GENERAL VERTEX DISJOINT PATHS IN SERIES-PARALLEL GRAPHS

by

E. Korach and A. Tal

Technical Report #559
May 1989
General Vertex Disjoint Paths in Series-Parallel Graphs

Ephraim Korach and Ady Tal

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

Abstract
Let $G=(V,E)$ be an undirected graph and let $(s_i,t_i) \ 1 \leq i \leq k$ be $k$ pairs of vertices in $G$. The vertex disjoint paths problem is to find $k$ paths $P_1, \ldots, P_k$ such that $P_i$ connects $s_i$ and $t_i$ and any two of these paths may intersect only at a common end point. This problem is NP-complete even for planar graphs. Robertson and Seymour proved that when $k$ is a fixed integer this problem becomes polynomial. We present a linear time algorithm for solving the general problem when the input graph is a series-parallel graph.

1. Introduction
The vertex disjoint paths problem (VDPP) is the following:
given an undirected graph $G=(V,E)$ and a set of pairs of terminals $\mathcal{O} = \{(s_i,t_i) : s_i,t_i \in V, 1 \leq i \leq k\}$ find whether there exist $k$ pairwise internally vertex disjoint paths $P_1, \ldots, P_k$ in $G$ such that $P_i$ connects $s_i$ and $t_i$. The pair $(G,\mathcal{O})$ is called a terminated graph [RS2].

In this paper we do not restrict all terminals to be distinct and we only require that the paths in the solution be internally vertex disjoint. This problem appears in practical problems like routing and VLSI layout design. Karp [K] showed that this problem is NP-complete and Lynch [L] proved that it remains NP-complete even if the input graph is planar. Some restricted versions of this problem have been solved polynomially: In [Se,Sh,T] a polynomial solutions are given for the case $k=2$, some other special cases where $k$ is a fixed integer are given in [RS1,RS2] and lately Robertson and Seymour solved this problem where the only restriction is that $k$ is any fixed integer [RS3]. Although their algorithm is a low degree polynomial, the constants are very large and unpractical even for small $k$'s. In [RS4] they provide a polynomial algorithm for the VDPP when $G$ is planar and $k$ is a variable part of the input, but all the terminals lie on the boundary of one or two faces of $G$.

In this paper, we present a practical linear algorithm for the VDPP where the given graph is restricted to be series-parallel graph (SPG) but $k$ is not restricted at all. In order to get the solution we introduce a technique of colouring edges in reductions which enable us to continue the series-parallel reductions while retaining information about the terminals for making decisions in a later stage. A preliminary results show that the constructive version of this problem is also linear time solvable. Since outer-planar graphs are subfamily of SPG our result is also a linear algorithm for these graphs. Note that a polynomial algorithm for
outer-planar graphs exists in [RS4] since in outer-planar graphs all the vertices lie on the outer face. Series-parallel graphs appears in many research studies and many NP-complete problem were shown to have polynomial, and even linear, algorithms when the problem is restricted to SPG (see e.g. [J,SY,TNS]). It is known that SPG has treewidth≤2 [RS2] and many NP-complete problems were shown to have polynomial algorithm when the graph is restricted to have a fixed tree width (e.g. [A,ALS,B]). In particular VDPP for a fixed k in graphs with bounded tree width was shown to be polynomial ([ALS,RS2]), but these do not include VDPP where k in not restricted.

2. Preliminaries

We assume that the reader is familiar with the standard graph theoretical definitions (see e.g. [BM]). We assume that all the graphs here are finite, they may have multiple edges and loops.

Definition 2.1. A series and a parallel constructions in a graph \( G=(V,E) \) are defined in the following way:

(i). A series construction Replace an edge \( e=(u,v) \in E \) by two edges in series: \((u,w) , (w,v)\) where \( w \) is a new vertex.

(ii). A parallel construction Add a new edge \( e'=(u,v) \) to \( E \) where \((u,v) \in E \) and \( e' \notin E \).

Definition 2.2. A 2-connected series-parallel graph (2SPG) is defined recursively as follows:

(i). An edge is a 2SPG.

(ii). The graph \( G=((u,v) , (e_1=(u,v),e_2=(u,v))) \) where \( e_1\neq e_2 \) is a 2SPG.

