mathematics

MDPI

Article

Algorithm for the Accelerated Calculation of Conceptual Distances in Large Knowledge Graphs

Rolando Quintero, Esteban Mendiola, Giovanni Guzmán *, Miguel Torres-Ruiz and Carlos Guzmán Sánchez-Mejorada

Centro de Investigación en Computación (CIC), Instituto Politécnico Nacional (IPN), Unidad Profesional Adolfo López Mateos (UPALM)-Zacatenco, Mexico City 07320, Mexico; rquintero@ipn.mx (R.Q.); emendiolat2020@cic.ipn.mx (E.M.); mtorresru@ipn.mx (M.T.-R.); cmejora@ipn.mx (C.G.S.-M.)

* Correspondence: jguzmanl@ipn.mx; Tel.: +52-(55)-5729-6000 (ext. 56617)

Abstract: Conceptual distance refers to the degree of proximity between two concepts within a conceptualization. It is closely related to semantic similarity and relationships, but its measurement strongly depends on the context of the given concepts. DIS-C represents an advancement in the computation of semantic similarity/relationships that is independent of the type of knowledge structure and semantic relations when generating a graph from a knowledge base (ontologies, semantic networks, and hierarchies, among others). This approach determines the semantic similarity between two indirectly connected concepts in an ontology by propagating local distances by applying an algorithm based on the All Pairs Shortest Path (APSP) problem. This process is implemented for each pair of concepts to establish the most effective and efficient paths to connect these concepts. The algorithm identifies the shortest path between concepts, which allows for an inference of the most relevant relationships between them. However, one of the critical issues with this process is computational complexity, combined with the design of APSP algorithms, such as Dijkstra, which is $\mathcal{O}(n^3)$. This paper studies different alternatives to improve the DIS-C approach by adapting approximation algorithms, focusing on Dijkstra, pruned Dijkstra, and sketch-based methods, to compute the conceptual distance according to the need to scale DIS-C to analyze very large graphs; therefore, reducing the related computational complexity is critical. Tests were performed using different datasets to calculate the conceptual distance when using the original version of DIS-C and when using the influence area of nodes. In situations where time optimization is necessary for generating results, using the original DIS-C model is not the optimal method. Therefore, we propose a simplified version of DIS-C to calculate conceptual distances based on centrality estimation. The obtained results for the simple version of DIS-C indicated that the processing time decreased 2.381 times when compared to the original DIS-C version. Additionally, for both versions of DIS-C (normal and simple), the APSP algorithm decreased the computational cost when using a two-hop coverage-based approach.

Keywords: conceptual distance; shortest path algorithms; accelerated calculation; computational complexity

MSC: 68Q25

Citation: Quintero, R.; Mendiola, E.; Guzmán, G.; Torres-Ruiz, M.; Guzmán Sánchez-Mejorada, C. Algorithm for the Accelerated Calculation of Conceptual Distances in Large Knowledge Graphs. *Mathematics* **2023**, *11*, 4806. https://doi.org/10.3390/math11234806

Academic Editors: András Benczúr, Domenico Ursino and Bálint Molnár

Received: 10 October 2023
Revised: 16 November 2023
Accepted: 25 November 2023
Published: 28 November 2023

1. Introduction

The spread of information and communication technologies has significantly increased the amount of available information. Thus, we can decode this information through concepts. Conceptual matching is essential for human and machine reasoning, enabling problem-solving and data-driven inference. Researchers studying the human brain use the term "concept" to describe a unit of meaning stored in memory. The ability to relate

concepts and generate knowledge is inherent to humans. Giving computers this capability is a significant advance in computer science.

Transferring the relationship between concepts from the human domain to the computational domain is complex. Sociocultural and language aspects are closely related to the types of concepts acquired by human beings. Determining the similarity between concepts so that a computer can process them is one of the most fundamental and relevant problems in various research fields. Over time, people have explored multiple structures and methods to represent the semantic relationships underlying conceptual distance. The graph is one of the most commonly used abstract data types to represent this information, designed to model connections between people, objects, or entities. Nodes and edges are the two main elements of a graph. Moreover, graphs have specific properties that make them unique and important in different applications.

According to Mejia Sanchez-Bermejo (2013) [1], measuring the conceptual distance and semantic similarity between words has been studied for many years in the fields of linguistic computing and artificial intelligence (AI) because it represents a generic procedure in a wide variety of applications, such as natural language processing, word disambiguation, detection and error correction in text documents, and text classification, among others. Based on psychological assessments, semantic similarity refers to how humans categorize and organize objects or entities [2]. Semantic similarity is a metric determining how closely two terms representing objects from a conceptualization are related. It is determined by analyzing whether the terms share a common meaning [3].

Reducing the computational cost of computing conceptual distances for each pair of concepts in massive graphs will accelerate the evaluation of similarity. As a result, it will have practical benefits and contribute to the development of more efficient applications to solve practical problems, for instance, reducing the computational time required to search for pairs of documents with high semantic similarity in large datasets composed of millions of documents and analyzing the transport network obtained automatically using AI techniques, as described by Chen et al. (2022) [4]. It also extends the understanding of the phenomenon of semantic similarity in computing systems based on conceptual graphs. The research and development of efficient algorithms that reduce computation time by combining approximation or analytical techniques is essential to advance the study of semantic similarity.

In recent years, conceptual graphs generated from ontologies have gained importance. These graphs are strongly connected, where each concept is a vertex, and the two edges represent each relationship (one in each direction). Consequently, it is possible to apply various graph theory search algorithms to study and analyze the relations represented in a conceptual graph. DIS-C Quintero et al. (2019) [5] is a method with which to measure the conceptual distance between concepts. It requires a conversion of a conceptualization into a conceptual graph by calculating the All Pairs Shortest Path (APSP) through the graph. Thus, shortest path algorithms are crucial for solving various optimization problems. The primary purpose of these algorithms is to find the path with the minimum cost or distance between a given pair of vertices in a graph or network. The study of shortest path algorithms and the proposal of different approaches to reduce computational cost have been largely studied, as per the research works proposed by Dreyfus (1969) [6], Gallo and Pallottino (1988) [7], Magzhan and Jani (2013) [8], and Madkour et al. (2017) [9].

Different research works have proposed several techniques and optimizations for efficient, shortest path algorithm performance to handle massive data. Fuhao and Jiping (2009) [10] proposed a novel algorithm using adjacent nodes that was more effective at analyzing networks with massive spatial data. Chakaravarthy et al. (2016) [11] introduced a scalable parallel algorithm that significantly reduces internode communication traffic and achieves high processing rates on large graphs. Yang et al. (2019) [12] focused on routing optimization algorithms based on node-compression in big data environments, addressing the common problem of finding the shortest path in a limited time and the number of connected nodes. Liu et al. (2019) [13] presented a navigation algorithm based on a navigator

data structure that effectively navigates the shortest path with high probability in massive complex networks.

On the other hand, research has been developed that generates complex graphs with information that represents vehicle paths or sensor networks, such as the work of Ma et al. (2021) [14]. Li et al. (2023) [15] investigated a massive MIMO uplink system, where a transmitter with two antennas has to upload data in real time to a BS with a more significant number of antennas. Techniques and algorithms are required for high-speed and accurate graph processing in their different forms.

Unfortunately, the shortest path algorithm can propagate local distances to determine the distance between two concepts that are not directly connected by a relation in the ontology. This process is applied to each pair of concepts, allowing semantic proximity to be established. However, these techniques compute such distances at a high computational rate since traditional algorithms, like Dijkstra, have a complexity of $\mathcal{O}(n^3)$, which is disadvantageous for massive graphs. The computational complexity of DIS-C is $\mathcal{O}(n^2 \log(n))$ (using the Floyd–Warshall algorithm); therefore, using techniques such as DIS-C in its original version is inadequate to handle very large graphs directly. This paper proposes the use of other approaches to solve the APSP calculation. The pruned Dijkstra and the Sketch-based algorithms were used in the first approach. Considering that the analysis of the obtained results includes different datasets and that the generated graphs are sparse, a centrality-based strategy is presented to reduce the computational complexity.

The manuscript is organized as follows: Section 2 summarizes the state-of-the-art related to this field. Section 3 presents an analysis of the complexity of the DIS-C algorithm to reference those parts that need to be accurately improved, the algorithms proposed to enhance performance, and the demonstration of its precision and the associated computational complexity. In Section 4, we establish the execution scenarios, which include the total number and type of tests, present the experimental results, and discuss their significance. Finally, Section 5 outlines the conclusion and future work.

2. Related Work

2.1. Semantic Similarity

Conceptual distance is closely related to semantic relationship and semantic similarity. The semantic relationship, on the other hand, is not necessarily conditioned by a taxonomic relationship; in general terms, these relationships are given by any relationship between concepts, such as meronymy, antonymy, functionality, cause-effect, holonymy, hyponymy, hyperonymy, and so on. Moreover, we define *conceptual distance* as the space that separates two concepts within a conceptualization; in other words, it is the degree of proximity between concepts Quintero et al. (2023) [3]. We can establish a rule that indicates conceptual closeness in terms of how close to 0 (in terms of distance) the assigned value is. However, computing such distances is entirely dependent on the context of the concepts. In order to calculate it, the concepts must first be represented in a model that allows for the expression of the semantic relations and defining the measurement metric from the different underlying structures and representations.

In the network model, ontologies are used to address the problem of formally representing the semantics of information and its relationships Meersman (1999) [16]. The mentioned structures evaluate the conceptual distance between terms. The present work considers the assumption that an ontology can be represented as a strongly connected graph (conceptual graph); in this context, some authors have proposed many approaches to evaluate the closeness between concepts. The contributions of Bondy (1982) [17], West et al. (2001) [18], Bollobás (1998) [19], and Gross et al. (2018) [20] describe the basic concepts, mathematical foundations, and representative problems of graph theory.

On the other hand, since their conception in 2012 by Google [21], knowledge graphs have been the research subject in several articles. This term has been frequently used in academic and business contexts, typically associated with semantic web technologies, linked data, large-scale data analytics, and cloud computing Ehrlinger and Wöß (2016) [22].

The works presented by Fensel et al. (2020) [23] and Ehrlinger and Wöß (2016) [22] focused on reviewing several definitions of a knowledge graph to develop a formal definition. A Knowledge graph describes structured information and can be applied to specific domains such as answering questions, recommending, and retrieving information Zou (2020) [24]. In addition, Pujara et al. (2013) [25] present a method for transforming uncertain extractions about entities and their relationships into a knowledge graph by filtering noise, inferring missing information, and identifying the relevant candidate facts.

In the literature, the authors have identified four groups of methods, models, and techniques for computing semantic similarity: (a) edge counting-based, (b) feature-based, (c) information content-based, and (d) hybrids. In this work, information content-based methods are used to determine semantic similarity based on their direct relationship to the measurement of conceptual distance. The underlying knowledge base provides these methods with a structured representation of terms or concepts connected by semantic relationships. In addition, it proposes an unambiguous semantic measure since it considers the real meaning of the terms Sanchez et al. (2012) [26].

Edge counting-based methods consider an ontology a connected graph, where the nodes are concepts, and the edges are relationships (almost always taxonomic). The number of edges separating two concepts determines the similarity between them. Table 1 summarizes some representative edge counting-based methods. Furthermore, it describes the expression used to calculate the similarity (sim)/distance (dis) and a summary of the variables involved.

Table 1. Edge counting-based methods. $dis \equiv$ conceptual distance; $sim \equiv$ similarity between two concepts: a and b.

Reference	Expression	Description
Rada et al. (1989) [27]	$dis_{rada}(a,b) =$ length of the path from a to b	
Wu and Palmer (1994) [28]	$sim_{wu}(a,b) = \frac{2N_c}{N_a+N_b+2N_c}$	a and b are concepts within the hierarchy; c is a less common super concept of a and b. N_a is the number of nodes on the path from a to c, N_b is the number of nodes in the path from b to c, and N_c is the number of nodes in the path from c to the hierarchy root.
Hirst and Stonge (1995) [29]	$sim_{hirst}(a,b) = C - \lambda(a,b) - K * C_h(a,b)$	C and K are constants, $\lambda(a,b)$ is the length of the shortest path between a and b, and $C_h(a,b)$ is the number of times that the path changes direction.
Li et al. (2003) [30]	$sim_{li}(a,b) = e^{-\alpha\lambda(a,b)}\frac{e^{\beta h}-e^{-\beta h}}{e^{\beta h}+e^{-\beta h}}$	λ is the length of the shortest path between a and b, h is the minimum depth of LCS (the more specific concept that is an ancestor of a and b) in the hierarchy, and $\alpha \geq 0$ and $\beta > 0$ are parameters that scale the contribution of the length and depth of the shortest path, respectively.
Shenoy et al. (2012) [31]	$sim_{shenoy}(a,b) = \frac{(2Ne^{-\frac{\gamma L}{N}})}{N_a+N_b}$	L is the shortest distance between a and b, calculated by taking into account the direction of the edges. Each vertical direction is assigned a value of 1; one is added for every direction change. N is the depth of the entire tree. N_a and N_b are the distances from the root to the concepts a and b, respectively. γ is 1 for the adjacent concepts and 0 for the rest.

Feature-based methods are based on the Tversky (1977) [32] similarity model, obtained from set theory, which assesses the similarity between concepts based on their properties, subtracts the common and uncommon features of the terms, and attempts to overcome the limitations of edge count-based measures regarding the fact that taxonomic links in an ontology do not necessarily represent uniform distances; Sanchez et al. (2012) [26]. These types of measurement have the potential to be applied in the calculation of semantic similarity supported by crossed ontological structures. The main limitation is that it requires ontologies or knowledge sources with semantic features, such as dictionaries, and most ontologies rarely contain semantic features other than relations.

The Lesk study [33] proposed the homonymous computing method, which consists of calculating the Lesk measure to disambiguate words in texts. The work presented by Banerjee and Pedersen (2003) [34] is an extension of this and used the terms or synsets provided by WordNet. However, in the modern world, there are multiple sources of information, such as social networks and internet blogs, which provide a large amount of information. For this reason, other works have used different sources of knowledge, such as Wikipedia. In this context, the study of Jiang et al. (2015) [35] evaluated the existing specific methods based on the features, using a formal representation of concepts.

Information content-based approaches improve some of the limitations of edge-counting-based methods. These methods are based on computing information content (IC) and similarity, (sim). Table 2 summarizes the IC-based methods. In addition, this table describes the expressions used to calculate IC and briefly explains the variables involved.

Hybrid methods combine two or more techniques for calculating semantic similarity. Quintero et al. (2019) [5] proposed the DIS-C method to compute the conceptual distance between two concepts in an ontology based on a fundamental approach that uses the topology of an ontology to compute the weight of relationships between concepts. DIS-C can incorporate various relationships between concepts, such as meronymy, hyponymy, antonymy, functionality, and causality.

