Proceedings of 2017 Tenth International Conference on Contemporary Computing (IC3), 10-12 August 2017, Noida, India

A Study on the Minimum Dominating Set Problem Approximation in Parallel

Mahak Gambhir and Kishore Kothapalli
International Institute of Information Technology, Hyderabad
Gachibowli, Hyderabad, India 500 032.
Email: mahak.gambhir@research.iiit.ac.in, kkishore@iiit.ac.in

***Abstract*— A dominating set of a small size is useful in several settings including wireless networks, document summarization, secure system design, and the like. In this paper, we start by studying three distributed algorithms that produce a small sized dominating sets in a few rounds. We interpret these algorithms in the natural shared memory setting and experiment with these algorithms on a multi-core CPU. Based on the observations from these experimental results, we propose variations to the three algorithms and also show how the proposed variations offer interesting trade-offs with respect to the size of the dominating set produced and the time taken.**

Keywords: Dominating set, parallel algorithms, Multicore CPU algorithms

I. Introduction

A dominating set of a graph $G(V, E)$ is a subset D of its vertex set V such that every vertex of the graph either belongs to D, or has at least one neighbour in D. In other words, the vertices in D cover the entire vertex set V. The minimum dominating set is one of the smallest such set possible for a given graph.

Dominating sets find numerous applications, in fields as varied as document summarization [8], design of secure systems for electrical grids [10], wireless networking [9], etc. Particularly in the distributed setting, a fast and effective method to find dominating sets is needed for applications like finding routes in ad-hoc mobile networks [11], constructing an efficient routing backbone [2], as well as partitioning a network into small local clusters [4].

For general graphs, it is known to be a NP-hard problem to obtain a minimum dominating set of a graph, even approximately. An important result in this regard was obtained by Raz and Safra (1997) [7] which established that no polynomial time algorithm can achieve an approximation factor better than $c \log |V|$ for some $c > 0$ unless $P = NP$. Almost ironically, a simple sequential greedy algorithm, which just picks the node covering the maximum number of yet uncovered nodes in each iteration, achieves a logarithmic approximation.

Over the years, in the sequential setting, the problem has been solved with better approximation ratios, and even exactly for various graph classes. One of the broadest category of graphs for which a high quality approximate solution can be obtained very efficiently, is the set of graphs with bounded arboricity as shown by Lenzen and Wattenhoffer [6, Algorithms 1 and 2]. The class of graphs of bounded arboricity subsumes a large class of graphs, including planar graphs, graphs of bounded tree-width, bounded genus, and the like.

The problem turns out to be much more difficult to handle in the distributed or parallel setting. Any solution has to address the critical problem of symmetry breaking where the algorithm has to make an intelligent choice over a set of potential options. In the case of the minimum dominating set (MDS) problem, the algorithm should not add too many candidate nodes to the MDS as the size of the resulting set can get unduly large.

In the distributed setting, Jia et al. [5] is the first to design a randomized distributed algorithm that runs in poly-logarithmic time, independent of the diameter of the network, and returns a dominating set of size within a logarithmic factor from optimal with high probability.

Starting with the work of Jia et al. [5], there have been a few recent improvements. The most recent solution for this is as given by Lenzen and Wattenhoffer [6] where two algorithms are presented.

One of them is designed specifically to work for graphs of bounded arboricity exploiting the fact that such graphs have a forest decomposition (also known as the Nash-Williams decomposition). This algorithm runs in $O(\log n)$ rounds and produces an MDS of size no more than $O(a^2)$ times the best possible MDS, with n being the number of nodes in the graph and a being the arboricity of the graph. The second algorithm from [6] is a greedy algorithm that can run in a distributed setting using $O(\log \Delta)$ rounds and produce an MDS with an approximation ratio of $O(a \log \Delta)$, where Δ is one more than the highest node degree in the graph.

The algorithms mentioned in above have comparable asymptotic complexities and approximation ratios. However, we are not aware of any experimental studies on these, or other algorithms for MDS. Such a study can help one consider the practical performance of the algorithms. We also go on to try and locate bottlenecks, and suggest heuristics to improve the practical performance of the algorithms.

