Variable reordering strategies for SLAM

Pratik Agarwal and Edwin Olson

***Abstract*— State of the art methods for state estimation and perception make use of least-squares optimization methods to perform efficient inference on noisy sensor data. Much of this efficiency is achieved by using sparse matrix factorization methods. The sparsity structure of the underlying matrix factorization which makes these optimization methods tractable is highly dependent on the choice of variable reordering; but there has been no systematic evaluation of reordering methods in the SLAM community.**

In this paper we evaluate the performance of various reordering techniques on benchmark SLAM data sets and provide definitive recommendations based on our results. We also compare these state of the art algorithms against our simple and easy to implement algorithm which achieves comparable performance. Finally, we provide empirical evidence that few gains remain with respect to variants of minimum degree ordering.

I. Introduction

Graph-based SLAM methods are becoming increasingly popular for mapping problem in robotics. In these approaches, each robot pose is represented as a node in the graph and each constraint as an edge. This approach was first proposed by Lu and Milios using the classic least squares formulation [1]. Their approach required the inversion of the full (dense) covariance matrix which is too expensive to compute for even modestly sized problems.

$\sqrt{SAM}$ showed that instead of the expensive matrix inversion, sparse matrix factorization with an efficient variable reordering could be used to solve the least squares SLAM formulation in a tractable way even for large problems [2]. It proposed using a sparse Cholesky or QR decomposition with column approximate minimum degree (COLAMD) variable reordering [3].

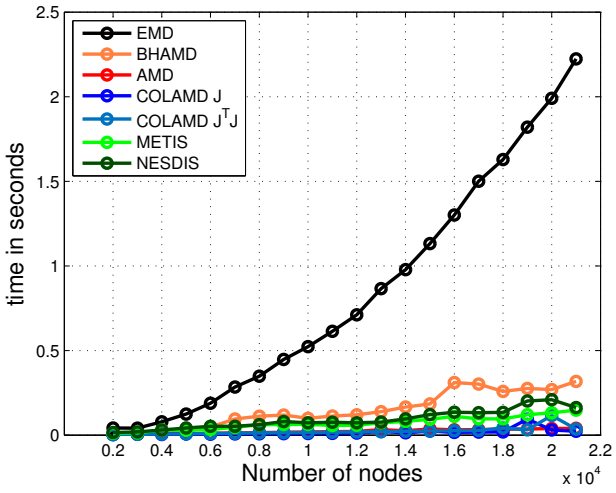
Variable reordering is equivalent to row and column exchanges on a matrix. A good variable reordering helps increase the sparsity structure of the factorized matrix. Different reordering techniques produce different sparsity structure, but all of them try to minimize fill-in.

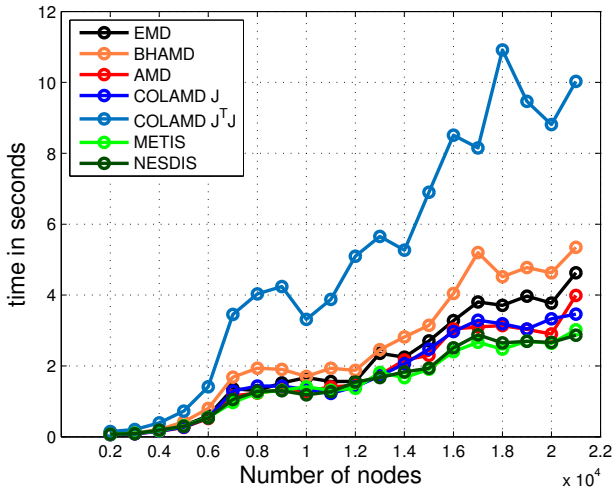
Reordering has a direct impact on the fill-in caused by matrix factorization, which affects the solve time for SLAM problems. The fact that affects of variable reordering algorithms have not been investigated despite their critical role in graph SLAM algorithms motivates this work.

The central contributions of this paper are:

1) We evaluate existing state of the art variable reordering strategies on standard SLAM benchmarks.
2) We propose an easy to implement reordering strategy that yields competitive performance.

