Performance of general graph isomorphism algorithms

Sara Voss
Coe College, Cedar Rapids, IA

I. Problem and Motivation

Graphs are commonly used to provide structural and/or relational descriptions. A node in a graph can be anything as long as it can be valuated either quantitatively or qualitatively. The relational information is stored in the set of edges [8]. From the point of view of pattern analysis and recognition, the most important graph-processing challenge is matching graphs for comparison [3]. One of the most relevant sub-problems in this category is matching a sample with a reference graph. One example would be comparing a communication structure with a reference topology as would be done in performance skeleton creation.

Performance prediction of a parallel application is challenging for foreign environments and is difficult to model. A performance skeleton is a short running program that models the dominate communication structure and computational behavior of a parallel application. Monitored execution of a performance skeleton in a new environment gives an estimate of the performance of the original application in the new environment. The execution time of the performance skeleton need only be scaled to obtain the estimated execution time of the application the skeleton represents. The basic procedure for constructing a performance skeleton consists of: 1) generating a trace for each process, 2) converging the traces into a single logical program trace, 3) compressing the logical trace by identifying the loop structure, and 4) converting this information into an executable program [13].

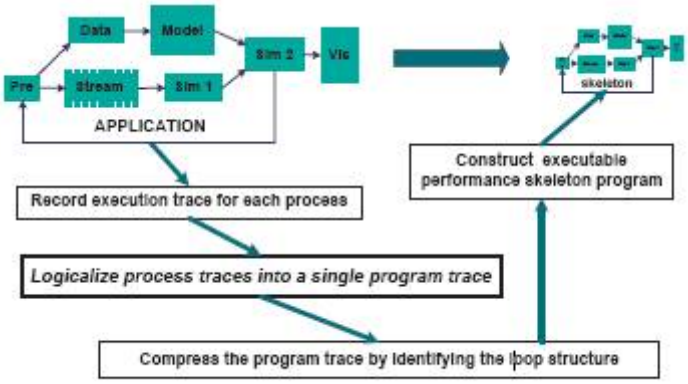


Figure 1 - Performance skeleton construction.

The highlighted logicalization step in Figure 1 is the focus of [12]. Logicalization converges the individual process traces into a single logical program trace. To obtain the logical trace the underlying communication structure must be identified. The process traces are represented in an adjacency matrix and compared with a library of known topologies. The topologies are narrowed down based on a set of invariants such as the number of nodes, the number of edges, the degree of the nodes, and the set eigenvalues of the communication matrix. After these invariants are tested, only a few library topologies remain, and in many cases only one topology remains as a candidate for the communication structure. Once the candidate topologies are identified, exact graph matching must be performed to guarantee the match. Exact graph matching is an isomorphism problem.

In the performance skeleton research, the VF2 algorithm was chosen to do the exact graph matching; however, the runtime of the algorithm was slower than expected in some cases. This study was created to explore factors that may affect the execution of the algorithm. One of the main theories to explore was whether process labeling affects runtime.

II. Background and Related Works

There is no known polynomial time algorithm for graph isomorphism and it is well known that subgraph isomorphism is an NP-complete problem. It is still open to question whether graph isomorphism is NP-complete. The exponential time requirements of isomorphism algorithms have been a major obstacle for applications that require the use of large.

Considerable research efforts have been put into improving the performance of isomorphism algorithms in terms of computational time and memory requirements. Some algorithms manage to reduce computational complexity by imposing topological restrictions (e.g. planar graphs [7] and trees [1]). However, in many circumstances graphs are not guaranteed to fall into any specific topological category. This is the case with communication structures in performance skeleton creation.

When deciding to work with a graph-based method, choosing an appropriate algorithm for the given application is essential. Only a few papers [6, 2] compare the general graph isomorphism algorithms in terms of key performance indices such as time requirements. This study examines five general graph isomorphism algorithms and compares their time requirements over a variety of topologies, sizes, and node labelings.