Another reason to consider experimental studies, especially for the MDS problem is that it seems to offer a trade-off in terms of the time spent and the size of the MDS obtained. A smaller dominating set is particularly useful in applications such as identifying cluster-heads in a wireless sensor network [2]. In such a setting, each of the selected nodes functions as a relay to each of the nodes it covers and disperse the messages received to the destined member of the cluster with

978-1-5386-3077-8/17/$31.00 ©2017 IEEE

minimal delay. Therefore, cluster heads are required to be equipped with additional routing hardware and hence add to the cost of the network. Thus, limiting the dominating set size provides long term economic benefits in terms of establishing and maintaining the network.

In this paper, we undertake such a study focusing on various classes of real-world graphs. Therefore, apart from focusing on the algorithms as they were designed in [5], [6] and comparing them against one another, we also suggest variations that trade-off time taken and the size of the MDS obtained. In some cases, our variations improve on both the run time and the size of the MDS obtained.

We implement the algorithms from [5], [6] and our variations on top of these algorithms on an Intel E5-2650 multi-core CPU running 40 threads of OpenMP. Our results on this platform on a wide variety of real world graphs indicate that [6, Algorithm 2] outperforms the other two algorithms we study in this paper. Our variations to [6, Algorithm 2] show how to obtain a dominating set that is 3x smaller in nearly the same time.

The rest of the paper is organized as follows. In Section II, we describe the three algorithms we consider. This is followed by a comparison of these three algorithms on our graph dataset, and draw inferences from them. These form the basis for the variations we introduce to the algorithms, as described in Section III. We also assess the performance of these variations and discusses their trade-offs. Section IV provides the concluding remarks and future work in the direction.

II. Distributed Algorithms for an MDS

In this section, we provide a brief overview of the three distributed algorithms that we study for obtaining a small sized dominating set. We refer the reader to the original papers [5], [6] for an in-depth understanding of these algorithms.

A. The Local Randomized Greedy Algorithm (JRS)

This algorithm given by Jia et al. [5] extends the natural sequential greedy algorithm to find a small dominating set in a distributed setting. The essential step in the algorithm is to select nodes in the dominating set based on the residual degrees, which is to say, the number of uncovered neighbours, of vertices in the 2-hop neighbourhood. A straightforward application of this selection criteria leads to a large number of iterations (rounds) in some particular graphs, like caterpillar graphs, and those containing a clique. To counter this, the algorithm uses a parameter b, or the base, to which all degrees are rounded off. This leads to a dominating set $O(\log \Delta)$ times the optimal in expectation and $O(\log n)$ with high probability in logarithmic runtime, where Δ is the highest degree plus one.

The algorithm hence proceeds as follows. Each vertex of the graph starts by computing its residual degree to the closest next highest power of b, δ'_v. This is followed by each vertex finding the highest such degree Δ'_w in its two hop neighbourhood. If the vertex matches this highest with $\delta'(v) \geq \Delta'(w)$, it enters the candidate set C. Then, each of the yet uncovered nodes calculates its support value $s(v)$, which is the number of candidates, including itself, that can cover it. $med(v)$ is then defined as a measure of central tendency of the support values of all the neighbours of a candidate. Each node hence chooses to join the dominating set D with a probability $1/med(v)$. All the uncovered nodes repeat this procedure until there are none left, and we have the entire vertex set V covered. In our experiments, we use 2 as the value for the parameter b, which we found experimentally to give the best results on the considered graphs.

This algorithm provides a simple, yet effective way to compute the dominating set in a distributed setting and is one of the most widely employed method to achieve the same. It has a distributed round complexity of $O(\log n \log \Delta)$ rounds with high probability, when using the *median* for the $med(v)$ function, and exactly $O(\log n \log \Delta)$, when using the *mean*. Besides this, the *max* value can also be used, giving a theoretical time complexity of $O(\log^2 n \log \Delta)$. The algorithm obtains an MDS of size that is within an $O(\log \Delta)$ of the best possible size in expectation, and within $O(\log n)$ of the best possible size with high probability.

