Paralelización de un experimento para determinar la escalonabilidad de grafos bipartitos usando Apache Spark

Parallelizing an Experiment to Decide Shellability on Bipartite Graphs Using Apache Spark

Paralelização de um experimento para determinar a shellabilidade de grafos bipartidos usando Apache Spark

Julián D. Arango-Holguín[1]; Milena Cárdenas-Álzate[2]; Andrés D. Santamaría-Galvis[3,11]

[1] Ing Sist;Desarrollo Institucional;
[2] Ing Sist,Departamento Ingeniería de Sistemas, Facultad de Ingeniería, Universidad de Antioquia, Medellín, Colombia,
[3] Ing Sist, Msc, Departamento Ingeniería de Sistemas, Facultad de Ingeniería, Universidad de Antioquia, Medellín
Email: david.galvis@udea.edu.co

Recibido: febrero 23 de 2017 **Aceptado**: abril 14 de 2017

Resumen

La escalonabilidad* de grafos es un problema en NP del que se desconoce su inclusión en las clases de complejidad P o NP-completa. Con el fin de comprender su comportamiento computacional en el caso particular de los grafos bipartitos, podría ser de utilidad disponer de un método eficiente para generar y analizar instancias escalonables. La literatura reporta un experimento secuencial, y de costo exponencial, diseñado para determinar la escalonabilidad de un conjunto de instancias. En el presente trabajo, y con el fin de mejorar el desempeño experimento mencionado, proponemos tres alternativas utilizando Apache Spark: una multinúcleo, otra multinodo y otra completamente paralela. Además, comparamos el tiempo de ejecución de cada una de ellas respecto a la versión original en grafos bipartitos aleatorios con 10,12,15,20 y 50 vértices, y obtuvimos aceleraciones (speedups) entre 1.37 y 1.67 para la versión multinúcleo, entre 2.34 y 3.56 para la versión multinodo, y entre 2.37 y 3.12 para la versión completamente paralela. Los resultados sugieren que la paralelización del experimento podría mitigar los enormes tiempos de ejecución del enfoque original.

Palabras clave: Apache™ Hadoop®, Apache Spark™, escalonabilidad de grafos bipartitos, experimentos en paralelo, problemas NP sin clasificar.

Abstract

Graph shellability is an NP problem whose classification either in P or in NP-complete remains unknown. In order to understand the computational behavior of graph shellability on bipartite graphs, as a particular case, it could be useful to develop

* Nota: Es difícil hacer una traducción literal de la palabra shellability y sus derivadas en español sin sacrificar su significado original. Usamos la palabra escalonabilidad y derivadas como se hace en (Cruz & Estrada, 2008) para respetar la convención impuesta por los autores y porque nos parece que sugiere la idea de secuencia que es importante en la definición. Respecto al uso de la palabra shellabilidade en portugués, tomamos como precedente a (Lewiner, 2005)

an efficient way to generate and analyze results over sets of shellable and non-shellable instances. In this way, a sequentially designed exponential time experiment for deciding shellability on randomly generated instances was proposed in literature. In this work, with the aim of improving the performance of that experiment, we propose three alternative approaches using Apache Spark™, we called multi-core, multi-node and full-parallel. We tested and compared their execution time for bipartite graphs with 10,12,15,20 and 50 vertices with regard to the original version, and we got speedups between 1.37 and 1.67 for the first one, between 2.34 and 3.56 for the second one, and between 2.37 y 3.12 for the last version. The results suggest that parallelization could relieve the large execution times of the original approach.

Keywords: Apache™ Hadoop®, Apache Spark™, bipartite graph shellability, parallel experiments, unclassified NP problems.

