio of solvers iteratively (Column "Iter. Marg. Contr" in Table III), we would first add the best performing solver on the training instances (FastVC with $k = 13$) to the portfolio. Secondly, FastVC with $k = 2$ improves the virtual best solver performance by 732 more nodes and then we would add $k = 8$ to further improve by

273 nodes. Afterwards, FASTVC with $k = 26$ and $k = 4$ further slightly improves the portfolio. All other settings of k are subsequently not important anymore. So, only these five solvers out of the 9 studied solvers would be necessary to build a decent portfolio here.

D. Feature Importance

As the calculation of instance features forms an important step in the application of algorithm portfolios, we review the necessary calculation times. In the following, we analyze which features are the most important ones for algorithm selection and we investigate how subsets of features impact portfolio performance.

To investigate which features are actually needed, we compute for each of the 14 features the average Gini importance [7] across all cost-sensitive random forests models, that can predict for a pair of solvers which one will perform better [38]. The results are shown in Figure 3, revealing that there is not a single feature, but a set of about a dozen features which together describe a network. It is therefore hardly possible to manually come up with good strategies to choose the appropriate heuristic depending on the graph features.

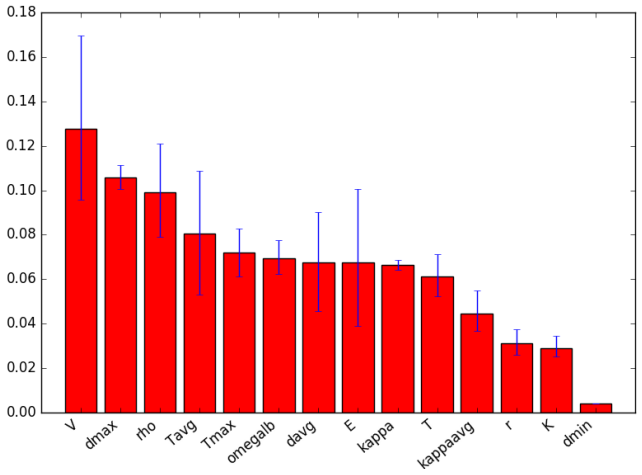

Fig. 3: Average feature importance of the 14 features based on Gini importance across all pair-wise random forest models. The error-bars indicate the 25th and 75th percentile.

As the calculation of instance features forms an important step in the application of algorithm portfolios, we briefly review the necessary calculation times in the following. Note that we keep this analysis fairly theoretical as the feature values were pre-computed and provided by an external source.

V and E can be retrieved in constant time as these are given in the header of the instance file. From these, the density ρ can also be computed in constant time. The maximum k-core number K can also be computed efficiently using a linear time $O(E)$ time algorithm [6]. In contrast to these, several other features take quadratic time or worse in the number of vertices or edges. For example, an efficient algorithm to date for computing the number of triangles has a complexity of $O(V^{2.376})$ [15]. With these, it is then possible to compute the local and global clustering coefficients, and other values.

As the previous portfolio investigations in Section IV-B are done using all 14 features, we now repeat the algorithm selection experiment using only the four features (K, V, ρ and E) that can be calculated efficiently. Surprisingly, even though AUTOFOLIO has no access to features which it believes to be important (d_{max} to r), the performance of AUTOFOLIO even improved by 191 in comparison to the full feature set, see Table II. From this experiment, we see that the exploitation of four (almost) immediately available instance features results in a portfolio performance that is comparable to (and actually even better than) a significantly more time-consuming one that requires the calculation of all 14 features.

V. CONCLUSIONS

We have studied the classical combinatorial optimization problem MINIMUM VERTEX COVER. We were able to improve the state-of-the-art local search solver FASTVC [9] on a large set of common benchmark instances. The optimal parameters of the solver depend on the kind of network analyzed. We derived an algorithm portfolio that performs better than any single variant. We also observed a high heterogeneity among the subclasses of instances.

