Finding two disjoint shortest paths in $O(|V| \cdot |E|)$

1 Introduction

The *2-disjoint shortest path problem*, or 2-DSP, is defined as follows: Given a graph $G = (V, E)$ and four vertices s_1, t_1, s_2, t_2, find two vertex-disjoint shortest paths P_1 and P_2, where P_1 goes from s_1 to t_1 and P_2 goes from s_2 to t_2. Here the graph can be directed or undirected. This problem was first considered by Eilam-Tzoreff ([1]), who provided a polynomial time algorithm for the case when the graph is undirected and the edge weights are positive. The running time was $O(|V|^8)$. For directed graphs with positive weights, Berczi and Kobayashi ([3]) were the first to propose a polynomial time algorithm, with running time $O(|V||E|^5)$. It should be noted that the goal of these algorithms was to prove that the problems are solvable in polynomial time, and they were not necessarily designed to be efficient beyond that. Nevertheless, it doesn't seem like the running times have improved much. The fastest seems to be the $O(|V|^2|E|)$-time algorithm in [4], for undirected graphs with unit edge weights.

In this paper, an $O(|V||E|)$-time algorithm for 2-DSP is proposed. The algorithm works for undirected graphs with positive edge weights.

2 Reducing to a problem on DAGs

Fix an instance of the 2-DSP problem (G, s_1, t_1, s_2, t_2) where the graph is undirected and the edge weights are positive.

The first step to solving it will be to consider all edges (u, v) that lie on a shortest path from s_1 to t_1. This creates a directed acyclic graph, or DAG, $G_1 = (V, E_1)$. We will then construct the corresponding DAG $G_2 = (V, E_2)$ for s_2, t_2.

Constructing these DAGs can be done in $O(|V||E|)$ (or faster) with the help of a shortest path algorithm (Dijkstra's).

We will now analyze how to find disjoint paths on these kinds of DAGs. This will require the following definition.

Definition 2.1. Let $G = (V, E)$ be a DAG, and $\phi : V \to \mathbb{R}$ a function. ϕ is said to be a

1. *strictly increasing rank function* if $\phi(v) < \phi(u)$ whenever u can be reached from v.
2. *weakly increasing rank function* if $\phi(v) \leq \phi(u)$ whenever u can be reached from v.
3. *strictly decreasing rank function* if $\phi(v) > \phi(u)$ whenever u can be reached from v.
4. *weakly decreasing rank function* if $\phi(v) \geq \phi(u)$ whenever u can be reached from v.

To illustrate the usefulness of this definition, consider the following theorem. This is a slightly more general version of Theorem 3.1 in [2], and the proof is similar.

Theorem 2.2. *Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be two DAG:s with the same vertex sets, and ϕ a rank function that is strictly increasing on both G_1 and G_2. Then two disjoint paths between some pairs s_1, t_1 and s_2, t_2 can be found in $O(|V||E|)$.*

Proof. This can be solved with dynamic programming. Let's define the following function

$$dp(v_1, v_2) = \begin{cases} 1 \text{ if there is a pair of disjoint paths } v_1 \to t_1, v_2 \to t_2 \\ 0 \text{ otherwise} \end{cases}$$

Here, all vertices with index i uses edges from E_i for $i \in \{1, 2\}$. To compute this, we first note that $dp(v, v) = 0$ and $dp(t_1, t_2) = 1$. Let $C_i(v)$ be the set of vertices that are adjacent from v using edges from E_i. Now, apart from the two base cases above, the following recurrence holds if $v_1 \neq t_1$ and either $\phi(v_1) \leq \phi(v_2)$ or $v_2 = t_2$:

$$dp(v_1, v_2) = \bigvee_{w_1 \in C_1(v_1)} dp(w_1, v_2).$$

Otherwise,

$$dp(v_1, v_2) = \bigvee_{w_2 \in C_2(v_2)} dp(v_1, w_2).$$

This is because if $v_1 \neq t_1$, then in any pair of disjoint paths $v_1 \to t_1, v_2 \to t_2$, the path on G_1 must go through some vertex $w_1 \in C_1(v_1)$, so if $dp(v_1, v_2) = 1$ then one of $dp(w_1, v_2)$ must also be 1. Moreover, since $\phi(v_1) \leq \phi(v_2)$ or $v_2 = t_2$,

we can take any pair of disjoint paths $w_1 \to t_1, v_2 \to t_2$, add $v_1 \to w_1$ to the first path, and get another pair of disjoint paths. This means that if any $dp(w_1, v_2) = 1$ then $dp(v_1, v_2) = 1$. Thus the equality holds. The reasoning as to why the second equality holds is similar. □

