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

Julián David Arango-Holguín, Milena Cárdenas-Alzate and Andrés David Santamaría-Galvis*

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 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 two alternative approaches using Apache Spark™: a multi-core one and a multi-node one. 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 former and between 2.34 and 3.56 for the latter. The results suggest that parallelization could relieve the large execution times of the original approach.**

Index Terms—**Bipartite graph shellability, graph shellability, unclassified NP problems, parallel experiments, Apache Spark™, Apache™ Hadoop®.**

I. 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, [1]–[4]). The concept also appears in graph theory where, through the Stanley–Reisner correspondence, a simplicial complex may be associated to a graph [5].

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) [6].

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 [5] and [7] that was further used in [8] 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 to construct some conjectures about the problem behavior for increasing values of its parameters. In this paper, with the aim of improving the performance of the sequential experiments performed in [8], we propose and implement two parallel alternatives using Apache Spark™ [9].

This paper is structured as follows: in section II, the main notions, required results, and the original experimental protocol are properly introduced. In section III, our alternative approaches are presented and, in section IV, we describe a way to compare their efficiency with respect to the original one. Section V presents the results and discusses the performance of our proposal. Finally, in section VI, the conclusions and some possibilities of future work are shown.

II. 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. A subset F of V is an *independent set* of G if $e \nsubseteq F$, for all $e \in E$; F is a *maximal independent set* if it is not properly included inside another independent set. Two vertices x_1 and x_2 are adjacent if $(x_1, x_2) \in E$. 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_a and V_b such that no edge exists in vertices of the same set. Fig. 1 (left) represents a bipartite graph. We say that a bipartite graph is *complete* if every vertex in V_a is adjacent with every vertex in V_b. By $K_{r,s}$ we denote a complete graph with $r = |V_a|$, $s = |V_b|$, and $r + s = n$.

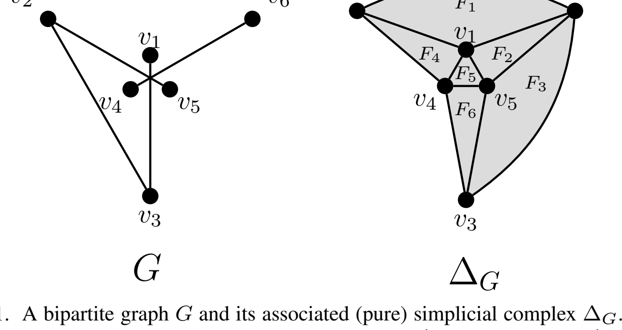

Fig. 1. A bipartite graph G and its associated (pure) simplicial complex Δ_G. The ordering $F_1, F_2, F_3, F_4, F_5, F_6$ is a shelling of Δ_G; consequently, Δ_G is a shellable simplicial complex and G is a shellable graph.

*Ingeniería de Sistemas, Universidad de Antioquia, Medellín, Colombia, `(julian.arango2|milena.cardenasa|david.galvis)@udea.edu.co`

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. Fig. 1 (right) shows a graphical representation of a specific simplicial complex. The dimension of a face A is defined as $\dim(A) := |A| - 1$ and the simplicial complex dimension is defined as $\dim(\Delta) := \max(\dim(A))$. The maximal faces in Δ are called *facets*. If every facet in Δ has dimension d, then Δ is *d-dimensional* and is called *pure*. For sets $A \subseteq B$, there exists the *boolean interval* $[A; B] = \{C \mid A \subseteq C \subseteq B\}$. Let $\bar{A} := [\varnothing; A]$. A complex of the form $\bar{A}$ is called a *simplex* [10], [11].

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

Definition 1 (Shellable simplicial complex [10])**.** 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 $\left(\bigcup_{i=1}^{k-1} \overline{F_i}\right) \cap \overline{F_k}$ is pure and $\dim(F_{k-1})$-dimensional for all $k = 2, \ldots, t$. Such an ordering of facets is called a *shelling*.

Problem 1 (SCS: simplicialComplexShellability [6])
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 [6]. 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[1].