Resumo

A shellabilidade dos grafos é um problema em NP, do qual é desconhecida sua inclusão nas classes da complexidade P ou NP-completo. A fim de compreender seu comportamento computacional no caso particular dos grafos bipartidos, poderia ser útil ter um método eficiente para gerar e analisar instâncias shellables. A literatura relata um experimento sequencial, e custo exponencial, projetado para determinar a escalabilidade do um conjunto de instâncias. Neste trabalho, e a fim do melhorar o desempenho do experimento mencionado, propomos três alternativas usando Apache Spark: uma multinúcleo, outra multinó e outra completamente paralela. Além disso, nós compararmos o tempo de execução de cada um deles respeito da versão original em grafos bipartidos com 10,12,15,20 e 50 vértices e obtivemos acelerações (speedups) entre 1.37 e 1.67 para a versão Multinúcleo, entre 2.34 e 3.56 para a versão Multinó, e entre 2.37 e 3.12 para a versão completamente paralela. Os resultados sugerem que a paralelização do experimento poderia atenuar os enormes tempos de execução da abordagem original.

Palavras-chave: Apache™ Hadoop®, Apache Spark™, shellabilidade de grafos bipartidos, experimento paralelo, problemas NP não classificado

Introduction

Simplicial complexes are combinatorial structures frequently used in geometrical applications because of their flexibility for modeling objects from different spatial dimensions. The presence of one of their combinatorial properties, known as shellability, has proved to be useful in practical situations (see, for example the works of Herlihy (2010) and Müller-Hannemann (2001)). The concept also appears in graph theory where, through the Stanley-Reisner correspondence, a simplicial complex may be associated to a graph (Van Tuyl & Villarreal, 2008).

Simplicial complex shellability and its graph counterpart have been well-studied and widely used in diverse mathematical and practical issues, but there exists relatively little work about their computational complexity. Although deciding shellablility requires significant amounts of computational time, it is currently unknown if the problem is either in P or in NPC (i.e. NP-complete) (Kaibel y Pfetsch, 2003).

In order to understand the computational behavior of graph shellability, and based on some combinatorial characterizations, the problem is commonly tackled by analyzing particular graph families. Fortunately, in the case of bipartite graphs a complete characterization was made in (Van Tuyl y Villarreal, 2008) and (Cruz y Estrada, 2008) that was further used in (Santamaria-Galvis, 2013) to propose a bipartite graph shellability solver called `isShellable_BG`.

`isShellable_BG` decides bipartite graph shellability in exponential time and was used in a sequentially designed experiment as a tool for collect some data that could be used in further research to construct some conjectures about the problem's behavior.

In this paper, with the aim of improving the performance of the sequential experiments performed in (Santamaria-Galvis, 2013), we propose and implement three parallel alternatives using Apache Spark™ (Spark™, 2016).

Methods and Materials

Main Notions and Required Results

A (simple and finite) graph G is a tuple of two finite sets (V,E) where V is the set of vertices and E is a set of unordered pairs over V called the edges of G; no edge is repeated and loops, i.e. edges from one vertex to itself, are not allowed. Two vertices $x_1,x_2 \in V$ are said to be *adjacent* in G if $(x_1,x_2) \in E$. A subset F of V is an *independent* set of G if not two vertices of F are adjacent in G; F is a *maximal independent* set if it is not properly included inside another independent set. If x is a vertex of V, we denote by $N(x)$ the *open neighborhood* of x, i.e., the set of all vertices adjacent to x, $N[x] := \{x\} \cup N(x)$ the *closed neighborhood* of x, and $\deg(x) := |N(x)|$ the *degree* of x.

A vertex with degree 1 is called a *pendant vertex*. A graph *G* is bipartite if its set of *n* vertices can be partitioned in two sets V_α and V_b such that no edge exists in vertices of the same set. Figure 1 (left) represents a bipartite graph. We say that a bipartite graph is complete if every vertex in V_α is adjacent with every vertex in V_b. We denote it by $K_{r,s}$, where $r = |V_\alpha|$, $s = |V_b|$.