The DIS-C method considers the conceptual distance as the space separating two concepts within a conceptualization represented as a directed graph, and it is related to the difference in the information content of the concepts in their definitions. The conceptual structure is irrelevant since the method assigns a distance value to each relationship type and transforms this into a weighted directed graph; as a consequence, DIS-C can be applied to any conceptual structure. The only requirement for the algorithm is that the conceptualization be a 3-tuple of elements $K = (C, \Re, R)$, where C is a set of concepts, $\Re$ is the set of relation types, and R is the set of relationships in the conceptualization. In summary, the method is described through the following steps:

1. Assign to each relation $\rho \in \Re$ as an arbitrary conceptual weight defined in each direction of the relationship.
2. A directed and weighted graph (conceptual graph) is created, where the vertices are the concepts contained in the conceptualization, and the edges are the relationships that connect each pair of concepts. Each edge has its counterpart in the opposite direction with a different weighting. Finally, to compute the weighting, the generality of each concept is necessary.
3. Calculate the APSP length between each pair of vertices, i.e., diffuse the conceptual distance to all concepts.

Table 2. Information content-based methods. $sim \equiv$ similarity; $IC \equiv$ information content.

Reference	Expression	Description
Resnik (1995) [36]	$sim_{resnik}(a,b) = I_C(L_{CS}(a,b))$	Based on the concept of the least common subsumer (LCS), information content (IC) is calculated when the terms share an LCS. A high IC value indicates that the term is more specific and clearly describes a concept with less ambiguity.
Jiang and Conrath (1997) [37]	$sim_{jiang}(a,b) = I_C(a) + I_C(b) - 2I_C(L_{CS}(a,b))$	It focuses on determining the link strength of an edge connecting a parent node to a child node. These taxonomic links between concepts are reinforced by the difference between a concept's IC and its LCS.
Gao et al. (2015) [38]	$sim_{gao}(a,b) = max(e^{-\alpha\lambda(a_s,b_s)} \mid a_s \in S(a), b_s \in S(b))$	Where $\alpha > 0$ is a constant and $S(x)$ is the set of meanings of the concept x.
Jiang et al. (2017) [39]	$I_{C_{jiang\_1}}(a) = -\log(p(a)) = -\log(\frac{\sum_{c\in h(a)\cup a}\lvert p_g(c)+1\rvert}{\sum_{c\in C_A}\lvert p_g(c)+1\rvert})$	Where $h(a)$ is the set of hyponyms for a, $p_g(c)$ is the set of pages in category c, and C_A is the set of categories.
	$I_{C_{jiang\_2}}(a) = 1 - \frac{\log(h(a)+1)}{\log(\lvert CA\rvert)}$	The second proposed approach is the combination of IC by using category structure and the extension of ontology-based methods.
	$I_{C_{jiang\_3}}(a) = \gamma\left(1 - \frac{\log(h(a)+1)}{\log(\lvert C_A\rvert)}\right) + (1-\gamma)\left(\frac{\log(d(a)+1)}{\log(d_{max})}\right)$	Generalization of the Zhou et al. (2008) [40] approach. Where γ is an adjustment factor for the weight of the two features involved in the IC calculation, $d(a)$ is the depth of the leaf a, and d_{max} is the maximum depth of a leaf.
	$I_{C_{jiang\_4}}(a) = -\log\left(\frac{\frac{\lvert l(a)\rvert}{\lvert H(a)\cup a\rvert}+1}{\lvert l_{max}\rvert+1}\right)$	Generalization of the Sanchez et al. (2011) [41] approach. Where $l(a)$ is the set of leaves of a in the category hierarchy, $H(a)$ is the set of hypernyms, and l_{max} is the maximum number of leaves in the hierarchy.

2.2. All Pairs Shortest Path Problem (APSP)

Computing the shortest path between each pair of concepts in large graphs is computationally expensive. This type of problem is known in graph theory as "All Pairs Shortest Path" (APSP) [42]. In [5], we used two well-known algorithms: the Floyd–Warshall and Johnson–Dijkstra algorithms. The Floyd–Warshall algorithm is based on transitive Boolean matrix closure [43], with a computational complexity of $\mathcal{O}(n^3)$ (n is the number of nodes, and m is the number of edges in the graph). On the other hand, Johnson-Dijkstra has a complexity $\mathcal{O}(mn + n^2\log(n))$ (using Fibonacci heaps); however, it requires that there be no negative cycle in the graph, and when $m \in \mathcal{O}(n^2)$, the complexity is equal to $\mathcal{O}(n^3)$ [44].

When using accelerated matrix multiplication algorithms, the cost is $\mathcal{O}(n^{2+\epsilon})$, $\epsilon < 1$ [45]. Techniques, such as divide and conquer [46], or metaheuristic algorithms [47] are also employed. Another approach is to use new-generation hardware, such as the GPU, to parallelize existing algorithms and reduce runtime [48]. According to Reddy (2016) [49], most algorithms are based on the following two types of computational models:

- Addition comparison model: Assume that the inputs are real weighted graphs, where the only operations allowed on real data are comparison and addition.
- Random access machine (RAM) model: Shortest path algorithms assume that the inputs are weighted graphs of integers manipulated by addition, subtraction, comparison, shift, and various logical operations [50] (the most commonly used model).

Despite the significant interest in the APSP problem, there have only been minor advances in computational complexity over $\mathcal{O}(n^3)$ for general graphs based on distance product computation, which is closely related to matrix multiplication. Although subcubic

algorithms exist, they only exploit the properties of specific graphs or advantage sparse graphs, so there is no real subcubic combinatorial algorithm. Table 3 describes the algorithms developed with their computational complexity over time, and Figure 1 shows a comparison of the execution time of each algorithm.

Although it is possible to solve the APSP problem in polynomial time, its application in large graphs is not practical. For this reason, it is interesting to analyze heuristic and metaheuristic methods and approximation algorithms with regard to this problem. The clear advantage of this approach is the speed gain, whereas the disadvantage is the accuracy of the computation. In the literature, the authors have employed some relevant investigations into this class of algorithms with regard to the APSP problem: genetic, evolutionary, and optimization algorithms per ant colony.

Attiratanasunthron and Fakcharoenphol (2008) [51] presented a rigorous analysis of the running time of ACO in the shortest path problem. The n-ANT method is inspired by the 1-ANT [52] and AntNet [53] algorithms. Horoba and Sudholt (2009) [54] formalized and optimized the n-ANT algorithm to solve the APSP. The M_{APSP} algorithm is a generalization of M_{SDSP} presented in the same research study. This algorithm decodes the single-source shortest path problem. The computational complexity of M_{APSP} is $\mathcal{O}\left(\Delta l l^* + \frac{l \log(\Delta l)}{p}\right)$, where Δ is the maximum degree of the graph, l is the maximum number of edges in any shortest path, and $l^* := max\{l, \ln(n)\}$.

Table 3. Analytical runtimes for different APSP algorithms.

$\mathcal{O}(.)$	Reference	Year
n^3	Dijkstra/Floyd–Warshall	1959/1962
$\frac{n^3 \log^{\frac{1}{3}}(\log(n))}{\log^{\frac{1}{3}}(n)}$	Fredman (1976) [55]	1976
$\frac{n^3 \log^{\frac{1}{2}}(\log(n))}{\log^{\frac{1}{2}}(n)}$	Takaoka (1992) [56]	1991
$\frac{n^3}{\log^{\frac{1}{2}}(n)}$	Dobosiewicz (1990) [57]	1990
$\frac{n^3 \log^{\frac{5}{7}}(\log(n))}{\log^{\frac{5}{7}}(n)}$	Han (2004) [58]	2004
$\frac{n^3 \log^2(\log(n))}{\log(n)}$	Takaoka (2004) [59]	2004
$\frac{n^3 \log(\log(n))}{\log(n)}$	Takaoka (2005) [60]	2005
$\frac{n^3 \log^{\frac{1}{2}}(\log(n))}{\log(n)}$	Zwick (2004) [61]	2004
$\frac{n^3}{\log(n)}$	Chan (2008) [62]	2005
$\frac{n^3 \log^{\frac{5}{4}}(\log(n))}{\log^{\frac{5}{4}}(n)}$	Han (2006) [63]	2006
$\frac{n^3 \log^3(\log(n))}{\log^2(n)}$	Chan (2010) [64]	2007
$\frac{n^3}{2^{\Omega(\log(n))^{1/2}}}$	Williams (2018) [65]	2014

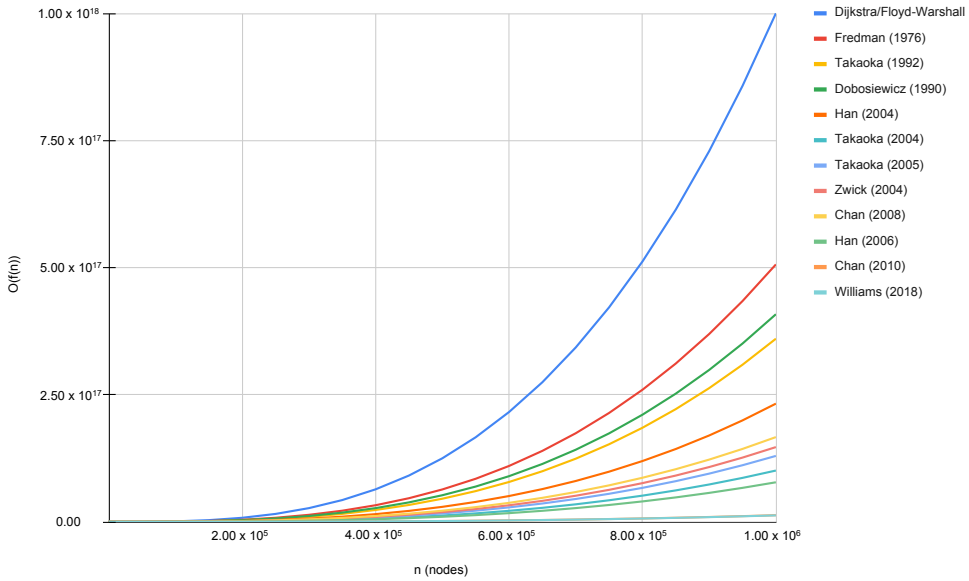


Figure 1. Comparison between the execution times of the APSP algorithms. This figure shows the number of operations (y-axis) required for each algorithm as the number of nodes increases (x-axis) [55–65].

On the other hand, Chou et al. (1998) [66] used a hierarchical approach to decompose a network (graph) into several smaller subnets. Therefore, when it is necessary to search for a shorter path, a smaller graph is evaluated. The authors compared the computational complexity between Dijkstra's algorithm (no error) and two versions of the algorithm called HA: Best HA (best results: 21.5% error) and Nearest HA (the fastest execution instance: 4.68% error). Mohring et al. (2007) [67] presented a similar report that focuses on the shortest path to a single source.

Other research studies that improve the asymptotic limit concerning the $\mathcal{O}(n^3)$ are Baswana et al. (2009) [68] and Yuster (2012) [69]. The first applied it to unweighted and undirected graphs and introduced the first algorithm, (n^2), to compute the APSP with a 3-approximation, meaning that for each pair of vertices $u, v \in V$, the distance is less than or equal to $3 * \delta(u, v)$, where $\delta(u, v)$ is the shortest distance of the path. In addition, two algorithms are presented for a 2-approximation, with a complexity of $\mathcal{O}(m^{2/3} n \log(n) + n^2)$ and $\mathcal{O}(n^2 \log(n)^{3/2})$, and the shortest distance being $2\delta(u, v) + 1$ and $2\delta(u, v) + 3$, respectively. In both cases, there is a predefined error bound.

Finally, there is the sketch-based approach, which requires two extensive processes. The first is to preprocess, in which the result is a data structure that allows the distance between two nodes to be consulted. This data structure and a distance oracle were introduced in [70]. The second process is related to queries, which is a case of making queries to the oracle, and it returns, in a specified time limit, the result of the shortest distance between two nodes.

In comparison to identical algorithms, these algorithms reduce the time by several orders of magnitude. Das Sarma et al. (2010) [71] is one of the first studies that applied this approach to complex graphs with hundreds of millions of nodes and billions of edges. The description of the algorithm is the following: sample a small number of node sets from the graph, called seed node sets. Then, for each node of the graph, the closest seed within each of these seeds is found. The sketch of a node is the set of the closest seeds and the distance to them. Thus, given a pair of nodes, u and v, the distance between them can be estimated by searching for a common seed in their sketches. However, there are no theoretical guarantees on the degree of optimization of the method for directed graphs. The results showed a better average close to the actual distance when applied to directed graphs.

Wang et al. (2021) [72] focused on finding all the shortest paths in large-scale undirected graphs. The principal difference, regarding the work of Das Sarma et al. (2010) [71],

is that Wang et al. (2021) [72] considered the computational complexity needed to perform the preliminary computation. The standard method employed in this problem involves applying a BFS algorithm to the seed node set. It is known that the complexity of BFS is $\mathcal{O}(m+n)$ when using an adjacency list or $\mathcal{O}(n^2)$ for an adjacency matrix. This complexity is a drawback when processing large graphs. For this reason, they investigated whether it is possible to speed up the preliminary computation using the [73] pruned reference point-labeling technique, i.e., applying a pruned BFS. The specific results of this stage showed linear time in terms of its construction, i.e., $\mathcal{O}(m)$. The results indicated a significant improvement, especially in terms of runtime for large graphs.

3. Materials and Methods

The DIS-C presented by Quintero et al. (2019) [5] uses two algorithms: a basic one, which calculates the conceptual distance within a conceptualization, $K = (C, \Re, R)$, and the weights δ^{ρ} for each relationship type $\rho \in Re$. This algorithm converts the conceptualization K into a graph $G = (V_G, A_G)$ with $V_G = C$ and $A_G = R$, through which we compute the conceptual distance by calculating the APSP in graph G. The solution described in this paper combines and evaluates several approaches to optimize the APSP. The second algorithm enables automatic weighting, which allows the conceptual distance to be computed automatically without having to provide the weights of the relationship types. Figure 2 shows the flowchart corresponding to the DIS-C algorithm with automatic weighting (where $i(a)$ is the entry degree of node a, and $o(a)$ is the exit degree of node a; $w_i(a)$ and w_o are the costs of entering and leaving node a, respectively; $g(a)$ is the generality of node a; V_j^{γ} and A_j^{γ} are the nodes and edges, respectively, of the graph Γ_K^j, which is the graph corresponding to the conceptualization K in the j-th iteration. Thus, $w_j(a,b)$ is the cost of the edge from node a to node b; p_w is a geometric weighting factor. Furthermore, $M_{\Gamma_K^j}$ is the APSP distance matrix, and ρ^* is the set of edges representing the relation ρ. Finally, ϵ_K is the convergence threshold of the algorithm), which will be referred to simply as DIS-C in this paper. Although the DIS-C algorithm consists of several processes, in terms of computational complexity, it is related to the execution of the APSP.

