Partitioning Dense Graphs with Hardware Accelerators

Xiaoyuan Liu[1,2], Hayato Ushijima-Mwesigwa[2], Indradeep Ghosh[2], and Ilya Safro[1]

[1] University of Delaware, Newark, DE, USA {joeyxliu,isafro}@udel.edu
[2] Fujitsu Research of America, Inc., Sunnyvale, CA, USA
{hayato,ighosh}@fujitsu.com

Abstract. Graph partitioning is a fundamental combinatorial optimization problem that attracts a lot of attention from theoreticians and practitioners due to its broad applications. In this work, we experiment with solving the graph partitioning on the Fujitsu Digital Annealer (a special-purpose hardware designed for solving combinatorial optimization problems) and compare it with the existing top solvers. We demonstrate limitations of existing solvers on many dense graphs as well as those of the Digital Annealer on sparse graphs which opens an avenue to hybridize these approaches.

Keywords: Graph Partitioning · Dense Graphs · Digital Annealer · Quantum-Inspired

1 Introduction

There are several reasons to be optimistic about the future of quantum-inspired and quantum devices. However, despite their great potential, we also need to acknowledge that state-of-art classical methods are extremely powerful after years of relentless research and development. In classical computing, the development of algorithms, the rich mathematical framework behind them, and sophisticated data structures are relatively mature, whereas the area of quantum computing is still at its nascent stage. Many existing classical algorithms do not have provable or good enough bounds on the performance (e.g., they might not have ideal performance in the worst case), but in many applications, the worst-case scenarios are rather rarely seen. As a result, such algorithms, many of which heuristics, can achieve excellent results in terms of the solution quality or speed. Therefore, when utilizing emerging technologies such as quantum-inspired hardware accelerators and quantum computers to tackle certain problems, it is important to compare them not only with possibly slow but provably strong algorithms but also with the heuristic algorithms that exhibit reasonably good results on the instances of interest.

The graph partitioning [2] is one of the combinatorial optimization problems for which there exists a big gap between rigorous theoretical approaches that ensure best known worst-case scenarios, and heuristics that are designed to cope

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_40

with application instances exhibiting a reasonable quality-speed trade-off. Instances that arise in practical applications often contain special structures on which heuristics are engineered and tuned. Because of its practical importance, a huge amount of work has been done for a big class of graphs that arise in such areas as combinatorial scientific computing, machine learning, bioinformatics, and social science, namely, *sparse graphs*. Over the years, there were several benchmarks on which the graph partitioning algorithms have been tested and compared with each other to mention just a few [1,3,21]. However, *dense graphs* can be rarely found in them. As a result, most existing excellent graph partitioning heuristics do not perform well in practice on dense graphs, while provable algorithms with complexity that depends on the number of edges (or non-zeros in the corresponding matrix) are extremely slow. As we also show in computational results, a graph sparsification does not necessarily practically help to achieve high-quality solutions.

Multilevel Algorithms This class of heuristics is one of the most successful for a variety of cut-based graph problems such as the minimum linear arrangement [15], and vertex separator [7]. Specifically for a whole variety of (hyper)graph partitioning versions [10, 11, 16, 18] these heuristics exhibit best quality/speed trade-off [2]. In multilevel graph partitioning frameworks, a hierarchy of coarse graph representations is constructed in such a way that each next coarser graph is smaller than the previous finer one, and a solution of the partitioning for the coarse graph can approximate that of the fine graph and be further improved using fast local refinement. Multilevel algorithms are ideally suited for sparse graphs and suffer from the same problems as the algebraic multigrid (which generalizes, to the best of our knowledge, all known multilevel coarsening for partitioning) on dense matrices. In addition, a real scalability of the existing refinement for partitioning is achieved only for sparse local problems. Typically, if the density is increasing throughout the hierarchy construction, various ad-hoc tricks are used to accelerate optimization sacrificing the solution quality. When such things happen at the coarse levels, an error is quickly accumulated. Here we compare our results with KaHIP [17] which produced the best results among several multilevel solvers [2].