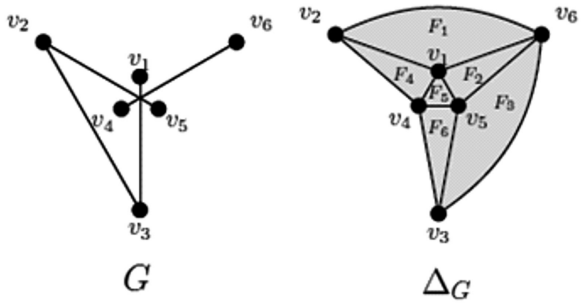

Figure 1. A bipartite graph *G* and its associated (pure) simplicial complex Δ_G. The ordering $F_1, F_2, F_3, F_4, F_5, F_6$ is a shelling of Δ_G; consequently, Δ_G is a shellable simplicial complex and *G* is a shellable graph.

A (*n abstract*) *simplicial complex* Δ over a set of vertices *V* is a finite and nonempty collection of subsets of *V* called faces, such that if *A* is a face of Δ, then so is every nonempty subset of *A*. Figure 1 (right) shows a graphical representation of a simplicial complex. The *dimension of a face A* is defined as dim(*A*) := |*A*| −1 and the *simplicial complex dimension* is defined as dim(Δ) := max(dim(*A*)). The maximal faces in Δ are called facets. If every facet in Δ has dimension *d*, is *d-dimensional* and is *called pure*. For sets $A \subseteq B$, there exists the *boolean interval [A; B]* := {*C*| $A \subseteq C \subseteq B$}. Let $\overline{\mathbf{A}}$:= [Ø; *A*]. A complex of the form $\overline{\mathbf{A}}$ is called *simplex* (Björner y Wachs, 1996), (Schläfli, 1901).

We can define shellable simplicial complex and its related decision problem as follows.

Definition 1 (Shellable simplicial complex (Björner y Wachs, 1996)). A simplicial complex is called *shellable* if its facets can be arranged in a linear order $F_1, F_2, \ldots, F_t$, in such a way that the subcomplex $(\bigcup_{i=1}^{k-1})\overline{F}_i \cap \overline{F}_k$ is pure and (dim(F_k) – 1)-dimensional for all k=2,...,t. Such an ordering of facets is called a *shelling*.

Problem 1 (*SCS*: SimplicialComplexShellability (Kaibel & Pfetsch, 2003)
INPUT: A simplicial complex Δ represented by a list of its facets.
QUESTION: Is Δ shellable? (return either *YES* or *NO*).

It is easy to show that *SCS* is a decision problem in NP, but it is currently unknown whether it is in P, NPC or even fits in another class into NP (Kaibel, y Pfetsh, 2003). With the aim of understanding the complexity of SCS we could deal with a related problem. The next definition introduces a kind of simplicial complex which could be generated from a given graph. Thus, SCS could be partially studied through some graph families where shellability is fully characterized, e.g. there are complete characterizations for the property on chordal, bipartite, arc-circular, vertex decomposable, simplicial and recursively simplicial graphs in (Van Tuyl y Villarreal, 2008), (Cruz y Estrada, 2008), (Woodroofe, 2009) and (Castrillón y Cruz, 2012).

Definition 2 (Independence (simplicial) complex of a graph (Van Tuyl y Villarreal, 2008)). Let *G* = *(V,E)* be a graph on the vertex set $V = \{x_1,\ldots,x_n\}$. By identifying vertex x_i with the variable x_i in the polynomial ring $R = k[x_1,\ldots,x_n]$ over a field *k*, we can associate *G* with a quadratic square-free monomial ideal *I(G)* = $(\{x_i\ x_j | (x_i,x_j) \in E\})$ where *E* is the edge set of *G*, the ideal *I(G)* is called the edge ideal of *G*. By using the Stanley-Reisner correspondence, we can also associate G with the simplicial complex Δ_G, called the *independence (simplicial) complex of the graph G*, where $I_{\Delta G} = I(G)$. Thus, the faces of Δ_G are the independent sets of *G*.

Now, we can define shellable graph and graph shellability as a decision problem.

Definition 3 (Shellable graph (Van Tuyl & Villarreal, 2008)). Let *G* be a graph and Δ_G its independence complex. *G* is shellable if Δ_G is a *shellable simplicial complex*.