B. Lenzen-Wattenhofer Decomposition Algorithm (LW1)

This algorithm, [6, Algorithm 1] is designed for graphs of bounded arboricity and uses a forest decomposition of a graph where the out-degree of each node in the oriented forest thus obtained is bounded by $O(a)$, a being the arboricity of the graph. This topological information is exploited to restrict the number of nodes selected in dominating set by categorizing them as those covered by an incoming edge, those covered by an outgoing one, or those in the dominating set themselves.

The algorithm begins with the computation of a forest decomposition in the distributed setting as shown by Barenboim and Elkin [1]. This is achieved by iteratively identifying vertices of (residual) degree at most a. The set of vertices identified in the ith iteration can be called as the vertices belonging to the ith bucket. The input graph G is thus decomposed into a directed forest F, with each vertex v having at most a outgoing edges. A vertex w is defined as a parent of v if there is a directed edge (v, w) in F. Then, an auxiliary graph $H(V, T)$ is constructed such that neighbours in H share parents in F. Now, a maximal independent set I of the graph H is computed using standard techniques, and all the parents from F of each of the vertices in this set I are added to the dominating set D. The final dominating set is obtained by adding the still uncovered nodes to it, without any other consideration.

This algorithm obtains and O(a^2) sized-approximation to the minimum dominating set in $O(\log n)$ distributed rounds, utilizing the distributed Nash-William decomposition as well as MIS routines. The message size is limited, as is the work per iteration. Also, it has the flexibility of being implemented as a centralized algorithm with linear time complexity. Given that there is a lower bound of $\Omega(\log n)$ for obtaining a forest decomposition, the runtime of this algorithm cannot be improved further asymptotically.

Proceedings of 2017 Tenth International Conference on Contemporary Computing (IC3), 10-12 August 2017, Noida, India

TABLE I: LIST OF GRAPHS USED IN OUR EXPERIMENTS

CATEGORY	GRAPH	NODES	EDGES	Δ(G)	*a(G)*
Census/Survey	tx2010	914,231	2,228,136	121	3
	ca2010	710,145	1,744,683	141	3
Numerical	NLR	4,163,763	12,487,976	20	3
	packing	2,145,852	17,488,243	18	6
Planar	delaunay_n24	16,777,216	50,331,601	26	3
	delaunay_n23	8,388,608	25,165,784	28	3
Random Geometric	rgg_n_2_19_s0	524,288	3,269,766	30	9
	rgg_n_2_16_s0	65,536	342,127	27	7
Road Networks	europe_osm	50,912,018	54,054,660	13	2
	road_usa	23,947,347	28,854,312	9	2
Walshaw	auto	448,695	3,314,611	37	6
	m14b	214,765	1,679,018	40	6

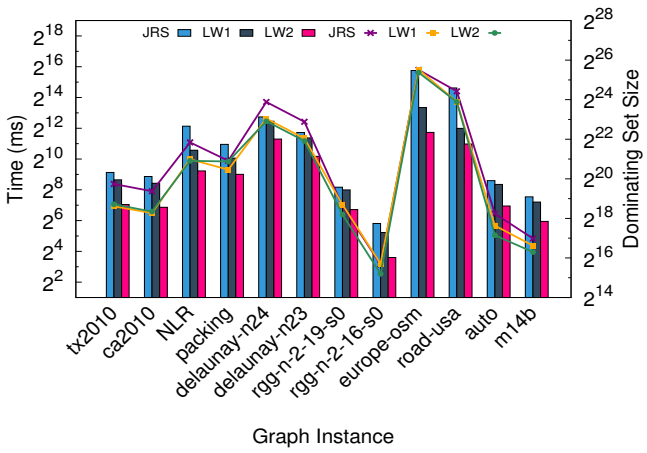

Fig. 1: Figure showing the time and dominating set size for the three original algorithms.

C. Lenzen-Wattenhofer Deterministic Algorithm (LW2)

This algorithm is presented in [6, Algorithm 2] and does not require knowledge of the arboricity of the input graph. The algorithm has some similarity to that of the JRS algorithm [II-A] in the way candidates are chosen and are added to the dominating set under construction.

