ON DISJOINT PATHS PROBLEMS IN GRAPHS WITH
BOUNDED TREE WIDTH

by

E. Korach, N. Solel and A. Tal

Technical Report #585

August 1989
On Disjoint Paths Problems in Graphs with Bounded Tree Width

(Extended Abstract)

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 $(s_i, t_i)$, 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 actually Karp [K] has shown that a simpler version of this problem, where all the sources and sinks are disjoint, is also 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). We also solve some related problems, for example Edge disjoint connecting paths (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, RS] 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 definition of the tree width of a graph [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 $\text{tw}(G)$ 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 (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 (for the input graph). 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 $\Pi \in \text{VDP-C}_{\Pi}$. For any instance $\Pi$ of VDP we show how to compute $C_{\Pi}$ and how to transform $\Pi$ to an instance of VDP-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 (exponential in C), gives a better explicit upper bound for solving II (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, ND 40].

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

Question: Are there k internally-vertex disjoint paths in G one connecting (s_i, t_i) for each i, 1 ≤ i ≤ 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 ≤ i ≤ k, for each vertex v, I{1(i: t_i = v)} ≤ 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 ≤ i ≤ k.

Question: Are there k edge-disjoint paths of G one connecting (s_i, t_i) for each i, 1 ≤ i ≤ 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 ≤ i ≤ 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 Δ = {δ_1, δ_2, ..., δ_k} (δ_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 ≤ i ≤ k, and each component T_i includes all vertices of δ_i and none of the vertices of δ_j, j ≠ 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, ND 42).

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 denote $S = s_1,s_2,...,s_k$, $T = t_1,t_2,...,t_k$, $S(v) = \{i : s_i=v\}$, $T(v) = \{i : t_i=v\}$ 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 $v_1^v,...,v_{\delta_v}^v$, and replacing the edges $(u,v) \in G$ with the edges $(u,v_1^v),..., (u,v_{\delta_v}^v)$.

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 $(u,v_1^v),..., (u,v_{\delta_v}^v)$ and each outgoing edge $(v,u)$ is replaced by $(v_1^v,u),..., (v_{\delta_v}^v,u)$.

Let $\delta \in \mathbb{Z}^*_+$ be a vector describing the size of vertex-multiplication for each vertex $v$ in a graph $G=(V,E)$, $|V| = n$, $V = \{v_1,...,v_n\}$. 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 $\Delta(\delta) = \max_{v \in V} (\delta_v)$ and let $U'(v) = \{v_1^v,...,v_{\delta_v}^v\}$.

Note that the graph $G^*$ is independent of the order of performing vertex multiplication (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 $\text{tw}(G) \leq k$.

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

Proof: A tree decomposition $((X_i^*:i \in I^*), T^* = (I^*,F^*))$ of $G^*$ can be obtained as follows:

Let $T^* = T$, $\forall i \in I^*$: $X_i^* = \cup \{U'(v) : v \in X_i\}$.

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

$|X_i^*| = \sum_{v \in X_i} |U'(v)| \leq |X_i| \cdot \Delta(\delta) \leq k \cdot \Delta(\delta)$.

Therefore $G^*$ has a tree decomposition with $\text{tw}(G^*) \leq \Delta(\delta) \cdot k$. □

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

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

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

1. \( \forall v \in \mathcal{V} \setminus \mathcal{S}, \ f_2(v) \) is a neighbor of \( v \), or \( f_2(v) = v \).
2. \( \forall v \in S, \ f_3(v) = 0 \).
3. \( \forall t \in T, \ f_4(t) = i \).
4. \( \forall v, w \in V, \ w \in S, \text{ if } f_2(w) = v \text{ and } f_1(w) > 0 \text{ then } f_3(w) = f_3(v) + 1 \).
5. \( \forall v, w \in V, \ w \in S, \text{ if } f_2(w) = v \text{ and } f_1(w) > 0 \text{ then } \)
   
   - if \( v \in S \) then \( f_1(w) = f_1(v) \)
   - else \( f_1(w) = i \), 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 \( \mathcal{V} \setminus \mathcal{S} \) to \( k+1 \) disjoint sets, and if \( f_1(v) = j, \ 1 \leq j \leq k, \ v \in \mathcal{V} \setminus \mathcal{S} \), then there is a path \( \pi(v) = (u_0, u_1, \ldots, u_l) \) connecting \( s_j \) to \( t_j \), whose length is \( f_2(v), \ u_0 = s_j \), \( u_l = v \), \( l = f_3(v) \), \( u_{i-1} = f_2(u_i), \ 1 \leq i \leq l \), and \( f_1(u_i) = j, \ 1 \leq i \leq l \).

