Machine-Learning-Driven Matrix Ordering for Power Grid Analysis

Ganqu Cui*, Wenjian Yu*, Xin Li†, Zhiyu Zeng‡, Ben Gu‡
*BNRist, Dept. Computer Science & Tech., Tsinghua Univ., Beijing, China.
†iAPSE, Duke Kunshan Univ., Kunshan, China. ‡Cadence Design Systems, Inc., Austin, USA.
cgq15@mails.tsinghua.edu.cn, yu-wj@tsinghua.edu.cn, xinli.ece@duke.edu, zzeng@cadence.com, gxin@cadence.com

***Abstract*—A machine-learning-driven approach for matrix ordering is proposed for power grid analysis based on domain decomposition. It utilizes support vector machine or artificial neural network to learn a classifier to automatically choose the optimal ordering algorithm, thereby reducing the expense of solving the subdomain equations. Based on the feature selection considering sparse matrix properties, the proposed method achieves superior efficiency in runtime and memory usage over conventional methods, as demonstrated by industrial test cases.**

***Index Terms*—Classification, direct sparse solver, fill-reducing ordering, machine learning, power grid analysis.**

I. Introduction

Modern very large-scale integration (VLSI) design relies heavily on efficient power grid analysis [1]–[4]. Because the effect of voltage fluctuations becomes more and more significant, how to retain the switching speeds and satisfy the noise margin is challenging. This requests accurate and efficient simulation of the large-scale power grids.

To tackle the challenge of large-scale power grid analysis, many contributions have been developed. Direct solvers are based on the direct methods for solving sparse linear systems [5], [6], but cannot scale well with the problem size for large-scale power grids. Iterative solvers employ the Krylov subspace method with preconditioners. Although they are more competitive than direct solvers for large-scale cases, they often suffer from instability. An inappropriate preconditioner can cause the convergence issue, inducing unacceptable runtime and memory cost. A number of specialized methods have also been proposed, including domain decomposition method (DDM) [2] and matrix-oriented algebraic methods like sparse approximate inverse [3] and hierarchical matrix methods [4]. The DDM is based on graph partitioning. With the non-overlapping DDM technique, the whole power grid is partitioned into subdomains, and by solving the linear equations for the subdomains and the interface among them respectively the whole solution of problem can be obtained. With the DDM, each sub-problem can be robustly solved with direct sparse solver. Its parallelizability also makes it practically used.

Even with the DDM technique, the power grid analysis consumes a large portion of efforts for VLSI design nowadays. To find the worst-case scenario, a large number of simulations are performed for a power grid design which also endures successive modifications within the design cycle. So, it is crucial to reduce the computational expense of DDM-based power grid analysis, where the major task is to solve the subdomain equations. Notice that the efficiency of direct solver largely depends on matrix ordering, while finding the optimal ordering with minimum fill-ins is an NP-hard problem [7]. Although some ordering approaches, e.g. approximate minimum degree (AMD) [8], have been proposed, none of them is always optimal when applied to different matrices. On the other hand, the effectiveness of matrix ordering only depends on the matrix's sparse pattern. Hence, a better ordering is beneficial to power grid analysis during the whole design cycle as the power grid's structure and partitioning which determine the sparse patterns are usually fixed at the beginning.

In this work, we focus on the DDM based power grid analysis. A machine-learning-driven approach is proposed to select the optimal matrix-ordering methods among three widely-used candidates, to reduce the cost of solving the subdomain equations. Our proposed approach is composed of several steps. Firstly, a set of subdomain matrices collected from large-scale power grid analysis are tested with different matrix orderings. According to the fill-in reduction or peak memory usage the optimal ordering method is labeled for each matrix. Then, the set is spitted into two subsets, one as the training set and the other as the testing set. With the training set, two supervised learning methods based on support vector machine (SVM) and multilayer perception neural network (MLP-NN) respectively, are developed to train two possible classifiers for choosing the optimal ordering method. Our experiments on the testing set show that the accuracy of both classifiers is 94% or higher. And, with an effective feature selection step the proposed method brings little overhead. Compared with the conventional approach with a fixed ordering method applied to all cases, the proposed approach reduces the memory cost for the power grid analysis by about 30%.