Future Work: There are different avenues for future work on further improving algorithm portfolios for vertex cover. For example, one could use automatic per-instance algorithm configuration methods, such as ISAC [22] or Hydra [36]. These methods automatically determine complementary portfolios of parameter settings and avoid the need of expert knowledge to partition the instances. We simply use in our experiments the instance classes to partition our instances and tuned k on each class; however, the partition based on the classes is not reflected in the instance feature space, which makes it later harder for the algorithm selector to learn a mapping from instance features to optimized parameter settings. Therefore, automatic methods could maybe even find better instance partitions and hence improve the performance of an algorithm selector on vertex cover.

A second interesting future avenue would be to develop better instance features. So far, we used only static instance features (such as counting the number of edges or vertices); however, in some domains such as SAT, MIP or TSP, probing features (i.e., running for a short amount of time a solver and extract solving characteristics from it) are essential to predict the performance of a parameter setting well [28], [20]. This seems promising, and we have already explored such probing features for minimum vertex cover and traveling salesperson problem to learn a reactive restart strategies [33].

ACKNOWLEDGMENT

M. Lindauer was supported by the DFG (German Research Foundation) under Emmy Noether grant HU 1900/2-1. M. Wagner was supported by the Australian Research Council (DE160100850).

REFERENCES

[1] F. N. Abu-Khzam, M. A. Langston, P. Shanbhag, and C. T. Symons, "Scalable parallel algorithms for FPT problems," *Algorithmica*, vol. 45, no. 3, pp. 269–284, 2006.

[2] T. Akiba and Y. Iwata, "Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover," *Theor. Comput. Sci.*, vol. 609, pp. 211–225, 2016.

[3] C. Ansótegui, Y. Malitsky, M. Sellmann, and K. Tierney, "Model-based genetic algorithms for algorithm configuration," in *Proc. of IJCAI'15*, Q. Yang and M. Wooldridge, Eds., 2015, pp. 733–739.

[4] C. Ansótegui, M. Sellmann, and K. Tierney, "A gender-based genetic algorithm for the automatic configuration of algorithms," in *Proc. of CP'09*, 2009, pp. 142–157.

[5] M. Barthélemy, "Spatial networks," *arXiv*, vol. 499, pp. 1–101, Feb. 2011.

[6] V. Batagelj and M. Zaveršnik, "Fast algorithms for determining (generalized) core groups in social networks," *Advances in Data Analysis and Classification*, vol. 5, no. 2, pp. 129–145, 2011.

[7] L. Breimann, "Random forests," *MLJ*, vol. 45, pp. 5–32, 2001.

[8] E. Brochu, V. Cora, and N. de Freitas, "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *CoRR*, vol. abs/1012.2599, 2010.

[9] S. Cai, "Balance between complexity and quality: Local search for minimum vertex cover in massive graphs," in *Proc. of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, Q. Yang and M. Wooldridge, Eds., 2015, pp. 747–753.

[10] ——, "Local Search for Minimum Vertex Cover (Website)," http://lcs.ios.ac.cn/~caisw/MVC.html, 2015, [Online; accessed 11-September-2016].

[11] S. Cai, J. Lin, and K. Su, "Two weighting local search for minimum vertex cover," in *Proc. of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015, pp. 1107–1113.

[12] S. Cai, K. Su, C. Luo, and A. Sattar, "NuMVC: An efficient local search algorithm for minimum vertex cover," *Journal of Artificial Intelligence Research*, vol. 46, no. 1, pp. 687–716, 2013.

[13] S. Cai, K. Su, and A. Sattar, "Two new local search strategies for minimum vertex cover," in *Proc. of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[14] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proc. of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, 2016, pp. 785–794.

[15] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *Journal of Symbolic Computation*, vol. 9, no. 3, pp. 251 – 280, 1990.

[16] T. Friedrich, T. Ktzing, and M. Wagner, "A generic bet-and-run strategy for speeding up stochastic local search," 2017, accepted for publication. [Online]. Available: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14290

[17] F. C. Gomes, C. N. Meneses, P. M. Pardalos, and G. V. R. Viana, "Experimental analysis of approximation algorithms for the vertex cover and set covering problems," *Compututers and Operations Research*, vol. 33, no. 12, pp. 3520–3534, 2006.

