Extended Abstract for Lightning Talk: A Conflict-Free Replicated Data Type for Graph Rewriting

Luca Mondada
University of Oxford
Oxford, UK
luca@mondada.net

Graph rewriting is the problem of finding the best equivalent graph, given an input graph and a set of equivalence relations, the rewrite rules. In the absence of theoretical guarantees such as the confluence of rewriting systems [2], graph rewriting becomes a complex optimisation problem. In this scenario a naive exploration of the search space that applies one rewrite at a time suffers from two problems:

Locality. Cost functions are often local properties, i.e. they can be given as an expression of the cost function evaluated on subgraphs

$$\mathscr{C}(G) = \mathscr{A}(\{\mathscr{C}(H) \mid H \in P(G)\})$$

where G is a graph, $P(G)$ is a partition of G into subgraphs, $\mathscr{C}$ is the cost funtion and $\mathscr{A}$ is some cost aggregation function. Not exploiting this property results in search trees with high branching factors that scale with the size of G, resulting in up to exponential overheads.

Parallelism. A$*$ and related backtracking search algorithms [1, 3] are inherently sequential in nature, making them hard to parallelise [10].

This extended abstract is a proposal for a novel data structure that explicitly leverages the locality of graph rewrites to simultaneously reduce the search space and parallelise the search algorithm. We expect speedups in particular in the regime of large graphs and constant size rewriting rules.

Background: Equality saturation and CRDTs

We take ideas from the equality saturation technique used in term rewriting [6, 8] and the concept of a conflict-free replicated data type (CRDT) popular in distributed computing applications [5].

Term rewriting can be viewed as a special case of graph rewriting in which the input and the rewrite rules are algebraic terms; every expression is thus an algebraic syntax tree (AST). Every node of an AST represents a subterm of the expression, given by the subtree that is rooted in that node. By storing the equivalence class of subterms obtained through rewrites in every node of the AST, we obtain a persistent data structure that efficiently encodes all equivalent expressions. Once such a data structure has been populated, an extraction step can search for the optimal expression. This approach to term rewriting is called equality saturation. Unfortunately, equality saturation does not generalise well to graphs [9].

Persistent data structures for rewriting can be combined with CRDTs, distributed data structures that allow for concurrent edits. Every node maintains a local copy of the data and changes are propagated by broadcasting edits using a communication protocol such as [4]. Incoming edits are then applied to the local copy, effectively syncing the data. Importantly, every element in the data structure is immutable and assigned a globally unique ID (GUID) to avoid edit conflicts. Persistent data structures are particularly suited as CRDTs as addition is the only modification of the data that must be handled.

Submitted to:
GCM 2024 (Lightning)

This work is dedicated to the public domain.

Local graph diffs

For a given input graph G, our data structure $\mathscr{D}$ stores a hierarchy of *local graph diffs* of G, that is, subgraphs of G along with a rewrite for each subgraph. In other words, each diff is given by a set of vertices and edges (the rewritten graph) as well as a boundary relation that maps the boundary of the diff to edges in one of its parent diffs. We support three elementary operations on $\mathscr{D}$. These operations are persistent, i.e. they leave their arguments unchanged and insert new diffs into $\mathscr{D}$:

REWRITE$(H,(V',E'),\varphi)$**.** Insert a new diff on the same subgraph as H that rewrites it to the new graph (V',E'). A map φ must be provided that maps the boundary nodes in H to new nodes in V'. This is used to update the boundary relation of the new diff. The parent of the new diff will be the parent of H. The new vertices inserted must be given fresh globally unique IDs and the vertices in H are marked as deleted in the new diff.

MERGEALONG(H_i,H_j,e)**.** If H_i and H_j are diffs on disjoint subgraphs and share a common boundary edge e, then insert a new diff on the union of the subgraphs that combines both rewrites. The resulting boundary relation is the union of the boundary relations minus the edge e. Both H_i and H_j are parents of the new diff.

MERGEWITHPARENT(H_i)**.** Merge the diff H_i into the subgraph of its parent. The new diff will keep the rewrite of H_i but its subgraph will match the parent of H_i's.