II. Fill-Reducing for the DDM and Classification

We consider the DC analysis of power grid though the DDM is also capable of simulating the transient analysis [2]. The problem can be described using the nodal analysis as $\boldsymbol{G}\boldsymbol{x} = \boldsymbol{f}$, where $\boldsymbol{G}$ is the conductance matrix, $\boldsymbol{x}$ is the vector of node voltages and $\boldsymbol{f}$ denotes the loads of current source. Suppose that the power grid is partitioned into m subdomains by reordering the variables in $\boldsymbol{x}$. The nodes in the original system are classified into interior nodes of subdomains and interface nodes. So, the equation has the blocked sparse structure:

978-3-9819263-2-3/DATE19/©2019 EDAA

$$\begin{pmatrix} \boldsymbol{G}_1 & & & \boldsymbol{E}_1 \\ & \ddots & & \vdots \\ & & \boldsymbol{G}_m & \boldsymbol{E}_m \\ \boldsymbol{F}_1 & \cdots & \boldsymbol{F}_m & \boldsymbol{G}_\Gamma \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_m \\ \boldsymbol{x}_\Gamma \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_1 \\ \vdots \\ \boldsymbol{f}_m \\ \boldsymbol{f}_\Gamma \end{pmatrix} . \quad (1)$$

Here, the matrices $\boldsymbol{G}_1, \ldots, \boldsymbol{G}_m$ correspond to the m subdomains, and $\boldsymbol{G}_\Gamma$ corresponds to the interface nodes. Matrices $\boldsymbol{E}_i$ and $\boldsymbol{F}_i$ $(i = 1, \ldots, m)$ reflect the connections between the interface and the i-th subdomain.

Equ. (1) can be rewritten as

$$\begin{pmatrix} \boldsymbol{A}_D & \boldsymbol{E} \\ \boldsymbol{F} & \boldsymbol{A}_\Gamma \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_D \\ \boldsymbol{x}_\Gamma \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_D \\ \boldsymbol{f}_\Gamma \end{pmatrix}, \quad (2)$$

where $\boldsymbol{A}_D$ is the blocked diagonal matrix including $\boldsymbol{G}_1$, ..., $\boldsymbol{G}_m$. From the first equation, $\boldsymbol{x}_D$ can be expressed as

$$\boldsymbol{x}_D = \boldsymbol{A}_D^{-1}(\boldsymbol{f}_D - \boldsymbol{E}\boldsymbol{x}_\Gamma) . \quad (3)$$

Substituting (3) into the second equation of (2), we obtain

$$(\boldsymbol{A}_\Gamma - \boldsymbol{F}\boldsymbol{A}_D^{-1}\boldsymbol{E})\boldsymbol{x}_\Gamma = \boldsymbol{f}_\Gamma - \boldsymbol{F}\boldsymbol{A}_D^{-1}\boldsymbol{f}_D , \quad (4)$$

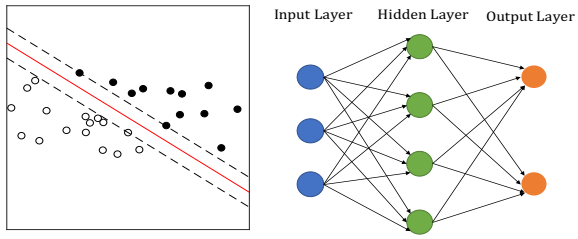
from which $\boldsymbol{x}_\Gamma$ can be solved. Notice that $\boldsymbol{A}_D^{-1}$ will not be explicitly computed. Due to the block diagonal property of $\boldsymbol{A}_D$ the computation is decomposed into solving the smaller equations with coefficient matrices $\boldsymbol{G}_1$, ..., $\boldsymbol{G}_m$. This can be easily parallelized, which is the reason why the DDM based approach is practical for large-scale power grid analysis.

