Abstract

Imagine a series of writeups with many links to different blogs in the writeup; how can we determine the best course of action for reading the writeup? How can we find which blog is the closest or “most similar to the current writeup”? The similarity analysis of nodes/links or writeups begins to formalise. The similarity between objects arises in many applications, like Web Search engines, Social Media, etc. This project takes one of many Similarity Algorithms, SimRank[1], and accelerates its computing using Graphics Processing Units (GPUs) over a CUDA parallel computing platform. Simrank calculates the similarity between any two nodes of a graph. It bases the similarity of the nodes on the notion that - “Two objects are similar if they are related to similar objects.” The CPU version[1] of the algorithm is slow to compute the similarity values; for accelerating the computation, a parallel computing platform, CUDA, is used.

Contents

List of Figures

List of Tables

Graph Similarity Analysis using High Performance Computing

1 Introduction and Background

Over the past decades, the number and size of graphs in the internet world have risen exponentially. Modern Internet searching websites, say Google, and DuckDuckGo, have adopted many algorithms to search and collect websites to display them to the user when needed. If we ponder how the websites collect data to reveal it to the user, some background calculation must be happening as the search engine knows what we want. The web pages are similar; they get displayed when a person searches for something on the engine.

Many algorithms are created to find how similar two graphs are. A few examples are - Edit Distance [2], PageRank [3], DivRank, PathSim [4] among others. This project takes a similarity algorithm - SimRank [1] and optimises it via GPUs to accelerate its computations and compare it with a CPU-based algorithm for the same.

In this project, frequent use of GPU Programming is involved; GPU programming is used for High-Performance Computing. It is a method of running highly parallel computations on GPU Cores/Accelerators. Although it was mainly used earlier for Graphics Processing, today, GPUs have found more jobs to do like Mathematical Modelling, Machine Learning and other parallelisation prone jobs. In the project, CUDA is used for harnessing the power of GPUs. C/C++ Languages can be used to talk with the CUDA APIs to access different levels of abstractions provided by the GPU.

SimRank is a similarity measure by which we can define how similar are 'any' two nodes in a graph. Nevertheless, the question arises, why do we need to calculate similarity scores between any pair of nodes? Take a real-world application, Social Media; we have many friends over social media, or the book reading website a person would use would have many followers who like some authors' work. So, if a person searches for, say, a book named - 'How to eat more and still be fat?' in the search bar, the backend engine returns all the related or wildly similar pages to the search entry. How does it calculate what pages to return? It should have some criteria for sorting out the pages which need to be displayed and not display redundant information. A similarity score is needed to ensure that the user's requirements are fulfilled by showing that the most accurate or slighting correlated pages are displayed. Simrank is one such similarity score.

Graph similarity is a very budding field of research nowadays, as the World Wide Web (WWW), is a tremendously sizably voluminous graph, and hence is of consequentiality to run graph algorithms over it.

Simrank was initially proposed in 2001 [1]. It was introduced to find similarity between nodes of a graph without having a bias. Over the years Simrank has found many variations[[5] [6]]. This work extends the base Simrank algorithm over 'static' graphs and optimises its computation time over GPUs[7].

The base Simrank Algorithm[1] calculates simrank over all pair of nodes taking a Time Complexity of $O(k.n^4)$, Where n is the number of vertices in the graph and k is the number of iterations the algorithm converged. Here, the worked algorithm uses backtracking to calculate the simrank values for each pair of nodes over the iterations. The backtracking traverses through all the iterations till the last iteration. Our algorithm stores the antecedent iteration values and uses Dynamic Programming methods to fasten the computation. Utilizing Dynamic Programming[8] methods to memoize the solution expedites it without backtracking over the call stack, reducing our complexity by a magnitude of O(n), taking down the complexity to - $O(k.n^3)$.