The authors are affiliated with the department of Computer Science and Engineering, University of Michigan, 2260 Hayward Street, Ann Arbor, Michigan. {pratikag,ebolson}@umich.edu





(b) Solve time

Fig. 1: Reorder and solve time for different variable reordering methods. Results shown for simulated grid world data sets containing 2000 to 20000 nodes with an average node degree of 3. AMD and COLAMD J compute reordering quicker than other algorithms. METIS and NESDIS have the least solve time. Solve time for COLAMD $J^T J$ is worse than all the algorithms.

3) We provide evidence showing that few gains remain with respect to variants of minimum degree ordering.
4) We provide definitive recommendation for choosing a particular reordering given a SLAM graph.

II. Background

This paper investigates the sparsity of the matrices that result from formulating SLAM as a least-squares problem. We begin by showing the origin of these matrices.

A. Non linear SLAM using matrix factorization

Each edge (constraint) in a SLAM graph is a set of simultaneous equations f that relates a set of state variables x to some observed quantity z. The difference between the observed and the predicted value of an observation is the residual r_i. These constraints are generally associated with an uncertainty covariance Σ. Scaling the residual by the constraint's confidence Σ_i^{-1} results in the χ_i^2 error.

$$\chi_i^2 = (z_i - f_i(x))\Sigma_i^{-1}(z_i - f_i(x)) \tag{1}$$

The observation equations are non-linear due to the effect of rotation. We can linearize f_i around the current estimate x_0, as $f_i(x) \approx f_i(x_0) + J_i(x - x_0)$ where J_i is the Jacobian. Substituting $r_i = z_i - f_i(x_0)$ and $d = (x - x_0)$, we obtain:

$$\chi^2 \approx \sum_i (J_i d - r_i)^T \Sigma_i^{-1} (J_i d - r_i) \tag{2}$$

If we stack the J_i and r_i matrix, and create a block-diagonal matrix from the Σ_i^{-1} matrices, we can write:

$$\chi^2 \approx (Jd - r)^T \Sigma^{-1} (Jd - r) \tag{3}$$

The maximum likelihood solution can be obtained my minimizing χ^2, with $\Sigma^{-1} = LL^T$, this over-constrained system of equations can be written as:

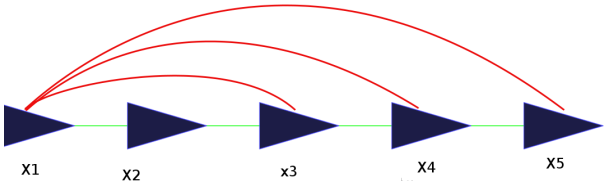
$$L^T J d = L^T r \tag{4}$$

The solution to such a system can be found through a QR decomposition. Alternatively we can form the normal equations by multiplying both sides with $J^T L$:

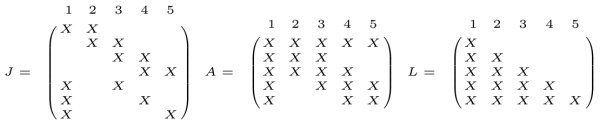
$$\begin{aligned} J^T \Sigma^{-1} J d &= J^T \Sigma^{-1} r \\ A x &= b \end{aligned} \tag{5}$$

The matrix $J^T \Sigma^{-1} J$ is the information matrix, which we will generally refer to as A in this paper. A is square and symmetric, with dimensions equal to the number of unknowns and the solution to the system can be found through a Cholesky decomposition. In contrast, J is tall and skinny: the number of rows in J is equal to the number of equations or edges in the graph and number of columns is equal to number of variables or nodes.

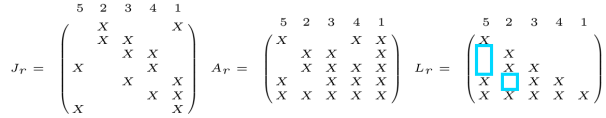
In both the case of QR and Cholesky factorization, performance is largely dictated by the sparsity of both the input (J or A) *and* the sparsity of the resulting factorization. The sparsity of the factorization depends on the ordering of the variables. We illustrate the dependence of variable ordering in the next section.



(a) Simple 5 robot SLAM graph



(b) Matrix structure using default ordering



(c) Matrix structure using a different ordering. The blue boxes show the sparse structure obtained due to reordering.

Fig. 2: A simple example showing the effects of reordering. Fig. 2(b) shows the matrix structure using original ordering. Fig. 2(c) shows the matrix structure after reordering. Reordering helped in the second case reduced fill in.