(iii). If \( G \) is a 2SPG with at least 2 edges, then \( G' \), the graph constructed from \( G \) by a series or a parallel construction, is a 2SPG.

Definition 2.3. \( G \) is a series-parallel graph (SPG) if every 2-connected component of \( G \) is a 2SPG.

Definition 2.4. Let \( e=(u,v) \) be an edge in \( G=(V,E) \). By "contraction of \( e \)" we mean removing \( e \) from \( E \), replacing \( u \) and \( v \) by a new vertex say \( w \) and replace all edges \((x,y)\) s.t. \( x \in \{u,v\} \), by a new edge \((w,y)\) if \( y \notin \{u,v\} \) or else by a loop \((w,w)\).

Definition 2.5. A graph \( H \) is a minor of \( G \) if \( H \) is isomorphic to the result of a series of deletions and contractions of edges in a subgraph of \( G \).

3.1 SPG Properties

Theorem 3.1.1. (Duffin [D]) \( G \) is a SPG if and only if \( G \) has no subgraph homeomorphic to \( K_4 \). □

Corollary 3.1.2. \( G \) is a SPG if and only if \( G \) does not have a minor isomorphic to \( K_4 \).
Proof. It is well known that $G$ has a minor $K_4$ if and only if it has a subgraph homeomorphic to $K_4$. □

Since the minor operation is transitive, we have the following corollary:

Corollary 3.1.3. The SPG class is closed under taking minors. □

Theorem 3.1.4. Let $G$ be a 2SPG then

(i). If $|V| \leq 3$ then every vertex has at most 2 neighbours other than itself.

(ii). If $|V| > 3$ then there are at least 2 non-adjacent vertices such that each one of them has exactly 2 neighbours. □

Proof

(i). Trivial.

(ii). The proof of this part is by induction on $|V| + |E|$. Note that the two graphs in Figure 1 are the only two simple 2SPG with 4 vertices. It follows that (ii) is true for all 2SPG with $|V| = 4$ and also for all 2SPG with $|V| + |E| \leq 9$.

![Figure 1](image)

Assume that the induction hypothesis is true for $|V| + |E| \leq k$. Let $G$ be a 2SPG, if $G$ was obtained from a 2SPG by a parallel construction then by the induction hypothesis (ii) holds also for $G$. Assume that $G$ was obtained from $G'$ by a series construction using the edge $e = (u, v)$ in $G'$, adding the new vertex $w$. By the induction, $G'$ had 2 non adjacent vertices $x$ and $y$ each with exactly 2 neighbours. Clearly $(x,y) \neq (u,v)$. In $G$ the set $(w,x,y) \backslash (u,v)$ has the following properties: there are at least 2 vertices, each vertex has exactly 2 neighbours and the vertices are pairwise non adjacent. This complete the proof of (ii). □

Corollary 3.1.5. Let $G$ be a SPG with $|V| \geq 2$ then there are at least 2 vertices with 2 or less neighbours.

Proof: $G$ is a SPG so every 2-connected component of $G$ is 2SPG. If $G$ is itself 2-connected then this follows from theorem 3.1.4. Otherwise, it is well known that all 2-connected components of a graph can be represented in a tree-like super structure, hence there are at least two 2-connected components that each one of them is connected to the rest of the graph by exactly one vertex (the leaves of the super structure). Therefore by theorem 3.1.4 each one of these components has at least one vertex, which is not in any other 2-connected component and has at most 2 neighbours in the whole graph. □
3.2 The algorithm

Based on corollaries 3.1.3 and 3.1.5, we can describe the following algorithm:

Given a terminated graph \( (G, \mathcal{C}) \) where \( G=(V,E) \) is a SPG and \( \mathcal{C}=(\langle s_i,t_i \rangle : s_i,t_i \in V, 1 \leq i \leq k) \). In the initial stage we delete all the parallel edges and all loops in \( G \), and any pair of terminals that either located in the same vertex or connected by a parallel edge. All the remaining edges left are coloured white. We use the idea of colouring edges in order to keep the underline structure of the graph as a SPG and to be able to progress with the reductions. The colour of an edge gives information on the terminals "contained" in it (This will be clear from the reductions). In the iterative stage, we find a vertex with at most 2 neighbours, reduce \((G, \mathcal{C})\) according to the set of rules described in the tables. If parallel edges are created, we reduce them too. If any reduction indicates "no solution" then the reduced graph has no solution and therefore the original one has no solution. Otherwise, the obtained graph has a solution if and only if the original graph has one. Repeat this until we have a graph with 2 vertices, or an indication that there is no solution.