At this stage, we can assume that the computational complexity of the DIS-C algorithm for a conceptualization with n concepts is $\mathcal{O}(n^2 \log n)$(A detailed analysis of this topic is presented in [74]).

3.1. The Pruned Dijkstra Algorithm

The algorithm implemented is described in Algorithm 1, and the index construction algorithm is presented in Algorithm 2. Appendix A explains the pruned Dijkstra algorithm. We executed the pruned Dijkstra in the order $v_1, v_2, \ldots, v_n$, and the sequence of vertices is arbitrary but relevant for good algorithm performance. Ideally, starting from the central vertices, many shortest paths will cross them. There are many strategies to order the vertices; one of the most common is based on ordering by centrality. In this work, we choose to order the vertices by the degree of centrality. The graphs generated by the semantic networks behave like networks without scale, where high-degree vertices provide high connectivity to the network. In this way, the shortest path between two vertices is usually through these nodes.

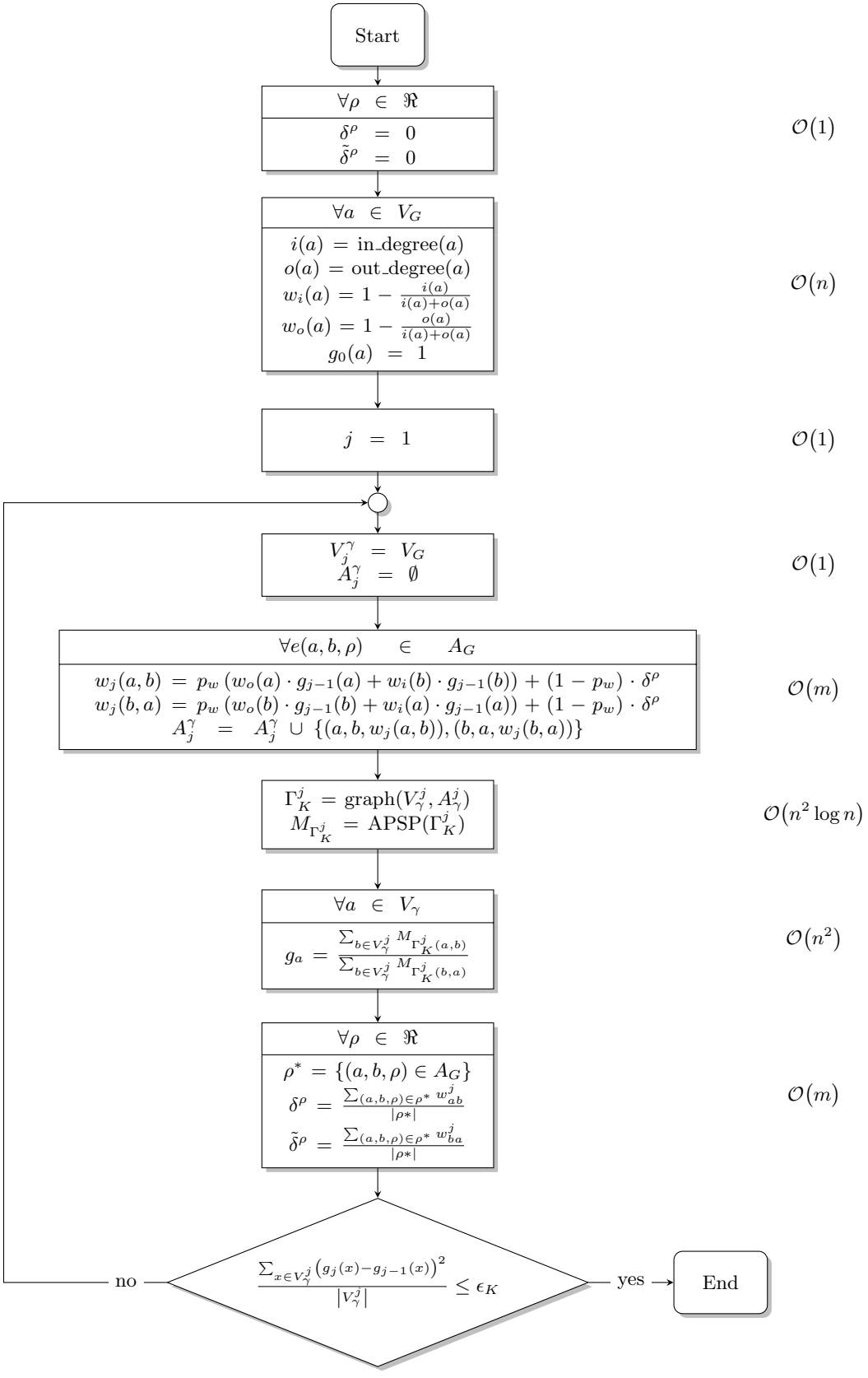


Figure 2. The DIS-C algorithm flow diagram.

Algorithm 1: The pruned Dijkstra

Function `PDijkstra`(G, u_k, L, dir)**:**

$Q \leftarrow (u_k, 0)$
$d \leftarrow \emptyset$
$S \leftarrow \emptyset$
while Q **do**
  $u, \delta \leftarrow \text{pop}(Q)$
  if $u \notin d$ **then**
    append u to d
    $d_u \leftarrow \delta$
    if *dir is 'in'* **then**
      `/* Determine if the best path is already known. In that case, it is not necessary to continue on that path, therefore it is pruned. */`
      if $\mathit{query}(u_k, u, L) \leq d_u$ **then**
        continue
      end
    else
      if $\mathit{query}(u, u_k, L) \leq d_u$ **then**
        continue
      end
    end
    `/* There is no previous path, so the current path is added to the index of the node being processed, and ordinary Dijkstra execution is performed. */`
    $L_u^{dir} \leftarrow L_u^{dir} \cup \{(u_k, d_u)\}$
    forall $v \in N_1^G(u)$ **do**
      if *dir is 'in'* **then**
        $d_{uv} \leftarrow d_u + w_{u,v}^G$
      else
        $d_{uv} \leftarrow d_u + w_{v,u}^G$
      end
      if $v \notin S$ *and* $d_{uv} < S_v$ **then**
        $S_v \leftarrow d_{uv}$
        push(Q, v, d_{uv})
      end
    end
  end
end
`/* Review the distance from the source to the target nodes according to a two-hop cover. */`
Function `query`(s, t, L)**:**
  $d \leftarrow \infty$
  foreach $k \in L_s^{in}$ **do**
    if $k \in L_t^{out}$ **then**
      $d \leftarrow \min\left(d, L_s^{in}(k) + L_t^{out}(k)\right)$
    end
  end
  return d
end

Algorithm 2: APSP_PDijkstra

Function `APSP_PDijkstra`(G, u_k, L, dir)**:**

$\forall v \in V, L_v^{in} \leftarrow \emptyset$
$\forall v \in V, L_v^{out} \leftarrow \emptyset$
for $k = 1, 2, \cdots, |V|$ **do**
  $L^{out} \leftarrow$ `PDijkstra` $(G, k, L, \text{'out'})$
  $L^{in} \leftarrow$ `PDijkstra` $(G, k, L, \text{'in'})$
end
return L
end

In order to formalize the proposed approach, it must be demonstrated that the algorithm is correct and produces the theoretically proposed result. In order to prove the correctness of the method, it suffices to show that the algorithm computes a two-hop coverage, i.e., $Q(s, t, L_k^{'}) = d_G(s, t)$ for any $s, t \in V$. This statement is equivalent to proving that, when given any vertex, $s \in V$, all other vertices belong to the input and output coverages of vertex s. Let L_k be the index built without the pruning process; so, it is a two-hop coverage; $L_k^{'}$ is the index generated by applying the pruning process. Since there exists a node, v_j, such that $(v_j, d_G(v_j, s)) \in L_k(s)^{in}$, $(v_j, d_G(v_j, t)) \in L_k(t)^{out}$ and $\delta_{v_j,s} + \delta_{v_j,t} = d_G(s, t)$, the goal is to show that v_j also exists in $L_k^{'}(s)^{in}$ and $L_k^{'}(t)^{out}$.

Theorem 1. *For $k = 1, \ldots, n$, and $s, t \in V$, $Q(s, t, L_k^{'}) = Q(s, t, L_k)$*

Proof. Let $s, t \in V$, assuming there is a path between them, and j be the smallest number, such that $(v_j, d_G(v_j, s)) \in L_k(s)^{in}$, $(v_j, d_G(v_j, t)) \in L_k(t)^{out}$ and $\delta_{v_j,s} + \delta_{v_j,t} = d_G(s, t)$. It must show that $(v_j, d_G(v_j, s))$ and $(v_j, d_G(v_j, t))$ are contained in $L_k^{'}(s)^{in}$ and $L_k^{'}(t)^{out}$, respectively; that is, $Q(s, t, L_k^{'}) = Q(s, t, L_k)$. In order to prove that $v_j \in L_k^{'}(t)^{out}$, first consider, for any $i < j$, in that $v_i \in P_G(s, v_j)$, where $P_G(a, b)$ is a path between a and b. Suppose $v_i \in P_G(v_j, t)$; through inequality, we obtain the following:

$$\begin{aligned} Q(s, t, L_k) &= d_G(s, v_j) + d_G(v_j, t) \\ &= d_G(s, v_j) + d_G(v_j, v_i) + d_G(v_i, t) \\ &\geq d_G(s, v_i) + d_G(v_i, t) \end{aligned}$$

Since $(v_i, d_G(v_i, s)) \in L_k(s)^{in}$ and $(v_i, d_G(v_i, t)) \in L_k(t)^{out}$, it contradicts the fact that j is the smallest number. Therefore, $v_i \notin P_G(v_j, t)$ for any $i < j$.

Now, we prove that $(v_j, d_G(v_j, t)) \in L_k^{'}(t)^{out}$. Suppose we perform the j-th pruned Dijkstra iteration from node v_j to construct index $L_j^{'out}$. Let $t \in P_G(v_j, t)$. Since there is no $v_i \in P_G(v_j, t)$ for $i < j$, this means that there is no vertex in the i-th pruned Dijkstra iteration that covers the shortest path from v_j to t, and, therefore, $Q(v_j, t, L_{j-1}^{'}) > d_G(v_j, t)$. Consequently, vertex t is visited without pruning, and $(v_j, d_G(v_j, t)) \in L_j(t)^{out} \subseteq L_k(t)^{out}$ is added.

Finally, suppose we execute the j-th pruned Dijkstra iteration from v_j, but this time, we construct $L_j^{'in}$. If s and t are reachable in G, it means that there is a path between v_j and s when considering the input edges or, equivalently, there is $P_G(s, v_j)$. Since we have already proved that $v_i \notin P_G(v_j, t)$ for $i < j$, this means that v_j is a neighbor of s or $v_j = s$, and since v_j is the initial vertex of the path $P_G(v_j, t)$, then v_j is the neighbor with the least distance sharing s. In all cases, $Q(s, v_j, L_{j-1}^{'}) > d_G(s, v_j)$; therefore, the node s is visited without pruning (remember that it considers the input edges). Consequently, $(v_j, d_G(v_j, s)) \in L_j(s)^{in} \subseteq L_k(s)^{in}$ is added. □

As mentioned earlier, the order in which the nodes are processed affects the performance of the algorithm. Is there an order of execution such that the construction time is minimized? The answer is 'yes', but finding such an order is challenging. So, there are two approaches: the first is to find the minor nodes that cover all the shortest paths, which is a much bigger problem since it is an instance of the *minimum vertex coverage problem*, which is NP-hard. The second approach is a parameterized perspective; specifically, the parameter explored is the *width of the tree*. It also relies on the process of the *centroid decomposition of a tree* [75]. This approach derives the following lemma; the formal proof is given in [73].

Lemma 1. *Let w be the width of the graph of G. There is a vertex order where the pruned Dijkstra method preprocesses in $\mathcal{O}\left(wm\log(n) + w^2n\log_2(n)\right)$ time, stores an index in $\mathcal{O}\left(wn\log(n)\right)$ space, and answers each query in $\mathcal{O}\left(w\log(n)\right)$ time.*

Considering that this research is not looking for minimum two-hop coverage, using the upper bound of $\mathcal{O}\left(wm\log(n) + w^2n\log_2(n)\right)$ for the algorithm can be considered an inconsistency. However, there is no guarantee that the selected order (degree centrality) is the same as the order required to achieve this computational complexity. We conclude that, according to our implementation, it cannot be faster than that theoretically obtained using the order with which we achieve $\mathcal{O}\left(wm\log(n) + w^2n\log_2(n)\right)$. This means that there is a lower bound, $\Omega(wm\log(n) + w^2n\log_2(n))$, in the implementation. In addition, when referring to the Dijkstra algorithm, we can hypothesize that the construction of the coverage using the pruned process, regardless of the order of execution, is much smaller than for the construction of the naive coverage in highly connected graphs, i.e., that $\mathcal{O}\left(mn + n^2\log(n)\right)$.

Thus, we concluded that this lemma indicates an optimal order for processing vertices. Unfortunately, it is extremely complicated to find it.

3.2. The Sketch-Based Algorithms

It is necessary to compute two sketches for each node. The first $\texttt{SKETCH}^{out}(u)$ and $\texttt{SKETCH}^{in}(u)$ represent the distance from node $u \in V$ to the nearest seed and inverse. This is carried out by running the search algorithm two different times, with one employing the input edges and the other using the output edges. We can randomly choose S when considering some criteria to guide the choice; for example, take into account discarding those nodes for which the degree is less than two. Alternatively, the structure of the graph and its specific properties (The strategy for choosing the seed nodes is to emphasize the nodes with the highest value of generality in the conceptual graph; this implies that the most general concepts will be chosen) can be employed. Appendix B provides some complementary aspects of sketch-based algorithms. Algorithm 3 describes the proposed pseudocode for generating the sketches of each node.

Algorithm 3: Sketches_DIS-C: offline sampling

Function `offlineSampling`(s, t, L)**:**

$R \leftarrow \log(|V|)$; `/* Number of seeds */`
$S \leftarrow \emptyset$
$samples \leftarrow \emptyset$
for $i = 0, 1, \cdots, R$ **do**
  $S_i \leftarrow choice(V, 2^i)$; `/* Choice a set of` 2^i `seed nodes */`
end
forall $s \in S$ **do**
  $\mathcal{S}^{in} \leftarrow$ `Dijkstra_Multisource`(G, s, in_edges)
  $\mathcal{S}^{out} \leftarrow$ `Dijkstra_Multisource`(G, s, out_edges)
end
return $\mathcal{S}$
end

For each sample, the closest seed node and its distance are obtained for each node. It is carried out by employing the classic Dijkstra multi-source algorithm. In this form, the seed nodes are computed for each $u \in V$ (see Algorithm 4).