It should be noted that in the case when $E_1 = E_2$, we can always find a strictly increasing rank function, for example by taking a topological ordering (in [2], the length of the longest path starting at v is used as a decreasing rank function, which also works). So that case can always be solved in $O(|V||E|)$ with the method above.

When the rank function is weakly increasing/decreasing, it is far less useful to us. For example, it could be constant on the whole graph in which case it wouldn't be of any use at all. However, two weakly increasing/decreasing rank functions can under some circumstances be useful:

Theorem 2.3. *Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be two DAG:s with the same vertex sets. Let ϕ be a rank function that is weakly increasing on both G_1 and G_2, and let ψ be a rank function that is weakly increasing on G_1 and weakly decreasing on G_2.*

Furthermore, assume that ϕ and ψ have the following property: if $u, v \in V$ are adjacent in either G_1 or G_2, then $\phi(u) \neq \phi(v)$ or $\psi(u) \neq \psi(v)$. In other words, when walking on an edge in $E_1 \cup E_2$, both ϕ and ψ can't stay the same.

Then two disjoint paths between some pairs s_1, t_1 and s_2, t_2 can be found in $O(|V||E|)$.

This is the main result of the paper, and its proof will be the focus of the next chapter. To conclude this chapter, let's see how it helps us solve the 2-DSP problem:

Theorem 2.4. *2-DSP can be solved in $O(|V||E|)$.*

Proof. Fix an instance (G, s_1, t_1, s_2, t_2) of the 2-DSP problem. Define $d(u, v)$ to be the length of the shortest path from u to v. We can find the numbers $d(s_i, v)$ and $d(v, t_i)$ for all $i \in \{1, 2\}$ and $v \in V$, by running a shortest path algorithm in $O(|V||E|)$ (or faster). This let's us construct the two DAG:s G_1, G_2 defined earlier.

Now, define the two rank functions ϕ, ψ as follows:

$$\phi(u) = d(s_1, u) + d(s_2, u)$$
$$\psi(u) = d(s_1, u) - d(s_2, u)$$

These functions have the properties described in Theorem 2.3. To see why, note that

1. if an edge with weight w goes from u to v in $E_1 \cup E_2$, then $|d(s_i, u) - d(s_i, v)| \leq w$ for $i \in \{1, 2\}$.
2. if an edge with weight w goes from u to v in E_i, then $d(s_i, v) - d(s_i, u) = w$.

So if an edge goes from u to v with weight w on G_1, then

$$\phi(v) - \phi(u) = d(s_1, v) - d(s_1, u) + d(s_2, v) - d(s_2, u) = w - (d(s_2, u) - d(s_2, v)) \geq w - |d(s_2, u) - d(s_2, v)| \geq 0$$

, which proves that ϕ is weakly increasing on G_1. By symmetry, ϕ is also weakly increasing on G_2. The corresponding proofs about ψ are similar.

Finally, we need to prove that if an edge goes from u to v in G_1 or G_2, then one of ϕ and ψ must change. Consider the quantities $\phi(u) + \psi(u) = 2d(s_1, u)$ and $\phi(u) - \psi(u) = 2d(s_2, u)$. When walking on an edge in E_1 with weight w, the first quantity will increase by $2w$, and when walking on an edge in E_2 the second quantity will increase by $2w$. Thus, one of them will change which means that either ϕ or ψ must change.

The proof now follows from Theorem 2.3.