Problem 2 (*GS*: graphShellability)
INPUT: A graph *G* or a list of all its maximal independent sets.
QUESTION: Is *G* shellable? (return either YES or NO).

It is also unknown whether *GS* is either in P or in NPC, and that is also the case for bipartite graphs; however, as we shall show in detail, the next theorem is useful to decide *GS* for bipartite graphs.

Theorem 4 (Van Tuyl y Villarreal, 2008), (Cruz y Estrada, 2008). Let *G* be a bipartite graph. Then *G* is shellable if and only if there are adjacent vertices x and y where *x* is a pendant vertex such that the graphs $G \setminus N_G[x]$ and $G \setminus N_G[y]$ are shellable.

Notation: From now on, and for the sake of brevity, we shall use $GS_{\text{bipartite}}$ when we refer to *GS* for bipartite graphs.

`isShellable_BG`: A solver for $GS_{\text{bipartite}}$

Let G be a bipartite graph on the vertex set V with $n:=|V|$ and $St(G)$ be the set of the maximal independent sets of G. A direct interpretation of the theorem 4 leads to the exponential time algorithm `isShellable_BG` (Procedure 1), which was proposed by (Santamaria-Galvis, 2013) as a tool to analyze the computational behavior of $GS_{\text{bipartite}}$ It was implemented in C/C++ using the igraph library (Csárdi y Nepusz, 2016). This is a faster alternative than calculate GS directly, i.e. by first obtaining all the maximal independent sets of G. Note that the mere problem of finding the maximum independent set in G is an NP-hard problem for the general case. Fortunately, `isShellable_BG` offers a way to solve $GS_{\text{bipartite}}$ directly from G by avoiding the heavy precalculations required to construct $St(G)$.

Procedure 1: Algorithm `isShellable_BG`(G)

Data: A bipartite graph $G = (V. E)$

Result: true if G is shellable, **false** otherwise

begin

1 **if** ($|V| \leq 2$) **then return true**

2 $x \leftarrow$ a pendant vertex in V_G

3 **if** (x = **null**) **then return false**

4 $y \leftarrow N_G(x)$

5 **if** (`isShellable_BG` ($G \backslash N_G[y]$) **and** `isShellable_BG` ($G \backslash N_G[x]$)) **then return true**
else return false

To understand the asymptomatic behavior of $GS_{\text{bipartite}}$ over increasing values of n and to build some conjectures, a sequentially designed experiment was also implemented by (Santamaria-Galvis, 2013) using `isShellable_BG`. The next section explains the protocol over which the experiment was performed and how we built our proposal.

Statistical analysis

Before describing our proposal, it is necessary to properly introduce the original experimental protocol in such a way that we can describe our three approaches.

Original experimental protocol

For several values of n, a set of t initial instances was created for each n. Each initial instance is a complete bipartite graph $K_{r,s}$, $r + s = n$ with its rs edges randomly stored in a file; besides, r and s are also randomly chosen from the given n. Every initial instance was used to generate a set of actual instances. Let $K_{r,s}^{(i)}$ denote the i-th initial instance and $G_{n,m}$ an (actual) instance of $GS_{\text{bipartite}}$ with n vertices and m edges. Once t and $\varepsilon \in (0,1)$ are fixed, an experiment could be performed for several values of n by using the next experimental protocol:

Procedure 2: Experimental protocol

Data: $n \in \mathbb{N}$, $t \in \mathbb{N}$, $\varepsilon \in (0,1)$

begin

1 Generate t initial instances

2 $m \leftarrow \lfloor \varepsilon n \rfloor$

3 **foreach** $i \in \{1,..., t\}$ **do**

4 Take the first m edges from $K_{r,s}^{(i)}$ and generate $G_{n,m}$.

5 Decide shellability of $G_{n,m}$ with `isShellable_BG`.

6 $m \leftarrow m + \lfloor \varepsilon n \rfloor$.