Algorithm 4: Sketches_DIS-C: k sketches offline

```
Function K_sketches(G, K):
    S ← ∅
    for k = 0, 1, ··· , K do
        S_k ← offlineSampling (G)
        S ← S ∪ S_k
    end
    return S
end
```

In each iteration, we generate a sketch from an offline sample; therefore, at the end of the algorithm, there will be $k \log(n)$ seeds for each node. The second process of the method is the online one. The calculation of the approximate minimum distance between nodes u and v is carried out by looking at the distance from u and v to any node w that appears in both $\mathcal{S}_u^{in}$ and $\mathcal{S}_v^{out}$. So, we compute the path length from u to v by adding the distance from u to w plus the distance from w to v. The minimum distance is taken if there are two or more shared nodes w. Algorithm 5 shows the proposal to perform this calculation.

Algorithm 5: Sketches_DIS-C: k sketches online

```
Function sketchesDistance(u, v, S):
    s_u ← S_s^in
    s_t ← S_t^out
    d ← ∞
    foreach w ∈ s_u do
        if w ∈ s_v then
            d ← min (d, s_u(w) + s_v(w))
        end
    end
    return d
end
```

Finally, to solve the APSP problem, it is necessary to create another algorithm that is executed for each pair of nodes (Algorithm 6).

Algorithm 6: Sketches_DIS-C: k sketches online APSP

```
Function sketchesDistance(u, v, S):
    S ← K_sketches (G, k)
    D ← ∅
    foreach u ∈ V do
        foreach v ∈ V do
            D_{u,v} ← sketchesDistance (u, v, S)
        end
    end
    return D
end
```

In the sketch-based method, there is no accuracy test for general graphs. The method does not guarantee exact node labeling to determine the distance between all vertex pairs.

On the other hand, one of the objectives of this paper is to improve the computational complexity of the original algorithm (which is $\mathcal{O}(n^3)$). The following theorem demonstrates that the sketch-based method accomplishes this aim.

Theorem 2. *For a set of nodes, $S \subseteq V$, such that the set size is 2^w for $0 \leq w \leq n$; the complexity of the algorithm is $\mathcal{O}(w(m + n\log(n)))$.*

Proof. We know that the sketch-based method selects a value, w, based on the number of nodes in the graph, i.e., $w = f(V) \leq |V|$. Then, given w, Q_i subsets of nodes are created for $i = 0, 1, 2, \ldots, \log n$, such that the size of each one is $|Q_i| = 2^i \mid i = 0, 1, \ldots, w$. It follows that $S = \bigcup_{i=0}^{w} Q_i$. For each element of the Q_i subsets, the multi-source Dijkstra algorithm is applied, with the purpose of finding, for all the nodes of G, the closest node of the Q_i set. The multi-source Dijkstra has a complexity of $\mathcal{O}(m + n\log(n))$; for each subset of nodes, it is necessary to apply a single execution of Dijkstra algorithm (regardless of the size). Therefore, since there are w subsets of nodes, w Dijkstra processes must be performed from multiple sources, i.e., in total, the complexity is $\mathcal{O}(w(m + n\log(n)))$. □

For the implementation employed in this work, we determined w by using the logarithmic function $w = \log_2(n)$, where $n = |V|$. It ensured that $|S| < |V|$. Therefore, for the implementation performed, the method has a complexity of $\mathcal{O}(\log_2(n)(m + n\log_2(n)))$.

4. Results

This section describes the results of the different experiments carried out. In the first section, the performance metrics of the generated graphs are analyzed, and the control algorithm is compared with the proposed one. The second part analyzes the results of the semantic metrics.

4.1. The Datasets

The used datasets contain pairs of words assigned to a numerical value by human evaluation. These sets are divided into two categories: those that measure semantic similarity and those that measure semantic relationship. These sets are described in Table 4.

Table 4. List of datasets used in the evaluation process.

Dataset	Content	Type	Word Pairs	Nodes	Edges
MC30 [76]	Nouns	Similarity	30	5496	9583
RG65 [77]	Nouns	Similarity	65	5080	9271
PS [78]	Nouns	Similarity	65	5080	9271
Agirre201 [79]	Nouns	Similarity	201	29,738	75,120
SimLex665 [80]	Nouns	Similarity	665	44,798	138,122
MTurk771 [81]	Nouns	Relation	771	55,403	185,523
MTurk287 [82]	Nouns	Relation	287	25,576	58,637
WS245Rel [83]	Nouns	Relation	245	24,029	56,983
Rel122 [84]	Nouns	Relation	122	23,775	58,322
SCWS [85]	Nouns	Relation	1,994	53,052	183,366
All	-	-	4,445	119,034	818,788

Thus, this process is executed by a script and, for each dataset, selects a few words, generating the conceptual graph that connects them by applying the DIS-C algorithm to find the conceptual distances. We repeat this process with all word pairs, and finally, all

graphs generated from a set of words are merged to obtain a graph with all word pairs in the set. We also apply the DIS-C algorithm to combine the graph that contains the conceptual distances. The purpose of applying DIS-C to those graphs holding a few words is to measure the runtime with different sizes of graphs and, therefore, obtain a significantly larger record.

4.2. Evaluation Metrics

The evaluation metrics used for this research are divided into performance and semantics. The first corresponds to the runtime measurement, which is related to the number of nodes and edges of the graph. The semantic metrics measure the precision in estimating similarity, using the Pearson correlation coefficient (Equation (1)), the Spearman rank correlation coefficient (Equation (2)), and the harmonic score (Equation (3)).

$$r = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{(Y_i - \overline{Y})^2}} \tag{1}$$

$$p = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)}, \quad d_i = (x_i - y_i) \tag{2}$$

$$h = \frac{2rp}{r+p} \tag{3}$$

The Pearson correlation compares human similarity vectors and has been widely used to evaluate methods that estimate similarity between words and concepts. The Spearman rank correlation provides an invariant ranking and allows for a comparison of the underlying quality of word similarity measures. Additionally, harmonic scoring that combines Pearson and Spearman correlations to provide a unique weighted score for evaluating word similarity methods has been used. These correlation measurements are essential for evaluating and comparing similarity methods in the literature.

4.3. Generated Graphs and Performance

For each dataset, we generated a graph connecting each pair of words. Later, we computed the corresponding unified graphs. Table 4 shows the size of these graphs. The last row of the table (labeled *'All'*) refers to the union of all graphs; in addition, the algorithms were applied to this graph.

The following is a series of charts on the running time of each proposed algorithm. Figure 3 shows the runtime of the control algorithm, the pruned Dijkstra, and sketch-based algorithms. At specific points, the runtime increases in an "abnormal" way; this is caused by a parameter that differs from the size of the graph; specifically, it is the *convergence threshold* of the algorithm. It indicates that more iterations are required for this dataset, in particular, the graph structure, to converge with the threshold.

The convergence threshold is a function of the total generality of the graph, i.e., the sum of all generalities of each node. The total generality of the graph depends on the distance value between each pair of nodes provided by the APSP algorithm; for this reason, the algorithms that obtained the shortest actual distance always reach the convergence threshold in the same number of iterations. In this case, it is the control algorithm and the pruned Dijkstra. On the other hand, the sketch-based algorithm varies the number of iterations to reach the threshold, but in general, it cannot be less than the number achieved by the control algorithm. The reason for this is that the control algorithm determines the precise shortest distances; thus, the total generality of the graph is the smallest possible. The generality of the graph cannot be less than the minimum generality since it is known that the sketch-based algorithm (and any other approximation algorithm) cannot compute a shorter distance than the real one.

In Figure 3, it is possible to recognize that most of the proposed algorithms have a longer runtime than the control algorithm; this is primarily attributed to graph density; that is, the generated graphs have few edges and can be considered sparse graphs. Notice that only the sketch-based algorithm with $k = 1$ is faster than the control algorithm. However, the acceleration is not significant. In order to significantly improve the runtime, it must be reduced by at least one order of magnitude, considering that the runtime in the studied results is measured in days. Since the proposed algorithms are divided into two significant processes (construction and query), the runtimes of these two processes are measured separately. Figure 4 depicts this specific operation.

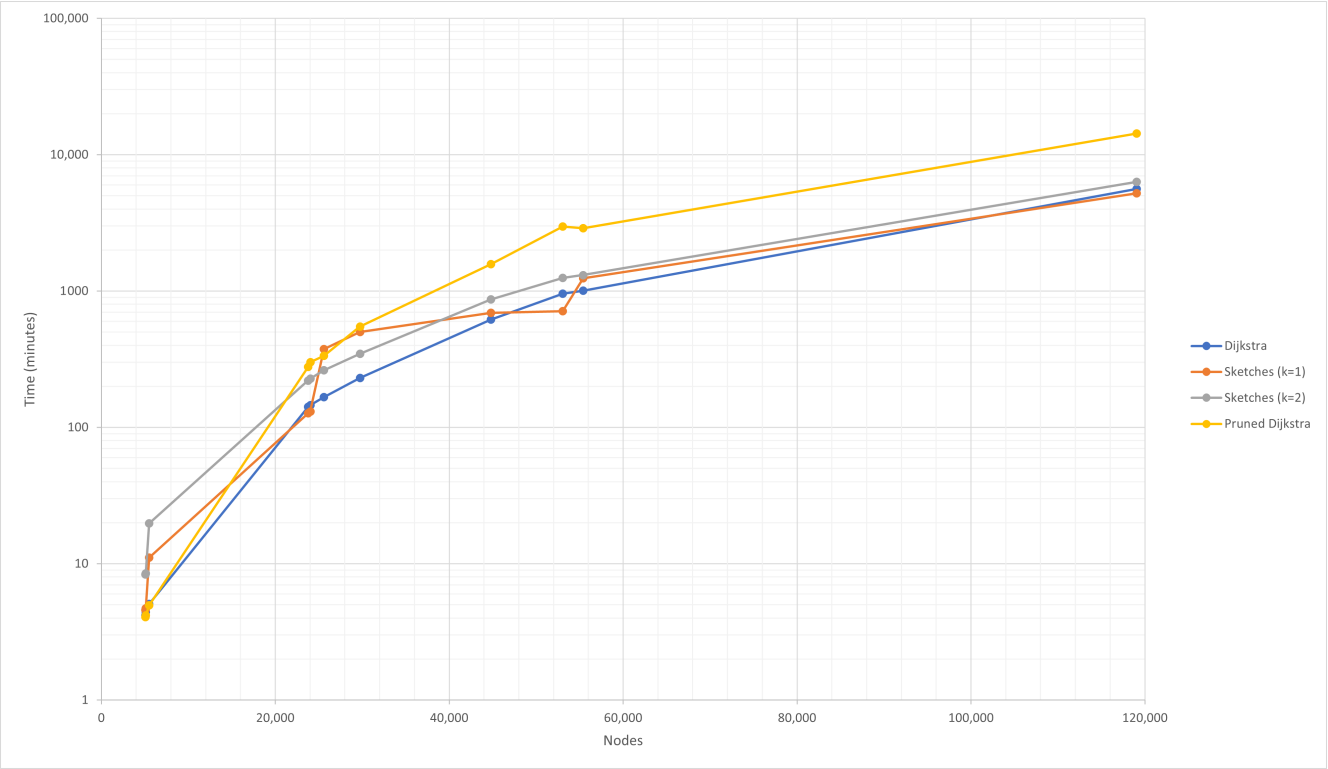
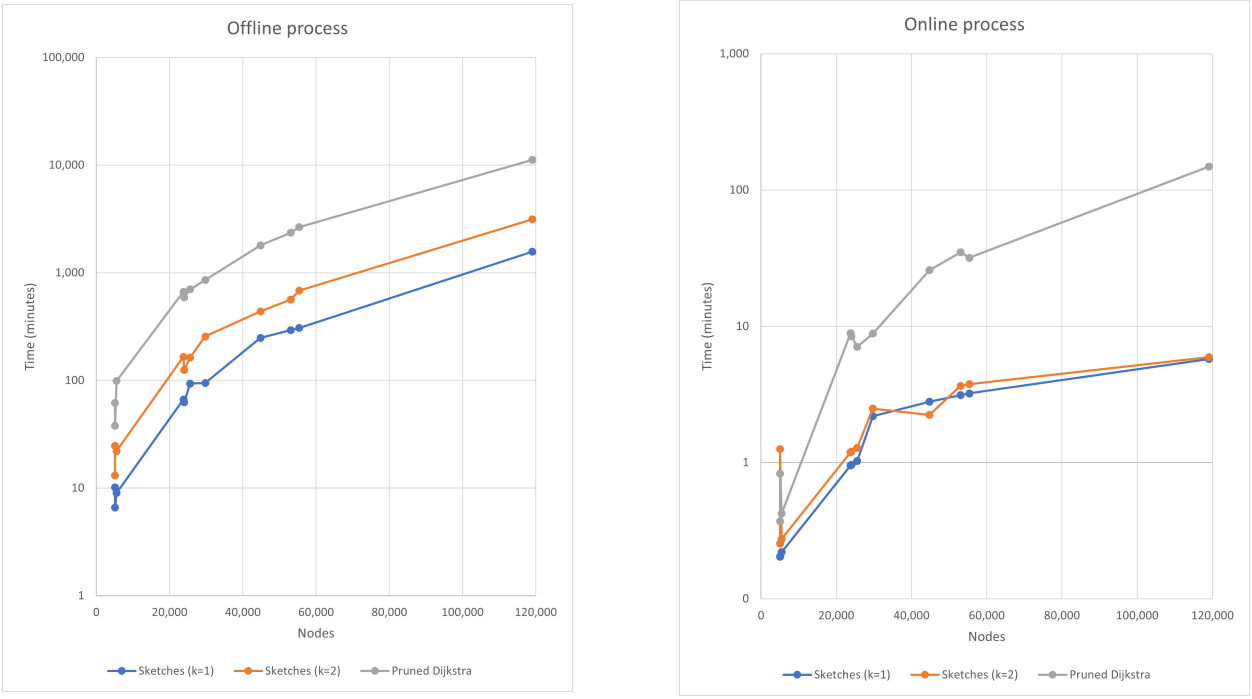


Figure 3. DIS-C runtime for the Dijkstra, pruned Dijkstra, and sketch-based algorithms.

The query time is the most important constraint in the algorithms. When analyzing the complexity of the query process, we found that the complexity of the sketch-based algorithm depends on the size of the coverages, and these are a function of k, so the response when increasing the value of k is predictable, as shown in Figure 4. In the case of the pruned Dijkstra algorithm, the coverage size is huge for more general nodes and decreases as a function of such generality. It induces some nodes to find their shortest distance, which takes longer; this pattern speeds up the overall query time. Although a binary search can be used instead of a linear one, it does not underestimate the quadratic factor inherent in the APSP, the behavior of which is reflected in the results. Considering this inconvenience, we conjecture that graph generality can be computed using only node coverages. This implies that solving the APSP to determine the generality is unnecessary, eliminating the quadratic factor of the APSP.