Data structure:

We use the following data structure: a table of vertices \( TV \), a table of edges \( TE \), a table of terminals \( IT \), 0-degree stack, 1-degree stack and 2-degree stack of vertices, and parallel edges stack.

For each vertex \( u \) in \( TV \): keep a table of neighbouring vertices \( v \) such that each entry represents the number of edges \((u,v)\), this table is handled by a method that has an \( O(1) \) initiating feature. A list of incident edges such that each entry has the edge name. If vertex \( u \) contains terminals then point to a terminal in the terminals table (and keep all \( u \)'s terminals in a bi-directional list in \( TT \)) also keep variables for the degree of \( u \), the number of terminals it contains and if it is deleted or not.

For each edge \( e \) in \( TE \): keep \( e \)'s ends, its colour, the terminals it represents (at most 2 terminals) and if it is deleted or not.

For each terminal in \( IT \): keep pointers for the bi-directional linked list of terminals, the name of the vertex/edge it is contained in, and if it has been connected to its matching terminal or not.

Description of the algorithm:

1. Initially, all the edges are coloured white. Build the data structure with a search algorithm based on Depth First Search (performed in each connected component in the input graph or which is created during the search). During the search
   (i). If a vertex contains some matching pairs of terminals then if it contains also a terminal without its matching one, remove all the matching pairs and leave the other terminals in it. If it contains only matching terminals then delete the vertex.
   (ii). If an edge \( e=(u,v) \) is parallel to an edge \( e=(u',v') \) then delete \( e_1 \) and if \( u \) contains a terminal \( a \) and \( v \) contains its matching terminal \( b \), then remove \( a \) and \( b \) from \( u \) and \( v \) respectively (only one pair of terminals) and if \( u \) (\( v \)) has no terminals left in it delete it.
   (iii). Remove loops.

At the end \( G \) contains neither loops nor parallel edges, and if the number of pairs of terminals > \( 1|E| \) then there is no solution. Finally scan the vertex table and collect all vertices of degree \( i \) (\( i=0,1,2 \)) to the appropriate stacks and set parallel stack to be empty.
2.1. While parallel stack is not empty call Reduce-Parallel\((u,v,e_1,e_2)\).
2.2. While 1-degree stack is not empty call Reduce-1-Degree\((u,e)\).
2.3. If 2-degree stack is not empty then call Reduce-2-Degree\((u,v,w,e_1,e_2)\).
2.4. For each vertex v in 0-degree stack reduce v in the natural way.
3. If the reduction indicates “no solution” then stop, answer NO.
4. If \(|V| \leq 2\) then solve directly and stop, if solution was found answer YES, else answer NO.
5. Goto step 2.1.

The following procedures are the reduction rules.

Reduce-Parallel\((u,v,e_1,e_2)\)  \((u\ and\ v\ are\ the\ ends\ of\ e_1,e_2)\)
1. Reduce-Colour-Terminal\(e_1\).
2. Reduce-Colour-Terminal\(e_2\).
3. If \(e_1\ and\ e_2\ were\ not\ reduced\ (in\ 1\ or\ in\ 2)\ then\ find\ the\ appropriate\ reduction\ according\ to\ table\ 1\ and\ reduce\ the\ graph.\)
4. If “no solution” was found return NO SOLUTION.
5. Update the data structure.

Reduce-1-Degree\((u,e)\)  \((u\ has\ degree=1\ and\ e\ is\ its\ incident\ edge)\)
1. Reduce-Colour-Terminal\(e\).
2. If e was not reduced (in 1) then find the appropriate reduction according to table 2 and reduce the graph.
3. If “no solution” was found return NO SOLUTION.
4. Update the data structure.