7 If $m \leq rs$, then go back to line 4, otherwise, continue with the next instance (line 3)

Thus, a sequentially designed experiment was performed in (Santamaria-Galvis, 2013) after setting the values $\varepsilon = 0.1$, $t = 200$, and $n = 50,100,150$ over the previous protocol. Here, the word sequential is employed to mean that in the whole experiment just a single computer (node) was used and, inside it, only one core; the other cores remained idle. Because of the low values of n that could be tested there, no conclusive results were obtained regarding the asymptotic behavior of $GS_{\text{bipartite}}$.

In this work, under the assumption that parallelization could relieve to some extent the computational burden involved in the original sequentially performed experiment, we propose three parallel ways of running the aforementioned experiment by slightly modifying some steps in the protocol. Here, it is important to stress that our parallel approaches are intended for the experiments themselves, not for the exponential time routine `isShellable_BG`.

Our proposal

To achieve parallelization over the experimentation, we propose three options by using Apache Spark™ (Spark™, 2016) as a framework for cluster compu-

ting and Apache™ Hadoop® (Hadoop®, 2016.) as underlying distributed file system. We call them *multi-core approach, multi-node approach* and *full-parallel approach*. They are completely defined by the modifications they impose over the protocol of Procedure 2. In this description, we intentionally omit those accessory details we consider strictly operative:

1) After line 1, the whole set of actual instances is generated from the initial ones. They are stored in Hadoop.

2) Depending on the approach, Spark runs line 5 —originally intended to be run under a sequential scheme— in one of these ways: (i) In the multi-core approach, Spark uses just one node during all the experimentation, but each core inside is always busy running `isShellable_BG` over a different instance. Although the cores are independently used, the other resources (main memory, cache, etc.) are shared. (ii) In the multi-node approach, Spark uses several nodes: one node as master and several nodes as slaves. Every slave can decide GS_bipartite by using its resources, but just one core per node is actually used. (iii) The full-parallel approach, proceeds as the previous one, but with two cores per node.

Performance Test

Let us fix the values ε = 0.1 and t = 200. Then, to contrast the efficiency of our proposal, we proceed as follows:

For every value of n in {10,12,15,20,50}, a set of initial instances was created; after that, the set of actual instances was generated and stored in Hadoop in order to be evaluated by the solver afterwards. For increasing values of n, the sets of instances were run over the three approaches in this way: (i) The protocol from Procedure 2 was run as is since line 3 to line 7. The total time $T_O\ (n)$ (in seconds) that it took to accomplish the whole task (i.e., to solve the problem for every instance in the given n), was stored and used as reference. Note that this is, in fact, the original approach. (ii) In an analogous way, we run the modified protocols for the multi-core, multi-node and full-parallel approaches over the same lines and we store the respective total times $T_{MC}\ (n)$, $T_{MN}\ (n)$ and $T_{FP}\ (n)$ (in seconds). The multi-node and full-parallel approaches used one master and two slaves; besides, the same master is also used as the single node for the multi-core approach. The master node has 4 cores and 57GB of RAM, and the slave nodes have 2 cores and 15GB of RAM.

Once the experiments were finished, we had to choose an appropriate measurement for their relative efficiency. Thus, in the sense of (Hennessy, y Patterson, 2012), we opted for the *speedups* of the enhanced experiments. The speedup of some computational task reflects an improvement in speed of its execution and consequently means an improvement in its efficiency. The speedup could be properly defined as the ratio between the execution time for a task without using the enhancement and the execution time for the same task using it. Thereby, we could define $S_{MC} := \frac{T_0}{T_{MC}}$, $S_{MN} := \frac{T_0}{T_{MN}}$ and $S_{FP} := \frac{T_0}{T_{FP}}$ as the speedups for the multi-core, multi-node and full-parallel approaches, respectively. In this way, every speedup reflects how fast a routine is regarding some fixed reference.

Results

After setting the parameters, we run the aforementioned performance test, and then, we tabulated the results. Table 1 displays the execution time alongside the speedups of our proposal, both over increasing values of n. Time is rounded to the nearest second in each case, and speedups are rounded to two decimal places.