The algorithm starts with each vertex calculating its residual degree, that is the number of uncovered vertices that can be covered by this vertex. If a vertex has a residual degree that is the largest in its 2-hop neighbourhood, then the vertex enters the candidate set C. Each uncovered vertex v now arbitrarily chooses a vertex in the candidate set, if any, that covers v. All such chosen vertices enter the dominating set. This process continues all the vertices are covered.

The algorithm is shown to finish in $O(\log \Delta)$ iterations. This algorithm does not require symmetry breaking and hence the size of the dominating set obtained is larger than that guaranteed by the JRS algorithm. The size of the dominating set is however shown to be within $O(a \log \Delta)$ of the best possible dominating set.

D. Experimental Results

1) Platform: All our experiments are carried out on an Intel E5-2650 multi-core processor. The E5-2650 is a two socket processor with each housing 10 cores. These cores support simultaneous multi-threading (SMT), allowing a total of 40 logical threads to run at a time. Each core is clocked at a frequency of 2.3 GHz, which can be boosted up to 3 GHz. Each processor is supplemented by a 25 MB shared L3 cache, in addition to a 64 KB L1 cache and a 256 KB L2 cache per core. The system has 128 GB of RAM supported by a memory bus with a 68 GB/s bandwidth.

2) Dataset: The graph dataset we use is a selection of real world graphs from the Sparse Matrix Collection [3] suite covering a varied classes of applications. Table I lists the graphs used along with some basic information. The graph "packing-500x100x100-b050" is referred to as *packing* due to space considerations.

3) Experimental Observations: We start by comparing the three algorithms presented above with respect to the time taken and the size of the dominating set obtained. The three algorithms are implemented as parallel algorithms using OpenMP threads on the CPU described above. We use 40 threads as doing so often resulted in better performance.

Figure 1 shows the comparative performance of the three algorithms. We see that the JRS algorithm almost always takes the most time, as much as 8 to 10 times than the other two in cases like `europe-osm`. Despite this, the size of the dominating set is almost always largest for it. Algorithm LW1 is often faster than the LW2 algorithm and the size of the dominating set produced by both these algorithms are nearly similar.

In our implementation of the JRS algorithm, we tried all three choices for computing the $med(v)$ value, but found that we were getting the best results in the least amount of time using *max* as the function. Hence, we used this version for all our comparisons and studies. We also observed that a large portion of the runtime is consumed in the randomization step required to pick a node with the desired probability. But in general, the quality of the MDS is lower than the other two algorithms which follow. The performance is poor on sparse graphs like planar and near-planar graphs, including road networks and delaunay graphs. One of the reasons for this is the slow convergence of the candidate selection process in sparse graphs.

For the LW1 algorithm, we observed that while it gives a better dominating set size than the JRS algorithm, the runtime is highly dependent on the arboricity a as well as the highest degree Δ. This can be explained by the fact that although the decomposition time depends only on the arboricity, the density of the auxiliary graph depends on the highest degree, constructing which takes almost 50% of the compute time. Besides this, a major portion of the time is taken by the forest decomposition. It performs better than the JRS algorithm in case of planar and near-planar graphs, which have a low arboricity, particularly in cases with higher number of nodes. This behaviour is consistent over all graph types.

Proceedings of 2017 Tenth International Conference on Contemporary Computing (IC3), 10-12 August 2017, Noida, India

The LW2 algorithm can be viewed as a de-randomized version of the JRS algorithm and hence we see that the time taken by the LW2 algorithm is better compared to that of the JRS algorithm. The small degree of the graphs in the dataset also meant that the size of the dominating set obtained is also better than the other two algorithms in study. The algorithm is seen to consume 50% of the time in finding the residual degree and the maximum of these residual degrees in a 2-hop neighborhood of every vertex.

From these observations, we see scope for improvement, possibly both in terms of the runtime, as well as in terms of the size of the dominating set obtained. For instance, the JRS algorithm suffers from slow convergence especially once the remaining graph becomes very sparse.

