2006

A Note on Quasi-Triangulated Graphs

Ion Gorgos
Academy of Economic Studies of Moldova

Chính T. Hoàng
Wilfrid Laurier University, choang@wlu.ca

Vitaly Voloshin
Troy University

SIAM J. DISCRETE MATH.
Vol. 20, No. 3, pp. 597–602

A NOTE ON QUASI-TRIANGULATED GRAPHS*

ION GORGOS†, CHÍNH T. HOÀNG‡, AND VITALY VOLOSHIN§

Abstract. A graph is quasi-triangulated if each of its induced subgraphs has a vertex which is either simplicial (its neighbors form a clique) or cosimplicial (its nonneighbors form an independent set). We prove that a graph G is quasi-triangulated if and only if each induced subgraph H of G contains a vertex that does not lie in a hole, or an antihole, where a hole is a chordless cycle with at least four vertices, and an antihole is the complement of a hole. We also present an algorithm that recognizes a quasi-triangulated graph in $O(nm)$ time.

Key words. triangulated graphs, chordal graphs, quasi-triangulated graphs

AMS subject classifications. 05C75, 05C85

DOI. 10.1137/S0895480104444399

1. Introduction. In a graph G, a vertex x is *simplicial* if its neighborhood $N(x)$ induces a complete subgraph of G. A graph is *triangulated (chordal)* if it does not contain a chordless cycle of length at least four (a *hole*) as an induced subgraph. A famous theorem of Dirac [3] states that every triangulated graph has a simplicial vertex. Actually, Dirac proved more: every triangulated graph different from a clique contains two nonadjacent simplicial vertices. Let us say that a vertex is *cosimplicial* if its nonneighbors form an independent subset of vertices and that a graph is *cotriangulated* if it does not contain the complement of a chordless cycle on at least four vertices (an *antihole*). Dirac's theorem says equivalently that every cotriangulated graph has a cosimplicial vertex. Our purpose is to investigate the larger class of graphs which are called *quasi-triangulated* graphs (QT for short), defined as follows: a graph G is in class QT if and only if every induced subgraph H of G has a vertex which is either simplicial or cosimplicial in H. Quasi-triangulated graphs have been introduced by the third author in [9, 11] as a generalization of chordal graphs. The problem of characterizing the class QT was raised in [9] and, independently, in [7] (where they are called *good*). The reader is referred to [1] for more information on the class QT.

Following [7], we say that an order $v_1 < v_2 < \cdots < v_n$ on a graph G is *good* if, for any induced subgraph H of G, either the largest vertex of $(H, <)$ is simplicial or the smallest vertex of $(H, <)$ is cosimplicial. Good orders are perfect in the sense of [2].

Simplicial vertices cannot lie in a hole; and cosimplicial vertices cannot lie in an antihole. A graph with each vertex belonging to some hole and some antihole is called *latticed.*

The third author conjectured (unpublished) and the first author proved in [4, 5] the following.

*Received by the editors June 10, 2004; accepted for publication (in revised form) November 9, 2005; published electronically August 25, 2006.
http://www.siam.org/journals/sidma/20-3/44439.html

†Academy of Economic Studies of Moldova, 61 Banulescu-Bodoni str. MD-2005, Chisinau, Moldova.

‡Department of Physics and Computer Science, Wilfrid Laurier University, 75 University Ave. W., Waterloo, ON N2L 3C5, Canada (choang@wlu.ca). This author's research was supported by the NSERC.

§Department of Mathematics and Physics, Troy University, Troy, AL 36082 (vvoloshin@troy.edu). This author's research was partially supported by a Troy University research grant.

THEOREM 1. *For a graph G, the following three conditions are equivalent:*

(i) *G is quasi-triangulated.*

(ii) *G does not contain a latticed subgraph as an induced subgraph.*

(iii) *G admits a good order.*

As usual, n (respectively, m) denote the number of vertices (respectively, edges) of the input graph. For the quasi-triangulated graph recognition problem, the third author [10] proposed an $O(n^4)$ algorithm, Spinrad [12] proposed an $O(n^{2.77})$ algorithm, and the second author [6] independently proposed an $O(nm)$ algorithm.

THEOREM 2. *There is an $O(nm)$-time $O(n^2)$-space algorithm to recognize a quasi-triangulated graph.*

Theorems 1 and 2 are known by researchers in the field and have been referred to in the literature, but their proofs have never been published. The purpose of this paper is to provide the proofs of these two theorems.

2. Proof of Theorem 1. To prove Theorem 1, we will need the following lemma, which was included in the original proof in [4] and was rediscovered independently in [8].

