ON DISJOINT PATHS PROBLEMS IN GRAPHS WITH BOUNDED TREE WIDTH
(Extended Version of TR#585)

by

E. Korach, N. Solel and A. Tal

Technical Report #601
January 1990
On Disjoint Paths Problems in Graphs with Bounded Tree Width

E. Korach, N. Solel and A. Tal

Department of Computer Science
Technion - Israel Institute of Technology
Haifa 32000, Israel

Abstract

The vertex disjoint connecting paths problem is defined as follows: Given an undirected graph \( G = (V,E) \) and \( k \) pairs of vertices of \( V \) - (source,sink), find whether \( G \) contains \( k \) internally vertex disjoint paths, one connecting each pair. Karp has shown that this problem is NP-complete (even when all sources and sinks are distinct), and Robertson and Seymour solved this problem where the number of paths is constant. For graphs with bounded tree-width we solve in polynomial time a version of this problem: \( k \) is a part of the input and only the number of sinks in a vertex is bounded. We also solve some more problems related to the disjoint connecting paths problem on graphs with bounded tree-width.

1. Introduction

The vertex disjoint connecting paths problem (VDP) is defined as follows: given an undirected graph \( G = (V,E) \) and \( k \) pairs of vertices \( (s_i,t_i) \) \( 1 \leq i \leq k \), find whether there exist \( k \) internally vertex disjoint paths one connecting \( s_i \) (the source) and \( t_i \) (the sink) for each \( i \), \( 1 \leq i \leq k \). This problem is NP-complete and in fact Karp [K] has shown that a simpler version of this problem, where all the sources and sinks are disjoint, is NP-complete. We solve this problem polynomially when the number of sinks assigned to a vertex is bounded by a constant \( C \), on graphs with bounded tree-width (problem VDP-C). Independently in [MT] problem VDP, where the demand graph is simple, was solved by a polynomial algorithm using a different method on graphs with bounded tree-width. We also solve some related problems, for example - the edge disjoint connecting paths problem for graphs with bounded tree-width and bounded degree. Apart from the theoretical interest, these problems also have practical aspects in VLSI layout design, routing, network communication and other areas.

In [ALS,RSI] it was shown that solving the vertex disjoint connecting paths problem, where \( k \) - the number of paths - is constant, can be solved in polynomial time when restricted to graphs with bounded
tree-width. Robertson and Seymour [RS3] extended this result to all graphs (but \(k\) remains fixed). In [KT] problem VDP was solved on series parallel graphs in linear time.

The tree-width of a graph is defined as follows [RS1]:

Let \(G=(V,E)\) be a graph. A tree decomposition of \(G\) is a pair \((\{X_i: i \in I\}, T=(I,F))\), where \(\{X_i: i \in I\}\) is a family of subsets of \(V\), \(T=(I,F)\) is a tree with the following properties:

1. \(\bigcup_{i \in I} X_i = V\).
2. Every edge of \(G\) has both its ends in some \(X_i (i \in I)\).
3. For all \(i,j,k \in I\), if \(j\) lies on the path of \(T\) from \(i\) to \(k\), then \(X_i \cap X_k \subseteq X_j\).

The tree-width of a tree decomposition is \(\max_{i \in I} |X_i| - 1\). The tree-width of \(G\), denoted by \(tw(G)\), is the minimum tree-width of a tree decomposition of \(G\), taken over all possible tree decompositions of \(G\).

The class of graphs with tree-width \(\leq k\) is equal to the class of partial-\(k\)-trees [ACP] (and also to \(k\)-decomposable graphs [ACP]). The importance of this class is that many graph families have been proven to have bounded tree-width e.g. trees and forests have tree-width \(\leq 1\), series-parallel graphs have tree-width \(\leq 2\), halin graphs, \(k\)-outerplanar graphs and others have bounded tree-width (e.g. see [B2]).

Deciding the tree-width of a graph is NP-complete, however for a given constant \(k\) it was shown in [ACP,RS1] that deciding whether a graph \(G\) has tree-width \(\leq k\) and that finding a tree-decomposition with width \(\leq k\) (if such exists) can be solved polynomially.

Many NP-hard problems such as independent set, dominating set, graph coloring, clique and hamiltonian circuit, have been shown to have polynomial algorithms when restricted to graphs with bounded tree-width [A,ALS,AP].