Table 1. Execution Times and Speedups over increasing values of

n	T_o	T_{MC}	T_{MN}	T_{FP}	S_{MC}	S_{MN}	S_{FP}
10	59	43	23	24	1.37	2.57	2.46
12	82	49	35	33	1.67	2.34	2.48
15	121	75	34	47	1.61	3.56	2.57
20	116	73	34	49	1.59	3.41	2.37
50	396	290	168	127	1.37	2.36	3.12

Total execution time T_0, T_{MC}, T_{MN}, T_{FP}, (in seconds) for the original, multi-core, multi-node and full-parallel approaches, respectively. S_{MC}, S_{MN}, and S_{FP} for the multi-core, multi-node and full-parallel speedups, respectively

In order to evince the improvements, we plotted two graphics: Figure 2 contrasts the performance of every approach as function of n. Figure 3 compares the three speedups, also, as function of n.

The results suggest increasing efficiency regarding the original protocol design. The speedups fluctuate between 1.37 and 1.67 for the multi-core approach, between 2.34 and 3.56 for the multi-node, and between 2.37 and 3.12 for the full-parallel. This behavior can be

observed in Figure 2 where time usually reduces from one approach to another, and in Figure 3 where the speedups for the multi-core approach are always under the multi-node and full-parallel approaches. Maybe, the use of several machines with independent resources for the multi-node approach played the differential factor in the performance.

Discussion

This work was intended for verifying whether one of the enhanced approaches could be used instead of the original one. A first conclusion we can draw is that the computational burden of the sequential protocol could be effectively relieved by using the parallel techniques we consider here. The times and speedups in Table 1 confirm it. However, we can distinguish different levels of improvement.

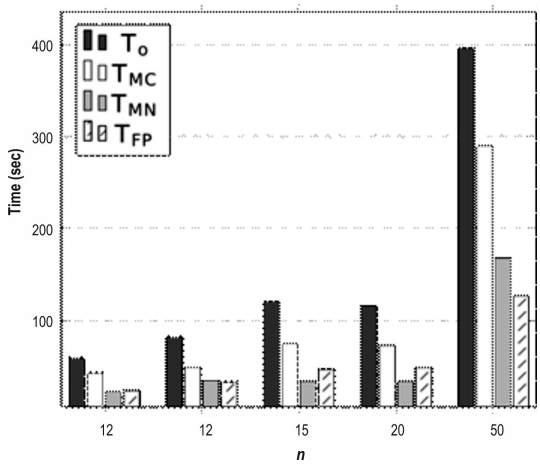

Figure 2. Total time from the different approaches over increasing values of .

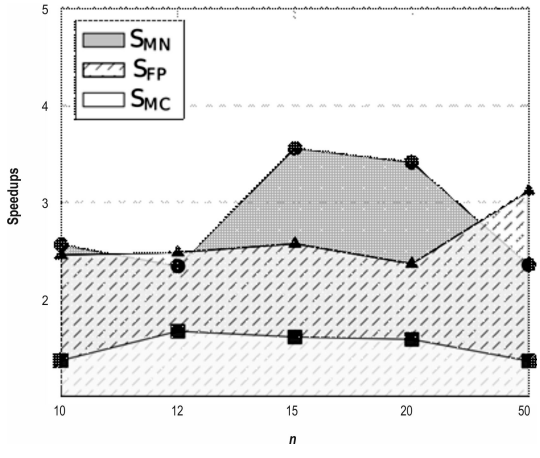

Figure 3. Speedups of each approach for increasing values of *n*

Figure 2 suggests that, for increasing values of *n*, the multi-core approach is outperformed by the others. Regarding them, we can see that multi-node and full-parallel approaches are almost in a draw for $n < 20$, then, for $n = 20$, the full-parallel approach is less efficient than the multi-node one, but the performance swaps when $n = 50$. Those differences are reinforced in Figure 3 but in terms of speedups, which range between 2.34 and 3.56 for the multi-node approach, and between 2.46 and 3.12 for the full-parallel one. Nevertheless, it could be hazardous to generalize a concrete behavior, mainly because the test was performed for relatively small values of *n*. Nevertheless, we could conjecture that for increasing values of n the difference between the approaches will stabilize. We suspect that the more parallel elements (nodes and cores) the approach includes, the less time is required to complete the whole experiment. Thus, the full-parallel approach looks like the best candidate to deal with higher values of *n*.