Use of CUDA parallel programming construct comes when the code is divided into independent work units[7].The parallel programming APIs are utilised in C++ Language. We find the units that are not homogeneous in any aspect of their computation. The GPU engenders an adequate number of threads to compute each independent work unit in one thread in parallel. We engender the GPU-predicated code for the verbally expressed Simrank Algorithm. By incorporating GPU over the computation, we parallelise the entire summation portion of the algorithm, tuning down the complexity to $O(k.1)$. Albeit we do not generally talk about GPU-predicated code involution analysis, we mention it to further understand the optimisations. In short, the work done in this project is summarised as follows:

1. We create a CPU based Algorithm for Simrank and optimise it to lower the computation time.
2. We also create the Python program using the in-built library of Simrank for correctness check and comparing the times with the current state of the art Python-based Simrank Algorithm.
3. We then create the GPU based code using proper General Purpose GPU Programming practices to compute the algorithms computations in parallel further. Hence reducing the time further.

2 The Model

In the project, a directed graph is denoted as $G(V, E)$, where V denotes the number of Nodes/Vertices in the Set, and E represents the number of edges in the set. Here, $E = V \times V$. Since the graph is directed, $e(a, b) \neq e(b, a)$. We are not assuming any other special condition for the graph.
Notations which are used further in the report are given below in the Table 2.1.

Symbol	Notation
G	A Graph
V	Given Set of Vertices
$\|V\|$	No. of Vertices
E	Given Set of Edges
$\|E\|$	No. of Edges
a,b	Given pair of nodes
S	Final Computed Simrank Matrix
S_k	Simrank Matrix for kth iteration.
S_T	Temporary Simrank Matrix for Computation
$\|S\|$	Strength of Simrank Matrix
η	Convergence Threshold of Algorithm.
In	Matrix of In-Neighbours for each node
$\|In(a)\|$	No. of In-Neighbours of Node 'a'
CV	Confidence Value
NF	Normalisation Factor
I_N	Identity Matrix of Size N $\times$ N
itr	Iteration Count
maxItr	Maximum Number of Iterations

Table 2.1: Notation Table

3 Simrank Similarity Measure

Many of today's applications require contextual 'similarity' between the corpora of objects. One of the self-evident examples would be the World Wide Web. The objects themselves have a wide range of criteria to define differences or kindred attributes; for example, in the restaurant recommender systems used by Google Maps or Apple Maps.

When a person searches for restaurants near them, the recommender system goes into the whole corpus of restaurants present in the near vicinity of the person, and according to the person's culls, verbalize Non-Veg, with an Air-Conditioner and many more factors, it finds the cluster of most similar restaurants matching the criteria, and then exhibits the results.

Such domain works represent graphs, where the nodes of the graph represent the objects, and the edges between the graphs mean the relationship between them and each relationship might have a weight or not. The Simrank[1] algorithm is mainly used to derive kindred attribute scores between data according to the structural context in which they appear. According to the definition of Simrank from [1], It calculates the similarity between two objects as - '**Two objects are similar if similar objects reference them.**'.

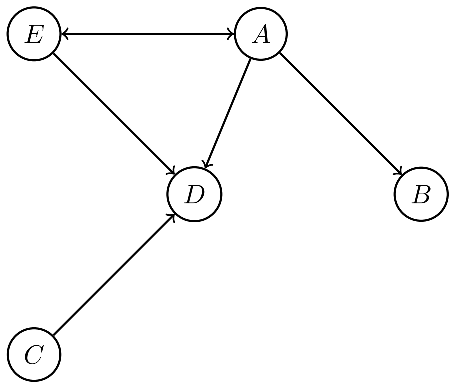


Figure 3.1: Similarity Analysis of above graph

If we consider the above-given graph[Fig:3.1] to calculate the simrank between any two nodes, We can see that the node '$\underline{\text{D}}$' and node '$\underline{\text{A}}$' are similar as they are directly related. Still, for the nodes D and B, according to the definition of Simrank, they are referenced by a mundane node, '$\underline{\text{A}}$', which in turn makes them kindred.

It is essential to ken that the given algorithm is generically proposed works; that is, the algorithm is applied to any domain where enough germane relationships between objects are there to find a structural correlation between them. For example, several web pages can be amalgamated to engender a web graph, and a kindred attribute can be found predicated on a homogeneous textual detail.