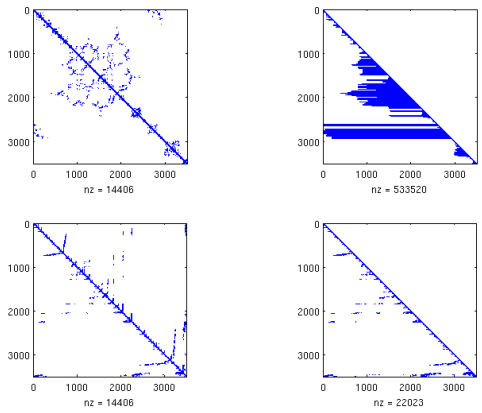
B. Variable Reordering

Variable reordering has been an active area of research in the field of Graph theory and sparse linear algebra. An efficient variable reordering can help the QR or Cholesky decomposition by minimizing the fill-in (see Fig. 3).

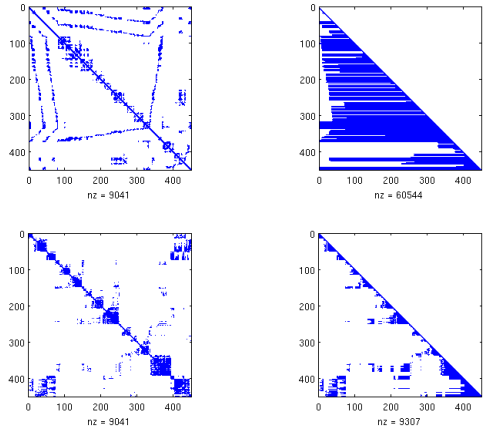
Consider a simple example as shown in Fig. 2(a). It consists of a SLAM graph with 5 poses and 6 edges. The green edges are odometry edges while red edges are loop closure edges. Fig. 2(b) shows the sparsity structure of the Jacobian matrix J and the matrix representing normal equations A. Each matrix column is labeled with the corresponding pose number. L is the result of factorizing A using a Cholesky decomposition. (This L is distinct form the one in Eqn. 4) Notice however the L factor is fully dense.

Now lets consider a different ordering of variables shown in Fig. 2(c). The reordered Jacobian and normal equations represent exactly the same problem: we have simply changed the order of the variables and equations. Critically, the sparsity structure of L_r is better than that of L: reordering J helped reduced fill-in. The goal of reordering algorithms is to minimize fill-in for L, speeding up the factorization process.

We can use permutation matrix P to transform between A and A_r. Permutation matrix are square binary matrices with exactly one entry 1 in each column and 0s elsewhere. Multiplying a matrix by a permutation matrix reorders the columns and rows. Since permutation matrices are orthogonal, their inverse is equal to their transpose. This trivially



(a) csw dataset fill-in



(b) Intel dataset fill-in

Fig. 3: Reordering helps to reduce fill-in. Top rows in each sub-figure shows the unordered matrix representing the normal equations (for csw and Intel dataset) and the corresponding lower triangular Cholesky factorized matrix. The bottom rows show the reordered matrix and the corresponding Cholesky factorized matrix. The fill-in for the reordered matrix is much smaller due to a good reordering.

allows to recover the solution to the original problem. If X_r is the solution to $P(A)x_r = Pb$, $x = P^T x_r$

Reordering methods try to find a permutation matrix P which maximizes the sparsity in L. Different reordering techniques do not change the solution but generate different sparsity patterns. Unfortunately, computing a reordering which minimizes fill-in is NP-complete[1][4]. Exact Minimum Degree (EMD) is widely used as a heuristic to minimize fill-in [5]. Various variants of EMD have been developed which are much faster than EMD itself, but achieve this due to approximations. In the next section we describe some of these variants including EMD itself.

[1]It is somewhat entertaining that matrix inversion is roughly $O(n^3)$, and the purpose of variable ordering is to reduce the run-time. However, we discover an NP-hard problem, which is harder than the original problem we set out to solve! Fortunately, an optimal variable ordering is not required; great gains can be obtained even with an imperfect ordering.