Definition 2 (Independence (simplicial) complex of a graph [5])**.** Let $G = (V, E)$ be a graph on the vertex set $V = \{x_1, \ldots, x_n\}$. By identifying the vertex x_i with the variable x_i in the polynomial ring $R = k[x_1, ..., x_n]$ over a field k, can be associated to G a quadratic square-free monomial ideal $I(G) = (\{x_i x_j \mid \{x_i, x_j\} \in E\})$ where E is the edge set of G, the ideal $I(G)$ is called the *edge ideal* of G. Using the Stanley–Reisner correspondence, can be associated to G 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 and, consequently, its facets are the maximal independent sets of G.

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

Definition 3 (Shellable graph [5])**.** Let G be a graph and Δ_G its independence complex. G is *shellable* if Δ_G is a shellable simplicial complex.

[1]There are complete characterizations for the property on chordal, bipartite, arc-circular, vertex decomposable, simplicial and recursively simplicial graphs are characterized in [5], [7], [12], [13].

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 & Villarreal [5], and Cruz & Estrada [7])**.** *Let G be a bipartite graph. Then G is shellable if and only if there are adjacent vertices x and y with $\deg_G(x) = 1$ such that the graphs $G \backslash N_G[x]$ and $G \backslash N_G[y]$ are shellable.*

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

A. `isShellable_BG`: *A solver for* $\mathsf{GS}_{\mathsf{bipartite}}$

Let G be a bipartite graph on the vertex set V with $n := |V|$ and $\mathrm{St}(G)$ be the set of 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 [8] as a tool to analyze the computational behavior of $\mathsf{GS}_{\mathsf{bipartite}}$. It was implemented in `C/C++` using the `igraph` library [14], [15]. It supposes an advantage regarding the direct way to deal with GS, i.e. by first obtaining all the maximal independent sets of G, but the mere problem of finding the maximum independent set in G is an NP-hard optimization problem for the general case. Fortunately, `isShellable_BG` offers a way to solve $\mathsf{GS}_{\mathsf{bipartite}}$ directly from G by avoiding the heavy precalculations required to construct $\mathrm{St}(G)$.

Procedure 1: Algorithm `isShellable_BG`(G) [8]

```
Data: A bipartite graph $G = (V, E)$
Result: true if $G$ is shellable, false otherwise
begin
1   if ($|V| \leqslant 2$) 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 $\mathsf{GS}_{\mathsf{bipartite}}$ over increasing values of n and to build some conjectures, a sequentially designed experiment was also implemented by [8] using `isShellable_BG`. The next section explains the protocol over which the experiment was performed.

B. Original experimental protocol

For several values of n, a set of t initial instances was created. 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 $G_{u,v}$ an (actual) instance of $\mathsf{GS}_{\mathsf{bipartite}}$ with u vertices and v 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 [8]