With respect to the LW1 algorithm, we notice that a large portion of the time is spent in obtaining the forest decomposition. We can however experiment with identifying techniques that reduce the size of the auxiliary graph. There is scope for improvement in terms of avoiding the decomposition, at least in part, as well as to limit the size of the auxiliary graph in terms of the edges involved.

In the case of the LW2 algorithm, while the time spent on finding the maximum degrees in the 2-hop neighbourhood cannot be easily reduced, as it is a very good heuristic for selecting *candidates*, we may attempt to reduce the time taken by the latter step. There might also be some scope for improving the quality of the MDS, as each of the nodes choosing a candidate can potentially add many distinct neighbours to the MDS unnecessarily.

III. Empirical Enhancements

Based on the observations from the experiments described in the previous section, we consider variations to each of the three algorithms described in Sections II-A–II-C.

A. Variations to the JRS Algorithm

1) Variation 1: (MIS) Conjugated Greedy Algorithm: Recall that in the JRS algorithm, in each iteration, each vertex computes its residual degree which is the number of uncovered nodes it covers. A vertex joins the dominating set if it has the maximum residual degree in its 2-hop neighbourhood, with ties being broken arbitrarily. This means that only one of the neighbours joins the dominating set in case both have the same residual degree.

We observed that, in the context of real-world graphs many of which are sparse, symmetry breaking based on the span value works well for the first few iterations of the algorithm, where the few very high degree vertices get selected and cover the majority of the graph. After the initial few iterations, the yet uncovered vertices usually have very low degrees. This leads to a very slow convergence and hence more time taken.

Thus, we propose a variation to the JRS algorithm that helps in faster convergence. We limit the number of iterations of the JRS algorithm and exit the JRS algorithm when the number of yet uncovered vertices fall below 50% of the original number of vertices in the graph. The partial dominating set obtained via the JRS algorithm is augmented with an MIS of the graph induced by the uncovered nodes. This set of vertices thus obtained can be seen to be a dominating set of the input graph. Since the remaining graph consists of low degree components, we also expect the MIS size to be close to the dominating set that may have been found by continuing the JRS algorithm.

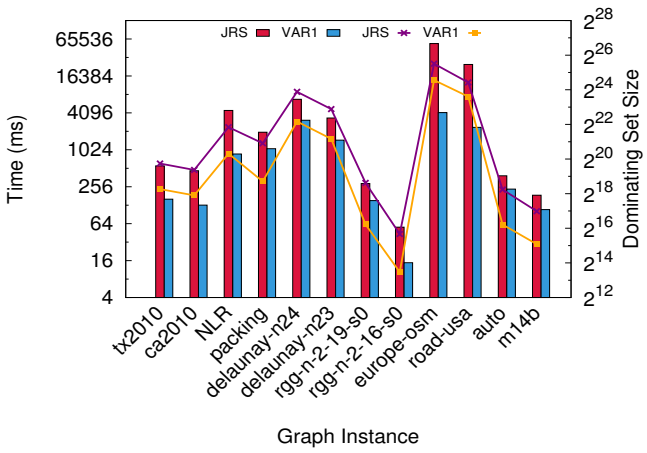

Fig. 2: Figure showing the run-time and dominating set sizes for the JRS algorithm and its variation.

2) Performance: Figure 2 compares the performance of Variation 1 and that of the JRS algorithm. For the dataset shown in Table I, the stopping criteria for the JRS algorithm as mentioned in Variation 1 is met in about three iterations. It is observed that the proposed variation is 4.3x faster than the JRS algorithm on average. In addition, the size of the dominating set obtained is 3.4x smaller. While road networks gain most in terms of time, with up to 13.3x reduction in time taken, random graphs are the greatest benefactors of result quality. Road networks, with a low number of high degree nodes, thus benefit the most. But the smaller sized dominating set is a result of all nodes left after the third iteration being considered as candidates.