□

3 Proof of Theorem 2.3

Definition 3.1. Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be two DAGs with rank functions ϕ and ψ satisfying the properties in Theorem 2.3. Let's call such DAGs *perpendicular*. The *perpendicular DAG disjoint paths problem* (PDDP) is the problem of finding two disjoint paths between some pairs (s_1, t_1), (s_2, t_2) of vertices in two perpendicular DAGs.

Our goal is to solve PDDP in $O(|V||E|)$. Let's start with some intuition of how to accomplish this.

Consider two paths P_1, P_2 that solve an instance of the PDDP problem. Imagine putting each vertex $v \in V$ in a coordinate system at the point $(\phi(v), \psi(v))$, and draw line segments between consecutive vertices in P_1 and P_2 (see Figure 1). Due to the properties of ϕ and ψ, P_1 will only move up and right, whereas P_2 will only move down and right. This means that there are only four ways the paths can intersect:

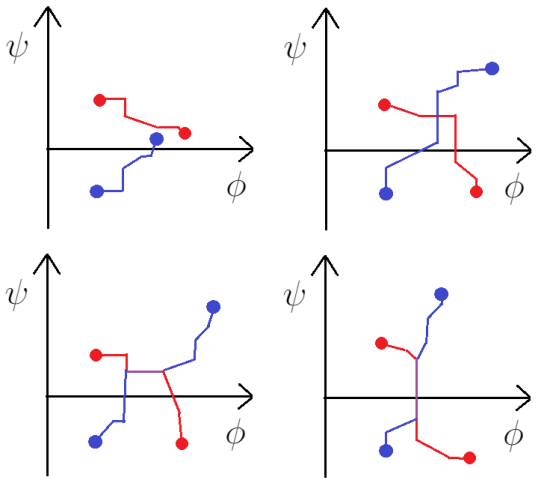

Figure 1

1. The paths don't intersect at all.
2. The paths intersect in a single point.
3. The paths intersect on a horizontal segment.
4. The paths intersect on a vertical segment.

These four cases are shown in Figure 1. Intuitively, in cases 1-3 the DAGs are oriented in the same direction where they overlap, which should make it possible to find them with dynamic programming in a way similar to Theorem 2.2. Case 4 is much harder to apply dynamic programming to directly, since the DAGs move in different directions. However, there is an easy way to get rid of this case: reverse all edges in E_2 and consider paths from t_2 to s_2 instead. We then get another instance of PDDP where previously vertical overlaps have turned into horizontal overlaps.

In short, the strategy will be to solve the instance $(V, E_1, E_2, \phi, \psi, s_1, t_1, s_2, t_2)$ under the additional constraint that overlaps as in case 4 are not allowed. We then run the same algorithm on the instance $(V, E_1, \bar{E}_2, \psi, \phi, s_1, t_1, t_2, s_2)$. Since a solution cannot have both horizontal and vertical overlaps, this is guaranteed to catch all possible solutions. Here $\bar{E}_2$ is the set of reversed edges of E_2. Also, note that ϕ and ψ have switched roles in the second problem instance.

What we end up with is a problem similar to Theorem 2.2, but with a few extra constraints. The solution will be similar, but more complicated.

To make this more formal, we need some definitions:

Definition 3.2. Let P_1 and P_2 be two paths that solve an instance of the PDDP problem. They are said to be *horizontally overlapping* if there exists vertices $u_1, v_1 \in P_1$ and $u_2, v_2 \in P_2$ such that $\psi(u_1) = \psi(v_1) = \psi(u_2) = \psi(v_2)$, and the intersection of the intervals $[\phi(u_1), \phi(v_1)]$ and $[\phi(u_2), \phi(v_2)]$ is non-empty and has positive length.

Similarly, P_1 and P_2 are said to be *vertically overlapping* if the above is true after swapping ϕ and ψ.

Lemma 3.3. *Let P_1 and P_2 be two paths that solve an instance of the PDDP problem. Then they cannot be both horizontally and vertically overlapping.*

Definition 3.4. Let PDDP* be the problem of finding a solution to PDDP, such that the paths P_1 and P_2 are not vertically overlapping.