As new diffs are inserted into $\mathscr{D}$, these additions can be communicated to other nodes with copies of $\mathscr{D}$. The use of unique IDs and the explicit marking of deleted vertices make it easy to insert any incoming addition in the local copy of the data structure. In the case where rewrites may not be arbitrary but correspond to the application of one of a predefined list of rewrite rules, rewrites may be identified by a rewrite rule ID. Thus, identical rewrites performed by multiple nodes concurrently can be identified and merged. In this case, a union-find data structure must be used as vertices may be assigned more than one ID[1].

Finally, to speed up pattern matching and rewriting we can keep an index that tracks the inverse boundary relations: for each edge e in a diff H_i, FINDBOUNDARIES(H_i,e) returns the list of its descendant diffs for which e is a boundary. As we traverse the graph of a diff H_i, we can thus find all other diffs that can be merged with H_i, until all possible rewrites were found and applied exhaustively.

Conclusion

Using $\mathscr{D}$, graph rewriting can be made local and persistent. Changes on one subgraph can easily be propagated to distributed nodes and combined with concurrent changes to other parts of the graph. Furthermore, FINDBOUNDARIES queries and MERGEALONG operations enable rewrites that span neighbouring subgraphs: as a result, any sequence of rewrites that is possible on a graph G will be possible to perform on $\mathscr{D}$. However, only rewrites within a neighbourhood of a diff will be considered for merging. The branching factor of the search space will thus be reduced, especially for large input graphs G. The combination of parallelisation and smaller search space should result in speedups in practice.

[1]There will be at most one ID per concurrent node. Using a conlifct resolution strategy in such cases (e.g. using the smallest assigned ID) will ensure that the same IDs will be used eventually.

References

[1] Peter E. Hart, Nils J. Nilsson & Bertram Raphael (1968): *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. *IEEE Transactions on Systems Science and Cybernetics* 4(2), pp. 100–107, doi:10.1109/TSSC.1968.300136.

[2] Gerard Huet (1977): *Conflunt reductions: Abstract properties and applications to term rewriting systems*. In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pp. 30–45, doi:10.1109/SFCS.1977.9.

[3] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia & Alex Aiken (2019): *TASO: optimizing deep learning computation with automatic generation of graph substitutions*. In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, Association for Computing Machinery, New York, NY, USA, p. 47–62, doi:10.1145/3341301.3359630.

[4] Giovanna Márk Jelasity, Di Marzo Serugendo, Marie-Pierre Gleizes & Anthony Karageorgos (2011): *Gossip*, pp. 139–162. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-642-17348-6_7.

[5] Hyun-Gul Roh, Myeongjae Jeon, Jin-Soo Kim & Joonwon Lee (2011): *Replicated abstract data types: Building blocks for collaborative applications*. *J. Parallel Distrib. Comput.* 71(3), p. 354–368, doi:10.1016/j.jpdc.2010.12.006.

[6] Ross Tate, Michael Stepp, Zachary Tatlock & Sorin Lerner (2009): *Equality saturation: a new approach to optimization*. *SIGPLAN Not.* 44(1), p. 264–276, doi:10.1145/1594834.1480915.

[7] Ross Tate, Michael Stepp, Zachary Tatlock & Sorin Lerner (2009): *Equality saturation: a new approach to optimization*. In: *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '09, Association for Computing Machinery, New York, NY, USA, p. 264–276, doi:10.1145/1480881.1480915.

[8] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock & Pavel Panchekha (2021): *egg: Fast and extensible equality saturation*. *Proc. ACM Program. Lang.* 5(POPL), doi:10.1145/3434304.

[9] Yichen Yang, Mangpo Phitchaya Phothilimtha, Yisu Remy Wang, Max Willsey, Sudip Roy & Jacques Pienaar (2021): *Equality Saturation for Tensor Graph Superoptimization*. *CoRR* abs/2101.01332. arXiv:2101.01332.

[10] Yichao Zhou & Jianyang Zeng (2015): *Massively parallel A* search on a GPU*. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, AAAI Press, p. 1248–1254, doi:10.1609/aaai.v29i1.9367.