[18] F. Hutter, H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Proc. of LION'11*, 2011, pp. 507–523.

[19] F. Hutter, H. Hoos, K. Leyton-Brown, and T. Stützle, "ParamILS: An automatic algorithm configuration framework," *Journal of Artificial Intelligence Research JAIR*, vol. 36, pp. 267–306, 2009.

[20] F. Hutter, L. Xu, H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods and evaluation," *Artificial Intelligence Journal (AIJ)*.

[21] R. Impagliazzo and R. Paturi, "On the complexity of k-SAT," *J. Comput. Syst. Sci.*, vol. 62, no. 2, pp. 367–375, 2001.

[22] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney, "ISAC - instance-specific algorithm configuration," in *Proc. of ECAI'10*, 2010, pp. 751–756.

[23] L. Kotthoff, "Algorithm selection for combinatorial search problems: A survey," *AI Magazine*, pp. 48–60, 2014.

[24] G. Lancia, V. Bafna, S. Istrail, R. Lippert, and R. Schwartz, *SNPs Problems, Complexity, and Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 182–193.

[25] A. Lancichinetti, M. Kivela, J. Saramaki, and S. Fortunato, "Characterizing the community structure of complex networks," *PLOS ONE*, vol. 5, no. 8, pp. 1–8, 08 2010.

[26] M. Lindauer, H. Hoos, F. Hutter, and T. Schaub, "Autofolio: An automatically configured algorithm selector," *Journal of Artificial Intelligence Research (JAIR)*, vol. 53, pp. 745–778, Aug. 2015.

[27] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

[28] E. Nudelman, K. Leyton-Brown, A. Devkar, Y. Shoham, and H. Hoos, "Understanding random SAT: Beyond the clauses-to-variables ratio," in *Proc. of CP'04*, 2004, pp. 438–452.

[29] W. Pullan, "Phased local search for the maximum clique problem," *Journal of Combinatorial Optimization*, vol. 12, no. 3, pp. 303–323, 2006.

[30] J. Rice, "The algorithm selection problem," *Advances in Computers*, vol. 15, pp. 65–118, 1976.

[31] S. Richter, M. Helmert, and C. Gretton, "A stochastic local search approach to vertex cover," in *KI 2007: Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, J. Hertzberg, M. Beetz, and R. Englert, Eds. Springer Berlin Heidelberg, 2007, vol. 4667, pp. 412–426.

[32] R. A. Rossi and N. K. Ahmed, "The Network Data Repository with Interactive Graph Analytics and Visualization (Website)," http://networkrepository.com, 2015, [Online; accessed 11-September-2016].

[33] M. Sellman and M. Wagner, "Learning a reactive restart strategy to improve stochastic search," in *Learning and Intelligent Optimization (LION)*, 2017, accepted for publication. [Online]. Available: http://cs.adelaide.edu.au/~markus/pub/2017lion-reactiveRestarts.pdf

[34] K. Smith-Miles, "Cross-disciplinary perspectives on meta-learning for algorithm selection," *ACM*, vol. 41, no. 1, 2008.

[35] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 06 1998.

[36] L. Xu, H. Hoos, and K. Leyton-Brown, "Hydra: Automatically configuring algorithms for portfolio-based selection," in *Proc. of AAAI'10*, 2010, pp. 210–216.

[37] L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown, "SATzilla: Portfolio-based algorithm selection for SAT," *Journal of Artificial Intelligence Research (JAIR)*.

[38] ——, "Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming," in *RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[39] ——, "Evaluating component solver contributions to portfolio-based algorithm selectors," in *Proc. of SAT'12*, 2012, pp. 228–241.