The algorithms explored are Ullmann, Schmidt and Druffel (SD), VF, VF2, and Nauty. Brute forcing graph isomorphism results in a dept-first search tree. Ullmann reduces the search space through backtracking [11]. SD is another backtracking algorithm; however, it relies on information contained in the distance matrix representation of the graphs [10]. VF is based on a depth-first strategy with a set of rules to prune the search tree [4]. VF2 works on the same concept but stores the information in more efficient data structures [5]. Nauty is based on a set of transformations that reduce the graphs to a conical form on which isomorphism tests are faster [9]. These algorithms, while exponential strive to be efficient in practice.

III. Uniqueness of Approach

Previous research on the performance of graph isomorphism algorithms had little focus on highly structured graphs. In [3], it is admitted highly structured graphs can be some of the most difficult to match, but the research only tested one highly structured topology. The underlying communication structure of parallel applications tends to be a low degree stencil [13]. Consequently, this study concentrates on highly structured topologies.

The topologies tested are 2D grids, 2D tori, six-point and eight-point stencils on the two dimensional topologies, 3D grids, 3D tori, binary trees, and hypercubes. The 2D topologies are square, the 3D topologies cubes, and the binary trees are complete.

Various sizes of each topology are created ranging from seven to over 16,000 nodes. Within each size, five different versions of the graph are created. One has a well-ordered node numbering, meaning the nodes are labeled in a systematic way. The others have a degree of randomness introduced. Randomness is introduced by swapping a percent of node labels. The percents tested are 25, 50, 75, and 99. Testing is performed between the well-ordered graph and a graph with randomness introduced. A test between the well-ordered graph and itself is executed as a baseline.

Degrees of randomness are introduced to test whether or not node labeling has an effect on the runtime of the algorithms. Figure 2 shows a systematic renumbering of the nodes and its affect.

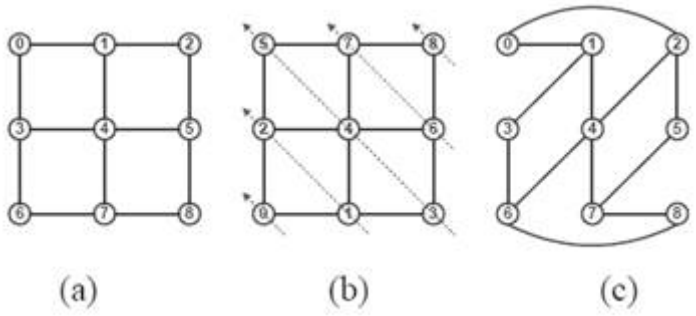


Figure 2 - Illustration of the affect of node labeling.