III. Variable ordering methods

A. Exact Minimum degree ordering

EMD is based on the observation that, when a variable is eliminated, a clique is formed between its neighbors. Each of the edges in this clique contributes to fill-in. Thus, the exact minimum degree ordering aims to minimize fill in by forming the smallest possible clique at each step.

Algorithm 1 Exact Minimum Degree

1: **while** variable nodes remain **do**
2: Choose a node y with minimum node degree
3: remove node y from the graph
4: add edges between all of y's neighbors
5: **end while**

EMD greedily picks the best degree node at each step; it does not "plan ahead" and thus does not generally produce the optimal fill-in for reducing ordering. Empirically however, it performs well. The order in which the nodes are eliminated is used for reordering the matrix. The pseudocode is shown in Alg. 1.

The following reordering algorithms are variants of the exact minimum degree algorithm.

- Approximate minimum degree ordering (AMD) [6]
- Column approximate minimum degree algorithm (COLAMD) [3]
- Nested Dissection (NESDIS) [7]
- Serial Graph Partitioning and Fill-reducing Matrix Ordering (METIS) [8]
- Bucket Heap AMD (BHAMD)

These algorithms are modifications of the Alg. 1 for faster computation, either using approximations or intelligent datastructures. The first three methods are state of the art sparse reordering algorithms available as a sparse matrix library – *SuiteSparse* [7]. METIS is available separately or through *SuiteSparse*. We additionally propose BHAMD as a simple alternative to these methods with comparable performance. We first summarise the existing reordering techniques and then go on to describe our implementation of BHAMD.

B. Approximate minimum degree ordering

AMD approximates EMD in line 4 of Alg. 1. EMD updates the node degree values of uneliminated node *exactly* after each elimination. AMD on the other hand computes an upper bound on this value. It uses a heuristic to determine if the node degree of a node needs to be updated even though its neighbor was eliminated. The heuristics are based on active submatrix and worst case fill-in.

C. Column approximate minimum degree ordering

COLAMD is an approximate reordering algorithm optimized for non-symmetrical matrices. While AMD operates only on symmetric matrices, COLAMD generates the column permutations without explicitly computing the normal

equations. This is especially helpful for QR factorization where the normal equations are never created. Though COLAMD was developed for non-symmetric matrices, it can also be used to order symmetric matrices. It is one of the reordering algorithms available to *CHOLMOD*, which is *SuiteSparse's* sparse Cholesky solver [9]. The use of COLAMD was first suggested by $\sqrt{SAM}$ and was later also used in used in iSAM1.0 [10].

D. Reordering methods based on Graph partitioning

METIS and NESDIS are both graph partitioning based reordering schemes [8]. They use the same graph partitioner but different minimum degree reordering on each subpartition.

METIS includes a reordering routine called *METIS_NodeND*. It first recursively partitions the graph and then computes a minimum degree ordering on each partition. METIS uses its own variant of minimum degree to reorder each partitioned subgraph.

NESDIS is the reordering strategy based on graph partitioners provided by *SuiteSparse*. It computes the partitions using METIS and then uses Constrained Column Approximate Min Degree ordering (CCOLAMD) to compute the ordering of individual partitions. In CCOLAMD, constraints can be added to specify specific elimination order. Certain nodes can be constrained to be eliminated before or after others. NESDIS uses CCOLAMD to specify that the partition nodes are to be eliminated only after all nodes in the subgraph partitioned by it are eliminated. CCOLAMD has been used in iSAM2.0 to specify that recent variables are to be eliminated last [11].

E. BHAMD

BHAMD, our variant of EMD, allows eliminating multiple nodes in a single iteration using heaps. A simple implementation using heaps, initializes the heap with all the nodes ordered by node degree. At each step, we extract the node with minimum node degree, eliminate it, and update its neighbors. This would be inefficient for a graph with a large number of nodes. Updates and reinsertion would scale logarithmically with the number of nodes.

Instead, we initialize the heap with lists where each list is a bucket of nodes. Each bucket contains nodes with the same node degree. At each iteration, we extract the list form the heap, with minimum node degree. This list contains putative nodes whose node degree may have changed since last insertion due to other nodes getting eliminated. Hence we evaluate each node in that bucket. Nodes whose true node degrees are equal to or less than the current minimum are eliminated as shown in Alg. 2. If not, the node is inserted back into a list containing nodes of equivalent node degree into the heap. The advantage of BHAMD over EMD is being able to eliminate multiple nodes in a single search step.