Reduce-2-Degree\((u,v,w,e_1,e_2)\)  \((e_1=(u,v), e_2=(v,w)\ and\ v\ has\ degree=2)\)
1. Reduce-Colour-Terminal\(e_1\).
2. Reduce-Colour-Terminal\(e_2\).
3. If \(e_1\ and\ e_2\ were\ not\ reduced\ (in\ 1\ or\ in\ 2)\ then\ find\ the\ appropriate\ reduction\ according\ to\ table\ 3\ and\ reduce\ the\ graph.\)
4. If “no solution” was found return NO SOLUTION.
5. Update the data structure.

Reduce-Colour-Terminal\(e\)
If colour\(e\)=white and at least one of \(e\)'s ends has a terminal then find the appropriate reduction according to table 4.
Delete-Vertex($u$)

1. For each edge $e$ incident to $u$ do
   1.1. If colour($e$) = white then remove it and update the data structure.
   1.2. Else, find the appropriate reduction according to table 5.
2. Remove $u$.

3.2.1 Properties of the reduced graphs

(1). A blue edge $(u,w)$ represents two edges in series $(u,v)$ and $(v,w)$, where the common vertex $v$ contains only one terminal $a$, $u$ and $w$ contain no terminals. In this case the path connecting $a$ to its matching terminal must use either $u$ or $w$ (but the other vertex is free).

(2). A green edge represents the two following cases:
   (i). Two blue edges in parallel, each blue edge represents a terminal say, $a$ and $b$, and terminal $a$ does not match terminal $b$
   (ii). Two white edges in series and the common vertex $v$ contain two non-matching terminals say, $a$ and $b$.

Any solution in both cases uses one end vertex ($u$ or $w$) for connecting $a$ to its matching pair and the other vertex for connecting $b$.

(3). A red edge represents two blue edges in parallel, each blue edge represents a terminal say, $a$ and $b$, and terminal $a$ matches terminal $b$. Therefore in any solution the path from $a$ to $b$ uses either $u$ or $v$.

(4). During the whole algorithm
   (i). A non terminal vertex can receive at most one terminal.
   (ii). A terminal vertex can receive no more terminals.
(iii). When all the terminals in a vertex are removed it is deleted.

(5). When a coloured (non-white) edge is created its ends contains no terminal. During the algorithm, any end of a non-white edge may receive a terminal but then this edge will be reduced.

(6). During the whole algorithm (except for the initial stage) there are no 3 parallel edges.

(7). Parallel edges are reduced before series edges, therefore two parallel edges that one of their ends has degree=2, are reduced as parallel, therefore no loops are created.

(8). An edge can be coloured only in one of these cases: W → B , W → G , B → R , B → G.

Definition 3.3.1 A coloured VDPP is VDPP where G has coloured edges, this problem can be transfered to a VDPP on a non-coloured graph G' where any coloured edge in G is substitute by the appropriate subgraph according to 3.2.1 (this substitution is not dependent in the series of reductions which created the coloured edge). In this way |G'|=O(|GI|) where |GI| = |VI|+ |EI|.

Definition 3.3.2 If the edge (u,v) is contracted according to table 1 reductions 1.4 or 1.5 then we say that all the edges that are incident to u or v are influenced by this contraction.

The following lemma is needed for the linear complexity proof in theorem 3.3.4.

Lemma 3.3.3 An edge can be influenced by at most two contractions during the algorithm.

Proof: The contractions in case 1.4 and case 1.5 of table 1 take place when we have 2 parallel edges and both ends are non-terminal vertices. Any time an edge is influenced by one of its ends, this end becomes a terminal and therefore, by property 4 in 3.2.1 it will not be influenced by this end again. □

Theorem 3.3.4. The algorithm solves the VDPP in linear time.

Proof: The algorithm solves the VDPP since each reduced graph has a solution to the reduced coloured VDPP if and only if the original graph has one. The algorithm terminates in one of the two cases:

(i). "No solution" in one of the reduced graphs.

(ii). A small reduced graph in which a positive solution is clear.