APPENDIX: List of 123 instances used in this research and the corresponding feature values as downloaded from the Network Repository [32].

instance	V	E	ρ	d_{max}	d_{min}	d_{avg}	r	T	T_{avg}	T_{max}	κ_{avg}	κ	K	ω_{lb}
bio-dmela	7393	25569	0.000935753	190	1	6	-0.05	8697	1	225	0.01	0.01	12	7
bio-yeast	1458	1948	0.00183401	56	1	2	-0.21	618	0	18	0.07	0.05	6	6
ca-AstroPh	18772	198050	0.00112411	504	0	21	0.21	4054323	215	11269	0.63	0.32	57	28
ca-citeseer	227320	814134	0.0000315	1372	1	7	0.07	8139894	35	5398	0.68	0.46	87	69
ca-coauthors-dblp	540486	15245729	0.000104378	3299	1	56	0.51	1332285174	2464	284067	0.8	0.66	337	218
ca-CondMat	23133	93439	0.000349231	279	1	8	0.13	520083	22	1615	0.63	0.26	26	23
ca-CSphd	1882	1740	0.000983039	46	1	1	-0.2	24	0	4	0.01	0	3	3
ca-dblp-2010	226413	716460	0.000028	238	1	6	0.3	4793937	21	5947	0.64	0.38	75	59
ca-dblp-2012	317080	1049866	0.0000209	343	1	6	0.27	6673155	21	8345	0.63	0.31	114	102
ca-Erdos992	6100	7515	0.00040399	61	0	2	-0.44	4830	0	99	0.07	0.04	8	7
ca-GrQc	5242	14484	0.0010544	81	0	5	0.66	144780	27	1179	0.53	0.63	44	44
ca-HepPh	12008	118489	0.00164363	491	0	19	0.63	10075497	839	39633	0.61	0.66	239	239
ca-hollywood-2009	1069126	56306653	0.0000985	11467	1	105	0.35	14748661845	13795	3977656	0.77	0.31	2209	53
ca-IMDB	896305	3782456	9.41657E-06	1590	1	8	-0.05	13074	0	78	0	0	24	3
ca-MathSciNet	332689	820644	0.0000148	496	1	4	0.1	1730334	5	1564	0.41	0.14	25	25
ca-netscience	379	914	0.0127598	34	1	4	-0.08	2763	7	75	0.74	0.43	9	9
ia-email-EU	32430	54397	0.000103449	623	1	3	-0.38	146976	4	1615	0.11	0.03	23	11
ia-email-univ	1133	5451	0.00850021	71	1	9	0.08	16029	14	261	0.22	0.17	12	12
ia-enron-large	33696	180811	0.000318501	1383	1	10	-0.12	2175933	64	17744	0.51	0.09	44	17
ia-fb-messages	1266	6451	0.00805625	112	1	10	-0.08	7473	5	242	0.07	0.04	12	5
ia-reality	6809	7680	0.000331351	261	1	2	-0.68	1200	0	52	0.02	0	6	5
ia-wiki-Talk	92117	360767	0.000085	1220	1	7	-0.03	2509401	27	17699	0.06	0.05	59	10
inf-italy-osm	6686493	7013978	3.1376E-07	9	1	2	0.25	22230	0	5	0	0	4	3
inf-openflights	2939	30501	0.00706468	473	1	20	0.05	853241	290	14954	0.4	0.25	56	20
inf-power	4941	6594	0.000540303	19	1	2	0	1953	0	21	0.08	0.1	6	6
inf-roadNet-CA	1957027	2760388	0.00000144	12	1	2	0.12	361476	0	7	0.05	0.06	4	4
inf-roadNet-PA	1087562	1541514	0.00000261	9	1	2	0.12	201336	0	8	0.05	0.06	4	3
rec-amazon	91813	125704	0.0000298	5	1	2	0.19	103023	1	9	0.27	0.35	5	5
sc-ldoor	952203	20770807	0.0000458	76	0	43	-0.12	567147375	595	1135	0.61	0.57	35	21