(**a**) Construction time (offline process).

(**b**) Query time (online process).

Figure 4. Comparative offline and online processes using pruned Dijkstra and sketch-based algorithms.

Based on the previous assumption, a framework for computing generality was established that only considers node coverage because generality was defined as the average of the conceptual distances from concept x to all other concepts, divided by the sum of the average conceptual distances and all concepts in the ontology. In the case of pruned Dijkstra, the more general concepts contain more significant coverage; their *influence area* is much larger than the less general concepts. This area can be used in a form where the generality of a node depends on itself. The principle of generality is respected by considering the information that other concepts provide for concept x (the distances of the coverage L_x^{in}) and the information that concept x itself provides to its related concepts (the distances of the coverage L_x^{out}). We can consider it as a generality measure based on the extended neighborhood of the nodes represented by their coverages. This is possible because the covers map the topology of the graph. So, we performed the generality computation, as per Equation (4). Henceforth, versions of DIS-C that use this type of generality are denoted as *simple*.

$$g(x) = \frac{\sum_{b \in L_x^{out}} \delta(b, x)}{\sum_{b \in L_x^{in}} \delta(b, x)} \tag{4}$$

In Figure 5, it is possible to study the total cost of running the simple algorithms in comparison with the others. An evident acceleration is obtained concerning the first versions of the proposed algorithms.

In the case of the simple pruned Dijkstra algorithm, there is a significant improvement in the runtime, reducing it by an order of magnitude from days to hours for the most extensive graph. In the case of the sketch-based algorithms, there is a more predictive, straight-line relationship with k, i.e., the runtime for $k = 2$ is two times lower than for $k = 1$, and so on.

An effective speedup is observed when using the simple version of the algorithms. Similarly, the use of the proposed algorithms in their original form is not recommended in a time-limited context. It is important to emphasize that all algorithms do not use any heuristic; therefore, all are made precise by employing the same input, and the results are obtained consistently. Consequently, a statistically significant analysis does not allow for the detection of differences in the proposed algorithms; in other words, we have the same

input, and there is no sensed or monitored data to apply descriptive or inference statistics. The following section examines the usefulness of conceptual distance in practical situations, how much precision is lost in the computation, and whether this is compensated for by the acceleration obtained.

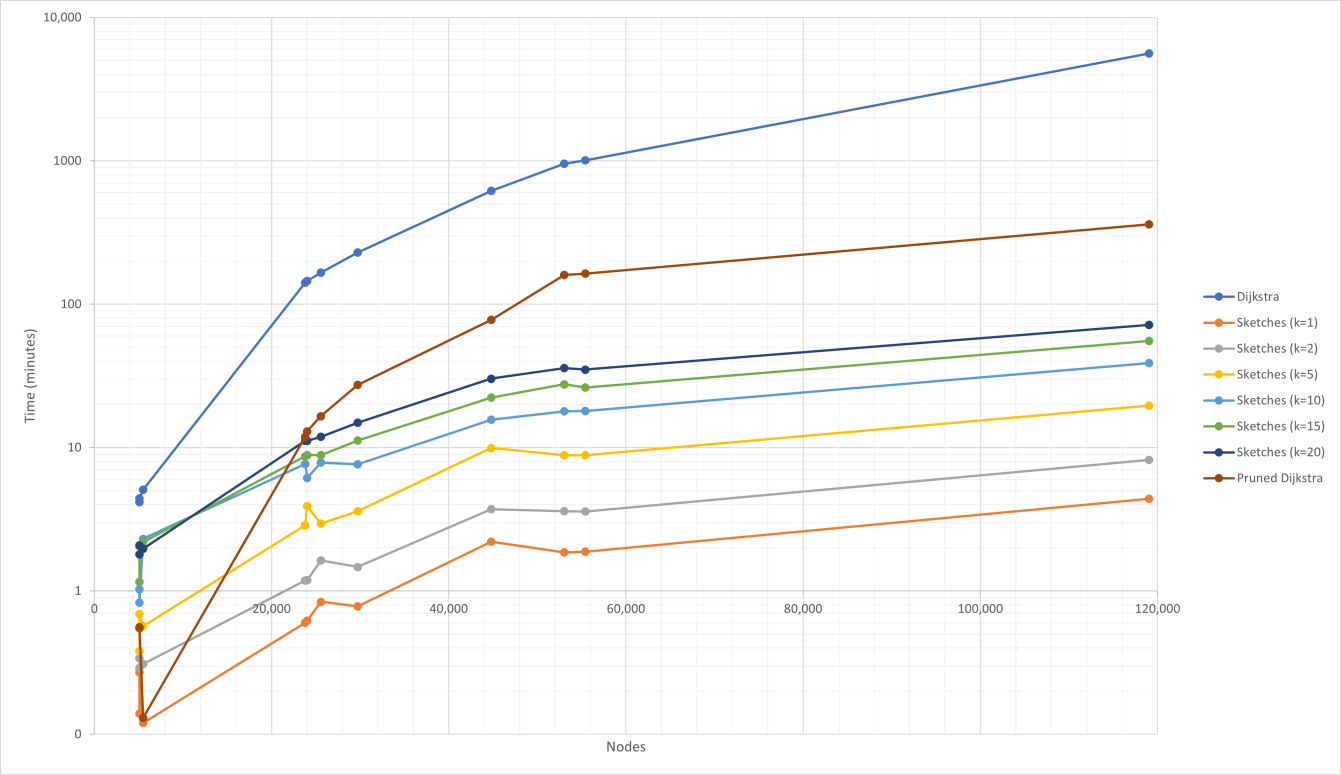


Figure 5. Simple DIS-C runtime for the following algorithms: Dijkstra, pruned Dijkstra simple, and sketch-based simple.

4.4. Conceptual Distance Results

In this section, the results of the conceptual distance obtained in the different datasets evaluated are analyzed, considering the previously proposed metrics. Correlation coefficients are frequently applied in knowledge representation systems; therefore, these coefficients are employed to compare the obtained results with other approaches. Before analyzing the group of datasets, we analyzed the results of a single dataset: MC30. The results of the pruned Dijkstra algorithm correspond directly to the results of the control algorithm because the process obtains the identical shortest distances. The sketch-based algorithm adequately adjusts to the conceptual control distance results; the accuracy of these results depends, essentially, on the value of k; at higher values, the difference from the original results is decreased. This behavior enables us to detect a high correlation with the control algorithm and, in general, with the other versions of the algorithms.

Figures 6 and 7 show the correlation matrix of Person and Spearman, respectively. The upper part of the diagonal represents the correlation between the words in the $a - b$ direction (conceptual distance from the word "a" to the word "b"). In contrast, the lower half of the diagonal is the correlation of $b - a$. Appendix C provides both the Pearson and Spearman correlation results.

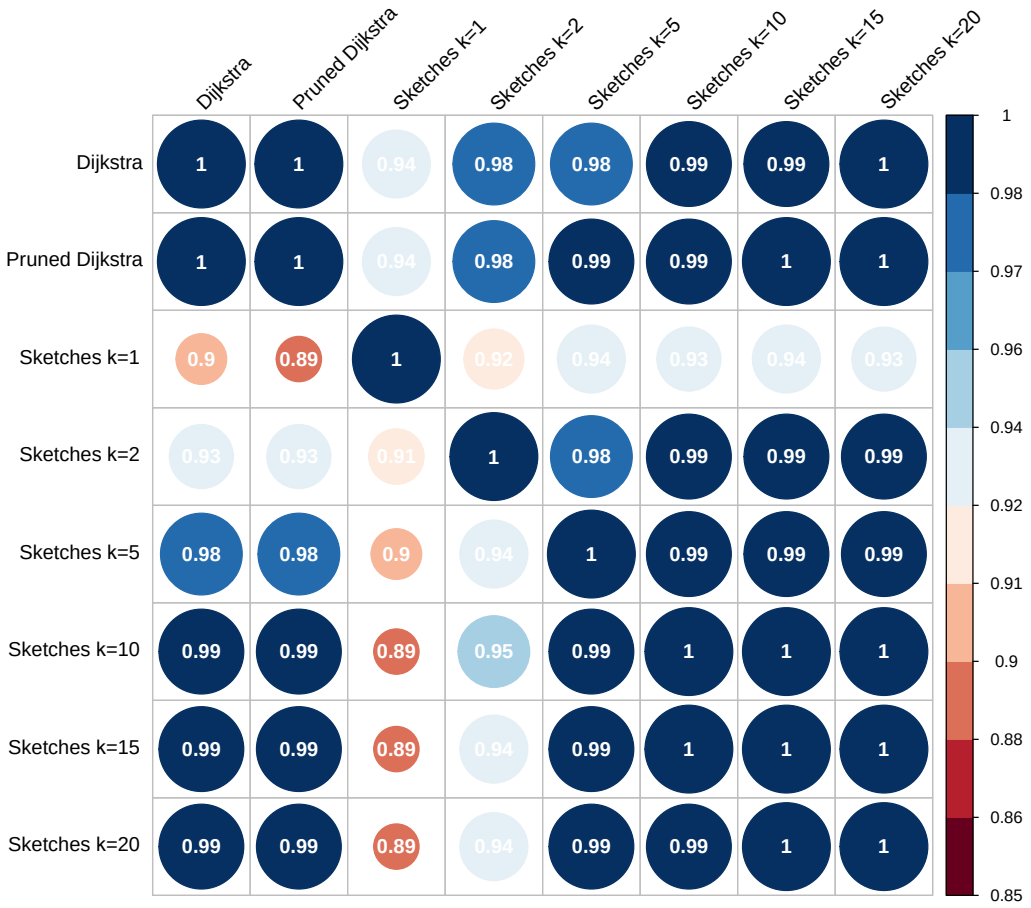


Figure 6. The Pearson correlation matrix for the MC30 dataset.

The matrices reveal a high correlation for all algorithms. Although the results are excellent in all cases, the correlation values related to the control algorithms are the most relevant for the analysis; these correspond to row 1 and column 1, with values of $a - b$ and $b - a$, respectively. By correlating the results of each algorithm with the control algorithm, an analysis of the error of each algorithm is implicitly performed. Therefore, as the correlation value is higher in the experiment, the error is decreased, and vice versa, with the advantage of using a metric limited to a specific interval.

In the first case, the lowest correlation value (0.94) corresponds to the sketch-based algorithm with $k = 1$; however, the correlation increases according to the value of k. The same behavior is replicated in the Spearman matrix and the results of the direction $b - a$ of the concepts. When considering the pruned Dijkstra algorithms, the correlation for the standard version is 1, and for the simple version, it obtained values close to 1 (0.99). These results indicate that the use of the algorithms in the standard or simple version is indifferent because, in both cases, the obtained correlation is very similar or, in some cases, identical.

All the algorithms produced a correlation that was considered high (greater than 0.7). For both types of correlations, the algorithm with the best performance was pruned Dijkstra. The algorithm that produced the lowest correlation was the sketch-based algorithm, with $k = 1$, and values ranging between 0.77 and 0.81; even if they cannot be considered bad results, they show disparity when compared to all the others. If the analysis is focused on the sketch-based algorithms, the pattern observed for the MC30 dataset can be verified, i.e., increasing the value of k also increases the probability of obtaining a better correlation.

Finally, the harmonic values are shown in Table 5; these values unify the performance obtained by both correlations. As mentioned in the previous sections, this table shows that the pruned Dijkstra algorithm achieves the highest correlation values. In contrast, the sketch-based algorithms perform better depending on the given value of its k parameter.

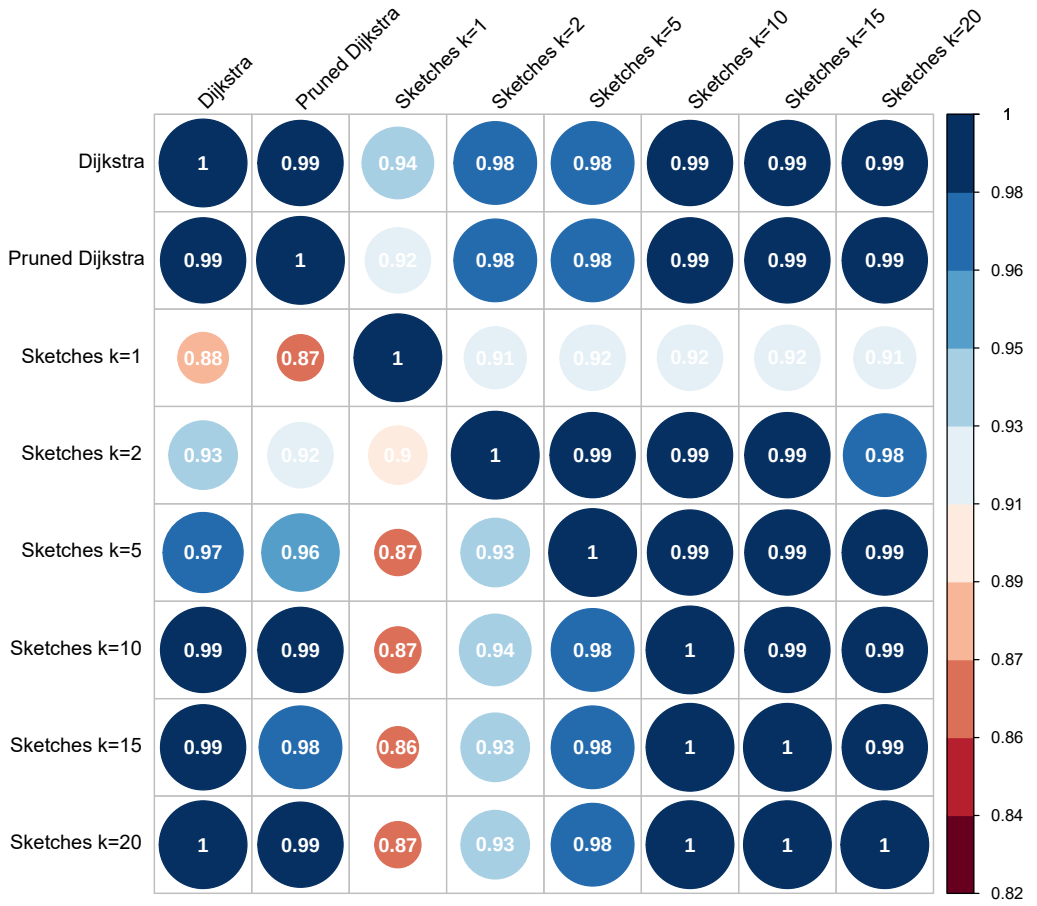


Figure 7. The Spearman correlation matrix for the MC30 dataset.