The algorithm runs in linear time because:

1. In the first stage we construct the data structure and remove all the parallel edges, loops and pairs of terminals which are both in the same vertex or connected by a parallel edge. This takes a linear time in the size of the input. After this stage we have SPG with |EI|=O(|V|), the number of terminals ≤ |EI| (Otherwise there is no solution) therefore, the data structure contains O(1V) initiating entries. Note that we use an O(1) initiating technique for the arrays therefore although we have for each vertex an array of size 1V for its neighbours which sums to O(1V^2) we do not spend more than a
total of $O(|V|)$ time to initiate all of them.

2. Each reduction reduces the size of the graph by at least 1 (an edge or a vertex).

3. Each reduction except for the deletions and for the special contractions (table 1 case 1.4 and case 1.5) takes a constant time (including updating the data structure).

4. Any edge can be influenced by a special contraction (table 1 case 1.4 or case 1.5) at most twice (by lemma 3.3.3). Therefore, the time spent in this reduction is constant on each edge influence by it, this implies that the total time spent by these reductions during the whole algorithm on edges influenced by these reductions is $O(|E|)$.

5. Any edge or vertex can be deleted at most once during the whole algorithm therefore the total time spent on deletions is $O(|E| + |V|)$.

From 1.-5. above we have that the algorithm has a linear running time. □

3.4 Tables of reductions

The following reductions are grouped in 5 tables according to their structure.

Notations used in the tables:

- $\otimes$: non terminal vertex
- $\bigcirc$: terminal or non terminal vertex
- $\square$: vertex with exactly one terminal
- $\triangleleft$: a vertex with one terminal called a
- $\square\Box$: vertex with d terminals
- $\bullet$: deleted vertex (for table 6)
- $\longrightarrow$: an edge (coloured white)
- $\rightarrow$: an edge (coloured C ∈ \{B,R,G\})
- $\bigcirc\square$: local solution and “reduction” according to the explanations

$\otimes a = b$: terminal a matches terminal b (analogously for sets)
3.4.1 Table 1 - Reductions of parallel edges.

This subtable represents all possible combinations of two parallel edges. Note that entry 1 indicates also the case where the vertices contain terminals, all other entries are just for non terminal ends. In case that an end vertex contains a terminal any coloured edge was reduced according to table 4.

<table>
<thead>
<tr>
<th></th>
<th>W</th>
<th>W</th>
<th>W</th>
<th>W</th>
<th>B</th>
<th>B</th>
<th>B</th>
<th>R</th>
<th>R</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Edge f</td>
<td></td>
<td>B</td>
<td>R</td>
<td>G</td>
<td>B</td>
<td>R</td>
<td>G</td>
<td>R</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>Table entry</td>
<td>1.1</td>
<td>1.3</td>
<td>1.9</td>
<td>1.10</td>
<td>1.2</td>
<td>1.4</td>
<td>1.5</td>
<td>1.6</td>
<td>1.7</td>
<td>1.8</td>
</tr>
</tbody>
</table>

1.1) Delete edge e. If u and v contain a pair of matching terminals do the following: connect them, delete the pair from u and v and if u (v) contains no terminals delete u (v).

1.2) Edges e and f are blue so each one represents a terminal say b₁ and b₂ respectively, delete e, and if b₁ matches b₂ then change f's colour to red, else change f's colour to green.

1.3) Delete the white edge.

1.4) The blue edge represents a terminal say b and the red edge represents 2 matching terminals r₁ and r₂, any solution has one path connecting terminal b and another path connecting r₁ and r₂, if the first path uses vertex u then the second path must use vertex v and vice versa, so do the following: delete one edge, contract the other one, make a new vertex, rename it v, and put terminal b in it. (The solvability of the reduced problem is equivalent to the solvability of the resulted problem.)
The blue edge represents a terminal say $b$ and the green edge represents 2 non-matching terminals say $g_1$ and $g_2$, if $b$ matches $g_1$ or $g_2$ then this case is exactly like case 1.4, else $b$ does not match neither $g_1$ nor $g_2$ then no solution is possible.

Any red edge represents 2 matching terminals so each vertex is used in the solution of each pair so connect each pair using one vertex and delete it from the graph.

The red edge represents 2 matching terminals this means that in any solution they use $u$ or $v$ but the green edge represents 2 non-matching terminals so must use $u$ and $v$ therefore no solution is possible.