Lemma 3.5. *If PDDP* can be solved in $\mathcal{O}(|V||E|)$, then so can PDDP.*

Proof. Assume that it is possible to solve PDDP* in $\mathcal{O}(|V||E|)$. Fix an instance of PDDP $I = (V, E_1, E_2, \phi, \psi, s_1, t_1, s_2, t_2)$. Now let's solve the corresponding instance of PDDP* in $\mathcal{O}(|V||E|)$. This finds any solutions that don't have vertical overlaps. To find the ones with vertical overlaps, we solve the instance $I' = (V, E_1, \bar{E_2}, \psi, \phi, s_1, t_1, t_2, s_2)$ of PDDP*. In this modified instance I', solutions to PDDP correspond bijectively to solutions in the instance I. The bijection is to reverse the path P_2 and let P_1 be unchanged. Furthermore, solutions to the instance I with vertical overlaps will be mapped to solutions with horizontal overlaps in I', and vice versa. Therefore, after solving PDDP* for the instance I', we will obtain all solutions to the instance I with vertical overlaps, which means that we have solved PDDP for that instance in $\mathcal{O}(|V||E|)$. □

3.1 Dynamic programming

Finally we have arrived at the problem PDDP*, which is solvable with dynamic programming. Throughout this section, an instance of PDDP* $I = (V, E_1, E_2, \phi, \psi, s_1, t_1, s_2, t_2)$ is fixed. Just like in Theorem 2.2, we will try to define a function $dp(v_1, v_2)$ that determines if there exists a solution to PDDP* with starting points v_1, v_2. However, the transitions of a DP with so few states will be rather complicated and inefficient, especially when $\phi(v_1) = \phi(v_2)$. Instead, we will have 5 functions $dp_0, dp_1, dp_2, dp_3, dp_4$, that determines if solutions exist with various extra constraints. The main reason for introducing more states like this is to ensure that the transitions in each DP state only iterates through the neighbours of one of v_1, v_2, thereby making the algorithm run in $\mathcal{O}(|V||E|)$ time.

1. $dp_0(v_1, v_2)$ determines if there exists a solution to PDDP* with starting points (v_1, v_2).

2. $dp_1(v_1, v_2)$ determines if there exists a solution to PDDP* with starting points (v_1, v_2) such that the path P_1 contains no vertices $u_1 \neq v_1$ with $\phi(u_1) = \phi(v_1) = \phi(v_2)$. This function is only defined when $\phi(v_1) = \phi(v_2)$.

3. $dp_2(v_1, v_2)$ determines if there exists a solution to PDDP* with starting points (v_1, v_2) such that the path P_2 contains no vertices $u_2 \neq v_2$ with $\phi(u_2) = \phi(v_1) = \phi(v_2)$. This function is only defined when $\phi(v_1) = \phi(v_2)$.

4. $dp_3(v_1, v_2)$ determines if there exists a solution to PDDP* with starting points (v_1, v_2), such that there are no vertices $u_1 \in P_1$, $u_2 \in P_2$, with the property that $\phi(u_1) = \phi(u_2) = \phi(v_1) = \phi(v_2)$, and $\psi(u_1) > \psi(u_2)$. This function is only defined when $\phi(v_1) = \phi(v_2)$.

5. $dp_4(v_1, v_2)$ determines if there exists a solution to PDDP* with starting points (v_1, v_2), such that there is no vertex $u \in P_1 \cup P_2$ satisfying $\phi(u) = \min(\phi(v_1), \phi(v_2))$, other than the starting points.

Here "determines if" means that the function returns 1 if the condition is met, and 0 otherwise. The value we are interested in is $dp_0(s_1, s_2)$. Where the other functions come into the picture is probably not clear right now, but will hopefully make more sense when we examine the transitions of $dp_0(v_1, v_2)$. But first, let's get the base cases out of the way.