B. Variations to the LW1 Algorithm

One possibility to improve the performance of the LW1 algorithm is to reduce the number of edges of the auxiliary graph. In order to reduce the number of edges in the auxiliary graph, we consider the removal of some of the highest degree vertices of the graph from the entire process of finding the dominating set, and instead adding them to the dominating set D a priori. We call these vertices as the *seed* vertices. The seed vertices can be identified from the forest decomposition computed as part of the LW1 algorithm. Some variations we studied on how to label a vertex as a seed vertex are described in the following.

1) Variation 2 : A seed of size (n/Δ)*:* It is apparent that we stand to benefit more from a larger seed set, which will reduce the computation to a larger extent, as compared to a smaller one. Hence, we first address the question of the maximum number of vertices we can mark as seed vertices without affecting the asymptotic size of the MDS obtained. We note that the size of any minimum dominating set can

Proceedings of 2017 Tenth International Conference on Contemporary Computing (IC3), 10-12 August 2017, Noida, India

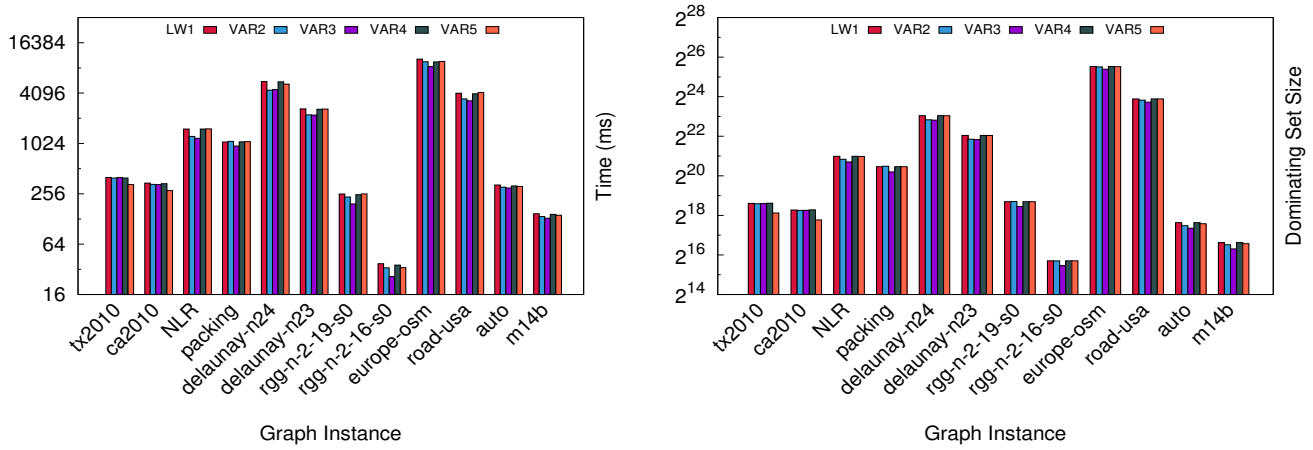

Fig. 3: Figure showing (a) the run-times and (b) the dominating set sizes for the LW1 algorithm along with its Variations 2–5.

help us in this regard. It is easy to see that in any graph, any minimum dominating set has to have at least $n/(\Delta+1)$ vertices, where Δ is the maximum degree of the graph. This follows from the observation that each vertex of in a graph can cover $\Delta+1$ other vertices at best, including itself.

In the light of this, picking a seed of size n/Δ is thus bound to affect the dominating set size only by a factor of $O(1)$, making it a good and reasonably large choice for the size of the seed set.

2) Variation 3 : A seed of size (n/Δ) with no mutual neighbours: We realized that at the cost of increasing some computation at the time of selecting the seed elements, a better choice can be made among the candidates. Instead of choosing all the elements in the latter buckets of the decomposition, we can afford to choose only those which are not themselves neighbours with each other. An efficient way to achieve this is to start with the last bucket, and find a maximal independent set of the nodes in this bucket. This way, we get a subset of its nodes which are representative of the entire bucket, as well as good candidates for a seed. This variation however may increase the runtime as it involves an MIS computation.