The matrix $\boldsymbol{G}$ is symmetric positive definite (SPD) and matrix $\boldsymbol{G}_i$ $(i = 1, \ldots, m)$ inherits this property. Therefore, the solution of each subdomain equation is realized by Cholesky factorization of $\boldsymbol{G}_i$ followed by forward/backward substitutions. This direct method is robust, but it often suffers from large memory consumption caused by fill-ins during factorization. The memory issue may not be pronounced, if the subdomain is small enough. However, to restrict the size of interface which affects the non-parallel part in the solution of (4), the size of $\boldsymbol{G}_i$ cannot be very small. Therefore, appropriately ordering $\boldsymbol{G}_i$'s rows and columns which is able to reduce the fill-ins and thus the runtime and memory cost, becomes important for the solution of a subdomain equation.

The fill-reducing ordering problem is described as follows. *Given a sparse SPD matrix $\boldsymbol{G}_i$, find a row permutation $\boldsymbol{P}$ such that the number of nonzeros in (or the amount of work required to compute) the factorization of $\boldsymbol{P}\boldsymbol{G}_i\boldsymbol{P}^T$ is minimized.* It is equivalent to finding an order of the nodes in the undirected graph corresponding to the sparse matrix. As this is an NP-hard problem, heuristics are used to reduce fill-ins. Three basic strategies exist [7]: (1) ordering the nodes with minimum degree first and its variants, (2) nested dissection which recursively partitions the graph with node separators, (3) band reduction which constrains nonzeros to a small band region around the diagonal. The representative of the first strategy is the AMD algorithm [8], while that of the second is *metis* [9]. The reverse Cuthill-McKee algorithm belongs to the last one, used for matrices generated from structural mechanism problem. Although AMD is the most widely-used, the optimal ordering algorithm varies case by case. Consequently, there is a strong need to develop an efficient approach to automatically choose the optimal ordering algorithm in practice.

In machine learning, classification is a typical instance of supervised learning. It identifies which of a set of categories (labels) a new observation (data) belongs to, on the basis of a training set of data containing observations whose category (label) is known. Support vector machine (SVM) is a widely used machine-learning technique for classification. As shown in Fig. 1(a), it computes an optimal hyperplane to separate two groups of labeled data. If the sets of data to discriminate are not linearly separable, the kernel trick can be used. It maps the original data to a higher dimensional space presumably making the separation easier. After this mapping the inner product in the original space needs to be changed to a new one, which is called kernel function and used for training the SVM classifier.

Mutilayer erceptron (MLP) is a kind of simple artificial neural network, which can also be used for classification. With linear transfomation and nonlinear activation function, the MLP neural network (MLP-NN) maps inputs into outputs (see Fig. 1(b)). As deep-learning methods, MLP uses the error backpropagation to train its parameters. At the output layer, the softmax function is often used for classification.



(a) SVM (b) MLP-NN

Fig. 1. The classifiers based on SVM and MLP-NN.

III. Methodology

Our idea is to automatically select an appropriate matrix ordering algorithm from the three discussed in Section II, via a well-trained machine learning classifier. The key points include the training data with labels, the representation of a data (subdomain matrix here), and the technique for classification. For the training data, we can obtain them from industrial design of power grid, with the help of our industrial partner. Then, we shall test these matrices with different ordering approaches to get the label (which one is the best ordering algorithm). For our problem, the number of fill-ins and memory usage during the Cholesky factorization are the two criteria for judgment.

To make the proposed approach useful for the power grid analysis, its runtime efficiency at the inference stage is of major concern. The representation of a test data and the classifier both largely affect the efficiency. In the problem, data is the subdomain matrix, whose dimension may be several thousands or larger. Instead of directly depicting the matrix we consider its graph counterpart. Each matrix corresponds to an undirected graph, and the property of graph affects the choice

of fill-reducing ordering. So, we use a couple of features from the view point of graph to represent the matrix data. Finally, both SVM and MLP-NN are considered. They have relatively simple structure, and ensure less computation in the inference.