Bodlaender in [B1] introduced two large classes of graph decision problems called ECC (edge condition composition) and LCC (local condition composition) and has shown that these problems can be solved polynomially when restricted to graphs with bounded tree-width (and also bounded degree for LCC). This was done by using non-serial dynamic programming methods, which combine local solutions to a global one. He has proven that many NP-complete problems are members of ECC or LCC and therefore have polynomial solutions when restricted to graphs with bounded tree-width (and bounded degree for LCC).

For example vertex-cover, independent-set, hamiltonian-path and many more problems listed in [B1].

In this paper we use Bodlaender's method to solve polynomially a restricted version of VDP and extend this to problem VDP-C and to some other related problems, on graphs with bounded tree-width.

Consider the problem VDP-C: if \(C\) is a variable \((1 \leq C \leq n, n\) the number of vertices in the input graph) then any instance \(\Pi\) of VDP has a minimal \(C_\Pi\) such that \(\Pi\) \(\in\) VDP-C. For any instance \(\Pi\) of VDP we show how to compute \(C_\Pi\), and how to transform \(\Pi\) to an instance of VDP-C if \(C_\Pi\) in polynomial time. This, together with the fact that the complexity of VDP-C for graphs with bounded tree-width is polynomial when \(C\) is...
constant gives a better explicit upper bound for solving \( \Pi \) (using our method).

2. Definition of the problems

In this paper we assume all graphs are simple (i.e. with no loops nor multiple edges).

**Problem (2.1):** Vertex disjoint connecting paths with shared sources (Variant of [GJ], problem ND40).

*Instance:* An undirected graph \( G = (V, E) \), \( k \) pairs of vertices of \( G \) \( (s_i, t_i) \), \( 1 \leq i \leq k \), for each \( i \neq j \), \( s_i \neq t_j \), \( t_i \neq t_j \), and for each \( i \), \( s_i \neq t_i \).

*Question:* Are there \( k \) internally-vertex disjoint paths in \( G \) one connecting \( (s_i, t_i) \) for each \( i \), \( 1 \leq i \leq k \)?

**Problem (2.1.1):** Vertex disjoint connecting paths with at most \( C \) (\( C \) is constant) sinks in (i.e. assigned to) a vertex (VDP-C).

*Instance:* An undirected graph \( G = (V, E) \), collection of \( k \) pairs of vertices of \( G \) \( (s_i, t_i) \), \( 1 \leq i \leq k \), for each vertex \( v \), \( \{i : t_i = v\} \leq C \).

*Question:* Same as the question in problem (2.1).

**Problem (2.2):** Edge disjoint connecting paths.

*Instance:* An undirected graph, \( G = (V, E) \), \( k \) pairs of vertices of \( G \) \( (s_i, t_i) \), \( 1 \leq i \leq k \).

*Question:* Are there \( k \) edge-disjoint paths of \( G \) one connecting \( (s_i, t_i) \) for each \( i \), \( 1 \leq i \leq k \)?

**Problem (2.3):** Maximum length bounded vertex disjoint connecting paths.

*Instance:* Same as in problem (2.1), and an integer \( l \).

*Question:* Are there \( k \) internally vertex disjoint paths between \( s_i \) and \( t_i \), \( 1 \leq i \leq k \) each involving \( l \) vertices, or less?

**Problem (2.4):** Disjoint connecting subgraphs [RS2].

*Instance:* An undirected graph \( G = (V, E) \), a set of \( k \) pairwise disjoint sets of vertices \( \Delta = \{\delta_1, \delta_2, \ldots, \delta_k\} \) (\( \delta_i \) is a set of vertices of \( G \)).

*Question:* Is there a subgraph of \( G \) which is a forest of \( k \) connected components \( T_i \), \( 1 \leq i \leq k \), and each component \( T_i \) includes all vertices of \( \delta_i \) and none of the vertices of \( \delta_j \), \( j \neq i \)?

**Problem (2.5):** Disjoint strongly connecting subgraphs (natural analog to problem (2.4) for directed graphs).

*Instance:* Same instance of problem (2.4), but \( G \) is a directed graph.

*Question:* Same question of problem (2.4), and each component \( T_i \) is strongly connected.
Problem (2.6): The \( J \) fixed length \( s-t \) disjoint paths (NP-C variant of [GJ]_problem ND42).

**Instance:** An undirected graph \( G=(V,E) \) specified vertices \( s \) and \( t \), positive integers \( J \leq |V| \), \( 1 < k \leq |V| \).