3) Variation 4 : A seed with size restricted to the last 1 bucket: In the above variations, seed vertices can be from multiple buckets. To keep the size of the dominating set small, it is advisable to finish processing higher numbered buckets before moving to a lower numbered bucket. This mechanism however can slow down the algorithm as it introduces a strict sequential ordering in how the seed vertices from various buckets are identified.

To avoid this overhead, in Variation 4, we pick seed vertices from the last k buckets. Since the highest degree vertices are in the very last bucket itself, it might suffice to just include all elements in it as the seed. The decomposition algorithm guarantees that the size of this set is limited to the arboricity a of the graph, and hence is safe for the asymptotic considerations.

4) Variation 5 : A seed with size restricted to the last 2 buckets: Continuing the line of Variation 4, in Variation 5, we set $k = 2$ thereby limiting seed vertices to the last two buckets.

5) Performance: We compare the performance of the LW1 algorithm with that of Variations 2–5. Figure 3 illustrates the results of this comparison for the time taken and the size of the dominating set obtained. We see that Variation 3 outperforms all others in terms of time as well as quality of result in most instances. Only in the case of survey graphs, we see *Variation 6* doing better both in terms of time and quality, while all others are almost equally good. This might be due to the relatively high degree for these particular graphs.

Thus, we see that a seed size of n/Δ with no mutual neighbours is a good approach to improve the performance of the LW1 algorithm. It can be seen from Figure 3 that this variation reduces the time taken by 30% on average without any significant increase the size of the dominating set obtained.

C. Variations to the LW2 Algorithm

The LW2 algorithm has the following drawback. As each yet uncovered vertex independently chooses a candidate that covers it, the resulting dominating set can be larger than what can be otherwise achieved. We therefore look for variations to reduce the size of the dominating set obtained without sacrificing on the time taken. Another possibility is to alter the conditions for selecting candidates in each iteration so as to reduce the number of iterations.

1) Variation 6: Modifying Candidate Selection: If the runtime is a concern, to aim for a smaller runtime, one way is to reduce the number of iterations. To this end, we increase the size of the candidate set, we say that a vertex can become a candidate if the vertex has a residual degree that is larger than the logarithm of the maximum of the residual degrees of vertices in its 2-hop neighbourhood. This criteria enhances the size of the candidate set and hence uncovered vertices are likely to find some candidate vertex that can cover it. This intuitively would therefore reduce the number of iterations.

2) Variation 7: An MIS of the candidates: If the goal is to reduce the size of the dominating set obtained, one possibility is to obtain an MIS of the chosen candidates in every round. Doing so may leave some vertices uncovered compared to the vertices covered by the corresponding round according

