Implementing the Minimum Dominating Set Problem into Graph Databases

P.Balaghan & K.A.Hawick

Computer Science, University of Hull

Introduction

Graph Databases have recently become a feasible alternative to relational Databases. As suggested by the name, Graph Databases store data in a Graph-Like structure. This allows the potential to support a range of graph algorithms, something which is difficult to implement in other database structures. We analyse the case of the greedy dominating set problem and measure the ease of expressiveness and performance implications of implementing the problem into graph database systems. We discuss comparisons between various approaches and the limitations of present graph databases and their interfaces for state-full algorithms.

The Dominating Set Algorithm

The minimum dominating set problem finds the lowest amount of nodes which are adjacent to every other node in a graph. It is a known to be a classical NP-Complete problem [1]. A heuristic used for finding the dominating set in polynomial time is the greedy algorithm [2], a definition of which can be found in Eq. 1.

Algorithm 1 The Greedy heuristic algorithm [2] for finding the minimum dominating set of a graph

$S := \emptyset$
while $\exists$ *white nodes* **do**
choose $v \in \{x | w(x) = max_{u \in V}\{w(u)\}\}$
$S := S \cup v$
end while

All nodes in the graph are initially coloured white. All isolated nodes are coloured black. We then find the node with the highest amount of white-node connections. We colour this node Black and colour it's adjacent nodes grey. This is repeated until no more white-coloured nodes exist. In figure 1 we show the minimum dominating set of an instance. The nodes coloured black are the minimum dominating set of the graph.

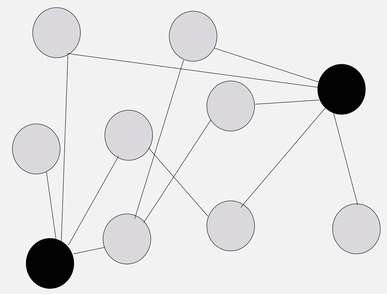

Figure 1: The two black nodes represent the dominating set of the graph above. They are adjacent to every other node in the graph

Implementations

Graph Databases can be generally sorted into 2 different types of database. They can be a *client-server* or an *embedded* database. *Client-server* host the database engine on a *server* where a *client* can send queries. *Embedded* databases are stored within a high level language, and everything is locally stored. The apparatus provided by the systems are also based on the type of database. *Client-Server* databases supply a query language, while *embedded* servers provide a library which have some query-like function calls.
At the time of writing, the most popular *client-server* database is Neo4j and the most popular *embedded* database is Sparsity. We use these systems to represent their database types.
For Neo4j we implement the dominating set algorithm in the query language Cypher. For Sparsity, we use equivalent library calls combined with Java. We found that Cypher does not currently contain a feature that allows a part of a query to be looped. Therefore we built a single Cypher query using a python program. We also created a Python wrapper that allowed us to loop around the Cypher query. Algorithm 2 shows the part of the query that was combined with the wrapper.

Algorithm 2 The "bulk" part of the query in Cypher. This part of the query finds the node with the most white node connections, and sets the node to black, and it's neighbourhood grey.

```
MATCH (n)–>(m)
WHERE n.blackness <> 1
WITH collect(m) as neighbourhood, n
WITH reduce(totalweight = n.whiteness, j in neighbourhood
| totalweight + j.whiteness) as weightings, n
WITH n, weightings
ORDER BY weightings desc limit 1
MATCH(n)–>(m)
WHERE m.blackness <> 1
SET n.blackness = 1
SET n.whiteness = 0
SET m.whiteness = 0
```

Figure 2: The time taken to find a dominating set in a selection of instances. N/A represents the implementation not being able to find a dominating set. The first column shows how long the single Cypher query took to be built. The second column is how long the final query took.

Data Set	CYPHER		CYPHER	Sparsity	C++
	Whole Program	Final Query	with wrapper	with wrapper	Implementation
Barabasi -100 -4 [3]	6.27s	0.89s	0.46s	0.02s	<1ms
Barabasi -1000 -2 [3]	N/A	N/A	20.87s	2.28s	0.1s
Barabasi -1000 -3 [3]	9305.22s	245.01s	23.1s	1.74s	0.01s
Barabasi -1000 -4 [3]	5696.14s	63.91s	27.24s	1.42s	0.01s
Barabasi -10000 -4 [3]	N/A	N/A	N/A	157.77s	10.04s
pokec_2000 [4]	11175.53s	198.71s	91.89s	1.90s	0.13s
gplus_200 [4]	27.86s	1.89s	1.16s	0.05s	<1ms
gplus_2000 [4]	N/A	N/A	94.83s	4.26s	0.26s

Results and Discussion

In the table in fig 2, we show the time taken for the implementations to find a dominating set for a selection of instances. We found that for all of the instances, a single Cypher query is much slower than the other implementations. However once a wrapper was introduced, a solution for some of the instances could now be found. For *barabasi_1000_2* and *gplus_2000*, a dominating set could be found. Sparsity was found to be much faster than Cypher. This could be due to the imperative style of the query language. It is also worth mentioning that when the algorithm is expressed in Cypher, it uses two lines of code less than a vanilla implementation in a high level language. As Cypher is a declarative language, we expected the language to be able to express the algorithm in fewer lines of code.

Conclusion

In summary, we found that Graph Databases are not yet reaching their full potential when implementing computational algorithms using the apparatus provided by the databases. One of the major advantages of Graph databases over other database families is the graph structure. This should allow an efficient implementation of a variety of graph algorithms. While the current apparatus does allow this to an extent, there are some key features that are missing. We believe this deficiency represents a significant opportunity for graph database systems to implement and address these issues.

References

[1] Michael R. Garey and David S. Johnson.
Computers and Intractability; A Guide to the Theory of NP-Completeness.
W. H. Freeman & Co., New York, NY, USA, 1990.

[2] V. Chvatal.
A greedy heuristic for the set-covering problem.
Mathematics of Operations Research, 4(3):233–235, 1979.

[3] Albert-László Barabási and Réka Albert.
Emergence of scaling in random networks.
Science, 286(5439):509–512, 1999.

[4] David Chalupa and Christian Blum.
Mining k-reachable sets in real-world networks using domination in shortcut graphs.
Journal of Computational Science, 22:1 – 14, 2017.