Hardware Accelerators for Combinatorial Problems Hardware accelerators such as GPU have been pivotal in the recent advancements of fields such as machine learning. Due to the computing challenges arising as a result of the physical scaling limits of Moore's law, scientists have started to develop special-purpose hardware for solving combinatorial optimization problems. These novel technologies are all unified by an ability to solve the Ising model or, equivalently, the quadratic unconstrained binary optimization (QUBO) problem. The general QUBO is NP-hard and many problems can be formulated as QUBO [14]. It is also often used as a subroutine to model large neighborhood local search [13]. The Fujitsu Digital Annealer (DA) [4], used in this work, utilizes application-specific integrated circuit hardware for solving fully connected QUBO problems. Internally the hardware runs a modified version of the Metropolis-Hastings algorithm for simulated annealing. The hardware utilizes massive parallelization and a novel sampling technique. The novel sampling technique speeds up the traditional Markov Chain Monte Carlo by almost always moving to a new state instead of being stuck in a local minimum. Here, we use the third generation DA, which is a hybrid software-hardware configuration that supports up to 100,000 binary variables. DA also supports users to specify inequality constraints and special equality constraints such as 1-hot and 2-way 1-hot constraints.

Our contribution The goal of this paper is twofold. First, we demonstrate that existing scalable graph partitioning dedicated solvers are struggling with the dense graphs not only in comparison to the special-purpose hardware accelerators but even sometimes if compared to generic global optimization solvers that are not converged. At the same time, we demonstrate a clear superiority of classical dedicated graph partitioning solvers on sparse instances. Second, this work is a step towards investigating what kind of problems we can solve using combinatorial hardware accelerators. Can we find problems that are hard for existing methods, but can be solved more efficiently with novel hardware and specialized algorithms? As an example, we explore the performance of Fujitsu Digital Annealer (DA) on graph partitioning and compare it with general-purpose solver Gurobi, and also graph partitioning solver KaHIP.

We do not attempt to achieve an advantage for every single instance, especially since at the current stage, the devices we have right now are still facing many issues on scalability, noise, and so on. However, we advocate that hybridization of classical algorithms and specialized hardware (e.g., future quantum and existing quantum-inspired hardware) is a good candidate to break the barriers of the existing quality/speed trade-off.

2 Graph Partitioning Formulations

Let $G = (V, E)$ be an undirected, unweighted graph, where V denotes the set of n vertices, and E denotes the set of m edges. The goal of perfect balanced k-way graph partitioning (GP), is to partition V into k parts, $V_1, V_2, \cdots, V_k$, such that the k parts are disjoint and have equal size, while minimizing the total number of *cut edges*. A *cut edge* is an edge that has two end vertices assigned to different parts. Sometimes, the quality of the partition can be improved if we allow some imbalance between different parts. In this case, we allow some imbalance factor $\epsilon > 0$, and each part can have at most $(1+\epsilon)\lceil n/k \rceil$ vertices.

Binary Quadratic Programming Formulation of GP We first review the integer quadratic programming formulation for k-way GP [8,20]. When $k = 2$, we introduce binary variables $x_i \in \{0, 1\}$ for each vertex $i \in V$, where $x_i = 1$ if vertex i is assigned to one part, and 0 otherwise. We denote by $\mathbf{x}$ the column vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$. The quadratic programming is then given by

$$\min_{\mathbf{x}} \mathbf{x}^T L \mathbf{x} \quad \text{such that } x_i \in \{0,1\},\ \forall i \in V, \tag{1}$$

where L is the Laplacian matrix of graph G. For perfect balance GP, we have the following equality constraint:

$$\mathbf{x}^T \mathbb{1} = \left\lceil \frac{n}{2} \right\rceil, \tag{2}$$

where $\mathbb{1}$ is the column vector with ones. For the imbalanced case, we have the following inequality constraint $\mathbf{x}^T \mathbb{1} \leq (1+\epsilon) \left\lceil \frac{n}{2} \right\rceil$.