IV. Experimental Evaluation

A. Experiments and Datasets

We evaluated 6 reordering techniques discussed previously on standard SLAM datasets. In all of our evaluations, we have shown the results of using COLAMD both with J and with $J^T J$. We compare the time required to reorder, factor and solve the graph. For all of these methods, a symbolic matrix was created to exploit the block structure of the constraints. A symbolic matrix encodes the sparsity pattern of a matrix without encoding the literal values. In short, each value in the symbolic matrix is a boolean encoding whether the value is zero or non-zero. In the SLAM domain, we additionally collapse variable nodes that are closely related, such as the x, y, and theta variables associated with a 2D robot pose [2]. By collapsing these nodes, we reduce the storage requirements and computational costs of computing an ordering.

Reorder time corresponds to time spent computing the variable reordering while solve time is the time to compute the sparse Cholesky decomposition. Total time includes

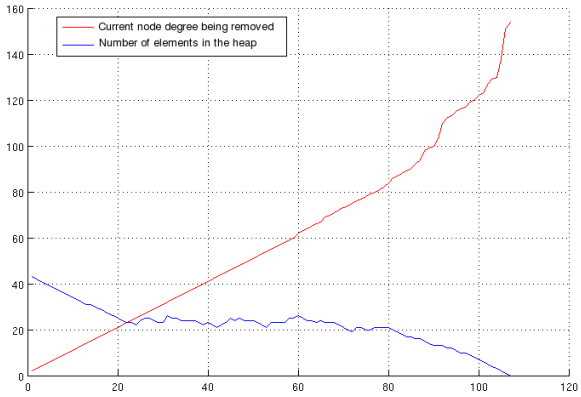


Fig. 4: Elimination pattern for w10000 data set using BHAMD. The x axis represents the number of heap queries. At each query a list is eliminated. y axis shows the statistics of each list in the heap. The total number of elements at any point in the heap does not cross 45 even though we begin with 10000 nodes. It also shows the maximum node degree for the last node eliminated is little less 160.

Algorithm 2 BHAMD

```
create a set of lists where each list contains nodes with equal number of neighbors
add all lists into a min heap - MH
while MH is not empty do
    bestList = getMinList from MH
    min = MH.getMinKey
    for each node nd in bestList do
        if nd.nodeDegree ≤ min then
            remove nd
            add pairwise edges between neighbors of nd
        else
            update nd.nodeDegree and push it back into the correct list
        end if
    end for
end while
```

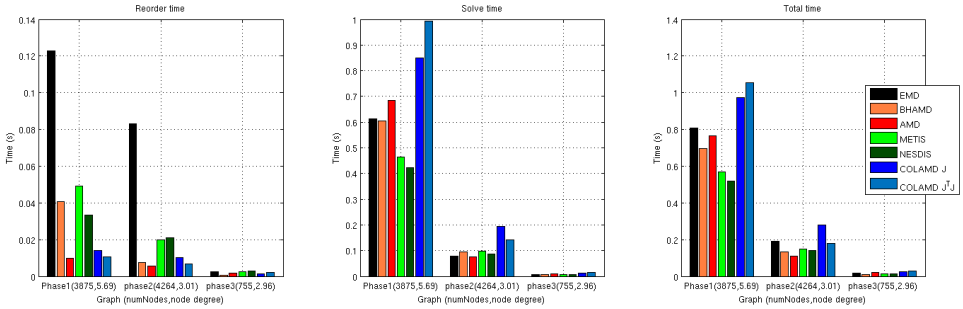


Fig. 5: Reorder, solve and total time on the MAGIC multi robot datasets. Note: Scales are different across figures.

