RC 20540 (9/9/96)
Computer Science/Mathematics

IBM Research Report

A Linear-Time Algorithm for Computing the Intersection of All Odd Cycles in a Graph

L. Cai

Dept. of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, New Territories, Hong Kong

B. Schieber

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

A Linear-Time Algorithm for Computing the Intersection of All Odd Cycles in a Graph

Leizhen Cai*   Baruch Schieber†

Abstract

We present a linear-time algorithm that finds all edges and vertices in the intersection of all odd cycles in a given graph. We also show an application of our algorithm to a variant of the satisfiability problem of Boolean formulas.

Key words: Graph algorithm, odd cycle, bipartite graph, satisfiability problem.

1 Introduction

We consider the problem of finding edges and vertices in a graph that are contained in every odd cycle of the graph. This problem is partially motivated by the following fixed-parameter version of the NP-complete problems BIPARTITE SUBGRAPH ([8], also [GT25] in [7]) and MAXIMUM CUT ([10], also [ND16] in [7]): *Does a given graph $G = (V, E)$ contain t edges, where t is a fixed integer, whose deletion from G results in a bipartite graph?* It is clear that this fixed-parameter problem for $t = 1$ is equivalent to the problem of determining if there is an edge in G that is contained in every odd cycle. (Recall that a graph is bipartite iff it contains no odd cycle.) We note that the above fixed-parameter problem can be easily solved in time $O(|E|^t(|E| + |V|))$ by exhaustive search; however, it is unknown whether the problem is *fixed parameter tractable*, i.e., can it be solved in $O(f(t)(|E| + |V|)^c)$ time for an arbitrary function $f(t)$ and an absolute constant c independent of the parameter t. Recently, there has been a growing interest in fixed-parameter tractability of various problems [3, 4, 6, 9].

In this paper, we present a linear-time algorithm that finds all edges and vertices in the intersection of all odd cycles in a given graph. Our algorithm can be used to reduce the time complexity of the fixed-parameter problem mentioned above to $O(|V|^{t-1}(|E| + |V|))$. We also give an application of our algorithm to a variant of the satisfiability problem of Boolean formulas.

*Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong. E-mail:lcai@cs.cuhk.edu.hk

†IBM Research Division, T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598, USA. E-mail:sbar@watson.ibm.com

The rest of the paper is organized as follows: We first fix definitions in Section 2. Then, in Section 3, we give a characterization of edges and vertices in the intersection of all odd cycles; and, in Section 4, we use the characterization to obtain a linear-time algorithm for computing the intersection. Finally, we show an application of our algorithm to a satisfiability problem in Section 5.

2 Definitions

We follow terminology in [5]. Graphs in this paper are simple, undirected, and connected graphs. Let $G = (V, E)$ be such a graph. For $v \in V$, let $G - v$ denote the subgraph of G induced by $V - \{v\}$; and for $e \in E$, let $G - e$ denote the partial subgraph of G obtained by removing edge e from G. A *path* P in G is a sequence $v_0, e_1, v_1, \ldots, e_k, v_k$ of distinct vertices and edges satisfying $e_i = v_{i-1}v_i$ for each $1 \leq i \leq k$. Each vertex but v_0 and v_k in P is an *internal vertex* of P. For convenience, we may also regard P as a graph or a set of vertices and edges. Let $V(P)$ and $E(P)$, respectively, denote the vertex and edge sets of P. A *cycle* is a path in which $v_0 = v_k$; it is an *odd cycle* if k is odd.

The intersection of all odd cycles in G is the set of vertices and edges that are contained in every odd cycle of G; any vertex (edge) in the intersection is called an *intersection vertex (edge)*.

Given a spanning tree T of G, the *T-span* of a non-tree edge e is the unique tree path in T between the endpoints of e.

A *k-colouring* of G is an assignment of k colours to the vertices of G such that each vertex receives a unique colour. Each set of vertices with the same colour forms a *colour class*. Given a k-colouring of G, an edge is *monochromatic* if its two endpoints have the same colour; otherwise, it is *bichromatic*. A k-colouring is *proper* if it does not produce any monochromatic edge. A graph G is a *bipartite graph* if its vertex set can be partitioned into two sets X and Y such that every edge of G has one end in X and the other end in Y; or equivalently, a graph is bipartite iff it admits a proper 2-colouring.

3 A characterization of the intersection

We start with some well known properties of or related to bipartite graphs.