When $k > 2$, we introduce binary variables $x_{i,j} \in \{0,1\}$ for each vertex $i \in V$ and part j, where $x_{i,j} = 1$ if vertex i is assigned to part j, and 0 otherwise. Let $\mathbf{x}_j$ denote the column vector $\mathbf{x}_j = (x_{1,j}, x_{2,j}, \cdots, x_{n,j})^T$ for $1 \leq j \leq k$. The quadratic programming formulation is then given by

$$\begin{aligned} \min_{\mathbf{x}} \ & \frac{1}{2} \sum_{j=1}^{k} \mathbf{x}_j^T L \mathbf{x}_j \\ \text{s.t.} \ & \sum_{j=1}^{k} x_{i,j} = 1, \quad \forall i \in V, \\ & x_{i,j} \in \{0,1\}, \quad \forall i \in V, \quad 1 \leq j \leq k. \end{aligned}$$

Again, for perfect balance GP, we have another set of equality constraints:

$$\mathbf{x}_j^T \mathbb{1} = \left\lceil \frac{n}{k} \right\rceil, \quad 1 \leq j \leq k.$$

For the imbalance case, we have the following inequality constraints:

$$(1-\epsilon) \left\lceil \frac{n}{k} \right\rceil \leq \mathbf{x}_j^T \mathbb{1} \leq (1+\epsilon) \left\lceil \frac{n}{k} \right\rceil, \quad 1 \leq j \leq k.$$

QUBO Formulation To convert the problem into QUBO model, we will need to remove the constraints and add them as penalty terms to the objective function [14]. For example, in the quadratic programming (1) with the equality constraint (2), we obtain the QUBO model as follows:

$$\begin{aligned} \min_{\mathbf{x}} \ & \mathbf{x}^T L \mathbf{x} + P \left(\mathbf{x}^T \mathbb{1} - \left\lceil \frac{n}{2} \right\rceil \right)^2 \\ \text{s.t.} \ & x_i \in \{0,1\}, \quad \forall i \in V, \end{aligned}$$

where $P > 0$ is a postive parameter to penalize the violation of constraint (2). For inequality constraints, we will introduce additional slack variables to first convert the inequality to equality constraints, and then add them as penalty terms to the objective function.

3 Computational Experiments

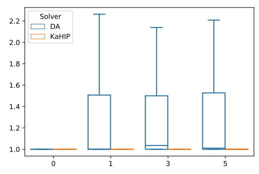
The goal of the experiments was to identify the class of instances that is more suitable to be solved using the QUBO framework and the current hardware. We compare the performance of DA with exact solver Gurobi [5], and the state-of-the-art multilevel graph partitioning solver KaHIP [17]. We set the time limit for DA and Gurobi to be 15 minutes. For KaHIP, we use KaFFPaE, a combination of distributed evolutionary algorithm and multilevel algorithm for GP. KaFFPaE computes partitions of very high quality when the imbalance factor $\epsilon > 0$, but does not perform very well for the perfectly balanced case when $\epsilon = 0$. Therefore we also enable a recommended by the developers KaBaPE ran with 24 parallel processes, and the time limit of 30 minutes.

To evaluate the quality of the solution, we compare the approximation ratio, which is computed using the GP cut found by each solver divided by the best-known value. For some graphs, we have the best-known provided from the benchmark [21], otherwise we use the best results found by the three solvers as the best known. Since this is a minimization problem, the minimum possible value of the approximation ratio is 1, the smaller the better. For each graph and each solver used, we also provide the objective function value, i.e., the number of cut edges. Due to space limitation, we present only the summary of the results. Detailed results are available in [12].
Main conclusion: We have focused on demonstrating practical advantage of software and hardware approaches for GP. We found that dense graphs exhibit limitations of the existing algorithms which can be improved by the hardware accelerators.