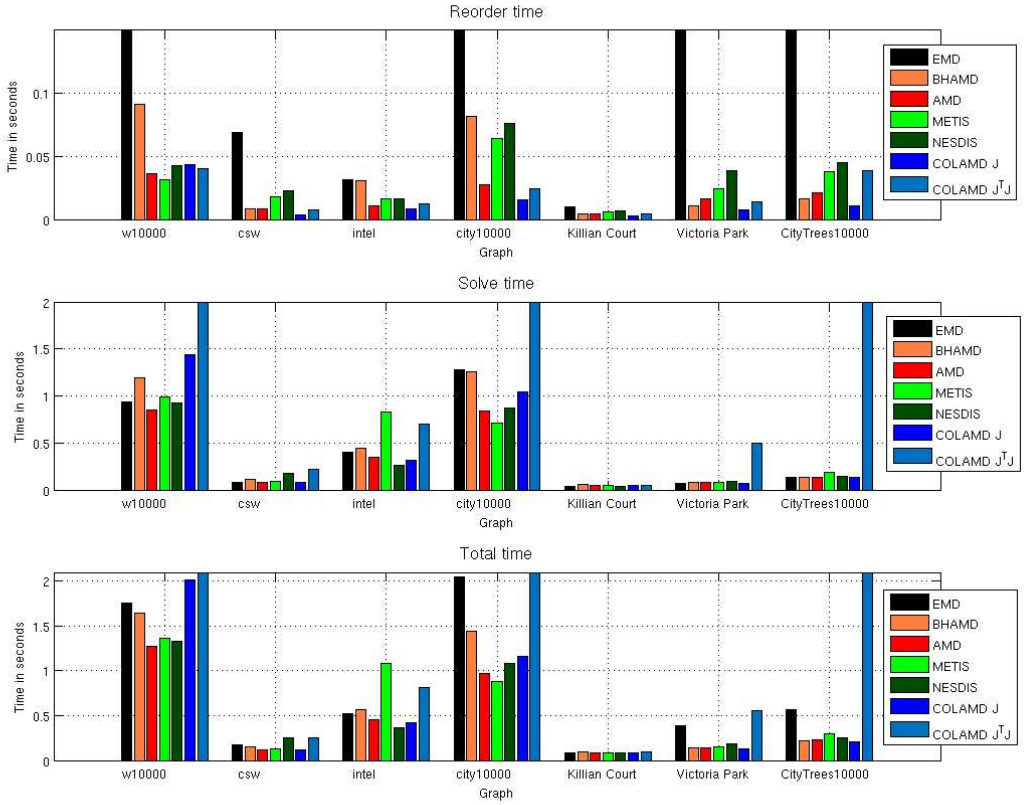


Fig. 6: Reorder, solve and total time on benchmark datasets. The y axis shows time in seconds. The same results are tabulated on the next page. Some bars are truncated to show details. All numerical results are available in Fig. 7

reorder, solve and other miscellaneous operations such as code instrumentation, creating the symbolic matrix and unpermuting the solution. These require similar time across all methods. The values shown are averaged over 10 runs.

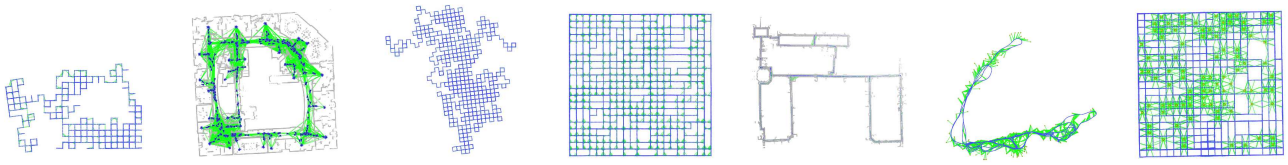
All our tests were run on Intel core i7 computer using the sparse Cholesky solver in April Robotics Toolkit (ART) [12]. The *SuiteSparse* libraries were interfaced with ART using Java Native Interface bindings.

B. Datasets

For our evaluation, we have used standard simulated datasets such as the Manhattan3500 (also known as the csw dataset) [13], world10000 [14], City10000 and CityTree10000 [10]. We created new simulated grid world datasets with varying numbers of nodes. Real-world datasets include Victoria park, Intel [15] and Killian Court [16]. These real world data sets are preprocessed by a front end to generate loop closures. We have also included graphs from our MAGIC datasets, including large multi-robot graphs [17].