3.1 Graph Model

Here, we are modelling the objects and their relationships in the form of Directed Graphs, $G = (V, E)$. In the case of the World Wide Web, the Vertices of the graph denotes the web documents, and the edge represents the document's references to other documents. Say in a domain which is seemingly bi-partite, for example, in the consumer seller domain, a seller is only giving to the consumer and not to the other sellers; an edge represents how many items the consumer bought from the seller. Below is an example of a bipartite graph denoting the consumer seller relationship.

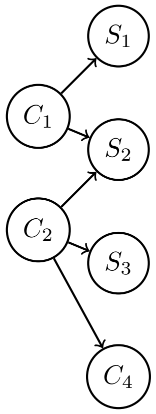
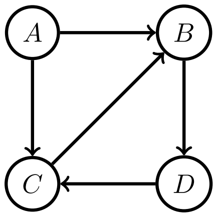
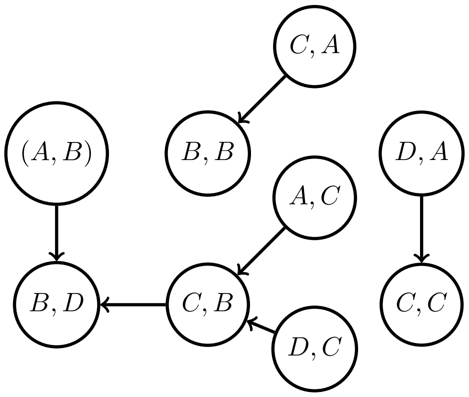


Figure 3.2: Bi-Partite Relationship

For the graph, we denote the In-Neighbours and Out-Neighbours of the nodes of the graph via - I(A) and O(A), where 'A' is any node of the graph; A ϵ V. In the Section 3, we talked that the Simrank Similarity measure is computed in terms of pair of nodes, i.e. Similarity is computed as a propagating pair of nodes in the given Graph. We can also think of it explicitly, by creating another graph, just for understanding purposes; this is a graph derived from the previous graph G, the new Graph G^2 denoted as (V^2, E^2). Where V^2 denotes all the pair of nodes, and E^2 denotes the edges between nodes of V^2. An edge between (a,b) and (c,d) is present in E^2 if and only if there is a node between, $< a, c >$ & $< b, d >$. Below two graphs, G and G^2 can be used to explain the above claim.



(a) G



(b) G^2

3.2 Simrank Algorithm

As we have discussed before 3, rephrasing the sole basis of Simrank - **Objects referring to similar objects are considered to be similar**. By this definition, the nodes will have the highest similarity with the node itself. The nodes with no common nodes will have a similarity score of 0. [Notation used below are from the notation Table 2.1].

$$S_k(a,b) = \begin{cases} \frac{CV}{\|In(a)\| \times \|In(b)\|} \times \sum_{i=1}^{\|In(a)\|} \sum_{j=1}^{\|In(b)\|} (S_{k-1}(In_i(a), In_j(b))), & \text{if } a \neq b \\ 1, & \text{if } a = b \\ 0, & \text{if } \|In(a)\| = 0 \textit{ or } \|In(b)\| = 0 \end{cases} \tag{1}$$

The above given equation (1) denotes the basic Simrank equation which was proposed in [1]. It currently has three cases :

- Nodes are not equal, and have some prevelant nodes between them.
- Nodes are equal.
- Nodes are not equal, but have no in-neighbours.