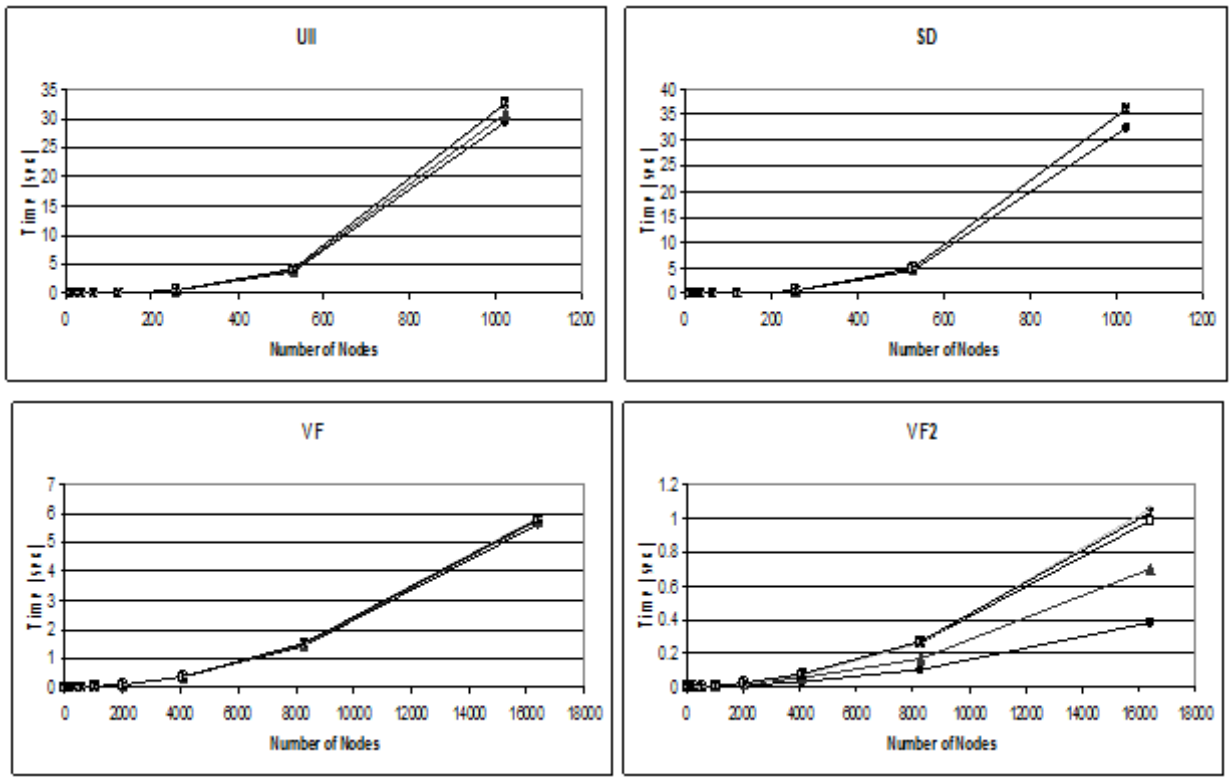
A topology is easy to identify when the nodes are assigned labels in a well-defined manner, but in general it is a harder problem. This is illustrated with the examples in Figure 2. The figure shows nine-node 2D grids. Figure 2(a) shows the underling 2D grid assigned labels in row-major order. However, in Figure 2(b) the nodes are numbered diagonally with respect to the underlying 2D grid pattern starting with the lower left hand corner. If the graph in Figure 2(b) were laid out in row major order, it would appear as Figure 2(c). The underlying 2D grid topology is easy to identify in the scenarios represented by Figures 2(a) and 2(b) but harder to identify in Figure 2(c). Identification would be even more difficult if the node labeling followed an unknown or arbitrary order.

Coding is done using a combination of C, C++, and Python. Ullmann, SD, VF, and VF2 are included in a graph isomorphism library, VFlib2.0, which is available online. VFlib2.0 was developed at the University of Naples "Federico II" and was chosen because of the ease of integration with C++ and it implements multiple algorithms. Nauty is available separately and is coded in C.
Each graph is stored as an adjacency list in a file. C++ code linked to VFlib2.0 reads these files. When a file is read, the graph is constructed using the built-in data structures provided by the library. Then a library provided function must be called to prepare the graphs for the isomorphism algorithm, from here on referred to as state construction. Then the isomorphism algorithm can be executed. Timings are taken using gettimeofday() throughout the code. These times are used to calculate the runtime of the algorithm being tested. Test for isomorphism is done by all five of the algorithms on pairs of isomorphic graphs. The runtimes are compared to determine which algorithm performs fastest for each given topology.

IV. Results and Contributions

Nauty could not be used to test for isomorphism because of the way isomorphism was created. This research is interested in whether or not node labeling affects the runtime of the algorithms so node labels are swapped around. Nauty produces the canonical form of graphs. If the canonical form of two graphs are the same then the graphs are isomorphic. However node labels swapped stay swapped in the canonical form. Hence the graphs are found to be non-isomorphic by Nauty.

The results of Ullmann, SD, VF, and VF2 are presented below. When examining the time it took to execute each step: loadtime of the graphs, state construction, and algorithm run it was discovered that not only did the algorithm time vary but so did the state construction. Due to the variation in state, results presented are a combination of state construction and algorithm time. There was insufficient memory on the computer used for testing to execute Ullmann on any graph beyond the 1,000-node test case. SD's performance significantly dropped off after the 2,000-node test case (just over six minutes for 2,000 nodes to just over an hour for 4,000 nodes). Therefore and testing beyond 2,000 nodes is not performed for SD due to time constraints.



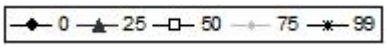


Figure 3 - Results on 2D grids.