A. Feature Selection

We need to select representative features from data. The following list summarizes a number of important features which can capture the major characteristics of a graph.

- n: the number of nodes (matrix dimension);
- e: the number of edges (number of nonzeros in matrix);
- avg_d: the average degree of a node;
- $density$: the density of the sparse matrix, i.e. $\frac{2e}{n(n-1)}$;
- max_d: the maximum degree of a node;
- min_d: the minimum degree of a node;
- $diameter$: the maximum distance between two nodes.

We also use the SVM-recursive feature elimination (SVM-RFE) technique to further reduce the features [10]. It first trains a SVM classifier with all features. Then, it estimates the relatively contribution of each feature and the least important features are removed. This procedure is repeated until the loss on the classification accuracy is above a threshold. Finally, five features are selected: n, e, avg_d, max_d and $diameter$.

B. Implementation Details

Calculating $diameter$ is of $O(n^2)$ complexity, where n is the number of nodes. This could be expensive for practical use. So, we devise an approximate approach to calculating the graph diameter (Algorithm 1). There, "distance(s,t)" stands for the length of the shortest path between nodes s and t. The idea is that we find the farthest node from an initial node and record their distance. Then, we repeatedly change the initial node and calculate its farthest node until the corresponding distance does not increases. The loop in Algorithm 1 usually converges in a couple of iterations, so that its complexity is about $O(n)$. Empirical results have shown that the approximate diameter well captures the feature of graph.

Algorithm 1 Approximate calculation of the graph diameter

Input: Graph (V, E)
Output: Graph diameter $diameter$

```
s is a node with the minimum degree, diameter = 0;
t_0 = argmax_{t∈V} distance(s, t);
d_0 = distance(s, t_0);
while d_0 > diameter do
    diameter = d_0; s = t_0;
    t_0 = argmax_{t∈V} distance(s, t);
    d_0 = distance(s, t_0);
end while
```

For SVM classifier, the kernel trick with the Gaussian radial basis function is applied. For MLP-NN, we build a three-layer network which includes only one hidden layer, as shown in Fig. 1(b). The size of hidden layer is 16. The activation function is set to ReLU. The gradient decent algorithm is used to training the weights and bias in the network. The learning rate is fixed at 1e-3. The maximum iteration number to set to 100. All tolerances for convergence are 1e-3.

IV. Experimental Results

Two large-scale power grids from real design (with 40 million and 22 million nodes), provided by our industrial partner, are tested. They are partitioned separately, resulting in 2660 subdomain matrices in total. The dimension of these matrices ranges from 11810 to 36473. A program is written in Matlab2016b to test them with three ordering approaches: AMD, *metis* and RCM. For AMD and RCM the build-in commands "`amd`" and "`symrcm`" are used respectively. For *metis*, the Matlab interface of metis-5.1.0 [9] provided in SuiteSparse [11] is employed. Applying the generated orders followed by Cholesky factorization ("`chol`" in Matlab) we measure the number of fill-ins or peak memory usage to determine the optimal ordering algorithm for each matrix. For the tested matrices, we find out that it is always AMD or *metis*. So, this leads to a binary classification problem.

We first construct two datasets: Dataset1 with 2128 randomly selected matrices (80% of the whole set) as the training subset and the remaining matrices as the testing subset, Dataset2 with 1862 matrices (70% of the whole set) as the training subset and the remainder as the testing subset. With the number of fill-ins and memory cost as the criterion separately, the numbers of matrices whose best orderings are AMD or *metis* are listed in Table I. We can observe that although AMD wins for most cases there are about 29% of the matrices whose best ordering algorithm is *metis*. And, the criterion of memory is similar to fill-in, as the number of fill-ins largely determines the peak memory cost.