**Question:** Does \( G \) contain \( J \) mutually vertex disjoint paths from \( s \) to \( t \), each involving exactly \( k \) edges?

For the above problems let \( S = s_1,s_2,...,s_k \), \( T = t_1,t_2,...,t_k \), \( S(v) = \{i: s_i=v\} \), \( T(v) = \{i: t_i=v\} \) be the list of sources, the list of sinks, the set of sources in \( v \) and the set of sinks in \( v \) respectively.

**Definition:** A \( \delta_v \)-**vertex multiplication** of a vertex \( v \) in an undirected graph \( G=(V,E) \) is the process of replacing \( v \) by \( \delta_v \) vertices \( v_1,...,v_{\delta_v} \), and replacing the edges \((u,v)\in G\) with the edges \((u,v_1),...,(u,v_{\delta_v})\).

Similarly we define \( \delta_v \)-**vertex multiplication** of a vertex \( v \) in a directed graph, where each incoming edge \((u,v)\) is replaced by the edges \((u,v_1),...,u,v_{\delta_v})\) and each outgoing edge \((v,u)\) is replaced by \((v,v_1),...,v,v_{\delta_v})\).

Let \( \delta \in \mathbb{Z}_+^n \) be a vector describing the size of vertex-multiplication for each vertex \( v \) in a graph \( G=(V,E) \), \( G \) directed or undirected \(|V|=n \), \( V=\{v_1,...,v_n\} \). A \( \delta \)-**multiplicated graph** \( G^* \) of \( G \) is a graph obtained as follows: consider the following sequence of graphs \( G_0,G_1,...,G_n \).

\( G_0=G \).

\( G_i \) is the result of \( \delta_v \)-vertex multiplication of \( G_{i-1}, v_i \in V \).

\( G^* = G_n \).

Denote \( \Delta(\delta) = \max \{\delta_v\} \) and let \( U'(v) = \{v_1,...,v_{\delta_v}\} \).

Note that the graph \( G^* \) is independent of the order of performing the vertex multiplications (i.e., reordering of \( V \)).

3. The main results

**Lemma (3.1):** Let \( G=(V,E) \) be a graph and \( \{X_i: i \in I\}, T=(I,F) \) a tree decomposition of \( G \) with \( tw(G) \leq k \).

Then, a \( \delta \)-multiplicated graph \( G^* \) of \( G \) has tree-width \( \leq \Delta(\delta) k \).

**Proof:** A tree decomposition \( \{X'_i: i \in I'\}, T'=(I',F') \) of \( G^* \) can be obtained as follows:

Let \( T' = T \cup \forall i \in I' : X'_i = \cup \{U'(v): v \in X_i \} \).

Clearly from the above process we obtain a tree decomposition of \( G^* \), and for each \( i \in I' \):

\[
|X'_i| = \sum_{v \in U'(v)} |X_i| \leq |X_i| \cdot \Delta(\delta) \leq k \cdot \Delta(\delta).
\]

Therefore \( G^* \) has a tree decomposition with \( tw(G^*) \leq \Delta(\delta) k \). \( \Box \)

**Theorem (3.2):** Problem (2.1) \( \in \) ECC.
Proof: Transform to the following problem:

**Instance:** The instance of problem (2.1), sets \( X = \{0,1,\ldots,k\}^* V^* \{0,1,\ldots,|V|-1\} \), \( Y = \{0\} \).

**Question:** Are there functions \( f : V \rightarrow X \), \( g : E \rightarrow Y \) such that:

1. \( \forall v \in V \setminus S \ f_2(v) \text{ is a neighbor of } v \), or \( f_3(v) = v \).
2. \( \forall v \in S \ f_3(v) = 0 \).
3. \( \forall i \in T \ f_3(i_i) = i \).
4. \( \forall v, w \in V \ v \in S \) or \( v \) is a neighbor of \( w \) if \( f_3(w) = v \) and \( f_3(w) > 0 \) then \( f_3(w) = f_3(v) + 1 \).
5. \( \forall v, w \in V \ v \in S \) or \( v \) is a neighbor of \( w \) if \( f_3(w) = v \) and \( f_3(w) > 0 \) then
   - if \( v \in S \) then \( f_3(w) = f_3(v) \)
   - else \( f_3(w) = i \), where \( i \in S(v) \).