Figure 3 shows the results of the algorithms on 2D grids. All grid topologies and the hypercube had similar results. Ullmann, SD, and VF were not significantly affected by node labeling. In all topologies tested, VF2 was significantly affected by node labeling. VF and VF2 were able to complete testing through 16,000 nodes while Ullmann and SD were only able to execute testing through 1,000 nodes.

The difference between the grid results and the results on binary trees is VF is affected by node labeling. However it still executed in under 12 seconds for 16,000 nodes. When it came to the torus topologies, Ullmann was significantly affected. It could only complete testing on smaller graphs and was affected by node labeling.

The most interesting results were for a 6-point stencil on 2D tori.

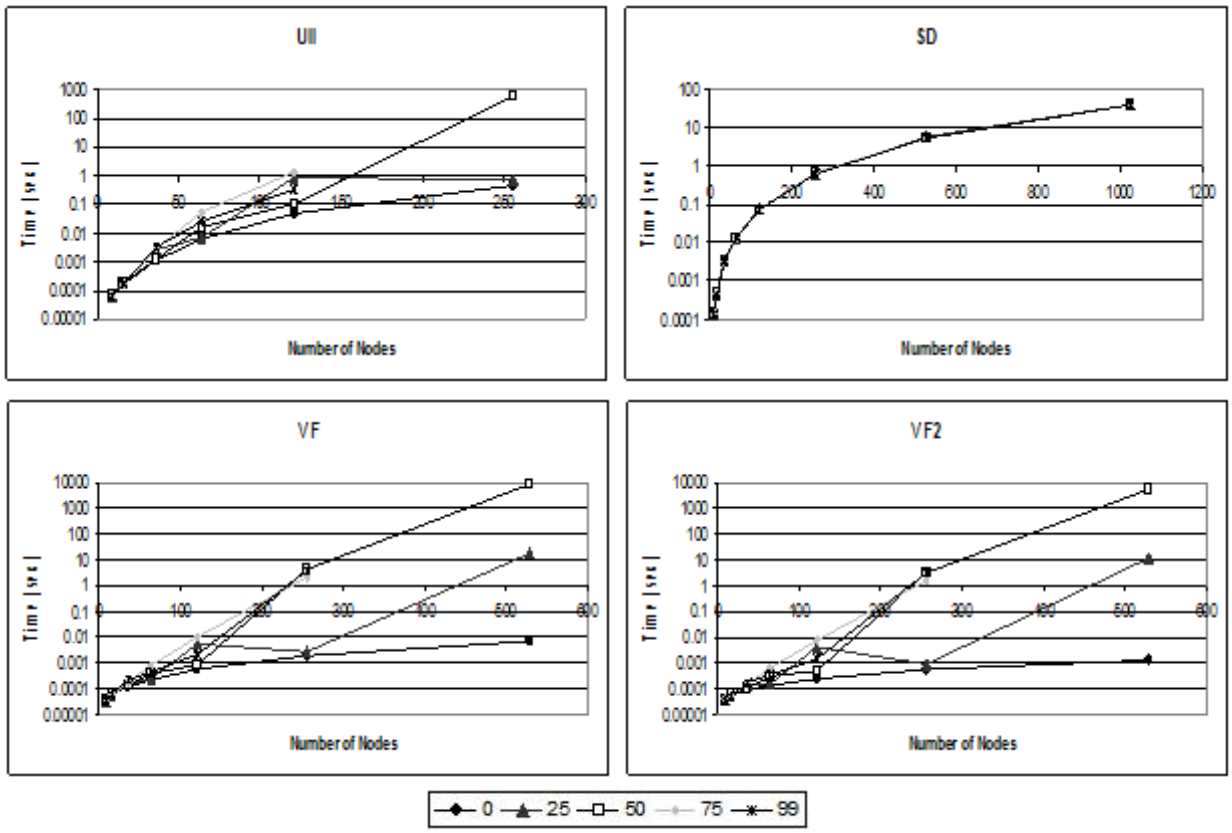


Figure 4 - Results on 6-point stencil on 2D tori.