The above equation is a part of the whole algorithm, iterated over all pairs of nodes (u, v).
As per the equation for calculating the Simrank values for any two nodes a & b, we have to backtrack till the first iteration using the 'In-Neighbour' nodes of a & b. *In-Neighbours* are defined as the nodes which have a direct edge to the corresponding node for which we are discovering the in-neighbours.
Computing the In-Neighbours for a large graph can be time-consuming [9]; we can also attempt to optimise the discovery algorithm faster. We know that we have to backtrack for calculating the Simrank for a pair of nodes; this computation reduces by storing the predecessor iteration simrank matrix, as it also carries over all the backtracked computation.
Now that we have found a way to reduce the number of computations over the summation part of the algorithm, we can now think about the convergence criteria. The authors [1] defined the convergence criteria over Norms of a matrix, and a matrix norm can be taken as the strength of a given matrix. Now, we can check the convergence criteria over the simrank matrix by calculating the norms of consecutive iterations Simrank and checking if the difference between the strengths of the matrix over consecutive iterations is less than or equal to a given threshold.
The threshold here is taken as 10^{-5}. This threshold is calculated via multiple experiments done by us over a wide range of graphs. We break the simrank algorithm when the strength of the matrix is less than the threshold.

According to the information and facts of the Simrank algorithm, discussed in the previous page, we can write the CPU based Simrank Algorithm as (for symbols & notations, see Table 2.1):

Algorithm 1: Simrank Algorithm

```
Input  : Maximum No. of Iterations(maxItr) & Confidence Value(CV)
Output: Converged Simrank Matrix
$itr \leftarrow 0$;
$S_0 \leftarrow I_v$;
$ConvergedAt \leftarrow 0$;
Default maxItr ← 1000;
Default CV ← 0.90;
for $itr \leftarrow 0$ to $maxItr$ by 1 do
    // Checking Convergence for Each Iteration
    $CovergeFlag \leftarrow$ CheckConvergence($S_{itr}$);
    if $ConvergeFlag$ is $True$ then
        // The Simrank Matrix has converged, break the algorithm
        $ConvergedAt \leftarrow itr$;
        Break Loop;
    end if
    else
        // Compute Simrank for further Iterations
        $ComputeSimrank(S_{itr}, itr, CV, )$[3];
    end if
end for
```

The above pseudo-code defines the rudimentary structure of a simrank program. The convergence function gets called where we commence the computation of the current iteration; and we will discuss How the convergence is performed and how Simrank computes in the CPU further. Here, the threshold is taken as $\eta = 0.00001$.

Algorithm 2: Convergence of Simrank Matrix

```
Input  : Current Iteration Simrank Matrix : $S_K$, PreviousNormValue : $\|S_{K-1}\|$
Output: Boolean Value showing whether the Simrank matrix converges or not.
// L1 Norm Calculation is the State of the Art Method[10] used customarily.
$\|S_K\| \leftarrow$ Calculate L1 Norm of the given Simrank Matrix;
$abs(\|S_K\| - \|S_{K-1}\|) \leftarrow$ absolute difference between previous norm and current norm;
$PreviousNormValue \leftarrow$ Update Norm Values;
if $\|S_K\| - \|S_{K-1}\| \leq \eta$ then
    Return True;
end if
Return False;
```

The above algorithm defines how the convergence criteria of a Simrank matrix are defined; we calculate the strength score of the simrank matrix for the current iteration and compare the scores of the current and its preceding iteration. If the score crosses the threshold we set earlier(3), we return the boolean flag to break the algorithm.

The Algorithm utilized for the computation of Simrank consists of Memoizing the anterior iteration and utilizing that memoized matrix to calculate the Simrank matrix for the next iteration.

The Simrank algorithm that is shown below 3 is the rudimentary Simrank algorithm which was discussed in the pristine paper[1]. It follows the iterative approach for calculating simrank for each pair of nodes hence taking the computational intricacy to the higher side. The algorithm is still optimised by memoization, i.e. storing the intermediate results.