The correspondence between a solution to the problem and the function \( f \) can be seen from the following observation:

- \( f_1(v) \) partitions \( V \setminus S \) to \( k+1 \) disjoint sets, and if \( f_1(v) = j \), \( 1 \leq j \leq k \), \( v \in V \setminus S \), then there is a path \( \pi(v) = (u_0, u_1, \ldots, u_l) \) connecting \( v \) to \( v \), whose length is \( f_3(v) \), \( u_0 = s_j \), \( u_l = v \), \( l = f_3(v) \), \( u_{i-1} = f_2(u_i) \), \( 1 \leq i \leq l \), and \( f_3(u_i) = j \), \( 1 \leq i \leq l \). 

For the above problem, we give an informal description of the algorithm, which is based on Bodlaender's algorithm, slightly simplified for our purpose. In a similar way, we can modify Bodlaender's algorithm so that it will solve the other problems in this paper shown to be members in LCC or ECC. The polynomial complexity of these problems is proven by showing their membership in one of the above classes.

Let \( G \) be the input graph with tree-width \( \leq w \). Let \( (X_i : i \in I, T = (I, F)) \) be a tree decomposition of \( G \) with-width \( \leq w \) and with \( \sum |I| = |V| - w \) (for the existence see [B1]). Arbitrarily choose one vertex in \( T \) to be the root, and direct the tree (to get a rooted tree). For each \( i \in I \) calculate recursively a Table for all sons of \( i \), calculate the Table of \( i \) (as explained in (1) below), and combine the Table of \( i \) and of it's sons (as explained in (2) below). Each row in the combined Table of \( i \) and all it's sons represents a legal solution that can be extended to the subgraph induced by branch \( (i) \) (branch \( (i) \) is the subgraph of \( T \) including \( i \) and all it's descendents. The subgraph induced by branch \( (i) \) is the subgraph of \( G \) induced by the vertices \( v \in X_j \), where \( j \in \text{branch} \ (i) \), but \( v \notin X_k \), \( k \) the father of \( i \) in \( T \) where the father of the root is the empty set). At the end of the above process a solution exists if and only if the Table of the root of \( T \) is not empty (i.e. contains a row which can be extended to the whole graph).

(1) Calculation of Table of \( X_i \):

- \( X = \{0,1,\ldots,k\}^* V^* \{0,1,\ldots,|V|-1\} \). \( X_i = \{u_1,\ldots, u_{m} \} \) \( m \leq w + 1 \). The Table of \( X_i \) has \( m \) columns and each entry in a column is a triplet as shown below. Write all the functions \( f : X_i \rightarrow X \) in the Table such that each row consists of one function, and has the following form:
  
  \[
  (p_1, p_2, j_1) \quad \ldots \quad (p_1, p_2, j_s) \quad \ldots \quad (p_m, p_m, j_m)
  \]
\[ p_i \in \{0,1,\ldots,k\} \rightarrow \text{intend to represent the path of } u_i \ (0 \text{ if } u_i \text{ does not participate in any of the paths}). \]

\[ n_i \in V \rightarrow \text{intend to represent the predecessor of } u_i \text{ in the path.} \]

\[ l_i \in \{0,1,\ldots,W-1\} \rightarrow \text{intend to represent the place of } u_i \ (\text{length of the path } p_i \text{ from the source to } u_i). \]

Initially, the **Table** includes candidates for partial solutions in \( X_i \), including rows which are illegal i.e rows that do not satisfy conditions 1-5 in the proof of theorem (3.2). Then rows that are found illegal are omitted from the **Table**.

(2) Combining the **Table** of \( X_i \) and \( X_j \) where \( j \) is the father of \( i \) in \( T \):

Suppose \( X_i \cap X_j = U \). For each row in the **Table** of \( X_j \) find whether there is a row in the **Table** of \( X_i \) with the same function \( f \) on the vertices of \( U \). If no such row exists (in the **Table** of \( X_j \)) delete the row from the **Table** of \( X_j \). The rows left in the **Table** of \( X_j \) can be extended to a legal partial solution in the graph induced by the branch of \( i \) and the branch including \( j \) and the previously combined sons of \( j \) and their descendents. A legal solution in the final **Table** of the root can be extended to a complete legal solution in the input graph.