TABLE I
The Distributions of the Optimal Ordering Algorithms

	Criterion	Training Subset		Testing Subset	
		AMD	*metis*	AMD	*metis*
Dataset1	Fill-in	1508	620	384	148
	Memory	1494	634	377	155
Dataset2	Fill-in	1329	533	563	235
	Memory	1319	543	552	246

All experiments are carried out on a Linux server with two 8-core Intel Xeon E5-2630 CPUs @2.40GHz.

The classifiers are implemented in Python 2.7.6, with Scikit-Learn package. The SVM based classifiers for optimal fill-ins and peak memory are denoted as Classifier1 and Classifier2 respectively. To evaluate their performance, we measure four metrics: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). The inference results on the two testing subsets are listed in Table II. Here, "positive" and "negative" refer to AMD and $metis$, respectively. From the results we observe that the classifiers perform very well.

TABLE II
The Performance of the Binary Classifiers Based on SVM

Classifier1	TP	TN	FP	FN	Classifier2	TP	TN	FP	FN
Dataset1	381	128	3	20	Dataset1	374	128	3	27
Dateset2	562	208	2	27	Dataset2	551	208	2	38

More computational results on the two datasets are given in Table III. "Time" means the inference time for the whole testing subset. "Accuracy" means the fraction correct, i.e.

(TP+TN)/(TP+TN+FP+FN), which is the fraction of all instances correctly categorized. "Average Mem." is the average memory cost for factorizing a matrix. For comparison, the average memory costs corresponding to the situation where AMD ordering is always applied and an oracle situation are also provided. The latter stands for the ideal scenario where the best among the three orderings is applied for each matrix. However, the knowledge of the best ordering algorithm is hardly achievable in practice.

TABLE III
RESULTS OF THE SVM BASED CLASSIFIERS ON THE TESTING SUBSETS

		Time (s)	Accuracy (%)	Average Mem.(MB)	Average Mem.(MB) AMD	Average Mem.(MB) Oracle
Dataset1	Classifier1	12.7	95.7	2265	3232	2192
	Classifier2	12.7	94.4	2266		
Dataset2	Classifier1	19.1	96.4	2240	3251	2168
	Classifier2	19.1	95.0	2242		

From the results we observe the high accuracy of inference. Compared to the total runtime for factorizing all matrices in the testing subset of Dataset1 (ordered by AMD), which is 45.0 seconds, the inference costs much less time. Notice that it can be further reduced if the feature selection and SVM classifier are implemented in C. Compared with the conventional approach where the AMD algorithm is applied to all cases, the proposed method reduces the memory cost by 30%. It is only slightly larger than the oracle case. For a brute-force approach where both ordering algorithms (i.e., AMD and $metis$) and Cholesky factorizations are run, it costs 67.2 seconds. Accordingly, the proposed method only consumes 47.0 seconds. These are the benefits of the proposed approach for automatically selecting the matrix ordering algorithm.

The MLP-NN based classifiers for optimal fill-ins and memory usage are denoted as Classifier3 and Classifier4 respectively. Their computational results are listed in Table IV. From it we observe that their runtime is slightly smaller than that of the SVM based classiers, while the accuracy is similar. As the accuracy ranges from 93.8% to 97.2%, the proposed method reduces the average memory cost by 29.5% $\sim$ 31.2%.

TABLE IV
RESULTS OF THE NN BASED CLASSIFIERS ON THE TESTING SUBSETS

		Time (s)	Accuracy (%)	Average Mem.(MB)	Average Mem.(MB) AMD	Average Mem.(MB) Oracle
Dataset1	Classifier3	12.0	94.9	2272	3232	2192
	Classifier4	12.0	93.8	2280		
Dataset2	Classifier3	18.1	97.2	2238	3251	2168
	Classifier4	18.1	95.0	2243		

To evaluate how the size of training subset affects the performance of the classifier, we construct more datasets with different size of training subsets. The smallest training set only contains 5% of the whole subdomain matrices (i.e. 133 matrices). Then, for each one we trained the classifiers based on SVM and MLP-NN and evaluate their classification accuracy on the corresponding testing subsets. The results are plotted in Fig. 2. From it, we see that the accuracy slowly decreases as the training subset becomes smaller. However, even trained with only 133 matrices both classifiers can have an accuracy around 93%. This implies that the proposed approach is very stable in terms of training set size.