**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 \( tw(G) \leq k_1 \), \( k_1 \) 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. \( \forall v \in \mathcal{V} \setminus \mathcal{S}, \ \delta_v = 1 \).
2. \( \forall v \in \mathcal{T} \setminus \mathcal{S}, \ \delta_v = |T(v)| \).
3. \( \forall v \in T \cap \mathcal{S}, \ \delta_v = |T(v)|+1 \).

For vertices satisfying 2) assign \( t_i \) to \( v_i^* \) for \( i = 1,...,|T(v)| \), and for vertices satisfying 3) assign \( T(v) \) as in 2) and assign \( S(v) \) to \( v_s \), where \( s = |T(v)|+1 \).

From Lemma 1 the graph \( G^* \) has tree width \( \leq \Delta(\delta) \cdot tw(G) \). But \( \Delta(\delta) \leq C+1 \), and \( tw(G) \leq k_1 \), therefore \( tw(G^*) \) 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) \((G^*, S^*, T^*) \in ECC \), 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) has a solution iff the \(I\)-multiplication graph with the new assignments \((G^*,S^*,T^*)\) of problem (2.2) has a solution. \(\Box\)

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 set 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, 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 \(F=\{(s_i,t_i) \mid 1 \leq i \leq k\}\), 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.

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

**Proof:** One can construct a polynomial algorithm based on network flow methods. \(\Box\)

From Theorem (3.4) it is straightforward that given an instance of problem VDP one can find polynomially a reassignment to \((S;\gamma)\), \(1 \leq i \leq k\), 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 \(l_v=|T(v)|\) for \(\forall v \in V\) where \(C\) is the constant required in the problem. In order to find the smallest possible value of \(C\) 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.

**Remark:** 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 is exponential in \(C\) we
reduce the running time). 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 $X = \{0\}$.

$$Y = \{0,1,\ldots,k\}^{*}E^{*}\{0,1,\ldots|E|-1\}$$

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 to $w$.
2. $\forall e = (v,w) \in E$, if $g_1(v,w) = i$ and $g_3(v,w) = 0$, 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 $g_3(e_2) = e_1$ and $g_1(e_2) > 0$ 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 relations between a solution and the function $g_1$: The function $g_1$ partitions $E$ to $k+1$ sets and each set $S_i$, $1 \leq i \leq k$ 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 $X = \{0,1,\ldots,k\}^{*}V^{*}\{0,1,\ldots|V|-1\}$, 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 $|V| - 1$ times, where at each iteration use the set $X = \{0,1,\ldots,k\}^{*}V^{*}\{1,\ldots,i\}$, $i = 1,\ldots,|V| - 1$. The first time the algorithm finds a solution (if such exists), gives the smallest possible $l$.

All the above proven 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 constant degree for problem (2.2)).

Theorem (4.3): Problem (2.4) $\in$ ECC.

Proof: Choose from each set $S_i$, $1 \leq i \leq k$ one vertex and assign $s_i$ to it. To all other vertices in $S_i$ 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: Use the same technique as in the proof of theorem (4.3). First on the original graph \( G=(V,E) \), i.e. check if there exists a directed path from each \( s_i \), \( 1 \leq i \leq k \) to the set \( \{ t_i : t_i \in \delta_i \} \). Then change the direction of all the edges in \( G \), and check if a solution exists in the reversed graph \( G'=\{V,E'\} \), \( E' = \{(u,v):(v,u) \in E\} \). If there is a path from each \( s_i \), \( 1 \leq i \leq k \) to \( \{ t_i : t_i \in \delta_i \} \) in the reversed graph then there is a path from the set of vertices \( \{ t_i : t_i \in \delta_i \} \) to \( s_i \) in the original graph \( G \).

There is a solution to problem (2.5) iff both in \( G \), and \( G' \), the algorithm finds a solution. \( \square \)

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

Proof: Transform to the following problem:

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

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

1. \( f_1(s) = 0. \)
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. \( \forall v,w \in V, v \neq s, v \neq t, \) if \( f_3(v) = w \) and \( f_1(v) > 0 \), then \( f_2(v) = f_2(w) - 1 \) and if \( w \neq t \) then \( f_4(w) = v \) and \( f_1(v) = f_1(w) \).
6. \( \forall v,w \in V, w \neq s, w \neq t, \) if \( f_4(w) = v \) and \( f_1(w) > 0 \) then \( f_2(w) = f_2(v) + 1 \) and if \( v \neq s \) then \( f_3(v) = w \) 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. \( v = s, w \neq t, f_4(w) = v, f_2(w) = 1, f_1(w) > 0 \), or
   c. \( v \neq s, w = t, f_3(v) = w, f_2(v) = k-1, f_1(w) > 0 \),
   then \( g(e) = 1 \), for all other edges \( g(e) = 0. \)
8. \( \sum_{e \in E} g(e) = k \cdot J. \) \( \square \)

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 \( g(e) = 0. \)
Acknowledgment

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

References


Technion - Computer Science Department - Technical Report CS0585 - 1989