The method we use enables us to add some measurement on the size of the problem. For example suppose in problem (2.1) we want to add an upper bound \( M \) on the number of internal vertices participating in all the paths. We can do so by adding the following condition in the proof of theorem (3.2):

(6) The number of vertices \( v \in V \setminus (S \cup T) \) having \( f_i(v)>0 \) is \( \leq M \).

Using Bodlaender's method we can add to each **Table** another column which for each row in the table will measure the minimum number of vertices implied by that row (denote the size of the solution). When calculating the initial **Table** of \( X_i \), in each row one can write that size by counting the number of vertices \( v \in X_i, v \in X_j, (k \text{ the father of } i \text{ in the tree decomposition}) \) having \( f_i(v)>0 \). When combining **Table** of \( j \) and **Table** of \( i \), a son of \( j \) in the tree decomposition, the size of each legal row left in the **Table** of \( j \) will be the sum of the size in \( j \) and the minimum size of a row in the **Table** of \( X_i \) having the same function \( f \) on the vertices of \( U = X_j \cap X_j \). Finally at the end of the algorithm there is a row in the **Table** of the root of \( T \) that it's size is \( \leq M \) if and only if there is a solution to the problem with no more then \( M \) internal vertices in all the paths.

Another possible extension (using the same technique) is the following: Suppose there is a cost function \( l \) on the edges of \( G, l: E \rightarrow \mathbb{R}^+ \). We can ask for a limit on the total cost of the edges used in the solution to the problem. Such a constraint could be expressed by changing the set \( Y \) to \( \{0,1\} \) and adding the following conditions in the proof of theorem (3.2):

(6) \[ \forall v, w \in V, \text{ if } v \text{ is a neighbor of } w \text{ and } f_i(v)>0 \text{ and } f_j(v)=w \text{ then } g((v,w))=1, \text{ otherwise } g((v,w))=0. \]

(7) \[ \sum_{e \in E} g(e)l(e) \leq M. \]
Theorem (3.3): Problem (2.1.1) has a polynomial algorithm when restricted to graphs with tree-width bounded by a constant.

Proof: Given a graph \( G=(V,E) \) with \( \text{tw}(G) \leq k \), constant, and \( k \) pairs as defined in problem (2.1.1) (Denote the instance \( (G,S,T) \)), transform to the following instance \( (G^*,S^*,T^*) \) (without loss of generality we assume that \( \forall i, 1 \leq i \leq k, s_i \neq t_i \), otherwise reduce the problem in the natural way): \( G^* \) is a \( \delta \)-multiplicated graph of \( G \), where \( \delta \) is defined as follows:

1. \( \forall v \in V \setminus T, \delta_v = 1. \)
2. \( \forall v \in T \setminus S, \delta_v = |T(v)|. \)
3. \( \forall v \in T \cap S, \delta_v = |T(v)|+1. \)

For every vertex \( v \) satisfying (2) assign a distinct terminal from \( T(v) \) to \( v^*_i \) for \( i = 1...|T(v)| \), and for a vertex \( v \) satisfying (3) assign \( T(v) \) as in (2) and assign \( s \) to \( v^*_i \) where \( s = |T(v)|+1. \)

From Lemma 1 the graph \( G^* \) has tree-width \( \leq \Delta(\delta) \cdot \text{tw}(G) \). But \( \Delta(\delta) \leq C+1, \) and \( \text{tw}(G) \leq k \), therefore \( \text{tw}(G^*) \) is also bounded by a constant. Note that the size of \( G^* \) is linear in the size of \( G \). Moreover, \( G^* \) with the new assignment of \( (s_i,t_i), 1 \leq i \leq k \) is an instance of problem (2.1) and therefore by Theorem (3.2) \( (G^*,S^*,T^*) \in \text{ECC} \), and has polynomial solution, when \( G^* \) has a constant bounded tree-width.

It is easy to see that an instance \((G,S,T)\) of problem (2.1.1) has a solution if and only if the \( \delta \)-multiplicated graph with the new assignments \( (G^*,S^*,T^*) \) of problem (2.1) has a solution. \( \square \)

In the above theorem we assume that given a pair of terminals \((s_i,t_i)\) the first terminal is the source, and the second one is the sink. Since the problem and the solution assume undirected graph, it is acceptable to interchange the role of source and sink in each pair, i.e., if \( v = s_i \) and \( u = t_i \), one can assign \( s'_i \) to \( u \), and \( t'_i \) to \( v \) and then replace \((s_i,t_i)\) by \((s'_i,t'_i)\). This can be done on any collection of pairs from \((s_i,t_i), 1 \leq i \leq k \), in order to get new lists of vertices, \( S', T' \).