```
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 $i \in \{1,\ldots,t\}$ 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 $m \leqslant rs$, then go back to line 4, otherwise, continue
        with the next instance (line 3)
```

Thus, a sequentially designed experiment was performed in [8] after setting the values $\varepsilon = 0.1$, $t = 200$, and $n = 50, 100, 150$ in the previous protocol. Here, the word *sequential* is employed to mean that in the whole experiment there was used just a single computer (node) 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 $\mathsf{GS}_{\mathsf{bipartite}}$.

Therefore, 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 two 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 experimentation itself, not for the exponential time routine `isShellable_BG`.

III. Our proposal

To achieve parallelization over the experimentation, we propose two options by using Apache Spark™ [9] as a framework for cluster computing and Apache™ Hadoop® [16] as underlying distributed file system. We call them *multi-core approach* and *multi-node approach*. Both are completely defined by the modifications they impose over the protocol[2]:

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 just one of these ways: (i) in the multi-core approach, Spark uses just one node during all the experimentation, but every single 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 $\mathsf{GS}_{\mathsf{bipartite}}$ by using all its resources, but just one core per node is actually used.

[2]In this description, we intentionally omit those accessory details we consider were strictly operative.

IV. Performance test

Let us fix the values $\varepsilon = 0.1$ 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 was 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 $\mathsf{T_o}(n)$ (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 and multi-node approaches over the same lines and we store the respective total times $\mathsf{T_{MC}}(n)$ and $\mathsf{T_{MN}}(n)$ (in seconds).

The multi-node approach used one master and two slaves; besides, the same master is also used as the single node for the other approaches. 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 [17], 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 the enhancement. Thereby, we could use it to define $\mathsf{S_{MC}}$, the *multi-core speedup*, and $\mathsf{S_{MN}}$, *multi-node speedup*, as $\mathsf{S_{MC}} := \frac{\mathsf{T_o}}{\mathsf{T_{MC}}}$ and $\mathsf{S_{MN}} := \frac{\mathsf{T_o}}{\mathsf{T_{MN}}}$.

V. Results

The table I displays the execution time for the aforementioned approaches alongside the speedups of our proposal, both over increasing values of n. Time is rounded to the nearest second in each case.

TABLE I
Execution Times and Speedups over Increasing Values of n

n	$\mathsf{T_o}$	$\mathsf{T_{MC}}$	$\mathsf{T_{MN}}$	$\mathsf{S_{MC}}$	$\mathsf{S_{MN}}$
10	59	43	23	1.372	2.565
12	82	49	35	1.673	2.343
15	121	75	34	1.613	3.559
20	116	73	34	1.589	3.412
50	396	290	168	1.366	2.357

Total execution times $\mathsf{T_o}$, $\mathsf{T_{MC}}$ and $\mathsf{T_{MC}}$ (in seconds) for the original, multi-core and multi-node approaches, respectively. $\mathsf{S_{MC}}$ and $\mathsf{S_{MN}}$ stand for the multi-core and multi-node speedups, respectively.

The results suggest significant improvements regarding the original version: the speedups fluctuate between 1.37 and 1.67 for the multi-core approach, and between 2.34 and 3.56 for the multi-node case; that means an improvement between 37% and 67% for the former and between 134% and 256% for the latter. This can be evinced in the Fig. 2 where time gradually reduces from one approach to the other, and in Fig. 3 where the speedups for the former case are always under the latter one. Maybe, the use of several machines with independent resources for the multi-node approach played the differential factor in the performance.

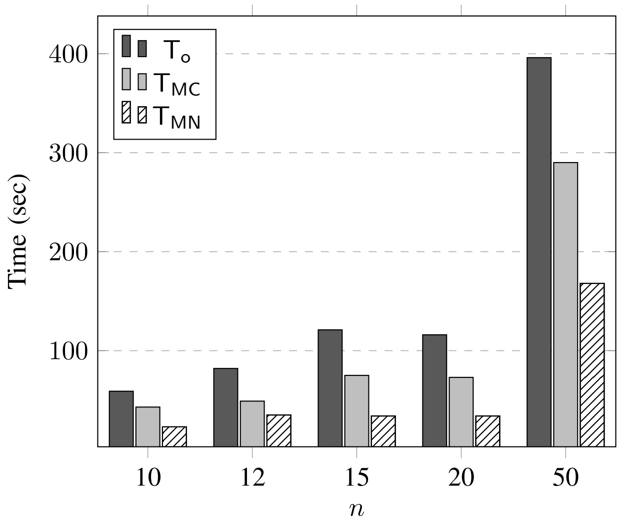

Fig. 2. Total time from the different approaches over increasing values of n.