Any green edge represents 2 non-matching terminals, if the 2 terminals in one edge matches the 2 terminals in the other edge then it is the same as case 1.6 otherwise no solution is possible.

The red edge represents 2 matching terminals it means that any solution uses one of the vertices then no path can use the white edge, so delete it.

In this case the white edge can’t be used in any path so delete it.
3.4.2 Table 2 - Reductions of 1 degree vertices.

2.1) Delete vertex u.

2.2) Vertex u contains a terminal, delete u and put u’s terminal in vertex v.

2.3) Vertex v contains d-1 terminals and vertex u contains a terminal say a, if a matches one of v’s terminals then connect them, delete them from the graph and put the rest d-1 terminals in v, (if d-1=0 then delete v). If a does not match one of u’s terminals then no solution is possible.

2.4) Since d>1 no solution is possible.

2.5) The blue edge represents a terminal say b then delete vertex u from the graph and put b in vertex v.

2.6) The red edge represents 2 matching terminals so connect them using vertex u and delete u from the graph.

2.7) No solution is possible because the green edge represents 2 non-matching terminals.
3.4.3 Table 3 - Reductions of 2 degree vertices.

The following two subtables represent all possible combinations of two edges in series where all three vertices are distinct. \( d_x (x \in \{u,v,w\}) \) is the number of terminals in vertex \( x \). The first subtable is for two white edges in series and the second one is where at least one edge is coloured and \( d \) is the number of terminals at the end of the white edge which is not common to the coloured edge. In case that a coloured edge has a terminal end it was reduced by table 4.

<table>
<thead>
<tr>
<th>Reductions of two edges in series both edges are white</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_u )</td>
</tr>
<tr>
<td>( d_v )</td>
</tr>
<tr>
<td>( d_w )</td>
</tr>
<tr>
<td>Table entry</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reductions of two edges in series at least one is non white</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colours</td>
</tr>
<tr>
<td>( d )</td>
</tr>
<tr>
<td>Table entry</td>
</tr>
</tbody>
</table>

3.1) \( \begin{array}{c}
\circlearrowleft \\
\circlearrowright
\end{array} \)
Contract edge \( e \).

3.2) \( \begin{array}{c}
\circlearrowleft \\
\circlearrowright
\end{array}
\)
\( \begin{array}{c}
\circlearrowleft \\
\circlearrowright
\end{array} \)
Contract edge \( e \) and change the colour of edge \( f \) to blue.

3.3) \( \begin{array}{c}
\circlearrowleft \\
\circlearrowright
\end{array}
\)
\( \begin{array}{c}
\circlearrowleft \\
\circlearrowright
\end{array} \)
Vertex \( v \) has 2 non matching terminals, contract edge \( e \) and change the colour of edge \( f \) to green.
3.4) Vertex w contains $d \geq 1$ terminals, if one of them say b matches terminal a (in vertex v) then connect them, delete v, and if d=1 then delete w, else (d>1) put the rest d-1 terminals in w. If a does not match any one of w’s terminals then delete the edge (v,w).

3.5) Vertex v contains 2 terminals and vertex w contains $d \geq 1$ terminals, if no one of v’s terminals matches one of w’s terminals then no solution is possible, else v and w contain a matching pair of terminals then connect them and remove the pair from v and w and delete the edge (v,w), if w contains no terminals (d=1) then delete w.

3.6) No solution is possible because only 2 paths can reach vertex v.

3.7) Edges (u,v) and (v,w) are blue so each one of them represents a terminal say $b_1$ and $b_2$ respectively, if $b_1$ matches $b_2$ then connects the terminals in a path and delete v, if $b_1$ does not match $b_2$ then put terminal $b_1$ in vertex u and terminal $b_2$ in vertex w and delete v.

3.8) Contract edge (u,v) and rename the new vertex u.
The set of terminals in vertex $u$, $T_u$, is not empty ($d \geq 1$), edge $(v,w)$ is blue so it represents a terminal say $b$, if terminal $b$ matches one of $u$'s terminals then connect them, delete $v$, remove this terminal from $u$. If this was the only terminal of $u$ ($d=1$) then delete $u$. If no terminal in vertex $u$ matches terminal $b$ then put terminal $b$ in vertex $w$ and delete $v$.