Given an instance of problem VDP one can make new assignments to some of the pairs \((s_i,t_i)\) as described above, such that for each vertex \( v \in V \), \(|T(v)| \) is bounded by a constant \( C \), (as required in the input of the problem VDP-C), if such assignment exists. Otherwise report that there is no such feasible assignment to sources and sinks. This can be done polynomially (as shown below), and also one can find the smallest \( C \), for which there is an assignment of terminals such that \( \forall v \in V, |T(v)| \leq C \), and therefore reduce the run time of the algorithm proposed here which is dependent on \(|T(v)| \). This is a direct consequence of theorem (3.4) below.

Let \( H=(V,F) \) be the demand graph where \( F=\{f_i:(s_i,t_i)\mid 1 \leq i \leq k\} \), and each edge \( f_i=(s_i,t_i) \in F \) is directed from \( s_i \) to \( t_i \). We call the tail of \( f_i \) the source and the head of \( f_i \) the sink; therefore any reorientation of \( F \) corresponds to reassignment of sources and sinks.
The \( l \)-bounded indegree reorientation problem: Let \( H=(V,F) \) be a directed graph and let \( l \in \mathbb{N}^+ \). Find a reorientation of the edges of \( H \) such that the indegree of each \( v \in V \) is bounded by \( l_v \) (if possible), or decide that no such orientation exists. Clearly every solution to this problem corresponds to reassignment of sources and sinks such that the number of sinks in every vertex \( v \) is bounded by \( l_v \).

Theorem (3.4): Let \( H=(V,F) \) be a directed graph and \( l \in \mathbb{N}^+ \). There is a polynomial algorithm to the \( l \)-bounded indegree reorientation problem.

Proof: Let \( \text{in}(v) \) be the indegree of \( v \), and \( \forall v \in H \) denote \( w(v) = l_v - \text{in}(v) \). Let \( X = \{ v \in V : w(v) < 0 \} \), \( \bar{X} = V \setminus X \), i.e \( \bar{X} = \{ v \in V : w(v) \geq 0 \} \). The following algorithm solves the \( l \)-bounded indegree reorientation problem, for \( H \):

\[
\text{While } X \neq \emptyset \text{ do }
\begin{align*}
\text{Let } x \in X. \text{ Using a regular labeling method (e.g. BFS) find a directed path } P \text{ from a vertex } y \in \bar{X} \text{ with } w(y) > 0 \text{ to } x. \\
\text{If no such path exists - stop! (there is no legal orientation) } \\
\text{Else - reverse the orientation of all edges in } P, \text{ update } w(x), w(y).
\end{align*}
\]

One can see that the algorithm correctly solves the problem, and the proof is left to the reader. \( \square \)

From Theorem (3.4) it is straightforward that given an instance of problem VDP one can find polynomially a reassignment to \((s_t, t_i), 1 \leq i \leq k\), that satisfies the requirements of the input of problem VDP-C (if such assignment exists). Use the algorithm of Theorem (3.4), and let \( l \) be the vector where \( l_v = C, \forall v \in V \) where \( C \) is the constant required in the problem.

Note that in order to find the smallest possible value of \( C \) one can run the algorithm of Theorem (3.4) with the vector \( l \) for \( C = 1, \ldots, |V| \), and stop whenever the algorithm finds an appropriate reorientation.

Remarks: Our method has an interesting feature: by solving an auxiliary problem we obtain a better complexity. For any instance we find the best possible \( C \) and since the running time of the algorithm for VDP-C is exponential in \( C \) we may reduce it. In fact when restricted to VDP-C for a given constant \( C \) this auxiliary problem provides means to test whether an instance of the problem VDP can be transformed to an instance of VDP-C and consequently can be solved by the algorithm.

4. Related problems

Theorem (4.1): Problem (2.2) \( \in \) LCC.

Proof: Transform to the following problem.
Instance: The instance of problem (2.2), sets \(X = \{0\}, Y = \{0,1,\ldots,k\}^* \{0,1,\ldots,k\}^*\).