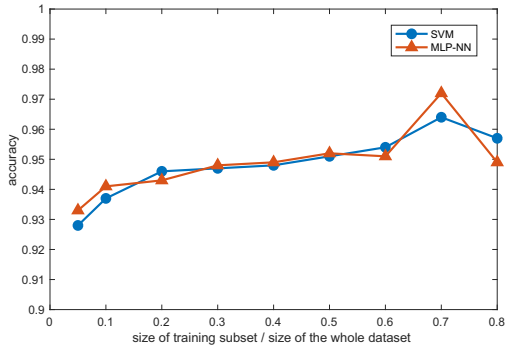


Fig. 2. The accuracy of the classifiers for optimal fill-ins trained with different sizes of training subsets.

V. CONCLUSIONS

An approach is proposed to choose the optimal matrix ordering algorithm for solving the subdomain equations in DDM based power grid analysis. It utilizes labeled matrix data to train a classifier based on support vector machine or neural network, for inferring the optimal ordering algorithm among several widely-used candidates. The experiments with industrial cases reveal that: 1) The classifier has a high accuracy of around 93% or higher, even with the training set as small as that including 133 matrices. 2) Compared with the brute-force approach running both AMD and $metis$ ordering algorithms prior to matrix factorization, it reduces the runtime of inference and factorization by at least 30%. 3) Compared with the conventional approach where the AMD algorithm is always applied, it reduces the memory cost by 30%.

REFERENCES

[1] H. Qian, S. Nassif, and S. Sapatnekar, "Power grid analysis using random walks," *IEEE Trans. Computer-Aided Design*, vol. 24, no. 8, pp. 1204–1224, 2005.

[2] K. Sun, Q. Zhou, K. Mohanram, and D. C. Sorensen, "Parallel domain decomposition for simulation of large-scale power grids," in *Proc. ICCAD*, Nov. 2007, pp. 54–59.

[3] S. Cauley, V. Balakrishnan, and C.-K. Koh, "A parallel direct solver for the simulation of large-scale power/ground networks," *IEEE Trans. Computer-Aided Design*, vol. 29, no. 4, pp. 636–641, 2010.

[4] J. M. Silva, J. R. Phillips, and L. M. Silveira, "Efficient simulation of power grids," *IEEE Trans. Computer-Aided Design*, vol. 29, no. 10, pp. 1523–1532, 2010.

[5] T. A. Davis and E. Palamadai Natarajan, "Algorithm 907: KLU, a direct sparse solver for circuit simulation problems," *ACM Trans. Math. Softw.*, vol. 37, no. 3, pp. 36–52, 2010.

[6] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam, "Algorithm 887: CHOLMOD, supernodal sparse cholesky factorization and update/downdate," *ACM Trans. Math. Softw.*, vol. 35, no. 3, p. 22, 2008.

[7] T. Davis, *Direct Methods for Sparse Linear Systems*. SIAM, 2006.

[8] P. R. Amestoy, T. A. Davis, and I. S. Duff, "Algorithm 837: AMD, an approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 381–388, 2004.

[9] G. Karypis, "METIS, a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," University of Minnesota, Tech. Rep., 2013.

[10] I. Guyon, et al., "Gene selection for cancer classification using support vector machines," *Machine Learning*, vol. 46, pp. 389–422, 2002.

[11] T. Davis, *SuiteSparse*, http://faculty.cse.tamu.edu/davis/suitesparse.html.