The results in Figure 4 are shown with a logarithmic scale. Ullmann is affected by this topology as it is with all tori topologies tested. VF and VF2 are also significantly affected and testing on this topology could only be completed through 500 nodes for this topology. Besides the binary tree, the 6-point stencil on a 2D torus was the only other topology tested where VF was affected by node labeling. At this time it is unknown why the VF algorithms are affected by this topology. One idea is that it has something to do with the symmetry of the topology as it is the only topology tested that does not have reflective symmetry. However, preliminary testing on non-reflective symmetric stencils applied to the 3D topologies has not produced any similar results.

The next set of results compares the algorithms across the topologies at approximately 100 nodes and 1,000 nodes. The degree of randomness used for these results is 99 so algorithms affected by node labeling are slightly penalized.

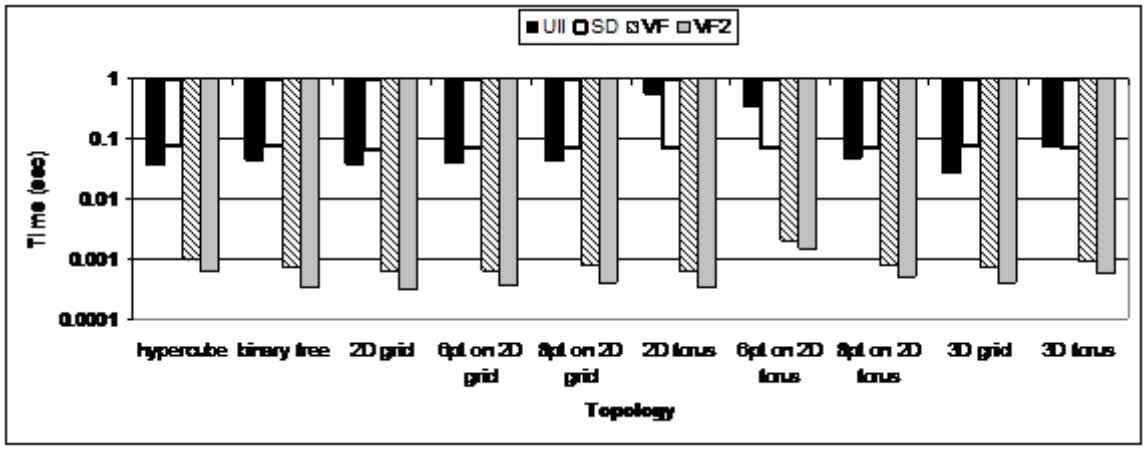


Figure 5a - Results of algorithms for approximately 100 nodes.

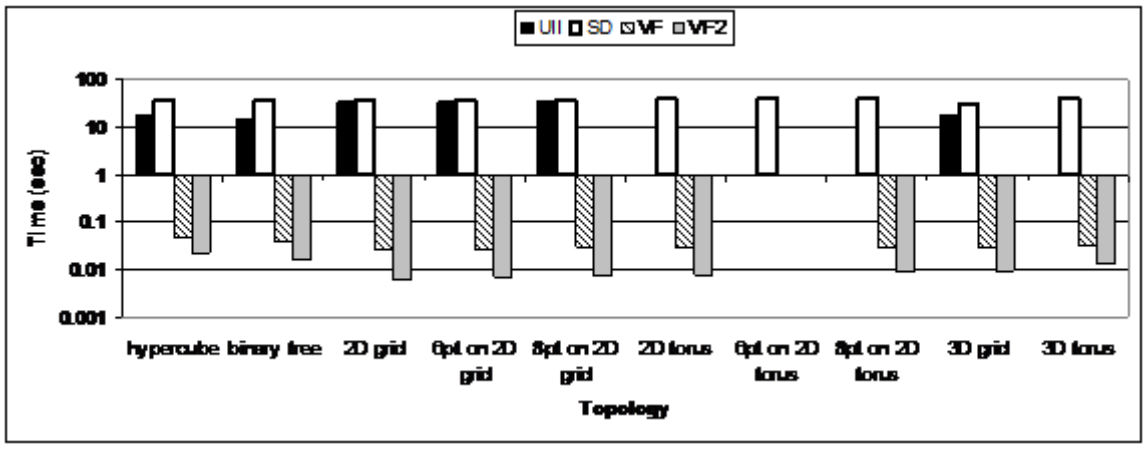


Figure 5b - Results of algorithms for approximately 1,000 nodes.