Proposition 3.1 *The following statements hold for any graph $G = (V, E)$:*

1. *G is bipartite iff it contains no odd cycle.*
2. *G is bipartite iff for any pair of vertices every path between them has the same parity.*

3. *An edge $e \in E$ is an intersection edge iff G is not bipartite and $G - e$ is bipartite.*

4. *A vertex $v \in V$ is an intersection vertex iff G is not bipartite and $G - v$ is bipartite.*

In order to compute the intersection of all odd cycles in G, we first find an arbitrary spanning tree T of G and properly colour it in two colours. This proper 2-colouring also gives a 2-colouring of G. Consider edges of G under this 2-colouring of G. We see that every tree edge is bichromatic, but some non-tree edges may be monochromatic. Furthermore, every monochromatic edge and its T-span form an odd cycle of G. Then, intersection edges are either monochromatic edges or contained in their corresponding T-spans. Similarly, intersection vertices are either incident with monochromatic edges or contained in their corresponding T-spans. See Figure 1 for an example. We summarize some useful observations in the following proposition.

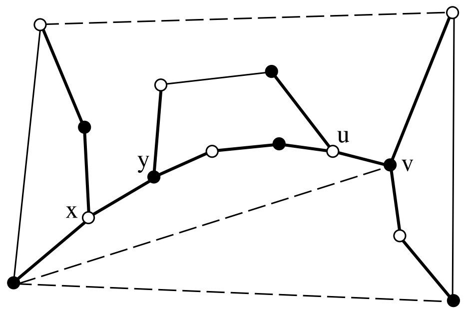

Figure 1: A spanning tree (thick lines) T of G with a proper 2-colouring of T. Dashed lines indicate monochromatic edges of G. Edges xy and uv are intersection edges, and vertices x, y, u, and v are intersection vertices.

Proposition 3.2 *Let T be a spanning tree of G. Properly colour T with two colours to form a 2-colouring of G. Let I be the intersection of the T-spans of all monochromatic edges. Then the following statements hold:*

1. *I is either empty or a path.*

2. *Every monochromatic edge and its T-span form an odd cycle of G.*

3. *The removal of all monochromatic edges from G results in a bipartite graph.*

4. *If G contains more than one monochromatic edge, then every intersection edge of G is contained in I.*

5. *Every intersection vertex of G is contained in I.*

It follows from statements (4) and (5) of Proposition 3.2 that to compute the intersection we need only consider edges and vertices in I. The following theorem characterizes the intersection edges and vertices in I.

Theorem 3.3 *Let T be a spanning tree of G. Properly colour T with two colours to form a 2-colouring of G. Let I be the intersection of the T-spans of all monochromatic edges, and G' be the partial subgraph of G obtained by removing all monochromatic edges. Then the following statements hold:*

1. *An edge of I is an intersection edge of G iff it is a cut edge of G'.*
2. *An internal vertex of I is an intersection vertex of G iff it is a cut vertex of G'.*

Proof. We first prove statement (1). Suppose that $e \in I$ is an intersection edge of G. Then, since I is contained in at least one odd cycle of G, there is a path P in $G - e$ between the endpoints of I such that the parity of P differs from that of I. (Note that I is a path.) Assume, to the contrary, that e is not a cut edge of G', then there is a path Q in $G' - e$ between the endpoints of I. Since G' is bipartite, I and Q have the same parity. Then, P and Q are two paths in $G - e$ with the same endpoints that have different parities. This implies that $G - e$ contains an odd cycle (Proposition 3.1), contradicting our assumption that e is an intersection edge of G. Therefore, e is a cut edge of G'.

Conversely, suppose that an edge $e \in I$ is a cut edge of G' and let X and Y be the two connected components of $G' - e$. Then, X and Y are bipartite graphs, and $G - e$ can be recovered from $G' - e$ by adding back all monochromatic edges of G. Since $e \in I$, every monochromatic edge of G has one endpoint in X and the other endpoint in Y. Therefore, we can obtain a proper 2-colouring of $G - e$ by interchanging the two colour classes in X but retaining the colour classes in Y. It follows that $G - e$ is bipartite and thus e is an intersection edge.

The proof of the second statement is similar. Suppose that $v \in I$ is an intersection vertex of G. Then, since v is not an endpoint of I, and I is contained in at least one odd cycle of G, there is a path P in $G - v$ between the endpoints of I such that the parity of P differs from that of I. Assume, to the contrary, that v is not a cut vertex of G', then there is a path Q in $G' - v$ between the endpoints of I. Since G' is bipartite, I and Q have the same parity. Then, P and Q are two paths in $G - v$ with the same endpoints that have different parities. This implies that $G - v$ contains an odd cycle, contradicting our assumption that v is an intersection vertex of G. Therefore, v is a cut vertex of G'.

Conversely, suppose that an internal vertex $v \in I$ is a cut vertex of G'. Then there are two connected components X and Y of $G' - v$ each of which contains an endpoint of I. Since $v \in I$, every monochromatic edge in $G - v$ has one endpoint in X and the other endpoint in Y. Furthermore, $G - v$ can be recovered from $G' - v$ by adding back these crossing monochromatic edges between X and Y. Note that each connected component of $G' - v$ is bipartite. Therefore, we can obtain a proper

2-colouring of $G - v$ by interchanging the two colour classes in X but retaining the colour classes in Y and all other connected components of $G' - v$. It follows that $G - v$ is bipartite and thus v is an intersection vertex. ∎

Remark An endpoint of I can be an intersection vertex even if it is not a cut vertex of G' (vertex x in Figure 2). On the other hand, an endpoint of I that is a cut vertex of G' may not be an intersection vertex (vertex y in Figure 2).