Table 5. Harmonic values of the correlations.

Algorithm	MC	RG	PS	Agirre	SimLex	MTurk771	MTurk287	WSRel	Rel	SCWS	All	Avg.
Dijkstra	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.0	1.00
Pruned Dijkstra	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.0	1.00
Sketches k = 1	0.94	0.86	0.86	0.72	0.75	0.61	0.85	0.69	0.82	0.82	0.0	0.79
Sketches k = 2	0.98	0.95	0.96	0.85	0.81	0.73	0.87	0.83	0.91	0.90	0.0	0.88
Sketches k = 5	0.98	0.98	0.98	0.92	0.89	0.88	0.93	0.91	0.95	0.95	0.0	0.94
Sketches k = 10	0.99	0.98	0.99	0.95	0.95	0.93	0.97	0.95	0.97	0.97	0.0	0.97
Sketches k = 15	0.99	0.99	0.99	0.97	0.97	0.96	0.97	0.97	0.98	0.98	0.0	0.98
Sketches k = 20	0.99	0.99	0.99	0.98	0.98	0.97	0.98	0.97	0.99	0.99	0.0	0.98

5. Conclusions and Future Work

This paper investigates the optimization of conceptual distance computation in an ontology using the DIS-C technique. When considering the case of computing conceptual distances by employing a method that solves the APSP problem, both the pruned Dijkstra and the sketch-based algorithms can substitute the Dijkstra algorithm. However, when analyzing the execution times obtained for the test datasets, the use of these two algorithms (pruned Dijkstra and sketch-based) is not recommended for situations where time constraints exist because the growth of the function (which estimates the execution time based on the number of nodes in the graph) is above that obtained by applying the standard Dijkstra algorithm.

Thus, to reduce the computational cost, a simple version of DIS-C was proposed by replacing the use of the APSP as a subprocess for propagating the conceptual distances between all pairs of concepts to compute generality, taking the vertex cover (sometimes referred to as node cover) into account. Based on this simple version, the experiments show that the processing time is up to 2.381 times faster than the original DIS-C approach. One of the main purposes of this paper was to maintain a reduced loss of precision in distance computation when compared to the original version of DIS-C, with correlation values of 1.0. The simple version can be useful in practical applications, and if the highest precision in distance calculation is required, this version using pruned Dijkstra is the best option. Moreover, using the complete topology of a graph to compute the conceptual distance in the DIS-C model is excessive and requires high computational cost. Therefore, we have experimentally demonstrated that the vertex cover is a robust approximation for computing the generality.

On the other hand, if speed is a priority, the simple version of the sketch-based algorithm with moderate values (greater than 1 but less than 20) for the parameter k is a good alternative. We must also consider the size of the conceptualization; for sizes of less than 1000 concepts, the type of algorithm is indifferent, and the use of Dijkstra is a better option since the computation time is not excessive and ensures the calculation, as proposed by the DIS-C model. An interesting way to take advantage of the speed of the sketch-based algorithm with low parameters (less than five) for k is when it is necessary to quickly perform tests on systems using DIS-C or directly when we want to compare the method with others. Implementing the sketch-based approach can provide a reference guide; for example, when good results are obtained with the simple version, it is reasonable to assume that the same results are obtained by executing the original DIS-C approach (when using the Dijkstra algorithm). The previous statement assumes that precision is the most crucial factor and that computation time is not a primary constraint. On the other hand, if time is a limitation, the pruned Dijkstra algorithm can be used as a replacement for Dijkstra, considering its high correlation.

Future work will be oriented toward evaluating the performance of these algorithms on other types of massive graphs, such as knowledge graphs, and analyzing the performance of the simple DIS-C approach with graphics unit processing (GPU) implementations. In addition, new advances in microprocessor architecture design, such as high-performance cores and high-efficiency cores, might be an exciting and suitable alternative to improve performance and computational time. Moreover, additional implementations to test this approach in other application domains involving information theory could be considered to evaluate the effectiveness of the proposed method.

Author Contributions: Conceptualization, R.Q. and E.M.; methodology, G.G. and M.T.-R.; software, E.M. and C.G.S.-M.; validation, R.Q., E.M., and G.G.; formal analysis, M.T.-R. and G.G.; investigation, R.Q. and C.G.S.-M.; resources, M.T.-R.; data curation, G.G.; writing—original draft preparation, R.Q. and C.G.S.-M.; writing—review and editing, M.T.-R. and G.G.; visualization, E.M.; supervision, R.Q.; project administration, C.G.S.-M.; funding acquisition, G.G. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially sponsored by the Instituto Politécnico Nacional under grants 20231372, 20230901, and 20230655, the Consejo Nacional de Humanidades, Ciencias y Tecnologías and Secretaría de Educación, Ciencia, Tecnología e Innovación de la Ciudad de México (SECTEI/182/2023).

Data Availability Statement: Data are contained within the article.

Acknowledgments: We are thankful to the reviewers for their time and invaluable and constructive feedback, which helped improve the quality of the paper.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. The Pruned Dijkstra

The pruned Dijkstra algorithm is a generalization of the PLL (pruned landmark labeling) method for undirected, weighted graphs [73]. The PLL method is based on the idea of two-hop coverage [86], which is an exact method. It is defined as follows: let $G = (V, E)$ be an undirected and weighted graph; for each vertex, $v \in V$, the cover $L(v)$ is a set of pairs, (u, δ_{uv}), where u is a vertex, and δ_{uv} is the distance from u to v in the graph, G. All coverage is denoted as indexed $L(v)_{v \in V}$. To compute the distance between the two vertices, s and t, we used the search function (Q), as shown in Equation (A1). L is a two-hop cover of G if $Q(s, t, L) = \delta_{st}$ for any $s, t \in V$.

$$Q(s,t,L) = \begin{cases} \infty & L(s) \cap L(t) = \emptyset \\ \min\{\delta_{vs} + \delta_{vt} | (v, \delta_{vs}) \in L(s), (v, \delta_{vt} \in L(t)\} & L(s) \cap L(t) \neq \emptyset \end{cases} \quad \text{(A1)}$$

In regard to weighted directed graphs, it is necessary to evaluate two indices; the first, L^{in}, contains the coverage of all the nodes, considering the incoming edges. The second, L^{out}, considers the outgoing edges. The procedure for consulting the distances has a simple modification but the idea remains the same. This change is shown in Equation (A2).

$$Q(s,t,L) = \begin{cases} \infty & L(s)^{in} \cap L(t)^{out} = \emptyset \\ \min\{\delta_{vs} + \delta_{vt} | (v, \delta_{vs}) \in L(s)^{in}, (v, \delta_{vt} \in L(t)^{out}\} & L(s)^{in} \cap L(t)^{out} \neq \emptyset \end{cases} \quad \text{(A2)}$$

A simple solution to build these indices is to employ two executions of the Dijkstra algorithm. Although this method needs to be more obvious and effective, we will explain the details when discussing the next method. Let $V = \{v_1, v_1, \ldots, v_n\}$; start with an empty index, L_0^{in}, where $\forall u \in V, L_0^{in}(u) = \emptyset$. Suppose Dijkstra is applied to each vertex in the order $v_1, v_2, \ldots, v_n$. After the k-th Dijkstra on a vertex v_k, the distances of v_k are added to the coverages of the vertices reached, where $L_k^{in}(u) = L_{k-1}^{in}(u) \cup \{(v_k, d_G(v_k, u))\}$ for each $u \in V$ with $d_G(v_k, u) \neq \infty$, where $d_G(u, v)$ is the distance from vertex u to vertex v within the graph, G. It is important to note that the coverages of the vertices reached are not changed, which implies that $L_k^{in}(u) = L_{k-1}^{in}(u)$ for each $u \in V$ with $d_G(v_k, u) = \infty$. Performing this procedure, starting from L_0^{out}, gives the same result but takes into account the output edges.

Therefore, L_n^{in} and L_n^{out} are the final indices contained in L_n. It follows that $Q(s, t, L) = d_G(s, t)$ for any vertex pair, so L_n^{in} and L_n^{out} are the two-hop covers of G. If s and t are reachable, then $(s, 0) \in L_n^{in}(s)$ and $(s, d_G(s, t)) \in L_n^{out}(t)$. Creating indices in this form might be more efficient. For this reason, *pruning* is added to the original method. This technique reduces the search space considerably. In the classic method, a pruned search is performed using the Dijkstra algorithm, processing the vertices in the following order: $v_1, v_2, \ldots, v_n$. Start with a pair of indices, L_0^{in} and L_0^{out}, and create indices L_k^{in}, L_k^{out} from L_{k-1}^{in} and L_{k-1}^{out}, respectively, using the information obtained by the k-th execution of pruned Dijkstra for vertex v_k.

The pruning process is as follows: Suppose we have the indices L_{k-1}^{out} and L_{k-1}^{in} and we execute the pruned Dijkstra from v_k to create a new index L_k^{out}. We assume that a vertex, u, with a distance of δ is visited. If $Q(s, t, L_{k-1}) \leq \delta$, then we prune u, i.e., not adding both (u_k, δ) to $L_k^{out}(u)$ and any neighbors of u to the priority queue. If the condition is not satisfied, then (u_k, δ) is added to the cover of u, and Dijkstra is performed as usual. Similar to the original method, for all vertices that were not reached in the k-th execution of the pruned Dijkstra, set $L_k^{in}(u) = L_{k-1}^{in}(u)$ and $L_k^{out}(u) = L_{k-1}^{out}(u)$. Finally, L_k^{out} and L_k^{in} are the terminal indices contained in $L_k^{'}$.

Figure A1 shows an example of the algorithm applied to the graph depicted in the figure, assuming that the order of execution is $1, 8, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12$. The first pruned Dijkstra from vertex 1 visits all other vertices forwards (Figure A2a) and backwards (Figure A2b). In the next iteration from vertex 8 (Figure A2c), when vertex 1 is visited, given that $Q(8, 1, L_1^{'}) = 1 = \delta(8, 1) = 1$, vertex 1 is pruned and neither of its neighbors is visited again; the same occurs in the opposite direction (see Figure A2d). With these two executions, it is possible to cover all the shortest paths, and after executing the process on the remaining vertices, they only refer to themselves.

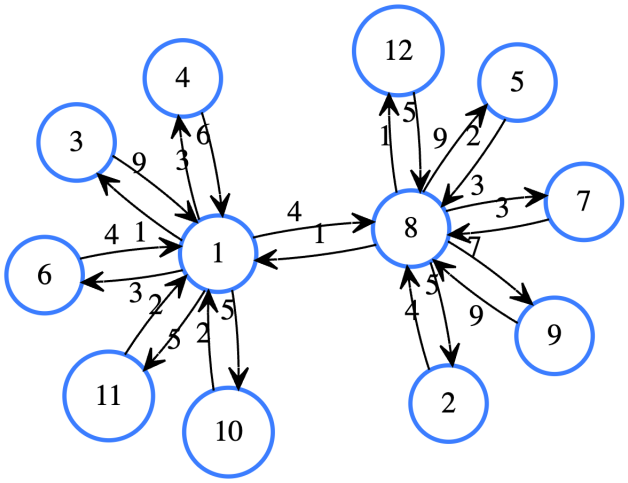


Figure A1. Example of a graph for the pruned Dijkstra algorithm.

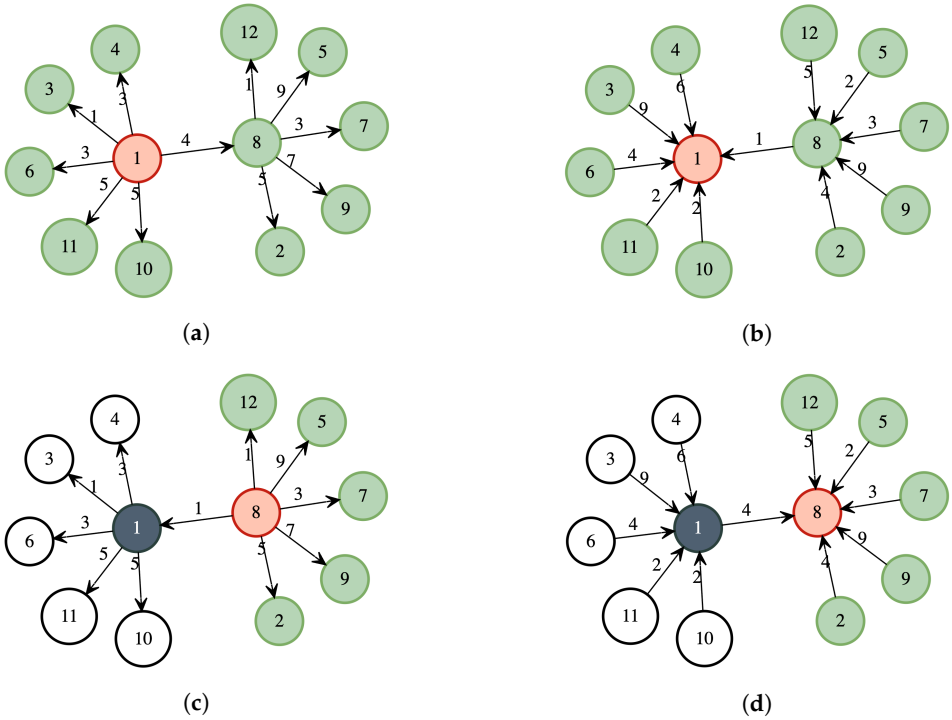


Figure A2. Example of a pruned Dijkstra. The red vertices denote the execution root, the green vertices represent those visited and added to the coverage, and the gray vertices are the roots. (**a**) The first pruned Dijkstra from vertex 1 with trailing edges. All vertices are visited. (**b**) The first pruned Dijkstra from vertex 1 with entry edges. All vertices are visited. (**c**) The second pruned Dijkstra from vertex 8 with exit edges. Only those adjacent to 8 are visited. (**d**) The second pruned Dijkstra from vertex 8 with entry edges. Only those adjacent to 8 are visited.

Appendix B. The Sketch-Based Algorithms

Sketch-based algorithms are methods that include two general processes: offline and online. In the offline process, a set of sketches represents the crucial nodes, defined randomly. By considering these sketches, the shortest path to all other vertices is determined. In the online process, the shortest distance between a pair of nodes, $u, v \in V$, is calculated from the closest common sketch of them.

The offline process consists of sampling $\log(n)$ sets of the seed vertex, each containing a subset, $S \subseteq V$, of size $2^i, i = 1, 2, 3, \ldots, \log(n)$. For each set, the closest seed, $s \in S$, of each vertex is stored, including its distance; i.e., for every node, u, a sketch of the form $\texttt{SKETCH}(u) = \{(s, \delta_{s,u})\}$ is created. Therefore, the final size of the sketch for each node is $\log(n)$. This technique can be performed using the Dijkstra algorithm. The vertex, $s \in S$, closest to u is denoted as the seed of u. This procedure is repeated k times with different sample sizes. At the end of the k iterations, each node has $k \log(n)$ seeds.