C. Variable ordering optimization

Given a variable ordering, it is interesting to ask whether that ordering can be incrementally improved. In other words, are the variable orderings computed by minimum-degree



(a) csw (b) Intel (c) w10000 (d) City10000 (e) Killian Court (f) Victoria Park (g) CityTree10000

Algorithm	csw	Intel	w10000	City10000	Killian Court	Victoria Park	CityTrees10000
P, E, N	3500,5598,3.2	875,15605,35.7	10000,64311,12.86	10000,20687,4.14	1462,6571,8.98	7120,10608,2.98	10100,14442,2.86
EMD	68, 84, 178	31, 398, 524	477, 940, 1757	663, 1277, 2052	9, **40, 80**	267, **68**, 389	360, 132, 562
BHAMD	8, 109, 148	30, 439, 564	91, 1195, 1645	81, 1253, 1439	4, 54, 92	10, 76, 138	16, 133, 219
AMD	7, **79,** 114	11, 344, 455	36, **855, 1269**	27, 838, 967	4, 44, 81	16, 75, 143	21, 135, 225
METIS	18, 89, 132	16, 827, 1087	**31**, 985, 1366	63, **706, 876**	6, 43, 83	24, 78, 154	37, 191, 294
NESDIS	22, 173, 247	16, **258**, **362**	42, 930, 1334	76, 872, 1078	7, 42, 82	38, 89, 181	45, 146, 250
COLAMDJ	**3**, 82, **113**	**7**, 314, 417	43, 1440, 2019	**15**, 1042, 1167	**3**, 49, 86	**7**, 70, **129**	**10, 129, 209**
COLAMDJ^TJ	7, 214, 250	12, 701, 811	40, 4285, 4718	24, 2746, 2883	4, 53, 90	13, 495, 561	39, 4168, 4353
NZ fill-in							
EMD	183493	303679	1306555	1147945	**93195**	**202252**	**290428**
BHAMD	194769	314444	1386612	1183202	105408	207378	290587
AMD	**178151**	275713	**1202455**	1026152	96928	207321	303743
METIS	204128	281152	1412363	1028779	97895	228574	369146
NESDIS	193519	**260179**	1307333	**1007935**	94302	226262	333494
COLAMDJ	181161	268261	1304646	1083914	100437	203441	290435
COLAMDJ^TJ	299219	412798	2567700	1957268	107729	510184	808918
Reorder/solve							
EMD	0.82	0.08	0.51	0.52	0.24	3.88	2.73
BHAMD	0.08	0.07	0.08	0.06	0.09	0.14	0.12
AMD	0.10	0.03	0.04	0.03	0.10	0.21	0.16
METIS	0.20	0.02	0.03	0.09	0.14	0.31	0.20
NESDIS	0.13	0.06	0.05	0.09	0.17	0.44	0.31
COLAMDJ	0.04	0.03	0.03	0.02	0.06	0.11	0.08
COLAMDJ^TJ	0.03	0.02	0.01	0.01	0.08	0.03	0.01

Fig. 7: Comparison of various reordering algorithms on standard SLAM datasets (P:number of poses, E:number of edges, N:average node degree). The top section shows the reorder, solve, total time in ms, taken by each algorithm for each data set. The middle section shows the fill-in in L after decomposition and the bottom section shows the ratio between reorder and solve time. COLAMDJ^TJ has maximum fill-in and worst solve time.

algorithms approximately locally optimal? Or, conversely, could small additional changes result in significantly better variable orderings?

In some cases, a variable ordering algorithm makes fairly arbitrary choices: there may be several variables that have the same degree. Different variants might select amongst these differently, with different long-term consequences to fill-in. Are some tie-breaking strategies better than others?

To explore this question, we generated minimum-degree orderings using AMD, then iteratively attempted to improve the orderings. In a framework resembling a genetic search, we randomly permuted pairs of variables and solved the resulting system, computing a fitness score as a function of the fill-in.

The results of our experiments showed that only very small reductions in fill-in resulted from this optimization process. Of course, many permutations resulted in worse fill-in, but even the best fill-in was reduced by at most 0.5% as shown in Table I.

This experiment provides evidence that small modifications to minimum-degree type variable ordering algorithms may not be able to achieve significant improvements. An experiment like this cannot be conclusive, but it does suggest that different ideas may be required to improve performance.

Dataset	csw	Intel	KillianCourt
Best nz improvement %	0.5	0.01	0.3

TABLE I: Datasets where fill-in could be reduced using genetic search.