LEMMA 1. *Let G be a graph and x be a vertex of G that does not lie in a hole. Then any minimal cutset C of G which is contained in the neighborhood $N(x)$ of x is a clique.*

Proof of Lemma 1. Define G, x, C as in the lemma. Let Y be a component of $G - C$ that does not contain x. We may assume that there are nonadjacent vertices u, v in C, for otherwise we are done. Since C is a minimal cutset, each of u and v has a neighbor in Y. It follows that there is a chordless path of length at least two joining u to v whose interior vertices lie in Y. This path together with x forms a hole, a contradiction to our assumption on x. □

Proof of Theorem 1. It is easy to see that (i) and (iii) are equivalent, and (i) implies (ii). So, we need only to prove that (ii) implies (i). We shall prove this by induction on the number of vertices. Let G be a graph satisfying (ii). We may assume G contains no simplicial vertex and no cosimplicial vertex, for otherwise we are done by the induction hypothesis. If G is disconnected, then each component of G contains a hole (for otherwise, it is triangulated and contains a simplicial vertex that remains simplicial in G); thus, G contains the union of two disjoint holes, a contradiction to (ii). So, G must be connected.

By replacing G by its complement $\overline{G}$ if necessary, we may assume that G contains a vertex that does not lie in a hole.

Define $X = \{x \mid x \text{ does not lie in a hole of } G\}$.

Our assumption on G implies that $X \neq \emptyset$. Let $G' = G - X$. G' is nonempty, for otherwise G is triangulated and thus contains a simplicial vertex by Dirac's theorem. By the induction hypothesis, G' contains a simplicial or cosimplicial vertex y. Since every vertex of G' lies in a hole, y is cosimplicial. We shall prove that y is adjacent to all vertices of X (this will imply y is cosimplicial in G, a contradiction).

Let x be a vertex in X. Since G is connected and x is not cosimplicial (by assumption), there is a nonempty set C of vertices in $N(x)$ that is a minimal cutset of G. By Lemma 1, C is a clique. Let G_1, G_2 be induced subgraphs of G such that $G = G_1 \cup G_2, G_1 \cap G_2 = C$, and there is no edge between $G_1 - C$ and $G_2 - C$.

Suppose G_1 is triangulated. We claim that there is a simplicial vertex s in $G_1 - C$. If G_1 is a clique, then the claim obviously holds; otherwise, by Dirac's theorem, G_1 contains two nonadjacent simplicial vertices, one of which must lie in $G_1 - C$ since C

is a clique. But s remains a simplicial vertex of G, a contradiction to our assumption on G. Thus G_1, and similarly G_2, cannot be triangulated.

Therefore, G_1 contains a hole. Since C is a clique, one edge, say e_1, of this hole lies completely in $G_1 - C$. Similarly, there is an edge, say e_2, that lies completely in $G_2 - C$ and belongs to a hole. Since y is cosimplicial in G' and all endpoints of e_1, e_2 are in G', y must be in C, and therefore adjacent to x, as desired. □

3. A recognition algorithm for quasi-triangulated graphs. In this section, we prove Theorem 2 by describing an algorithm that recognizes a quasi-triangulated graph in $O(nm)$ time using $O(n^2)$ space.

For a vertex x, an S-*obstruction* is a triple (a, b, x) that induces a P_3 with x being the interior vertex of the path; a C-*obstruction* is a triple (a, b, x) that induces an S-obstruction (a, b, x) in the complement.

A straightforward algorithm to recognize quasi-triangulated graphs proceeds as follows. First, for all vertices x, list all S- and C-obstructions. Then find a vertex y with no S- or C-obstructions; if no such vertex exists, then the graph is not quasi-triangulated. Remove y and update the lists of obstructions for the remaining vertices. Repeat this process to eliminate all vertices. If all vertices can be eliminated in this way, then the graph is quasi-triangulated; otherwise, it is not.

Since a vertex has $O(n^2)$ obstructions, we will need a data structure to store $O(n^3)$ obstructions of the graph. Thus the algorithm runs in $O(n^3)$ time using $O(n^3)$ space. We are going to show that the algorithm can be refined to run in time $O(nm)$ using $O(n^2)$ space.

Proof of Theorem 2. We may suppose there is a total order $<$ on the vertices of a given graph G. We say that (a, b, x) is less than (c, d, x), denoted by $(a, b, x) < (c, d, x)$, if $a < c$, or $a = c$ and $b < d$. To achieve the $O(nm)$ time bound, we will list only the smallest S-obstruction and C-obstruction for each vertex. When removing a vertex y, if a vertex x loses an obstruction, then we will find a smallest obstruction for x in the remaining graph. We shall show that, over the life of the algorithm, the time needed to list the (currently) smallest obstructions for a vertex x is $O(m)$. The outline of our algorithm is as follows.