sc-msdoor	415863	9378650	0.00010846	76	0	45	-0.12	260626518	626	1135	0.62	0.58	35	21
sc-nasasrb	54870	1311227	0.000871056	275	11	47	0.07	35155833	640	3649	0.56	0.55	36	24
sc-pkustk11	87804	2565054	0.000665431	131	17	58	0.28	79665120	907	3250	0.59	0.46	48	36
sc-pkustk13	94893	3260967	0.00072429	299	17	68	-0.04	135474234	1427	8506	0.65	0.5	42	36
sc-pwtk	217891	5653221	0.000238149	179	1	51	0.25	167944767	770	2342	0.58	0.58	36	25
sc-shipsec1	140385	1707759	0.000173308	67	2	24	0.59	18446736	131	989	0.39	0.39	25	24
sc-shipsec5	179104	2200076	0.00013717	75	1	24	0.56	23188062	129	1165	0.36	0.37	30	24
soc-BlogCatalog	88784	2093195	0.000531099	9444	1	47	-0.23	153580167	1729	804436	0.35	0.06	222	40
soc-brightkite	56739	212945	0.000132294	1134	1	7	0.01	1483224	26	11517	0.17	0.11	53	36
soc-buzznet	101163	2763066	0.000539986	64289	1	54	2.85	92759544	916	1099041	0.23	0.01	154	26
soc-delicious	536108	1365961	0.00000951	3216	1	5	-0.07	1463916	2	8009	0.03	0.01	34	21
soc-digg	770799	5907132	0.000019885	17643	1	15	-0.09	188132376	244	396575	0.09	0.05	237	45
soc-douban	154908	327162	0.0000273	287	1	4	-0.18	121836	0	394	0.02	0.01	16	11
soc-epinions	26588	100120	0.000283267	443	1	7	0.06	479100	18	5151	0.14	0.09	33	16
soc-flickr	513969	3190452	2.41551E-05	4369	1	12	0.16	176313864	343	524525	0.17	0.15	310	53
soc-flixster	2523386	7918801	0.00000249	1474	1	6	-0.32	23691366	9	15190	0.08	0.01	69	31
soc-FourSquare	639014	3214986	0.0000157	106218	1	10	-0.71	64953009	101	1996522	0.11	0	64	30
soc-gowalla	196591	950327	0.0000492	14730	1	9	-0.03	6819414	34	93817	0.24	0.02	52	29
soc-lastfm	1191805	4519330	0.00000636	5150	1	7	-0.14	11838621	9	38000	0.07	0.01	71	14
soc-livejournal	4033137	27933062	0.00000343	2651	1	13	0.27	250658109	62	79740	0.26	0.14	214	87
soc-LiveMocha	104103	2193083	0.000404728	2980	1	42	-0.15	10084953	96	36914	0.05	0.01	93	13
soc-pokec	1632803	22301964	0.0000167	14854	1	27	0	97672374	59	29290	0.11	0.05	48	29
soc-slashdot	70068	358647	0.000146105	2507	1	10	-0.07	1205643	17	13382	0.06	0.03	54	23
soc-twitter-follows	404719	713319	0.00000871	626	1	3	-0.88	88617	0	1687	0.01	0	29	6
soc-twitter-follows-mun	465017	835423	7.7268E-06	811	1	3	-0.87	135064	0	2926	0.02	0	31	6
soc-wiki-Talk-dir	2394385	5021409	1.75173E-06	100032	1	4	-0.29	53666618	22	359836	0.07	0	185	25
soc-youtube	495957	1936748	0.0000157	25409	1	7	-0.03	7331658	14	151081	0.11	0.01	50	16
soc-youtube-snap	1134890	2987624	0.00000464	28754	1	5	-0.04	9169158	8	180820	0.08	0.01	52	16