One hundred nodes was the largest test case all algorithms could perform for all topologies. In Figure 5a it can be seen that all algorithms were able to execute in less than a second for 100 nodes. VF2 was fastest followed by VF, then Ullmann and SD.

Figure 5b shows the results for 1,000 nodes. VF and VF2 are again fastest and are able to execute in less than a second for all cases except the 6-point stencil applied to the 2D torus. In this case, SD is fastest after 500 nodes. Ullmann is not able to perform efficiently on torus topologies at the 1,000-node level. At 16,000 nodes, only VF and VF2 are still able to perform and they run efficiently on all topologies tested except the 6-point stencil on the 2D torus.

VF2 is the most efficient algorithm followed by VF. However, both are affected by topology. Given the wrong topology, they will both in effect appear to crash. VF2 was the only algorithm affected by node labeling for all topologies, while VF and Ullmann were only affected by node labeling on a few topologies. Ullmann is faster than SD in cases where Ullmann is not affected by topology. SD is the most consistent as it is not affected by either topology or node labeling in any test case.

More research needs to be done on how these algorithms work and why they are affected by topologies and node labeling. The best algorithm can be chosen for a given situation by combing the results in [6, 2], which test mostly unstructured graphs, and this paper which focuses on highly structured graphs.

V. References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullmann, "The design and analysis of computer algorithms," Addison Wesley, 1974.

[2] H. Bunke, M.Vento, "Benchmarking of Graph Matching Algorithms," *Proceedings 2nd IAPR-TC15 Workshop on Graph-based Representations*, Handhorf, 1999.

[3] L.P. Cordella, P. Foggia, C. Sansone, M. Vento, "A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no, 10, pp. 1367-1372, 2004.

[4] L.P. Cordella, P. Foggia, C. Sansone, M. Vento, "Performance Evaluation of the VF Graph Matching Algorithm," *Proceedings of the 10th ICIAP*, IEEE Computer Society Press, pp. 1172-1177, 1999.

[5] P. Foggia, C. Sansone, M. Vento, "An Improved Algorithm for Matching Large Graphs," *Proceedings of the 3rd IAPR-TC15 International Workshop on Graph-based Representation*, Italy, 2001.

[6] P. Foggia, C. Sansone, M. Vento, "A Performance Comparison of Five Algorithms for Graph Isomorphism," *Proceedings of the 3rd IAPR-TC15 Workshop on Graph based Representation (GbR2001)*, Italy, 2001.

[7] J. Hopcroft, J. Wong, "Linear time algorithm for isomorphism of planar graphs," Proceedings of the 6th annual ACM Sym. Of Theory of Computing, pp. 162-184, 1974.

[8] M.J. Jolion, "Graph matching: what are we really talking about?," Third IAPR Workshop on Graph-based Representations in Pattern Recognition, Ischia, Italy, May 2001. Http://rfv.insa-lyon.fr/~jolion/PS/prlcp1.pdf

[9] B.D. McKay, "Practical Graph Isomorphism, *Congressus Numerantium*, 30, pp.45-87,1981.

[10] D.C. Schmidt, L.E. Druffel, "A Fast Backtracking Algorithm to Test Directed Graphs for Isomorphism Using Distance Matrices," *Journal of the Association for Computing Machinery*, 23, pp. 433-445, 1976.

[11] J.R. Ullman, "An Algorithm for Subgraph Isomorphism," *Journal of the Association for Computing Machinery*, vol. 23, pp. 31-42, 1976.

[12] Q. Xu, R. Prithivathi, J. Subhlok, R. Zheng, "Logicalization of {MPI} Communication Traces," Technical Report UH-CS-08-07, University of Houston, May 2008.

[13] Q. Xu, J. Subhlok, "Construction and Evaluation of Coordinated Performance Skeletons," Technical Report UH-CS-08-09, University of Houston, May 2008.