The red edge represents 2 matching terminals, the blue edge represents a terminal say $b$, because these 2 edges are in series all the paths for terminal $b$ must come from vertex $u$ so, we can connect the 2 red terminals using vertex $v$, delete $v$ from the graph and put terminal $b$ in $u$.

The blue edge represents a terminal say $b$ and the green edge represents 2 non-matching terminals say $g_1$ and $g_2$, if $b$ matches $g_1$ or $g_2$ then connect them, delete $v$, put the other terminal in vertex $w$. If $b$ matches neither $g_1$ nor $g_2$ then no solution is possible.

Any red edge represents 2 matching terminal so one pair can use vertex $v$ for the solution so only one of vertices $u$ or $w$ must be used in the solution for the other pair, after we know the global solution (for all the terminals) we can determine which pair used $v$, so remove the terminals in $(u,v)$, contract edge $(u,v)$ rename the new vertex $u$. (The pair of terminals in the remaining red edge represents the two pairs and for the decision problem this is enough.)
3.13) $\begin{array}{c}
\text{R} \\
\text{G}
\end{array}$ NO SOLUTION
The green edge represents 2 non-matching terminals say $g_1$ and $g_2$, no solution is possible because only one of $g_1$ and $g_2$ can be connected and no path can connect the other terminal because the red edge represents a pair of matching terminals.

3.14) $\begin{array}{c}
\text{G} \\
\text{G}
\end{array}$ NO SOLUTION
$\{g_1, g_2\} \cap \{g_3, g_4\} = \emptyset$
Any green edge represents 2 non-matching terminals, if one terminal of the edge $(u,v)$ matches one terminal of the edge $(v,w)$ then connect them, put the other 2 terminals in vertices $u$ and $w$ respectively and delete $v$, else no solution is possible.

3.15) $\begin{array}{c}
\text{R} \\
\text{G}
\end{array}$ NO SOLUTION
The red edge represents 2 matching terminals therefore no path for the terminals that are not in the red edge can use $v$, so connect the 2 terminals using the vertex $v$, and delete $v$.

3.16) $\begin{array}{c}
\text{G} \\
\text{G}
\end{array}$ NO SOLUTION
The green edge represents 2 non-matching terminals, so contract edge $(v,w)$ and rename the new vertex $w$.

3.17) $\begin{array}{c}
\text{G} \\
\text{G}
\end{array}$ NO SOLUTION
$\begin{array}{c}
\text{d} \\
\text{d-1}
\end{array}$
The green edge represents 2 non matching terminals say $g_1$ and $g_2$, vertex $w$ contains $d \geq 1$ terminals. If one of them matches $g_1$ or $g_2$ then connect them, put the other terminal ($g_1$ or $g_2$) in vertex $u$ and put the rest $d-1$ terminals in vertex $w$, delete $v$ and if no terminal left in $w$ then delete $w$. If neither $g_1$ nor $g_2$ matches any one of $w$’s terminals then no solution is possible.
3.4.4 Table 4 - Reductions of non-white edges with a terminal end.

From property 4 a coloured edge may have at most one terminal in each end vertex. Thus this table covers all possible cases.

4.1) \[ \begin{array}{c}
\text{blue edge} \\
\text{red edge} \\
\text{green edge} \\
\text{blue edge} \\
\text{green edge}
\end{array} \]

The blue edge represents a terminal say b, if the adjacent terminal say a matches b then connect them and delete v, if they do not match then delete the blue edge and put terminal b in its other end.

4.2) \[ \begin{array}{c}
\text{red edge} \\
\text{green edge} \\
\text{red edge} \\
\text{green edge}
\end{array} \]

The red edge represents 2 matching terminals say \( r_1 \) and \( r_2 \), because \( v \) has a terminal connect \( r_1 \) and \( r_2 \) using \( u \) and delete \( u \).

4.3) \[ \begin{array}{c}
\text{green edge} \\
\text{green edge} \\
\text{green edge} \\
\text{green edge}
\end{array} \]

The green edge represents 2 non-matching terminals say \( g_1 \) and \( g_2 \), and let \( a \) be the terminal in \( v \), if \( a \) matches \( g_1 \) or \( g_2 \) then connect them, delete \( v \) and put the other terminal in \( u \), else no solution is possible.