Although the results graphically suggest trends in the way that the multi-node approach outperforms the other (Fig. 2) and in the seemingly regular behavior of the speedups over a fixed approach (Fig. 3), it could be hazardous to generalize for every value of n: the experiments were run over small values of n and the test was intended to verify whether one of these enhanced approaches could be used instead of the original one.

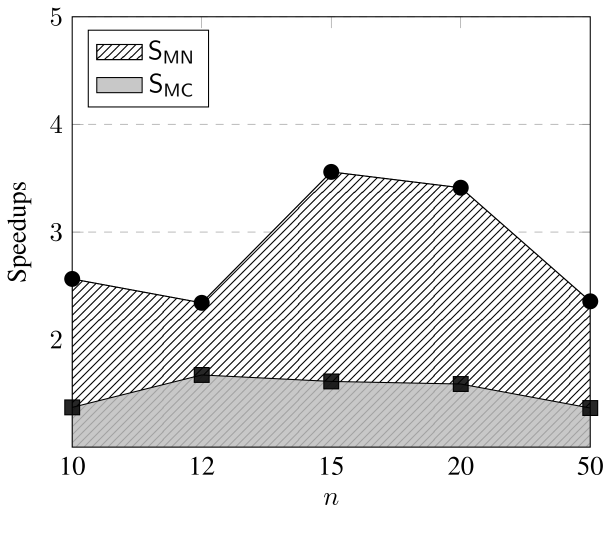

Fig. 3. Speedups of each approach for increasing values of n.

VI. Conclusions and Future Work

The speedups suggest that the proposed versions are fair options to relieve, to some extent, the computational burden of the original experiment by [8] which was initially intended to analyze the asymptotic behavior of $\mathsf{GS}_{\mathsf{bipartite}}$. The speedup ranges are notoriously higher in the multi-node approach: it gets execution times between 2.34 and 3.56 times faster than the sequential approach for the tested values of n; that makes this version the best candidate to deal with higher values of n in an asymptotic analysis of the problem.

As a future work, a full-parallel version with Apache Spark™ and Apache™ Hadoop® could be implemented by allowing parallelization in all the cores of a multi-node cluster; besides, the 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 coexists in the same architectural level of the framework.

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 conjectures or to analyze their asymptotic behavior.

References

[1] M. Herlihy, "Applications of shellable complexes to distributed computing – (invited talk)," in *CONCUR*, ser. Lecture Notes in Computer Science, P. Gastin and F. Laroussinie, Eds., vol. 6269. Springer, 2010, pp. 19–20.
[2] M. Herlihy and S. Rajsbaum, "Concurrent computing and shellable complexes," in *DISC*, ser. Lecture Notes in Computer Science, N. A. Lynch and A. A. Shvartsman, Eds., vol. 6343. Springer, 2010, pp. 109–123.
[3] L. De Floriani, P. Magillo, and E. Puppo, "Compressing triangulated irregular networks," *Geoinformatica*, vol. 4, pp. 67–88, 2000.
[4] M. Müller–Hannemann, "Shelling hexahedral complexes for mesh generation," *Journal of Graph Algortihms and Applications*, vol. 5, no. 5, pp. 59–91, 2001.
[5] A. Van Tuyl and R. H. Villarreal, "Shellable graphs and sequentially Cohen-Macaulay bipartite graphs," *J. Combin. Theory Ser. A*, vol. 115, no. 5, pp. 799–814, 2008.
[6] V. Kaibel and M. E. Pfetsch, "Some algorithmic problems in polytope theory," in *Algebra, Geometry, and Software Systems (outcome of a Dagstuhl Seminar)*. Springer-Verlag, 2003, pp. 23–47.
[7] R. Cruz and M. Estrada, "Vértices simpliciales y escalonabilidad de grafos," *Morfismos*, vol. 12, pp. 21–36, 2008.
[8] A. D. Santamaria-Galvis, "On algorithmic complexity of shellability in graphs and their associated simplicial complexes," Master's thesis, Facultad de Minas, Universidad Nacional de Colombia, Medellin, Colombia, 2013.
[9] Apache Spark™, Lightning-fast cluster computing. Available from http://spark.apache.org/. The Apache® Software Foundation, 2016.
[10] A. Björner and M. L. Wachs, "Shellable nonpure complexes and posets. I," *Trans. Amer. Math. Soc.*, vol. 348, no. 4, pp. 1299–1327, 1996.
[11] L. Schläfli, *Theorie der vielfachen Kontinuität; hrsg. im Auftrage der Denkschriften–Kommission der Schweizer. Naturforschenden Gesellschaft*. Zürich: Zürcher & Furrer, January 1901.
[12] R. Woodroofe, "Vertex decomposable graphs and obstructions to shellability," *Proc. Amer. Math. Soc*, vol. 137, no. 10, pp. 3235–3246, 2009.
[13] I. D. Castrillón and R. Cruz, "Escalonabilidad de grafos e hipergrafos simples que contienen vértices simpliciales," *Matemáticas: Enseñanza Universitaria*, vol. 20, no. 1, pp. 29–80, June 2012.
[14] G. Csárdi and T. Nepusz, "The `igraph` software package for complex network research," *InterJournal Complex Systems*, 2006. [Online]. Available: http://igraph.sf.net
[15] ——. (2012, June) The `igraph` software package for complex network research, (ver. 0.6). Available at http://igraph.sf.net. [Online]. Available: http://igraph.sf.net
[16] Apache™ Hadoop®. Available from http://hadoop.apache.org/. The Apache® Software Foundation, 2016.
[17] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed. Elsevier – Morgan Kaufmann, 2012.