Question: Are there functions \(f: V \rightarrow X\), \(g: E \rightarrow Y\) such that:

1. \(\forall e = (v,w) \in E\), \(g_2(e)\) is an edge adjacent to \(v\) or \(w\).
2. \(\forall e = (v,w) \in E\), if \(g_2(e) = i\) and \(g_3(e) = 0\), then \(v = s_i\) or \(w = s_i\).
3. \(\forall t_i \in T\), there is an incident edge \(e\) with \(g_1(e) = i\).
4. \(\forall e_1, e_2 = (u,v)(v,w) \in E\), if \(g_2(e_2) = e_1\) and \(g_3(e_2) > 0\) and \(g_3(e_2) > 0\) then \(g_3(e_2) = g_3(e_1) + 1\) and \(g_1(e_2) = g_1(e_1)\).

The relation between a solution and the function \(g_1\): The function \(g_1\) partitions \(E\) to \(k+1\) sets and each set \(i, 1 \leq i \leq k\) is a connected subgraph, containing \(s_i\) and \(t_i\), \(1 \leq i \leq k\).

Theorem (4.2): Problem (2.3) \(\in\) ECC.
Proof: Use the technique described in theorem (3.2), but let \(X = \{0,1,\ldots,k\}^* \{0,1,\ldots,k\}^*\), i.e., allow only paths of length \(l\), or less.

Note that by using the above method one can find the smallest integer \(l\) for which there is a solution, if such exists. This can be done by running the algorithm that checks if a solution exists at most \(lV_{l-1}\) times, where at each iteration use the set \(X = \{0,1,\ldots,k\}^* \{0,1,\ldots,k\}^*\), \(i = 1,\ldots, lV_{l-1}\). The first time the algorithm finds a solution (if such exists), gives the smallest possible \(l\).

All the above theorems can be modified with minor changes to show that the related problems on directed graphs are also polynomial when restricted to graphs with constant tree-width (and also constant degree for problem (2.2)).

Theorem (4.3): Problem (2.4) \(\in\) ECC.
Proof: Choose from each set \(\delta_i\), \(1 \leq i \leq k\) one vertex and assign \(s_i\) to it. To all other vertices in \(\delta_i\) assign \(t_i\). Now use the technique described in Theorem (3.2).

Theorem (4.4): Problem (2.5) has a polynomial algorithm when restricted to graphs with constant tree-width.
Proof: In a directed graph \(G=(V,E)\) \(u\) is a directed neighbor of \(v\) if \((v,u) \in E\) i.e there is a directed edge from \(v\) to \(u\).

First, choose from each set \(\delta_i\), \(1 \leq i \leq k\) one vertex and assign \(s_i\) to it. To all other vertices in \(\delta_i\) assign \(t_i\). Then transform to the following problem:

Instance: The instance of problem (2.5) and the assignment to \(s_i\) and \(t_i\) as described above, sets \(X = \{0,1,\ldots,k\}^* \{0,1,\ldots,k\}^* \{0,1,\ldots,k\}^* \{0,1,\ldots,k\}^*\), \(Y = \{0\}\).

Question: Are there functions \(f: V \rightarrow X\), \(g: E \rightarrow Y\) such that:
\(1\) \(\forall v \in V \setminus S \quad v\) is a directed neighbor of \(f_2(v)\), or \(f_2(v) = v\), and \(f_4(v)\) is a directed neighbor of \(v\) or \(f_4(v) = v\).

\(2\) \(\forall v \in S \quad f_2(v) = 0\) and \(f_3(v) = 0\).

\(3\) \(\forall t_i \in T, f_2(t_i) = i\) and \(\forall s_j \in S, f_1(s_j) = i\).

\(4\) \(\forall (v, w) \in E:\)
   a. If \(w \notin S\) and \(f_2(w) = v\) and \(f_1(w) > 0\) then \(f_2(w) = f_2(v) + 1\) and \(f_1(w) = f_1(v)\).
   b. If \( v \notin S\) and \(f_4(w) = v\) and \(f_1(w) > 0\) then \(f_4(w) = f_4(v) + 1\) and \(f_1(w) = f_1(w)\).

The proof idea is to check simultaneously, after partitioning the vertices of \(V\) to \(k+1\) disjoint sets (by the function \(f_1\)), if there exist directed paths from each \(S_i\) to the set \(\{t_i, t_i \in \delta_i\}\) for each \(i, 1 \leq i \leq k\), using only vertices from the \(i\)’th component (this is done by the functions \(f_2\) and \(f_3\)) and directed paths from all the vertices in \(\{t_i, t_i \in \delta_i\}\) to \(S_i\), for each \(i, 1 \leq i \leq k\), again, involving only vertices from the \(i\)’th component (this is done by the functions \(f_4\) and \(f_5\)).