4.4) \[ \begin{array}{c}
\text{blue edge} \\
\text{blue edge} \\
\text{blue edge} \\
\text{blue edge}
\end{array} \]

The blue edge represents a terminal say \( b \), and let \( a \) and \( c \) be the terminals at the ends of the edge. If \( b \) matches \( a \) or \( c \) then connect them and delete the corresponding end vertex, else no solution is possible.

4.5) \[ \begin{array}{c}
\text{red edge} \\
\text{green edge} \\
\text{red edge} \\
\text{green edge}
\end{array} \]

No solution is possible.

4.6) \[ \begin{array}{c}
\text{green edge} \\
\text{green edge} \\
\text{green edge} \\
\text{green edge}
\end{array} \]

The green edge represents 2 non-matching terminals say \( g_1 \) and \( g_2 \) and let \( a \) and \( b \) be the terminals at the ends of the edge. If the matching terminals of \( g_1 \) and \( g_2 \) are \( a \) and \( b \) then connect each pair and delete \( u \) and \( v \), else no solution is possible.
3.4.5 Table 5 - Deletions of edges.

After we delete a vertex u, we have to delete all its adjacent edges. This table is used to delete these edges; vertex u here is already marked "deleted".

5.1) \( \begin{array}{c}
\text{u} \\
\end{array} \quad \begin{array}{c}
\text{v} \\
\end{array} \quad \begin{array}{c}
\text{u} \\
\end{array} \quad \begin{array}{c}
\text{v} \\
\end{array} \)

If the edge is white and its other end v is either deleted or not, then remove the edge.

5.2) \( \begin{array}{c}
\text{u} \\
\end{array} \quad \begin{array}{c}
\text{C} \\
\end{array} \quad \begin{array}{c}
\text{u} \\
\end{array} \quad \begin{array}{c}
\text{v} \\
\end{array} \)

\( C \in \{B, R, G\} \) NO SOLUTION

If the edge has its both ends deleted and it is not coloured white then there is no solution possible.

5.3) \( \begin{array}{c}
\text{u} \\
\end{array} \quad \begin{array}{c}
\text{B} \\
\end{array} \quad \begin{array}{c}
\text{v} \\
\end{array} \quad \begin{array}{c}
\text{b} \\
\end{array} \)

A blue edge represents a terminal say b, if its other end v has no terminals then put terminal b in it and remove the edge from the graph.

5.4) \( \begin{array}{c}
\text{u} \\
\end{array} \quad \begin{array}{c}
\text{B} \\
\end{array} \quad \begin{array}{c}
\text{a} \\
\end{array} \quad \begin{array}{c}
\text{b} \\
\end{array} \)

\( \text{a} = \text{b} \)

A blue edge represents a terminal say b, if its other end contains a matching terminal then connect them and delete v, else no solution is possible.

5.5) \( \begin{array}{c}
\text{u} \\
\end{array} \quad \begin{array}{c}
\text{R} \\
\end{array} \quad \begin{array}{c}
\text{v} \\
\end{array} \quad \begin{array}{c}
\text{u} \\
\end{array} \quad \begin{array}{c}
\text{v} \\
\end{array} \)

A red edge represents 2 matching terminals, if the other end contains no terminals then connect them using v and delete v.

5.6) \( \begin{array}{c}
\text{u} \\
\end{array} \quad \begin{array}{c}
\text{R} \\
\end{array} \quad \begin{array}{c}
\text{v} \\
\end{array} \)

NO SOLUTION

A red edge represents 2 matching terminals, if the other end contains a terminal then no solution is possible.

5.7) \( \begin{array}{c}
\text{u} \\
\end{array} \quad \begin{array}{c}
\text{G} \\
\end{array} \quad \begin{array}{c}
\text{v} \\
\end{array} \)

NO SOLUTION

No solution is possible because a green edge represents 2 non-matching terminals.

4. Concluding remark

In this paper we solve the decision version of the VDPP in series-parallel graphs. It can be shown that our method can be modified to solve the constructive version of the VDPP in series-parallel graphs in polynomial time. A preliminary result shows that it can be done in linear time.
Acknowledgment

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

References