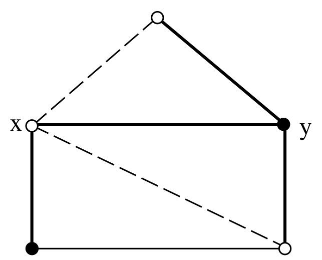

Figure 2: Endpoints of I vs. intersection vertices

4 A linear-time algorithm

Using the results in the previous section, we obtain the following algorithm for computing the intersection of all odd cycles in a graph.

Algorithm **Intersection** {Compute the intersection of all odd cycles}
Input: A simple, undirected, and connected graph G.
Output: The set I_e of intersection edges, and set I_v of intersection vertices of G.

1. Find a spanning tree T of G.
2. Properly colour T in two colours to obtain a 2-colouring of G.
3. Let M be the set of monochromatic edges in G and let $G' = G - M$.
4. Compute the intersection I of the T-spans of all monochromatic edges in M.
5. Find all cut vertices V_c and cut edges E_c of G'.
6. Compute $I_e = E(I) \cap E_c$ and $I_v = V(I) \cap V_c$.
7. If $|M| = 1$ then $I_e = I_e \cup M$.
8. For each endpoint x of I, if $G - x$ is bipartite then $I_v = I_v \cup \{x\}$ else $I_v = I_v - \{x\}$.
9. Return I_e and I_v.

Theorem 4.1 *Algorithm* **Intersection** *finds all intersection edges and vertices of G in linear time.*

Proof. The algorithm is clearly correct when G contains no monochromatic edge, since G is bipartite in this case. If G contains only one monochromatic edge e, then e is an intersection edge since $G - e$ is bipartite. Furthermore, any other intersection edge of G must be in the T-span of e, which equals I in this case since the only odd cycle of G consists of edge e and its T-span. If G contains at least two monochromatic edges, then any intersection edge of G belongs to I (Proposition 3.2). From this and Theorem 3.3, it is clear that I_e is the set of intersection edges of G.

For the intersection vertices part, we note that any intersection vertex is contained in I (Proposition 3.2). By Theorem 3.3, it is clear that Step 6 and Step 8 together correctly compute the set I_v of intersection vertices.

As far as linear time is concerned, we notice that all but Step 4 can be carried out in linear time by standard techniques [1]. We now give a linear-time implementation of Step 4. By statement (1) of Proposition 3.2, I is either empty or a path. Therefore it suffices to find the two endpoints of I. We do this as follows. Pick up an arbitrary vertex in V, and root T at this vertex. Arbitrarily order the edges in M as $e_0 = (x_0, y_0), \ldots, e_k$, where $|M| = k + 1$. Initialize I to be the path in T from x_0 to y_0 by designating x_0 and y_0, respectively, to be the left and right endpoints of I. To find the left endpoint of I, we do the following for $i = 1, \ldots, k$: Starting from the left endpoint of I, find the first vertex l_i on I that is an ancestor of either x_i or y_i (relative to the root vertex). If no such vertex exists then the intersection is empty. Otherwise, move the left endpoint of I to l_i. It is easy to see that after the k iterations l_k is the left endpoint of I. We can find the right endpoint of I in a similar manner. Note that given a preorder listing and a postorder listing [2] of T it can be determined in constant time whether one vertex is an ancestor of another. (Note that *x is an ancestor of y iff preorder(x) $\leq$ preorder(y) and postorder(x) $\geq$ postorder(y).*) Since a preorder (postorder) listing of T can be computed in linear time and I can be obtained easily from its endpoints in linear time, Step 4 takes linear time. ∎

Remark We can use algorithm **Intersection** to solve the fixed-parameter problem mentioned in the introduction in $O(|V|^{t-1}(|E| + |V|))$ time as follows: Use algorithm **Intersection** for $t = 1$. For any $t > 1$, we find an odd cycle O in G and then for each edge in O recursively solve the problem for $t - 1$.

5 An application to a satisfiability problem

We now consider an application of algorithm **Intersection** to a satisfiability problem of Boolean formulas. Let U be a finite set of Boolean variables, and c be a clause that contains exactly two distinct literals over U. Then, given a truth assignment for U, c is *mono-valued* if its two literals have the same value under the truth assignment. Consider the following problem:

MINIMUM MONO-VALUED 2SAT (MM2SAT)

INSTANCE: Finite set U of Boolean variables, collection C of clauses over U such that each clause in C contains exactly two distinct literals over U, positive integer $K \leq |C|$.