Algorithm 3: Computation of Simrank

```
Input   : Current Iteration Simrank Matrix : $S_K$, Graph : $G$, Iteration : $itr$
$S_T \leftarrow$ Matrix to store Intermediate Computations;
$i \leftarrow 0$;
$j \leftarrow 0$;
for $i \leftarrow 0$ to $\|V\|$ by 1 do
    for $j \leftarrow 0$ to $\|V\|$ by 1 do
        // $cmp \leftarrow$ Stores Computation
        $cmp \leftarrow 0.0$;
        // Base Condition check.
        if $i == j$ then
            $cmp \leftarrow 1.0$;
        end if
        else if $\|In(i)\| == 0$ or $\|In(j)\| == 0$ then
            $cmp \leftarrow 0.0$;
        end if
        else
            // Computing Simrank for pair :  (i, j)
            for $k \leftarrow 0$ to $\|In(i)\|$ by 1 do
                for $m \leftarrow 0$ to $\|In(j)\|$ by 1 do
                    $cmp \leftarrow cmp + S_K[In(i)_k, In(j)_m]$;
                end for
            end for
        end if
        $cmp \leftarrow cmp \times \frac{CV}{\|In(i)\| \times \|In(j)\|}$;
        // Update the Ad interim Simrank Matrix, $S_T$ with the above Computed value, $cmp$
        $S_T[i, j] \leftarrow cmp$;
    end for
end for
//  copy the temporary simrank values to the new Simrank Matrix.
$S_{K+1} \leftarrow S_T$ // Copy Matrix
Return;
```

The above algorithm defines how the simrank computes in the CPU. For each pair, it traverses through all its in-neighbours to compute the similarity between the two nodes.

4 Graphics Processing Unit (GPU)

A Graphics Processing Unit[11] is an electronic circuit capable of rendering graphics to a device. Rendering graphics denotes it can do a very high amplitude of intricate calculations very expeditious. Very expeditious designates it can do ostensibly independent calculations in parallel.

5 Problem definition and Objective

6 Methodology

7 Theoretical/Numerical/Experimental findings

Figure 7.1: Our logo

8 Summary and Future plan of work

This is how you cite a paper: Golve

Publications (if any)

Appendix (if any)

References

[1] G. Jeh and J. Widom, "Simrank: A measure of structural-context similarity," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 538–543. [Online]. Available: https://doi.org/10.1145/775047.775126

[2] W. Zheng, L. Zou, X. Lian, D. Wang, and D. Zhao, "Graph similarity search with edit distance constraint in large graph databases," in *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, ser. CIKM '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 1595–1600. [Online]. Available: https://doi.org/10.1145/2505515.2505723

[3] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Technical Report 1999-66, November 1999, previous number = SIDL-WP-1999-0120. [Online]. Available: http://ilpubs.stanford.edu:8090/422/

[4] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks," *Proc. VLDB Endow.*, vol. 4, no. 11, p. 992–1003, aug 2011. [Online]. Available: https://doi.org/10.14778/3402707.3402736

[5] C. Li, J. Han, G. He, X. Jin, Y. Sun, Y. Yu, and T. Wu, "Fast computation of simrank for static and dynamic information networks," in *Proceedings of the 13th International Conference on Extending Database Technology*, ser. EDBT '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 465–476. [Online]. Available: https://doi.org/10.1145/1739041.1739098

[6] M. Kusumoto, T. Maehara, and K.-i. Kawarabayashi, "Scalable similarity search for simrank," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 325–336. [Online]. Available: https://doi.org/10.1145/2588555.2610526

[7] E. K. Jason Sanders, *CUDA by Example: An Introduction to General Purpose GPU Programming.* Addison-Weasley Professional, 2010.

[8] D. P. Bertsekas, *Dynamic Programming & Optimal Control.* Athena Scientific, Belmont, Massachuchets, 2005.

[9] M. Koohi Esfahani, P. Kilpatrick, and H. Vandierendonck, "How do graph relabeling algorithms improve memory locality?" in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2021): Proceedings*, ser. IEEE International Symposium on Performance Analysis of Systems and Software: Proceedings. Institute of Electrical and Electronics Engineers Inc., Apr. 2021, pp. 84–86, publisher Copyright: © 2021 IEEE. Copyright: Copyright 2021 Elsevier B.V., All rights reserved.; 2021 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2021 ; Conference date: 28-03-2021 Through 30-03-2021.

[10] L. G. Belitskii, *Matrix Norms and their Applications.* Birkhäuser Verlag, 1988.

[11] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "Gpu computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.