instance	V	E	ρ	d_{max}	d_{min}	d_{avg}	r	T	T_{avg}	T_{max}	κ_{avg}	κ	K	ω_{lb}
socfb-A-anon	3097165	23667394	0.00000493	4915	1	15	-0.06	166819284	53	50241	0.1	0.05	75	9
socfb-American75	6386	217662	0.0106763	930	1	68	0.07	4400862	689	17432	0.24	0.16	57	9
socfb-Amherst41	2235	90954	0.0364327	467	1	81	0.06	2748795	1229	14640	0.31	0.23	64	10
socfb-Auburn71	18448	973918	0.00572371	5160	1	105	0	30329328	1644	198991	0.22	0.14	96	13
socfb-B-anon	2937612	20959854	0.00000486	4356	1	14	-0.11	155972349	53	36861	0.09	0.05	64	12
socfb-Baylor93	12803	679817	0.00829531	2109	1	106	0.08	20917197	1633	40865	0.21	0.16	97	11
socfb-BC17	11509	486967	0.00735347	1377	1	84	0.08	10548657	916	27761	0.21	0.14	70	12
socfb-Berkeley13	22900	852419	0.00325111	3434	1	74	0.01	16108779	703	69511	0.21	0.11	65	15
socfb-Bingham82	10004	362894	0.0072528	553	1	72	0.14	7163862	716	17652	0.22	0.16	58	10
socfb-BU10	19700	637528	0.00328563	1819	1	64	0.05	9227970	468	22496	0.19	0.12	51	12
socfb-Cal65	11247	351358	0.00555578	415	1	62	0.19	6361083	565	8243	0.23	0.16	64	19
socfb-CMU	6621	249959	0.0114056	840	1	75	0.12	6930159	1046	24065	0.28	0.19	70	15
socfb-Cornell5	18660	790777	0.00454239	3156	1	84	0.02	18337950	982	77234	0.22	0.14	85	13
socfb-Duke14	9885	506437	0.0103668	1887	1	102	0.07	15427674	1560	41982	0.25	0.17	86	9
socfb-FSU53	27737	1034802	0.0026902	2555	1	74	0.1	23575941	849	27950	0.22	0.15	82	24
socfb-Indiana	29732	1305757	0.00295433	1358	1	87	0.13	28173249	947	37292	0.2	0.14	77	11
socfb-Michigan23	30147	1176516	0.00258913	2031	1	78	0.12	24752640	821	43887	0.21	0.13	89	11
socfb-Mississippi66	10521	610911	0.0110391	1691	1	116	0.15	24761280	2353	60286	0.25	0.18	117	10
socfb-MIT	6402	251230	0.0122613	708	1	78	0.12	7111758	1110	27788	0.27	0.18	73	9
socfb-MU78	15436	649449	0.00545172	653	1	84	0.16	13945266	903	13592	0.21	0.15	71	11
socfb-NYU9	21679	715715	0.00304587	2315	1	66	0.01	10859937	500	39402	0.19	0.11	55	13
socfb-OR	63392	816885	0.000406564	1098	1	25	0.18	10504575	165	19467	0.22	0.15	53	10
socfb-Penn94	41536	1362220	0.0015792	4410	1	65	0	21623388	520	68097	0.21	0.1	63	17
socfb-Rutgers89	24580	784602	0.00259737	1642	1	63	0.08	13394787	544	21290	0.22	0.14	57	9
socfb-Stanford3	11586	568309	0.00846808	1172	1	98	0.1	17507610	1511	33177	0.24	0.16	92	10
socfb-Syracuse56	13653	543982	0.00583701	1340	1	79	0.06	13109538	960	18708	0.24	0.17	76	13
socfb-Temple83	13686	360795	0.00385274	1394	1	52	0.03	5592252	408	16993	0.21	0.13	62	10
socfb-Texas80	31560	1219650	0.00244909	1796	1	77	0.16	28868289	914	34619	0.22	0.15	79	12
socfb-Texas84	36364	1590651	0.00240588	6312	1	87	0	33535656	922	141050	0.19	0.1	82	9
socfb-UC33	16808	522147	0.00369672	1415	1	62	0.09	9670329	575	22686	0.23	0.15	59	15
socfb-UC61	13746	442174	0.00468061	687	1	64	0.2	10531566	766	22660	0.26	0.18	69	18
socfb-UCLA	20453	747604	0.00357445	1180	1	73	0.14	15339651	749	17534	0.21	0.14	66	9