QUESTION: Is there a truth assignment for U that makes at most K clauses in C mono-valued?

It is not surprising that the above problem is intractable.

Theorem 5.1 *The MINIMUM MONO-VALUED 2SAT is NP-complete.*

Proof. The problem is clearly in NP. We transform NOT-ALL-EQUAL-3SAT (N3SAT) ([11], also [LO3] in [7]) to MM2SAT. For an arbitrary instance (U, C) of N3SAT, we construct an instance (U', C', K) of MM2SAT as follows: set $U' = U$, $K = |C|$, and construct C' from C by replacing each clause $\{x, y, z\} \in C$ with three clauses $\{x, y\}, \{y, z\}$, and $\{z, x\}$. It is easy to see that for any truth assignment for U, $\{x, y, z\}$ has at least one true literal and at least one false literal iff exactly one of $\{x, y\}, \{y, z\}$, and $\{z, x\}$ is a mono-valued clause, and $\{x, y, z\}$ has three true literals or three false literals iff $\{x, y\}, \{y, z\}$, and $\{z, x\}$ are all mono-valued clauses. It follows that (U, C) is a "Yes" instance of N3SAT iff (U', C', K) is a "Yes" instance of MM2SAT. Since the transformation is polynomial, MM2SAT is NP-complete. ∎

On the other hand, we can reduce MM2SAT to a graph problem. Given an arbitrary instance (U, C, K) of MM2SAT, we construct a graph G, called the *literal-clause graph* of (U, C), as follows:

1. Each literal over U corresponds to a vertex of G.
2. There are two types of edges in G:
   (a) *literal edge*: for each variable in U, there is an edge joining the two vertices corresponding to the two literals of the same variable.
   (b) *clause edge*: for each clause in C, there is an edge joining the two vertices corresponding to the two literals in the clause.

Then, there is a one-to-one correspondence between solutions to MM2SAT and clause edges in G whose deletion makes G bipartite.

Proposition 5.2 *Let (U, C, K) be an arbitrary instance of MM2SAT. Then there is a truth assignment for U that makes at most K clauses in C mono-valued iff there are at most K clause edges in the literal-clause graph G of (U, C) whose deletion from G results in a bipartite graph.*

Proof. We only need to note that there is a one-to-one correspondence between a truth assignment ξ for U and a 2-colouring for the literal-clause graph G of (U, C) in which each literal edge is bichromatic. Therefore mono-valued clauses under ξ correspond to monochromatic edges in G, and the claim follows. ∎

It is now straightforward to use algorithm **Intersection** to solve MM2SAT. We first construct the literal-clause graph G of (U, C). For $K = 1$, we use algorithm

Intersection to find all intersection edges I_e of G, and return "Yes" if I_e contains a clause edge of G; otherwise return "No". For $K > 1$ we use recursion with $K = 1$ as our base case: Find an odd cycle O in G and then for each clause edge in O recursively solve the problem for $K - 1$. This yields an $O((2|U|)^{K-1}(|C| + |U|))$ time algorithm which is faster than exhaustive search.

Acknowledgement

The first author was partially supported by an Earmarked Grant from the Research Grants Council of Hong Kong and a Direct Grant for Research from the Chinese University of Hong Kong.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, Reading, Mass., 1983.

[3] L. Cai, Fixed-Parameter Tractability of Graph Modification Problems for Hereditary Properties, *Information Processing Letters*, Vol. 58 No. 4 (1996) 171-176.

[4] H.L. Bodlaender, On disjoint cycles, *International J. of Foundations of Computer Science,* Vol. 5 No. 1 (1994) 69-97.

[5] J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, North-Holland, New York, 1976.

[6] R.G. Downey and M.R. Fellows, Fixed-Parameter Tractability and Completeness I: Basic Results, *SIAM J. on Computing,* Vol. 24 (1995) 873-921.

[7] M.R. Garey and D.S. Johnson, *Computers and Intractability : A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, 1979.

[8] M.R. Garey, D.S. Johnson and L. Stockmeyer, Some simplified NP-complete graph problems, *Theoretical Computer Scienec,* Vol. 1 (1976) 237-267.

[9] H. Kaplan, R. Shamir, and R.E. Tarjan, Tractability of parameterized completion problems on chordal and interval graphs: minimum fill-in and physical mapping, *Proc. 35th IEEE Symp. on Foundations of Computer Science*, 780-791, 1994, Sante Fe, New Mexico, USA.

[10] R.M. Karp, Reducibility among combinatorial problems, in *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher, eds., Plenum Press, (1972), 85-103.

[11] T.J. Schaefer, The complexity of satisfiability problems, *Proc. 10th ACM Symp. on Theory of Computing*, 216-226, 1978.