V. Conclusions

Timing runs of all reordering algorithms on standard data sets as well as our new grid world datasets are shown in Fig. 1, Fig. 5, Fig. 6 and Fig. 7. We summarize the broad trends observed:

1) With the exception of COLAMD applied to J^TJ and EMD, the performance of the methods are comparable: They produce similar-quality variable reordering, and the variation in the computational time required to compute those orderings is generally very small in comparison to the solve time.

2) COLAMD with J^TJ produces a poor variable reordering, leading to significant fill-in and slow solve time: The second section in Fig. 7 shows that fill-in for COLAMD J^TJ is higher than that of other algorithms. It produces on an average more than 2 times the fill-in compared to other algorithms. Fig. 1(b) shows that the solve time for COLAMD

$J^T J$ is much slower. COLAMD must be used only on the Jacobian and not on the normal matrix $J^T J$.

3) BHAMD, despite its simplicity, is generally competitively in terms of computational time and the quality of the reordering: It is a viable option, particularly if the more complex packages are not available.

4) Further development of minimum-degree algorithm variants may not produce significant reductions in fill-in: Our attempts to search for better orderings than those computed by AMD consistently resulted in best-case reductions in fill-in of less than 0.5%

VI. Final Words

In this paper, we compared reordering and solve times for various sparse matrix reordering algorithms on standard SLAM data sets, including simulated, real and multi robot graphs. We showed a simple algorithm like BHAMD, having comparative performance to state of the art methods. We provided empirical results which proved that small variations in minimum degree algorithms do not decrease fill-in drastically.

References

[1] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, no. 4, pp. 333–349, April 1997.

[2] F. Dellaert, "Square root SAM," in *Proceedings of Robotics: Science and Systems (RSS)*, Cambridge, USA, June 2005.

[3] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, "Algorithm 836: Colamd, a column approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, pp. 377–380, September 2004. [Online]. Available: http://doi.acm.org/10.1145/1024074.1024080

[4] M. Yannakakis, "Computing the minimum fill-in is NP-complete," *SIAM Journal on Algebraic and Discrete Methods*, vol. 2, no. 1, pp. 77–79, Mar. 1981.

[5] W.F.Tinney and J.W.Walker, "Direct solutions of sparse network equations by optimally ordered triangular factorization," in *Proceeding of IEEE*, no. 55, 1976, pp. 1801–1809.

[6] 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, pp. 381–388, September 2004. [Online]. Available: http://doi.acm.org/10.1145/1024074.1024081

[7] T. Davis, "Suitesparse." [Online]. Available: http://www.cise.ufl.edu/research/sparse/SuiteSparse//

[8] G. Karypis and V. Kumar, "Metis - unstructured graph partitioning and sparse matrix ordering system, version 2.0," Tech. Rep., 1995.

[9] 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, pp. 22:1–22:14, October 2008. [Online]. Available: http://doi.acm.org/10.1145/1391989.1391995

[10] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Fast incremental smoothing and mapping with efficient data association," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Rome; Italy, April 2007. [Online]. Available: http://www.cc.gatech.edu/ dellaert/pub/Kaess07icra.pdf

[11] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree," *Intl. J. of Robotics Research, IJRR*, vol. 31, pp. 217–236, Feb 2012.

[12] E. Olson, "The APRIL robotics toolkit," 2010. [Online]. Available: http://april.eecs.umich.edu

[13] ——, "Robust and efficient robotic mapping," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, June 2008.

[14] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard, "A tree parameterization for efficiently computing maximum likelihood maps using gradient descent," in *Proceedings of Robotics: Science and Systems (RSS)*, Atlanta, GA, USA, 2007.

[15] A. Howard and N. Roy, "The robotics data set repository (radish)," 2003. [Online]. Available: http://radish.sourceforge.net/

[16] M. Bosse, P. Newman, J. Leonard, and S. Teller, "Simultaneous localization and map building in large-scale cyclic environments using the Atlas framework," *International Journal of Robotics Research*, vol. 23, no. 12, pp. 1113–1139, December 2004.

[17] E. Olson, J. Strom, R. Morton, A. Richardson, P. Ranganathan, R. Goeddel, M. Bulic, J. Crossman, and B. Marinier, "Progress towards multi-robot reconnaissance and the MAGIC 2010 competition," *Journal of Field Robotics*, September 2012.