**Theorem (4.5):** Problem (2.6) \(\in\) ECC.

**Proof:** Transform to the following problem:

**Instance:** The instance of problem (2.6), sets \(X = \{0, 1, \ldots, f, 0, 1, \ldots, k\}^*V\), \(Y = \{0, 1\}\).

**Question:** Are there functions \(f: V \rightarrow X, g: E \rightarrow Y\) such that:

\(1\) \(f_2(s) = 0\).

\(2\) \(f_2(t) = k\).

\(3\) \(\forall v \in V, f_3(v)\) is a neighbor of \(v\), or \(f_3(v) = v\).

\(4\) \(\forall v \in V, f_4(v)\) is a neighbor of \(v\), or \(f_4(v) = v\).

\(5\) \(\forall v, w \in V, v \neq s, v \neq t, v = w\) or \(v\) is a neighbor of \(w\), if \(f_3(w) = w\) and \(f_1(w) > 0\), then \((f_2(v) = f_2(w) - 1)\) and if \(w \neq t\) then \(f_2(w) = v\) and \(f_1(w) = f_1(v)\).

\(6\) \(\forall v, w \in V, w \neq s, w \neq t, v = w\) or \(v\) is a neighbor of \(w\), if \(f_4(w) = w\) and \(f_1(w) > 0\) then \((f_2(w) = f_2(v) + 1)\) and if \(v \neq s\) then \(f_2(w) = v\) and \(f_1(w) = f_1(v)\).

\(7\) For every edge \(e\) with ends \(v\) and \(w\) that satisfies one of the following conditions:
   a. \(v \neq s, w \neq t, w \neq s, w \neq t, f_3(v) = w, f_4(w) = v, f_1(v) = f_1(w) > 0,\) or
   b. \(v = s, w \neq t, f_3(w) = v, f_4(w) = 1, f_1(w) > 0,\) or
   c. \(v \neq s, w = t, f_3(v) = w, f_4[w) = k - 1, f_1(w) > 0,\)
   then \(g(e) = 1,\) for all other edges \(g(e) = 0.\)

\(8\) \(\sum_{e \in E} g(e) = k J.\)

\(\square\)
An intuitive explanation to the functions $f$ and $g$:

$f_1(v)$ is the index of the path that contains vertex $v$ (if $f_1(v) > 0$).

$f_3(v)$ is the neighboring vertex after $v$ on the path, and $f_4(v)$ is the neighboring vertex before $v$.

$f_2(v)$ gives the relative place of $v$ in the path starting from vertex $s$ with 0 and terminating at vertex $t$ with $k$. Therefore each path has exactly $k$ edges.

$g(e) = 1$ for all edges participating in a path, and otherwise $g(e) = 0$.

Consider the variant of problem (2.6) (denote problem (2.6a)) that we obtain by changing the requirement that each path contains exactly $k$ edges, to the requirement that each path has at most $k$ edges. This variant is NP-complete (see [GJ], problem ND41). Given an instance to problem (2.6a) one can polynomially transform the instance to an instance of problem (2.6) without increasing the tree-width of the input graph such that there is a solution to problem (2.6a) if and only if there is a solution to problem (2.6). Therefore problem (2.6a) can be solved polynomially on graphs with constant tree-width. We leave the details for the reader.

Recently it was shown in [LMS] that the following problem is strongly NP-complete: Given a directed or undirected graph $G=(V,E)$ and two vertices $s,t$ find two internally vertex disjoint paths between $s$ and $t$, such that the length of the longer path is minimized. We have solved this problem polynomially on graphs with bounded tree-width, where weights are given in unary (i.e we have solved this problem by a pseudo polynomial algorithm on graphs with bounded tree-width). Two other related problems are the directed and the undirected two-commodity integral flow, both are NP-complete (see [GJ], problems ND38 and ND39). We have shown that these problems, and some variants of them, have pseudo polynomial algorithms on graphs with bounded tree-width and bounded degree. The details for all these problems will appear elsewhere.

Acknowledgment

This research was partially supported by Technion V.P.R. Fund - Coleman Cohen Research Fund.
References