3.2 Base cases

The base cases for all five functions will be similar. First, we have

$$dp_i(v, v) = 0$$

and

$$dp_i(t_1, t_2) = 1,$$

just like in Theorem 2.2. In addition, we will treat some more situations as base cases in an attempt to make the transitions below cleaner. One such situation is $dp_i(v, t_2)$. Here, all that matters is if there exists a path from v to t_1 that avoids t_2. This is a graph reachability problem, and the answers for all such states can clearly be precomputed in $\mathcal{O}(|V||E|)$. The states $dp_i(t_1, v)$ will be treated similarly. Also, the states $dp_i(v_1, v_2)$ where $\psi(v_1) > \psi(v_2)$ can be treated as base cases in a similar way. The reason for this is that ψ is weakly increasing on (V, E_1) and weakly decreasing on (V, E_2). So if $\psi(v_1) > \psi(v_2)$, then there is no way the two paths can intersect and we just have to find two paths. Note that these graph reachability problems might look different depending on which of the five functions we are evaluating.

Now that those cases are out of the way, we can assume that $v_1 \neq t_1$, $v_2 \neq t_2$, and $\psi(v_1) \leq \psi(v_2)$, and analyze what the DP transitions should look like.

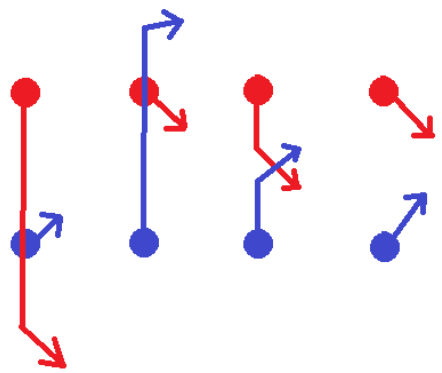

Figure 2

3.3 Transitions

Let's start with some intuition. To find $dp_0(v_1, v_2)$ we get two cases:

1. $\phi(v_1) \neq \phi(v_2)$.
2. $\phi(v_1) = \phi(v_2)$.

It turns out that the first case is easy, we can deal with it in the exact same way as in Theorem 2.2, by advancing the vertex with the lower rank (in terms of ϕ).

The second case is trickier. Now the two paths can move vertically towards each other, and we must ensure that they don't vertically overlap. One way to do that is for instance to prevent P_1 from advancing to a vertex w with $\psi(w) > \psi(v_2)$. However, if P_2 doesn't move vertically, then P_1 could move to such a vertex w, since in that case the intersection would be a single point which is allowed. Let's consider all four combinations of which paths move vertically along the $\phi = \phi(v_1) = \phi(v_2)$ line:

1. P_1 does not move vertically. Such solutions exist if and only if $dp_1(v_1, v_2) = 1$.
2. P_2 does not move vertically. Such solutions exist if and only if $dp_2(v_1, v_2) = 1$.
3. Possibly both P_1 and P_2 move vertically, but they don't contain vertices $u_1 \in P_1$, $u_2 \in P_2$, such that $\phi(u_1) = \phi(u_2) = \phi(v_1) = \phi(v_2)$ and $\psi(u_1) > \psi(u_2)$. Such solutions exist if and only if $dp_3(v_1, v_2) = 1$.
4. P_1 and P_2 do not move vertically. Such solutions exist if and only if

$dp_4(v_1, v_2) = 1$.

These four cases are shown in Figure 2. In short, this means that if $\phi(v_1) = \phi(v_2)$, then

$$dp_0(v_1, v_2) = dp_1(v_1, v_2) \vee dp_2(v_1, v_2) \vee dp_3(v_1, v_2) \vee dp_4(v_1, v_2).$$

It should be noted that there is some overlap between the four cases. For example, cases 1-3 also contain case 4, so we could drop $dp_4(v_1, v_2)$ from the transition and it would still be correct.

References

[1] T Eilam-Tzoreff. *The disjoint shortest paths problem.* Discrete applied mathematics 85 (1998): 113-138.

[2] Y Perl, Y Shiloach. *Finding Two Disjoint Paths Between Two Pairs of Vertices in a Graph.* Journal of the ACM 25 (1978): 1–9.

[3] K Berczi, Y Kobayashi. *The Directed Disjoint Shortest Paths Problem.* 25th Annual European Symposium on Algorithms (ESA 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[4] M Bentert, A Nichterlein, M Renken, P Zschoche. *Using a geometric lens to find k disjoint shortest paths.* arXiv preprint arXiv:2007.12502 (2020)