Outline of algorithm. We begin with the input graph G and proceed to eliminate vertices one by one using the following steps.

1. For each vertex x of graph G, list a smallest S-obstruction (a, b, x) and a smallest C-obstruction (g, d, x).

2. If every remaining vertex has an S-obstruction and a C-obstruction, then G is not quasi-triangulated.

3. If a vertex z has no S-obstruction or no C-obstruction, eliminate z from G, and for each remaining vertex x that loses an S-obstruction (respectively, C-obstruction), generate a new smallest S-obstruction (respectively, C-obstruction). Replace G by $G - z$, and repeat step 2.

The graph G is quasi-triangulated if and only if recursive applications of step 3 eliminate all vertices. To anticipate, our algorithm lists the S-obstructions in $O(nm)$ time using $O(n+m)$ space and the C-obstructions in $O(nm)$ time using $O(n^2)$ space.

Let $N(x)$ be the adjacency list of vertex x. Without loss of generality, we may assume for all x that the lists $N(x)$ are sorted in increasing order.

Listing the smallest S-obstruction for a vertex *x*. For each vertex x, we use two pointers, $\alpha(x)$ and $\beta(x)$. Initially $\alpha(x)$ points to the first vertex α in $N(x)$ and $\beta(x)$ points to the immediate successor β of α in $N(x)$ (for simplicity, we let α (respectively, β) denote the name of the vertex pointed to by the pointer $\alpha(x)$

(respectively, $\beta(x)$). If $\alpha(x)$ or $\beta(x)$ cannot be initialized, then x has no S-obstruction. We simply advance $\beta(x)$ on $N(x)$ until we find that α and β are nonadjacent. When $\beta(x)$ reaches the end of $N(x)$ (i.e., it has value *null*), we advance $\alpha(x)$ in $N(x)$ and initialize $\beta(x)$ (making $\beta(x)$ point to the immediate successor of $\alpha(x)$ in $N(x)$). If $\alpha(x) = null$, then x has no S-obstruction, and a message "No S-obstruction" is produced. We can summarize this process as follows. (In the following procedure, the function IsEdge(a, b) returns true if and only if ab is an edge.)

PROCEDURE LISTSMALLEST-S-OBSTRUCTION(x).

```
while true
{   if α(x) = null
        then return "no S-obstruction for x"
    if IsEdge(α, β) = true
        then advance β(x) in N(x)
    else
        return (α, β, x)
    while ((α(x) ≠ null) and β(x) = null))
    {   advance α(x) in N(x)
        initialize β(x)
    }
}
```

Suppose we eliminate a vertex z and x loses its S-obstruction (a, b, x) (because $a = z$ or $b = z$). If b (respectively, a) is eliminated, then we advance $\beta(x)$ (respectively, $\alpha(x)$) and call Procedure ListSmallest-S-Obstruction(x) to get the smallest S-obstruction for x. The number of movements of the pointers $\alpha(x), \beta(x)$ in $N(x)$ is proportional to $O(n + m)$ since we advance $\beta(x)$ only in the presence of an edge, and $\alpha(x)$ is reset at most the degree of x times. If we have the incidence matrix of G at our disposal, then each call to Procedure IsEdge takes only constant time, but this method needs $O(n^2)$ space. We are going to show that for each vertex x we can implement Procedure IsEdge in $O(n + m)$ time using only the adjacency lists of G ($O(n + m)$ space).

Now we describe Procedure IsEdge(α, β), which returns true if and only if $\alpha\beta$ is an edge. For a vertex x, there is a pointer $p(x)$ which initially points to the first vertex in $N(\alpha)$ (recall that $\alpha(x)$ is the pointer associated with vertex x). If $p(x)$ cannot be initialized, then $\alpha\beta$ is not an edge. Pointer $p(x)$ is advanced in $N(\alpha)$ until it points to either (i) β ($\alpha\beta$ is an edge) or (ii) the smallest vertex in $N(\alpha)$ that is greater than β ($\alpha\beta$ is not an edge). The vertex pointed to by $p(x)$ is denoted by p.

PROCEDURE ISEDGE(α, β).

```
while true
{   if p(x) = null
        return false
    if p < β
        advance p(x) in N(α)
    else if p = β
        return true
    else if p > β
        return false
}
```

For each vertex x and each $\alpha(x)$, $p(x)$ scans $N(\alpha)$ only once. Thus, for each x, the cost of testing for all edges $\alpha\beta$ is $O(n + m)$.

Listing the smallest C-obstruction for a vertex x. Assume that the vertices are numbered $1, 2, \ldots, n$. For each vertex x, we maintain an integer variable counter $\gamma(x)$ that refers to the smallest nonneighbor of x. We need to generate the smallest C-obstruction of the form (γ, δ, x) for some vertex δ (that must be adjacent to γ and nonadjacent to x). This can be done as follows.