socfb-UConn	17206	604867	0.00408654	1709	1	70	0.09	10252326	595	21515	0.2	0.13	66	9
socfb-UCSB37	14917	482215	0.00433448	810	1	64	0.18	9229971	618	16140	0.22	0.16	66	12
socfb-UCSD34	14948	443221	0.00396747	2165	1	59	0.03	7995471	534	25961	0.23	0.15	57	12
socfb-UF	35111	1465654	0.00237787	8246	1	83	-0.01	36440373	1037	159087	0.22	0.12	84	13
socfb-UGA50	24389	1174057	0.00394774	2926	1	96	0.09	30263433	1240	41323	0.21	0.14	87	12
socfb-UIllinois	30795	1264421	0.00266671	4632	1	82	0.03	28052775	910	66178	0.21	0.14	86	9
socfb-UNC28	18163	766800	0.00464901	3795	1	84	0	16275057	896	94190	0.2	0.12	83	11
socfb-USC35	17444	801853	0.00527057	4459	1	91	0.01	21485247	1231	106421	0.21	0.14	87	9
socfb-Vanderbilt48	8069	427832	0.0131437	2041	1	106	0.04	14224983	1762	60775	0.25	0.18	87	9
socfb-Virginia63	21325	698178	0.00307071	7206	1	65	-0.02	13453515	630	153243	0.22	0.11	71	9
socfb-WashU32	7755	367541	0.0122244	1794	1	94	0.04	10655649	1374	45503	0.26	0.17	78	9
socfb-Wisconsin87	23831	835946	0.00294403	3484	1	70	0	14586621	612	46791	0.21	0.12	61	12
tech-as-caida2007	26475	53381	0.000152321	2628	1	4	-0.19	109095	4	3813	0.21	0.01	23	16
tech-as-skitter	1694616	11094209	0.00000773	35455	1	13	-0.08	86309526	50	564609	0.26	0.01	112	55
tech-internet-as	40164	85123	0.000105539	3370	1	4	-0.18	189840	4	8513	0.21	0.01	24	16
tech-p2p-gnutella	62561	147878	0.0000756	95	1	4	-0.09	6072	0	17	0.01	0	7	4
tech-pgp	10680	24340	0.000426824	206	1	4	0.24	164934	15	2306	0.27	0.38	32	25
tech-RL-caida	190914	607610	0.0000333	1071	1	6	0.02	1364412	7	6044	0.16	0.06	33	17
tech-routers-rf	2113	6632	0.00297222	109	1	6	0.02	31212	14	588	0.25	0.23	16	16
tech-WHOIS	7476	56943	0.00203793	1079	1	15	-0.04	2347482	314	22271	0.49	0.31	89	49
web-arabic-2005	163598	1747269	0.000130568	1102	1	21	0.15	65011059	397	5884	0.65	0.95	102	102
web-BerkStan	685230	7600595	0.0000324	84290	1	22	-0.18	322306034	470	1004568	0.64	0.01	337	18
web-edu	3031	6474	0.00140985	104	1	4	-0.17	30174	9	523	0.56	0.27	30	16
web-google	1299	2773	0.00328924	59	1	4	-0.05	15204	11	189	0.35	0.53	18	18
web-google-dir	875713	5105039	1.33139E-05	6353	1	11	-0.05	69625427	79	48152	0.6	0.09	65	17
web-indochina-2004	11358	47606	0.000738119	199	1	8	0.12	630234	55	1475	0.71	0.57	50	35
web-it-2004	509338	7178413	0.000055341	469	1	28	0.99	1016652636	1996	93368	0.82	0.95	432	432
web-polblogs	643	2280	0.0110464	165	1	7	-0.22	9012	14	392	0.23	0.16	13	9
web-sk-2005	121422	334419	0.0000454	590	1	5	0.08	2980263	24	3481	0.23	0.47	82	53
web-spam	4767	37375	0.00329012	477	1	15	0	387051	81	6273	0.29	0.15	36	14
web-Stanford	281903	2312497	5.81987E-05	38626	1	16	-0.1	48741234	172	382002	0.59	0.01	87	17
web-uk-2005	129632	11744049	0.00139774	850	1	181	1	2513657160	19390	124251	0.99	1	500	500
web-webbase-2001	16062	25593	0.000198417	1679	1	3	-0.1	63345	3	1362	0.22	0.02	33	9
web-wikipedia2009	1864433	4507315	0.00000259	2624	1	4	0.05	6653808	3	12404	0.16	0.05	67	11