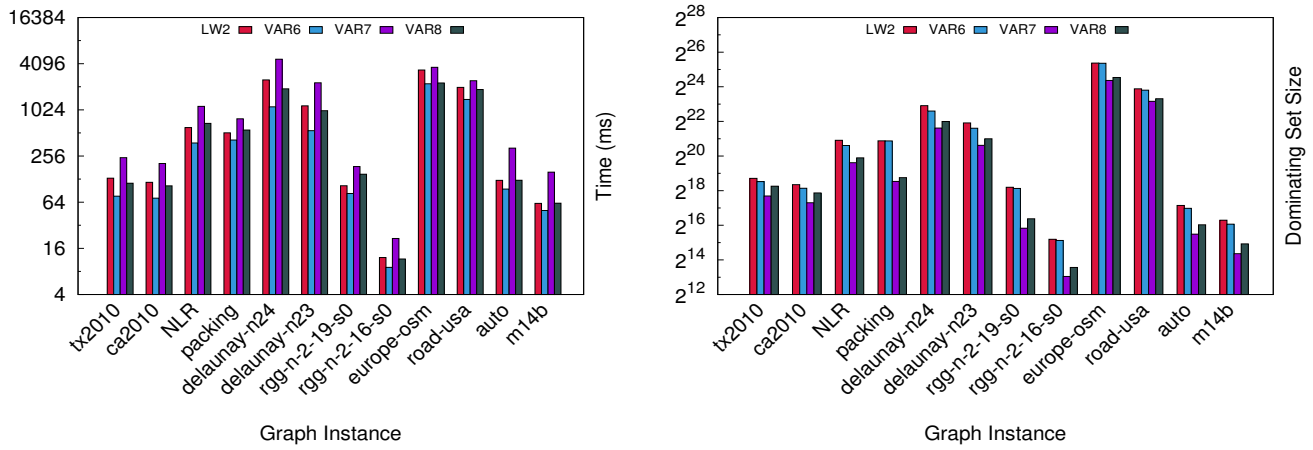

Fig. 4: Figure showing (a) the run-times and (b) the dominating set sizes for LW2 along with its variations.

to the LW2 algorithm. However, this would only mean that the algorithm now requires more iterations. This variation therefore offers a trade-off on the time spent by the algorithm and the quality of the obtained output.

3) Variation 8: Combining Variations 6 and 7: In this variation, we try a combination of the above two variations. On the one hand, we use the candidate selection criteria of Variation 6, and on the other hand, we add nodes to the dominating set after filtering away neighbours by using an MIS subroutine. We anticipate that the runtime of this variation as well as the size of the dominating set is between those of Variations 6 and 7.

4) Performance: Figure 4 shows how Variations 6–8 perform compared to the LW2 algorithm. We note that the three variations provide a good trade-off in terms of the size of the dominating set obtained and the time taken by the algorithm.

For instance, Variation 6 consumes lesser time than the LW2 algorithm but does not significantly lower the size of the dominating set produced. The time taken is on average 50% lesser compared to the LW2 algorithm. On the other had, Variation 7 is the slowest but this obtains the smallest sized dominating set of all the variations. The obtained dominating set is 3x smaller than the one obtained by the LW2 algorithm on average. Random graphs benefit the most, with a 5x better result. With no defined structure or properties, a choice of more candidates is indeed something random graphs can be expected to gain from the most.

Variation 8 provides a balance between Variations 6 and 7. While the time taken by Variation 8 is nearly the same as that of the LW2 algorithm, the obtained dominating set is 2.3x smaller on average. Thus, depending on the application requirement, one can pick one of the three variations.

IV. Conclusions

The paper started by implementing three popular distributed algorithms for finding a small dominating set in a given graph. The algorithms are interpreted as parallel algorithms and implemented as parallel algorithms in a shared memory environment. Based on the experimental observations of how these algorithms perform, the paper proposes variations to the algorithms so as to improve their practical performance and potentially offer trade-offs.

References

[1] L. Barenboim and M. Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using nash-williams decomposition. In *Proc. ACM PODC*, pages 25–34, 2008.

[2] B. Das and V. Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *Communications, 1997. ICC'97 Montreal, Towards the Knowledge Millennium. 1997 IEEE International Conference on*, volume 1, pages 376–380. IEEE, 1997.

[3] T. A. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

[4] Y. Fernandess and D. Malkhi. K-clustering in wireless ad hoc networks. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 31–37. ACM, 2002.

[5] L. Jia, R. Rajaraman, and T. Suel. An efficient distributed algorithm for constructing small dominating sets. *Distributed Computing*, 15(4):193–205, 2002.

[6] C. Lenzen and R. Wattenhofer. Minimum dominating set approximation in graphs of bounded arboricity. In *International Symposium on Distributed Computing*, pages 510–524. Springer, 2010.

[7] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 475–484. ACM, 1997.

[8] C. Shen and T. Li. Multi-document summarization via the minimum dominating set. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 984–992. Association for Computational Linguistics, 2010.

[9] I. Stojmenovic, M. Seddigh, and J. Zunic. Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. *IEEE Transactions on parallel and distributed systems*, 13(1):14–25, 2002.

[10] A. Teixeira, H. Sandberg, and K. H. Johansson. Networked control systems under cyber attacks with applications to power networks. In *American Control Conference (ACC), 2010*, pages 3690–3696. IEEE, 2010.

[11] J. Wu and H. Li. A dominating-set-based routing scheme in ad hoc wireless networks. *Telecommunication Systems*, 18(1):13–36, 2001.