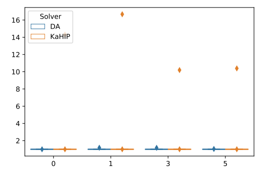
Graph Partitioning on Sparse Graphs We first test the three solvers on instances from the Walshaw graph partitioning archive [21]. We present the summary of the results with box plots in Fig. 1 (a), (d). We observe that in Figure 1 (d), where we compare DA and Gurobi, DA can find the best-known partition for most instances, and perform better compared to Gurobi. However, for several sparse graphs, i.e., $d_{avg} < 3$, for example, `uk`, `add32` and `4elt`, DA can not find the best-known solutions. For these sparse graphs, multilevel graph partitioning solvers such as KaHIP can usually perform an effective coarsening and uncoarsening procedure based on local structures of the graph and therefore find good solutions quickly. As shown in Fig. 1 (a), KaHIP performs better than DA. Based on the numerical results, we conclude that for the sparse graphs, generic and hardware QUBO solvers do not lead to many practical advantages. However, graphs with more complex structures, that bring practical challenges to the current solvers might benefit from using the QUBO and hardware accelerators.

Graph Partitioning on Dense Graphs To validate our conjecture, in the next set of experiments, we examine dense graphs from the SuiteSparse Matrix Collection [3] The experimental results are presented in Fig. 1 (b), (e). We observe that for these dense graphs, in general, DA is able to find solutions that are usually at least as good as those produced by KaHIP and Gurobi. In particular, we find that for one instance, `exdata_1`, KaHIP fails significantly. We therefore use a graph generator MUSKETEER [6] to generate similar instances[3]. The parameters used

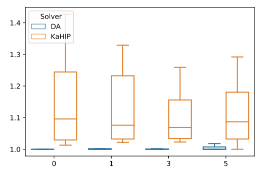
[3] The `exdata` graph files are available here: https://github.com/JoeyXLiu/dense-graph-exdata



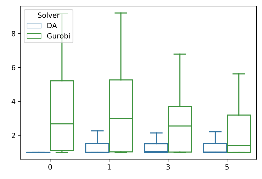
(a) Walshaw $k = 2$



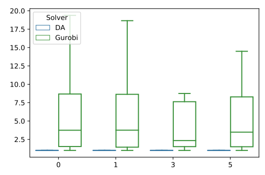
(b) Suite Sparse $k = 2$



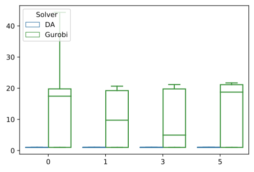
(c) exdata $k = 2$



(d) Walshaw $k = 2$



(e) Suite Sparse $k = 2$



(f) exdata $k = 2$

Fig. 1: Comparison of DA with KaHIP (dedicated GP solver), and Gurobi (general-purpose solver) for sparse and dense graphs respectively. The y-axis represents the approximation ratio (solution to best-solution ratio), the minimum possible value of the approximation ratio is 1, the smaller the better. The x-axis represents the imbalance factor as percentage

to generate the instances can be found in the appendix of the full version. In short, MUSKETEER applies perturbation to the original graph with a multilevel approach, the local editing preserves many network properties including different centralities measures, modularity, and clustering. The experiment results are presented in Fig. 1 (c), (f). We find that in most instances, DA outperforms KaHIP and Gurobi, demonstrating that in this class of problems, specialized hardware such as DA is having an advantage.

Currently, to tackle GP on dense graphs, the main practical solution is to first sparsify the graphs (hoping that the sparsified graph still preserves the structure of the original dense graph), solve GP on the sparsified graph, and finally project the obtained solution back to the original graph. We have applied the Forest Fire sparsification [9] available in Networkit [19]. This sparsification is based on random walks. The vertices are burned starting from a random vertex, and fire may spread to the neighbors of a burning vertex. The intuition is that the edges that are visited more often during the random walk are more important in the graph. In our experiments, we eliminate 30% of the edges. Then we solve GP using KaHIP (KaffpaE version) and project the obtained solution back to the original dense graph. Results and details of the experiments can be found in the full version of the paper.We find that for dense graphs with complex structures, KaHIP does not outperform DA, and graph sparsification does not help to achieve this goal. In this case, we advocate the use of the QUBO framework and specialized hardware.