Figure A3 shows an example of the implemented procedure. The graph in Figure A1 is used as a reference. Since there are 12 vertices, the number of seed sets is 3 ($\log_2(12) = 3$), and the size of the sets is 1, 2, and 4, respectively. The vertices in each set are 1 for the first, 3.1 for the second, and 8, 1, 2, 7 for the last.

In the first iteration (Figure A3a), all the vertices have vertex 1 in their coverage; this also happens for the input edges (see Figure A3b). In the second iteration (Figure A3c,d), the coverages do not change since vertex 3 does not cover any path shorter than those already covered by vertex 1. Finally, in the last iteration (Figure A3e,f), we have vertex 8 in the seed set; this covers the shortest path of its neighbors, except those that are also in the set; therefore, vertices 5, 12, and 9 will have 8 in their coverage. On the other hand, vertices 2 and 7 are included in their respective coverages.

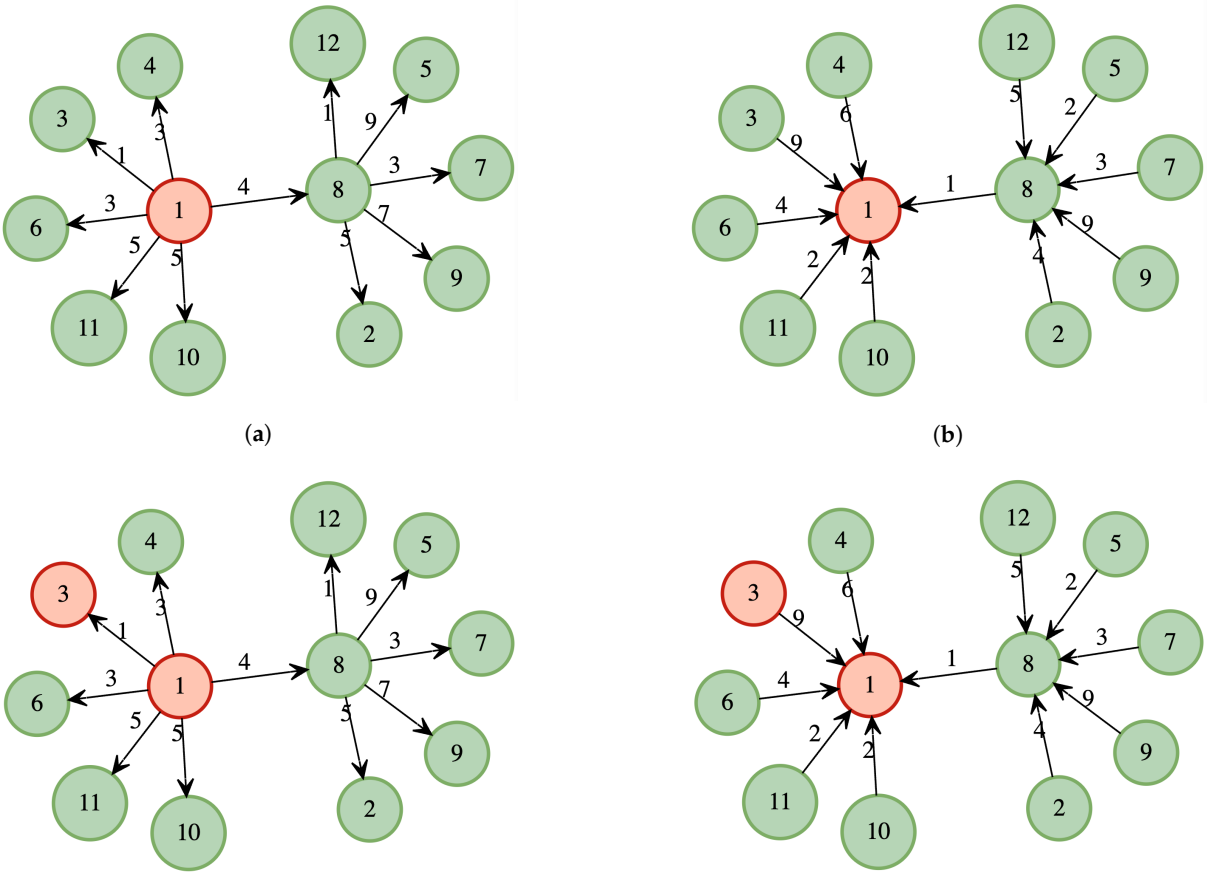


Figure A3. *Cont.*

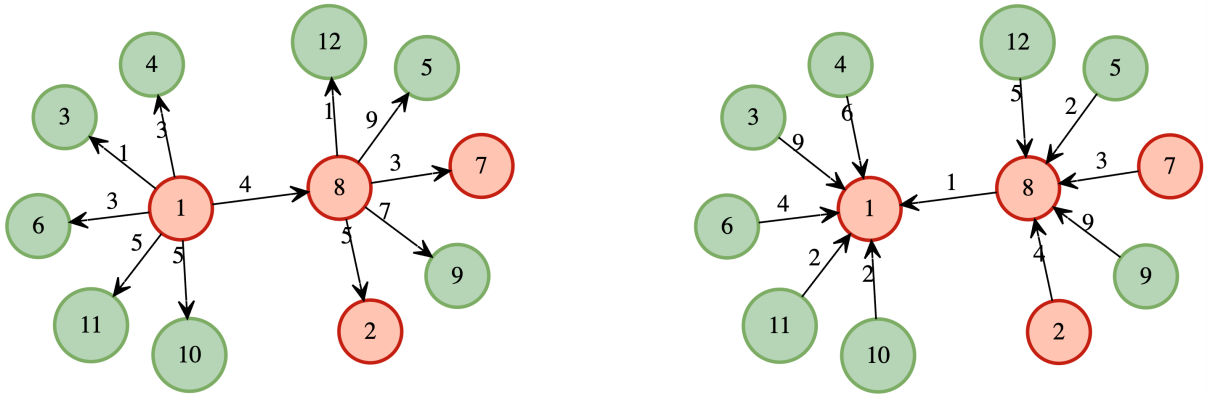


Figure A3. Sketch-based algorithm example. The red vertices denote the execution root, and the green ones represent those that are visited. (**a**) Sketches from set 1 with output edges. All vertices have node 1 in their coverage. (**b**) Sketches from set 1 with input edges. All vertices have node 1 in their coverage. (**c**) Sketches from set 2 with output edges. Coverages do not change. (**d**) Sketches from set 2 with input edges. Coverages do not change. (**e**) Sketches from set 3 with output edges. 8 is added to the coverage of vertices 5, 9, and 12. (**f**) Sketches from set 3 with input edges; 8 is added to the coverage of vertices 5, 9, and 12.

Appendix C. Pearson and Spearman Correlation Results

According to the described experiments, Tables A1 and A2 describe the results of the Pearson correlation obtained by applying all the algorithms and their configurations to the 11 datasets. In addition, Tables A3 and A4 show the values of the Spearman correlation.

Table A1. The Pearson correlation of all datasets in the direction $a - b$.

Algorithm	MC	RG	PS	Agirre	SimLex	MTurk771	MTurk287	WSRel	Rel	SCWS	All	Avg.
Dijkstra	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Pruned Dijkstra	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.997	1.000
Sketches k = 1	0.940	0.853	0.862	0.733	0.762	0.605	0.864	0.720	0.841	0.815	0.745	0.795
Sketches k = 2	0.977	0.947	0.961	0.870	0.813	0.731	0.888	0.842	0.922	0.898	0.802	0.877
Sketches k = 5	0.985	0.984	0.978	0.923	0.897	0.881	0.937	0.921	0.953	0.944	0.899	0.937
Sketches k = 10	0.991	0.979	0.995	0.961	0.955	0.931	0.967	0.959	0.976	0.971	0.942	0.966
Sketches k = 15	0.993	0.996	0.991	0.980	0.972	0.959	0.971	0.971	0.983	0.982	0.957	0.978
Sketches k = 20	0.996	0.992	0.994	0.983	0.974	0.971	0.979	0.975	0.989	0.986	0.972	0.983

Table A2. The Pearson correlation of all datasets in the direction $b - a$.

Algorithm	MC	RG	PS	Agirre	SimLex	MTurk771	MTurk287	WSRel	Rel	SCWS	All	Avg.
Dijkstra	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Sketches k = 1	0.897	0.777	0.877	0.734	0.722	0.562	0.862	0.757	0.857	0.793	0.733	0.779
Sketches k = 2	0.932	0.961	0.956	0.893	0.806	0.725	0.890	0.827	0.879	0.896	0.795	0.869
Sketches k = 5	0.979	0.984	0.981	0.909	0.893	0.865	0.946	0.936	0.947	0.945	0.897	0.935
Sketches k = 10	0.987	0.991	0.992	0.963	0.952	0.944	0.960	0.945	0.979	0.967	0.939	0.965
Sketches k = 15	0.992	0.998	0.994	0.968	0.971	0.958	0.975	0.966	0.978	0.980	0.960	0.976
Sketches k = 20	0.995	0.996	0.997	0.979	0.971	0.972	0.980	0.983	0.984	0.988	0.972	0.983

Table A3. The Spearman correlation of all datasets in the direction $a - b$.

Algorithm	MC	RG	PS	Agirre	SimLex	MTurk771	MTurk287	WSRel	Rel	SCWS	All	Avg.
Dijkstra	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Pruned Dijkstra	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.968	0.997
Sketches k = 1	0.938	0.876	0.858	0.702	0.742	0.605	0.843	0.671	0.799	0.831	0.736	0.782
Sketches k = 2	0.977	0.944	0.959	0.823	0.804	0.729	0.855	0.821	0.900	0.906	0.807	0.866
Sketches k = 5	0.978	0.978	0.984	0.911	0.887	0.876	0.925	0.907	0.950	0.954	0.897	0.931
Sketches k = 10	0.987	0.981	0.991	0.941	0.955	0.931	0.963	0.946	0.973	0.978	0.942	0.962
Sketches k = 15	0.987	0.993	0.985	0.968	0.971	0.969	0.963	0.960	0.980	0.986	0.954	0.974
Sketches k = 20	0.989	0.988	0.991	0.973	0.979	0.978	0.976	0.961	0.985	0.990	0.965	0.980

Table A4. The Spearman correlation of all datasets in the direction $b - a$.

Algorithm	MC	RG	PS	Agirre	SimLex	MTurk771	MTurk287	WSRel	Rel	SCWS	All	Avg.
Dijkstra	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Pruned Dijkstra	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.969	0.997
Sketches k = 1	0.875	0.826	0.843	0.735	0.700	0.573	0.823	0.693	0.817	0.807	0.732	0.766
Sketches k = 2	0.925	0.921	0.932	0.859	0.816	0.733	0.876	0.764	0.835	0.902	0.803	0.851
Sketches k = 5	0.972	0.970	0.961	0.898	0.884	0.866	0.920	0.906	0.936	0.953	0.896	0.924
Sketches k = 10	0.991	0.980	0.985	0.953	0.950	0.947	0.945	0.928	0.969	0.974	0.938	0.960
Sketches k = 15	0.991	0.990	0.981	0.959	0.974	0.960	0.961	0.952	0.969	0.984	0.958	0.971
Sketches k = 20	0.996	0.985	0.990	0.973	0.971	0.977	0.971	0.975	0.982	0.991	0.966	0.980

References

1. Mejia Sanchez-Bermejo, A. Similitud Semantica Entre Conceptos de Wikipedia. Bachelor's Thesis, Universidad Carlos III de Madrid, Getafe, Madrid, Spain, 2013.
2. Goldstone, R.L. Similarity, interactive activation, and mapping. *J. Exp. Psychol. Learn. Mem. Cogn.* **1994**, *20*, 3. [CrossRef]
3. Quintero, R.; Torres-Ruiz, M.; Saldaña-Pérez, M.; Guzmán Sánchez-Mejorada, C.; Mata-Rivera, F. A Conceptual Graph-Based Method to Compute Information Content. *Mathematics* **2023**, *11*, 3972. [CrossRef]
4. Chen, X.; Wang, Z.; Hua, Q.; Shang, W.L.; Luo, Q.; Yu, K. AI-empowered speed extraction via port-like videos for vehicular trajectory analysis. *IEEE Trans. Intell. Transp. Syst.* **2022**, *24*, 4541–4552. [CrossRef]
5. Quintero, R.; Torres-Ruiz, M.; Menchaca-Méndez, R.; Moreno-Armendariz, M.A.; Guzmán, G.; Moreno-Ibarra, M. DIS-C: Conceptual distance in ontologies, a graph-based approach. *Knowl. Inf. Syst.* **2019**, *59*, 33–65. [CrossRef]
6. Dreyfus, S.E. An appraisal of some shortest-path algorithms. *Oper. Res.* **1969**, *17*, 395–412. [CrossRef]
7. Gallo, G.; Pallottino, S. Shortest path algorithms. *Ann. Oper. Res.* **1988**, *13*, 1–79. [CrossRef]
8. Magzhan, K.; Jani, H.M. A review and evaluations of shortest path algorithms. *Int. J. Sci. Technol. Res* **2013**, *2*, 99–104.
9. Madkour, A.; Aref, W.G.; Rehman, F.U.; Rahman, M.A.; Basalamah, S. A survey of shortest-path algorithms. *arXiv* **2017**, arXiv:1705.02044.
10. Zhang, F.; Liu, J. A new shortest path algorithm for massive spatial data based on Dijkstra algorithm. *J. LiaoNing Technol. Univ. Sci. Ed.* **2009**, *28*, 554–557.
11. Chakaravarthy, V.T.; Checconi, F.; Murali, P.; Petrini, F.; Sabharwal, Y. Scalable single source shortest path algorithms for massively parallel systems. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *28*, 2031–2045. [CrossRef]
12. Yang, Y.; Li, Z.; Wang, X.; Hu, Q. Finding the shortest path with vertex constraint over large graphs. *Complexity* **2019**, *2019*, 8728245. [CrossRef]
13. Liu, J.; Pan, Y.; Hu, Q.; Li, A. Navigating a Shortest Path with High Probability in Massive Complex Networks. In Proceedings of the Analysis of Experimental Algorithms: Special Event, SEA^2 2019, Kalamata, Greece, 24–29 June 2019; Revised Selected Papers; Springer: Berlin/Heidelberg, Germany, 2019; pp. 82–97.
14. Ma, X.; Huo, E.; Yu, H.; Li, H. Mining truck platooning patterns through massive trajectory data. *Knowl. Based Syst.* **2021**, *221*, 106972. [CrossRef]
15. Li, S.; Sun, X.; Xiao, Y.H.; Guo, X.; Zhang, J. Noncoherent space-time coding for correlated massive MIMO channel with Riemannian distance. *Digit. Signal Process.* **2023**, *133*, 103876. [CrossRef]