For each vertex x, we maintain a 0-1 characteristic vector $I(x)$ of size n to represent the neighborhood of x (the jth entry of $I(x)$ is 1 if and only if vertex j is a neighbor of x; in other words, $I(x)$ is the xth row of the incidence matrix of G). This is necessary so that testing of an edge of the form xy can be done in constant time. Given $\gamma(x)$, we find the smallest neighbor δ of γ (the vertex referred to by $\gamma(x)$) such that $\gamma < \delta$ and δ is nonadjacent to x by scanning the list $N(\gamma)$ and, for each vertex y in this list, testing whether yx is an edge. We use a pointer $\delta(x)$ to point to the location of δ in $N(\gamma)$. If $\delta(x)$ cannot be initialized, then there is no C-obstruction of the form (γ, δ, x); in this case, we increase $\gamma(x)$ by one and repeat the process (the initial value of $\gamma(x)$ is one). This is summarized in the following procedure (we leave the pointer initialization problem to the reader).

PROCEDURE LISTSMALLEST-C-OBSTRUCTION(x).

```
while true
{   if δ(x) = null
        repeat
            increase γ(x) by one
            if (γ > n)
                return "No C-obstruction for x"
            let δ(x) point to the first vertex in N(γ)
        until (xγ is not an edge) and δ(x) is not null
    if (xδ is not an edge) and (γ < δ)
        return the C-obstruction (γ, δ, x)
    advance δ(x) in N(γ)
}
```

Suppose we eliminate a vertex z and x loses its C-obstruction (g, d, x) (because $g = z$ or $d = z$). If d is eliminated, then we advance $\delta(x)$ in $N(\gamma(x))$ and call Procedure ListSmallest-C-Obstruction. If g is eliminated, then we repeatedly increase $\gamma(x)$ by one until we get the next smallest nonneighbor of x and call Procedure ListSmallest-C-Obstruction.

For each vertex x and each $\gamma(x)$, the list $N(\gamma(x))$ is scanned at most once. Thus, for each x, we can list the smallest C-obstruction in $O(n + m)$ time over the life of the algorithm. This method requires $O(n^2)$ space. □

In the case of listing the C-obstructions, we do not know how to implement our algorithm in $O(nm)$ time using linear space. We leave this as an open problem. We note Spinrad's algorithm [12] uses $O(n^2)$ space since it relies on matrix multiplications.

REFERENCES

[1] A. BRANDSTÄDT, V. B. LE, J. P. SPINRAD, *Graph Classes: A Survey*, SIAM Monogr. Discrete Math. Appl. 3, SIAM, Philadelphia, 1999.

[2] V. CHVÁTAL, *Perfectly ordered graphs*, in Topics on Perfect Graphs, C. Berge and V. Chvátal, eds., Ann. Discrete Math. 21, North–Holland, Amsterdam, 1984, pp. 63–65.

[3] G. A. DIRAC, *On rigid circuit graphs*, Abh. Math. Sem. Univ. Hamburg, 25 (1961), pp. 71–76.

[4] I. M. GORGOS, *A Characterization of Quasi-triangulated Graphs*, Preprint 11B494, Kishinev State University, Kishinev, Moldova, 1984 (in Russian).

[5] I. M. Gorgos, *Method of Alternating Chains and Its Applications*, Ph.D. Thesis, Kishinev State University, Kishinev, Moldova, 1985 (in Russian).
[6] C. T. Hoàng, *Recognizing Quasi-triangulated Graphs in $O(nm)$ Time*, manuscript.
[7] C. T. Hoàng and N. V. R. Mahadev, *A note on perfect orders*, Discrete Math., 74 (1989), pp. 77–84.
[8] C. T. Hoàng, S. Hougardy, F. Maffray, and N. V. R. Mahadev, *On simplicial and co-simplicial vertices in graphs*, Discrete Appl. Math., 138 (2004), pp. 117–132.
[9] V. I. Voloshin, *Quasi-triangulated Graphs*, Preprint 5569-81, Kishinev State University, Kishinev, Moldova, 1981 (in Russian).
[10] V. I. Voloshin, *Quasi-triangulated Graphs Recognition Program*, Algorithms and Programs P006124, Moscow, Russia, 1983 (in Russian).
[11] V. I. Voloshin, *Triangulated Graphs and Their Generalizations*, Ph.D. Thesis, Kishinev State University, Kishinev, Moldova, 1983 (in Russian).
[12] J. Spinrad, *Recognizing quasi-triangulated graphs*, Discrete Appl. Math., 138 (2004), pp. 203–213.