References

1. Bader, D.A., Meyerhenke, H., Sanders, P., Wagner, D.: 10th dimacs implementation challenge-graph partitioning and graph clustering (2011), https://www.cc.gatech.edu/dimacs10/
2. Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., Schulz, C.: Recent advances in graph partitioning. In: Algorithm Engineering: Selected Results and Surveys. LNCS 9220, Springer-Verlag, pp. 117–158. Springer (2016)
3. Davis, T.A., Hu, Y.: The university of florida sparse matrix collection. ACM Transactions on Mathematical Software (TOMS) **38**(1), 1–25 (2011)
4. Fujitsu: Fujitsu Digital Annealer (2022), https://www.fujitsu.com/global/services/business-services/digital-annealer/
5. Gurobi Optimization, I.: Gurobi optimizer reference manual (2018), https://www.gurobi.com/
6. Gutfraind, A., Safro, I., Meyers, L.A.: Multiscale network generation. In: 2015 18th International Conference on Information Fusion. pp. 158–165. IEEE (2015)
7. Hager, W.W., Hungerford, J.T., Safro, I.: A multilevel bilinear programming algorithm for the vertex separator problem. Computational Optimization and Applications **69**(1), 189–223 (2018)
8. Hager, W.W., Krylyuk, Y.: Graph partitioning and continuous quadratic programming. SIAM Journal on Discrete Mathematics **12**(4), 500–523 (1999)
9. Hamann, M., Lindner, G., Meyerhenke, H., Staudt, C.L., Wagner, D.: Structure-preserving sparsification methods for social networks. Social Network Analysis and Mining **6**(1), 22 (2016)
10. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing **20**(1) (1999)
11. Karypis, G., Kumar, V.: Multilevel algorithms for multi-constraint graph partitioning. In: SC'98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing. pp. 28–28. IEEE (1998)
12. Liu, X., Ushijima-Mwesigwa, H., Ghosh, I., Safro, I.: Partitioning dense graphs with hardware accelerators. arXiv preprint arXiv:2202.09420 (2022)
13. Liu, X., Ushijima-Mwesigwa, H., Mandal, A., Upadhyay, S., Safro, I., Roy, A.: Leveraging special-purpose hardware for local search heuristics. Computational Optimization and Applications, to appear (2022)
14. Lucas, A.: Ising formulations of many np problems. Frontiers in Physics **2**, 5 (2014)
15. Safro, I., Ron, D., Brandt, A.: Graph minimum linear arrangement by multilevel weighted edge contractions. Journal of Algorithms **60**(1), 24–41 (2006)
16. Safro, I., Sanders, P., Schulz, C.: Advanced coarsening schemes for graph partitioning. ACM Journal of Experimental Algorithmics (JEA) **19**, 2–2 (2015)
17. Sanders, P., Schulz, C.: Think locally, act globally: Highly balanced graph partitioning. In: , SEA 2013. vol. 7933, pp. 164–175. Springer (2013)
18. Shaydulin, R., Chen, J., Safro, I.: Relaxation-based coarsening for multilevel hypergraph partitioning. SIAM Multiscale Modeling and Simulation **17**, 482–506 (2019)
19. Staudt, C.L., Sazonovs, A., Meyerhenke, H.: Networkit: A tool suite for large-scale complex network analysis. Network Science **4**(4), 508–530 (2016)
20. Ushijima-Mwesigwa, H., Negre, C.F., Mniszewski, S.M.: Graph partitioning using quantum annealing on the D-Wave system. In: Proceedings of the Second International Workshop on Post Moores Era Supercomputing. pp. 22–29 (2017)
21. Walshaw, C.: The graph partitioning archive. https://chriswalshaw.co.uk/partition/ (2009)