16. Meersman, R.A. Semantic ontology tools in IS design. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 1999; pp. 30–45. [CrossRef]
17. Bondy, J.A. *Graph Theory with Applications*; Elsevier Science Publishing Co., Inc.: New York, NY, USA, 1982.
18. West, D.B. *Introduction to Graph Theory*, 2nd ed.; Prentice Hall Inc.: Upper Saddle River, NJ, USA, 2001.
19. Bollobás, B. *Modern Fraph Theory*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 1998; Volume 184.
20. Gross, J.L.; Yellen, J.; Anderson, M. *Graph Theory and Its Applications*; Chapman and Hall/CRC: London, UK, 2018.
21. Juel Vang, K. Ethics of Google's Knowledge Graph: Some considerations. *J. Inf. Commun. Ethics Soc.* **2013**, *11*, 245–260. [CrossRef]
22. Ehrlinger, L.; Wöß, W. Towards a definition of knowledge graphs. *SEMANTiCS (Posters, Demos, SuCCESS)* **2016**, *48*, 2.
23. Fensel, D.; Şimşek, U.; Angele, K.; Huaman, E.; Kärle, E.; Panasiuk, O.; Toma, I.; Umbrich, J.; Wahler, A.; Fensel, D.; et al. Introduction: What is a knowledge graph? In *Knowledge Graphs: Methodology, Tools and Selected Use Cases*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 1–10.
24. Zou, X. A survey on application of knowledge graph. In *Proceedings of the Journal of Physics: Conference Series*; IOP Publishing: Bristol, UK, 2020; Volume 1487, p. 012016.
25. Pujara, J.; Miao, H.; Getoor, L.; Cohen, W. Knowledge graph identification. In Proceedings of the Semantic Web—ISWC 2013: 12th International Semantic Web Conference, Sydney, NSW, Australia, 21–25 October 2013; Proceedings, Part I 12; Springer: Berlin/Heidelberg, Germany, 2013; pp. 542–557.
26. Sanchez, D.; Batet, M.; Isern, D.; Valls, A. Ontology-based semantic similarity: A new feature-based approach. *Expert Syst. Appl.* **2012**, *39*, 7718–7728. [CrossRef]
27. Rada, R.; Mili, H.; Bicknell, E.; Blettner, M. Development and application of a metric on semantic nets. *IEEE Trans. Syst. Man Cybern.* **1989**, *19*, 17–30. [CrossRef]
28. Wu, Z.; Palmer, M. Verb semantics and lexical selection. *arXiv* **1994**, arXiv:cmp-lg/9406033.
29. Hirst, G.; Stonge, D. Lexical Chains as Representations of Context for the Detection and Correction of Malapropisms. *Wordnet Electron. Lex. Database* **1995**, *305*, 305–332.
30. Li, Y.; Bandar, Z.A.; Mclean, D. An approach for measuring semantic similarity between words using multiple information sources. *IEEE Trans. Knowl. Data Eng.* **2003**, *15*, 871–882. [CrossRef]
31. Shenoy, M.K.; Shet, K.; Acharya, U.D. A new similarity measure for taxonomy based on edge counting. *Int. J. Web Semant. Technol.* **2012**, *3*, 23. [CrossRef]
32. Tversky, A. Features of similarity. *Psychol. Rev.* **1977**, *84*, 327. [CrossRef]
33. Lesk, M. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In Proceedings of the 5th Annual International Conference on Systems Documentation, Toronto, ON, Canada, 8–11 June 1986; pp. 24–26.
34. Banerjee, S.; Pedersen, T. Extended gloss overlaps as a measure of semantic relatedness. In Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI, Acapulco, Mexico, 9–15 August 2003; Volume 3, pp. 805–810.
35. Jiang, Y.; Zhang, X.; Tang, Y.; Nie, R. Feature-based approaches to semantic similarity assessment of concepts using Wikipedia. *Inf. Process. Manag.* **2015**, *51*, 215–234. [CrossRef]
36. Resnik, P. Using Information Content to Evaluate Semantic Similarity in a Taxonomy. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI, Montreal, QC, Canada, 20–25 August 1995; Volume 1, pp. 449–453.
37. Jiang, J.J.; Conrath, D.W. Semantic similarity based on corpus statistics and lexical taxonomy. In Proceedings of the 10th Research on Computational Linguistics International Conference, ROCLING X, Taipei, Taiwan, 25–27 August 1997; pp. 19–33.
38. Gao, J.B.; Zhang, B.W.; Chen, X.H. A WordNet-based semantic similarity measurement combining edge-counting and information content theory. *Eng. Appl. Artif. Intell.* **2015**, *39*, 80–88. [CrossRef]
39. Jiang, Y.; Bai, W.; Zhang, X.; Hu, J. Wikipedia-based information content and semantic similarity computation. *Inf. Process. Manag.* **2017**, *53*, 248–265. [CrossRef]
40. Zhou, Z.; Wang, Y.; Gu, J. A New Model of Information Content for Semantic Similarity in WordNet. In Proceedings of the 2008 Second International Conference on Future Generation Communication and Networking Symposia, Sanya, China, 13–15 December 2008; Volume 3, pp. 85–89. [CrossRef]
41. Sanchez, D.; Batet, M.; Isern, D. Ontology-based information content computation. *Knowl. Based Syst.* **2011**, *24*, 297–303. [CrossRef]
42. Seidel, R. On the All-Pairs-Shortest-Path Problem. In Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing, STOC'92, New York, NY, USA, 4–6 May 1992; pp. 745–749. [CrossRef]
43. Warshall, S. A Theorem on Boolean Matrices. *J. ACM* **1962**, *9*, 11–12. [CrossRef]
44. Singh, P.; Kumar, R.; Pandey, V. An Efficient Algorithm for All Pair Shortest Paths. *Int. J. Comput. Electr. Eng.* **2010**, *2*, 984–991. [CrossRef]
45. Zwick, U. All Pairs Shortest Paths Using Bridging Sets and Rectangular Matrix Multiplication. *J. ACM* **2002**, *49*, 289–317. [CrossRef]
46. D'alberto, P.; Nicolau, A. R-Kleene: A high-performance divide-and-conquer algorithm for the all-pair shortest path for densely connected networks. *Algorithmica* **2007**, *47*, 203–213. [CrossRef]
47. Islam, M.T.; Thulasiraman, P.; Thulasiram, R.K. A parallel ant colony optimization algorithm for all-pair routing in MANETs. In Proceedings of the International Parallel and Distributed Processing Symposium, Nice, France, 22–26 May 2003.

48. Katz, G.J.; Kider, J.T. All-Pairs Shortest-Paths for Large Graphs on the GPU. In Proceedings of the EUROGRAPHICS/ACM SIGGRAPH Conference on Graphics Hardware 2008, Sarajevo, Bosnia and Herzegovina, 20–21 June 2008; pp. 47–55.
49. Reddy, K.R. A survey of the all-pairs shortest paths problem and its variants in graphs. *Acta Univ. Sapientiae Inform.* **2016**, *8*, 16–40. [CrossRef]
50. Aho, A.V.; Hopcroft, J.E. *The Design and Analysis of Computer Algorithms*; Pearson Education India: Chennai, India, 1974.
51. Attiratanasunthron, N.; Fakcharoenphol, J. A running time analysis of an ant colony optimization algorithm for shortest paths in directed acyclic graphs. *Inf. Process. Lett.* **2008**, *105*, 88–92. [CrossRef]
52. Neumann, F.; Witt, C. Runtime analysis of a simple ant colony optimization algorithm. In Proceedings of the International Symposium on Algorithms and Computation, Kolkata, India, 18–20 December 2006; Springer: Berlin/Heidelberg, Germany, 2006; pp. 618–627.
53. Di Caro, G.; Dorigo, M. AntNet: Distributed stigmergetic control for communications networks. *J. Artif. Intell. Res.* **1998**, *9*, 317–365. [CrossRef]
54. Horoba, C.; Sudholt, D. Running time analysis of ACO systems for shortest path problems. In Proceedings of the International Workshop on Engineering Stochastic Local Search Algorithms, Brussels, Belgium, 3–4 September 2009; Springer: Berlin/Heidelberg, Germany, 2009; pp. 76–91.
55. Fredman, M.L. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.* **1976**, *5*, 83–89. [CrossRef]
56. Takaoka, T. A new upper bound on the complexity of the all pairs shortest path problem. *Inf. Process. Lett.* **1992**, *43*, 195–199. [CrossRef]
57. Dobosiewicz, W. A more efficient algorithm for the min-plus multiplication. *Int. J. Comput. Math.* **1990**, *32*, 49–60. [CrossRef]
58. Han, Y. Improved algorithm for all pairs shortest paths. *Inf. Process. Lett.* **2004**, *91*, 245–250. [CrossRef]
59. Takaoka, T. A faster algorithm for the all-pairs shortest path problem and its application. In Proceedings of the International Computing and Combinatorics Conference, Jeju Island, Republic of Korea, 17–20 August 2004; Springer: Berlin/Heidelberg, Germany, 2004; pp. 278–289.
60. Takaoka, T. An O (n3loglogn/logn) time algorithm for the all-pairs shortest path problem. *Inf. Process. Lett.* **2005**, *96*, 155–161. [CrossRef]
61. Zwick, U. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. In Proceedings of the International Symposium on Algorithms and Computation, Hong Kong, China, 20–22 December 2004; Springer: Berlin/Heidelberg, Germany, 2004; pp. 921–932.
62. Chan, T.M. All-pairs shortest paths with real weights in O (n 3/log n) time. *Algorithmica* **2008**, *50*, 236–243. [CrossRef]
63. Han, Y. An o (n 3 (loglogn/logn) 5/4) time algorithm for all pairs shortest paths. In Proceedings of the European Symposium on Algorithms, Zurich, Switzerland, 11–13 September 2006; Springer: Berlin/Heidelberg, Germany, 2006; pp. 411–417.
64. Chan, T.M. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.* **2010**, *39*, 2075–2089. [CrossRef]
65. Williams, R.R. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.* **2018**, *47*, 1965–1985. [CrossRef]
66. Chou, Y.L.; Romeijn, H.E.; Smith, R.L. Approximating shortest paths in large-scale networks with an application to intelligent transportation systems. *INFORMS J. Comput.* **1998**, *10*, 163–179. [CrossRef]
67. Mohring, R.H.; Schilling, H.; Schutz, B.; Wagner, D.; Willhalm, T. Partitioning graphs to speedup Dijkstra's algorithm. *J. Exp. Algorithmics* **2007**, *11*, 2–8. [CrossRef]
68. Baswana, S.; Goyal, V.; Sen, S. All-pairs nearly 2-approximate shortest paths in O(n2polylogn) time. *Theor. Comput. Sci.* **2009**, *410*, 84–93. [CrossRef]
69. Yuster, R. Approximate shortest paths in weighted graphs. *J. Comput. Syst. Sci.* **2012**, *78*, 632–637. [CrossRef]
70. Thorup, M.; Zwick, U. Approximate distance oracles. *J. ACM* **2005**, *52*, 1–24. [CrossRef]
71. Das Sarma, A.; Gollapudi, S.; Najork, M.; Panigrahy, R. A sketch-based distance oracle for web-scale graphs. In Proceedings of the Third ACM International Conference on Web Search and Data Mining, New York, NY, USA, 3–6 February 2010; pp. 401–410.
72. Wang, Y.; Wang, Q.; Koehler, H.; Lin, Y. Query-by-Sketch: Scaling Shortest Path Graph Queries on Very Large Networks. In Proceedings of the 2021 International Conference on Management of Data, Virtual Event, 20–25 June 2021; pp. 1946–1958.
73. Akiba, T.; Iwata, Y.; Yoshida, Y. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, 22–27 June 2013; pp. 349–360.
74. Mendiola, E. Algoritmo Para el Cálculo Acelerado de Distancias Conceptuales. Master's Thesis, Instituto Politécnico Nacional, Mexico City, Mexico, 2022.
75. Robertson, N.; Seymour, P. Graph minors. III. Planar tree-width. *J. Comb. Theory Ser. B* **1984**, *36*, 49–64. [CrossRef]
76. Miller, G.A.; Charles, W.G. Contextual correlates of semantic similarity. *Lang. Cogn. Process.* **1991**, *6*, 1–28. [CrossRef]
77. Rubenstein, H.; Goodenough, J.B. Contextual correlates of synonymy. *Commun. ACM* **1965**, *8*, 627–633. [CrossRef]
78. Pirro, G. A semantic similarity metric combining features and intrinsic information content. *Data Knowl. Eng.* **2009**, *68*, 1289–1308. [CrossRef]
79. Agirre, E.; Alfonseca, E.; Hall, K.; Kravalova, J.; Pasca, M.; Soroa, A. A Study on Similarity and Relatedness Using Distributional and WordNet-based Approaches. In Proceedings of the Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Boulder, Colorado, 31 May–5 June 2009; pp. 19–27. Available online: https://aclanthology.org/N09-1003 (accessed on 9 October 2023).

80. Hill, F.; Reichart, R.; Korhonen, A. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Comput. Linguist.* **2015**, *41*, 665–695. [CrossRef]
81. Halawi, G.; Dror, G.; Gabrilovich, E.; Koren, Y. Large-scale learning of word relatedness with constraints. In Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Beijing, China, 12–16 August 2012; pp. 1406–1414.
82. Radinsky, K.; Agichtein, E.; Gabrilovich, E.; Markovitch, S. A word at a time: Computing word relatedness using temporal semantic analysis. In Proceedings of the 20th International Conference on World Wide Web, Hyderabad, India, 28 March–1 April 2011; pp. 337–346.
83. Finkelstein, L.; Gabrilovich, E.; Matias, Y.; Rivlin, E.; Solan, Z.; Wolfman, G.; Ruppin, E. Placing search in context: The concept revisited. In Proceedings of the 10th International Conference on World Wide Web, Hong Kong, China, 1–5 May 2001; pp. 406–414.
84. Szumlanski, S.; Gomez, F.; Sims, V.K. A new set of norms for semantic relatedness measures. In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, Sofia, Bulgaria, 4–9 August 2013; Volume 2: Short Papers, pp. 890–895.
85. Huang, E.H.; Socher, R.; Manning, C.D.; Ng, A.Y. Improving word representations via global context and multiple word prototypes. In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, Jeju Island, Republic of Korea, 8–14 July 2012; Volume 1: Long Papers.
86. Cohen, E.; Halperin, E.; Kaplan, H.; Zwick, U. Reachability and distance queries via 2-hop labels. *SIAM J. Comput.* **2003**, *32*, 1338–1355. [CrossRef]