As a future work, a new version should try the new library GraphX of Apache Spark™ in order to deal directly with some graph related functionalities. The use of GraphX could be better than our approach because the library seems to coexist in the same architectural level of the framework.

On the other hand, and in a more general sense, we suggest this parallel scope to deal with similar combinatorial problems where experimentation over massive sets of data is required either to construct mathematical conjectures or to analyze their asymptotic behavior.

Acknowledgment

This research was conceived inside the context of the course Proyecto Integrador 2 offered by the Computer Science Engineering program at the University of Antioquia. The University provided us with those computational and human resources required to accomplish the proposed goals through the research group Ingeniería y Software. We want to express our gratitude for their commitment to our work.

References

Björner A, Wachs ML. Shellable Nonpure Complexes and Posets I. Trans. Amer. Math. Soc. 1996;348(4):1299-1327.

Castrillón ID, Cruz R.). Escalonabilidad de grafos e hipergrafos simples que contienen vértices simpliciales. Matemáticas: Enseñanza Universitaria. 2012;20(1):29-80.

Cruz R, Estrada M. Vértices simpliciales y escalonabilidad de grafos. Morfismos. 2008;12:21-36.

Csárdi G, Nepusz T. 2006. The igraph software package for complex network research. InterJournal Complex Systems. Retrieved from http://igraph.sf.net

Csárdi G, Nepusz T. (2012, June). The igraph software package for complex network research, (ver. 0.6). Retrieved from http://igraph.sf.net

Hadoop®, A. 2016. The Apache® Software Foundation. Retrieved from The Apache® Software Foundation: http://hadoop.apache.org/

Hennessy JL, Patterson DA. 2012. Computer Architecture: A Quantitative Approach. 5th ed. Elsevier - Morgan Kaufmann.

Herlihy M. 2010. Applications of Shellable Complexes to Distributed Computing. CONCUR (pp. 19-20). Springer.

Herlihy, M., & Rajsbaum, S. 2010. Concurrent Computing and Shellable Complexes. DISC (pp. 109-123). Springer.

Kaibel V, Pfetsch ME. 2003. Some Algorithmic Problems in Polytope Theory. Algebra, Geometry, and Software Systems (outcome of a Dagstuhl Seminar) (pp. 23-47). Springer-Verlag.

Lewiner T. 2005. Complexos de Morse discretos e geométricos. master's thesis. Rio de Janeiro.

Müller-Hannemann M. Shelling Hexahedral Complexes for Mesh Generation. Journal of Graph Algortihms and Applications, 2001;5(5):59-91.

Santamaria-Galvis AD. 2013. On Algorithmic Complexity of Shellability in Graphs and their Associated Simplicial Complexes. Master's thesis, Facultad de Minas, Universidad Nacional de Colombia, Medellin, Colombia, Medellín.

Schläfli L. (1901, January). Theorie der vielfachen Kontinuität; hrsg. im Auftrage der Denkschriften_Kommission der Schweizer. Naturforschenden Gesellschaft. Zürich: Zürcher & Furrer.

Spark™ A. 2016. Apache Spark™ Lightning-fast cluster computing. (The Apache® Software Foundation) Retrieved from Apache Spark™ Lightning-fast cluster computing: http://spark.apache.org/

Van Tuyl A, Villarreal RH. Shellable graphs and sequentially Cohen-Macaulay bipartite graphs. J. Combin. Theory Ser. A, 2008;115(5):799-814.

Woodroofe R. Vertex decomposable graphs and obstructions to shellability. Proc. Amer. Math. Soc. 2009;137(10):3235-3246.