

# The hardness of the k-ALLOCATION problem

CS270 Class Project  
Michael Demmer, Bowei Du

May 5, 2005

## 1 Introduction

Recent work related to Delay Tolerant Networking (DTN) examined the problem of coping with path failures in a network [3]. The techniques presented in the work are based around the idea of improving delivery reliability of a message by using erasure coding and alternate paths for the redundant code blocks.

In this paper we explore the k-ALLOCATION problem, which arises in the routing of erasure code blocks in networks where the communication links among nodes fail with some known probability structure. Informally speaking, the k-ALLOCATION problem seeks to determine from a graph with capacities, a set of  $k$  paths which will maximize the probability that enough erasure-coded messages reach the destination to be reconstructed.

We begin our discussion by focusing on simplified cases of k-ALLOCATION, specifically leveraging properties of the problem when applied to a restricted class of *layer graphs* (Section 3). Using this framework, we show that 1-ALLOCATION can be solved in polynomial time (Section 4).

We then build on these results along with a reduction from the well-known NP-complete problem of finding disjoint paths in a graph to show that 2-ALLOCATION is NP-complete (Section 5). Using this reduction for the two path case, we then continue to show that k-ALLOCATION is NP-hard, but not necessarily NP-complete (Section 6). We conclude with a brief discussion of some related work (Section 7).

## 2 Network Model

We represent the communication network as a directed graph  $G(V, E)$ , with each edge  $(u, v) \in E$  having an associated failure probability  $\phi(u, v)$  as well as a capacity  $C(u, v)$ . The messages sent in the network will have a single source and sink, denoted as nodes  $s$  and  $t$  respectively.

The failure model represented by the probabilities behaves as follows. For each round of communication,  $k$  paths will be chosen along which to route packets. After the paths are chosen, a Bernoulli coin toss for each edge  $(u, v)$  probability  $\phi(u, v)$  is made to determine whether or not the edge has failed.

**Definition 1** A path  $P$  is **successful** if none of the edges in the path have failed.

**Definition 2** A set of  $k$  paths is **feasible** if for each edge  $(u, v)$ , the number of paths which use the edge is  $\leq C(u, v)$ .

**Definition 3** The k-ALLOCATION problem:

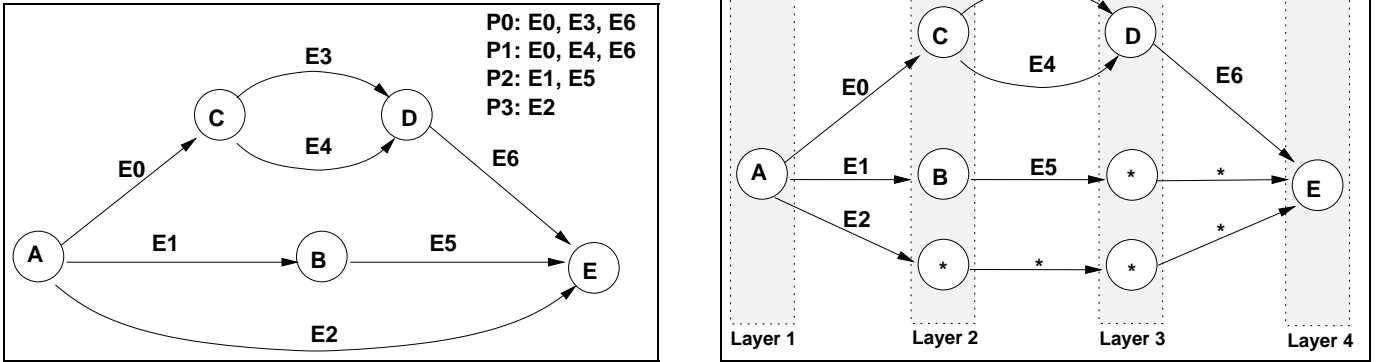


Figure 1: Example network with four paths and transformation into a layer graph. Each edge has an associated failure probability (not shown). Nodes and edges with a label '\*' were added by the transformation.

*Given:* Graph  $G = (V, E)$ , number of paths  $k$ , number of required successes  $n$ , edge failure probabilities  $\phi(u, v)$  and capacities  $C(u, v)$ .

*Find:* A set of  $k$  feasible  $s$  to  $t$  paths such that  $\mathbb{P}(S \geq n)$  is maximal, where  $S$  is a random variable denoting the number of successful paths.

This model represents the case of a single message which has been erasure coded into  $k$  messages,  $n$  of which need to arrive at  $t$  for successful reconstruction. The number of paths using that can use an edge  $(u, v)$  must be  $\leq C(u, v)$ , which reflects a capacity constraint on the bandwidth of the link.

Figure 1 shows a sample network graph that has four possible paths from the source node A to the sink node E. As can be seen from the graph, paths P0 and P1 both share edges E0 and E6. Therefore, the fate of messages sent over these two paths is dependent, unlike paths P2 and P3 which are independent.

### 3 Layer Graphs

Before we begin our discussion of various aspects of k-ALLOCATION, we first discuss some properties of the allocation problem when applied to *layer graphs*.

**Definition 4** A **layer graph** is a graph in which the nodes of the graph can be partitioned into  $n$  layers such that the only edges of the graph are those between nodes on layer  $i$  to nodes on layer  $i + 1$ .

#### 3.1 Layer Graph Construction

For any input graph that is a DAG, the algorithm for transforming the DAG to a layer graph is straightforward.

First, perform a standard topological sort on the DAG from the start node, which produces a sequential ordering  $order(u)$  for each node of index  $1..|V|$ . Note that we can immediately remove all vertices that come after the destination node  $t$  in the sorted order because they have no path to  $t$  and hence do not participate in any  $s$  to  $t$  path.

For any edge that exists between two vertices  $u$  and  $v$  whose order differs by  $k > 1$ , add to the layer graph

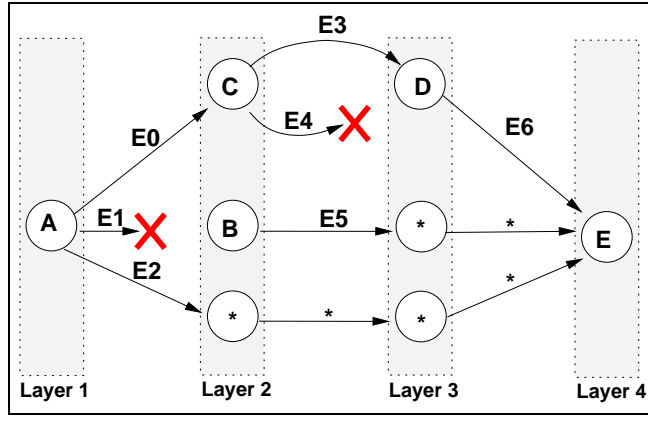


Figure 2: Example layer graph with failed edges. Referring back to the definitions of  $p_0 \dots p_3$  from Figure 1, we have:  $\vec{e}_0 = \{e_0 = 1, e_1 = 0, e_2 = 1\}$ ,  $\vec{p}_0 = (1, 1, 1, 1)$ , and  $\vec{p}_1 = (1, 1, 0, 1)$ , so therefore  $\vec{p}_1 = T(\vec{p}_0, \vec{e}_0)$ .

with  $k - 1$  additional edges and vertices in a linear sequence between  $u$  and  $v$ . Let the order of each new augmented node be exactly one greater than it's predecessor.

Figure 1 shows the result of transforming the original graph into a layer graph. After this transformation, all edges  $(u, v)$  (both augmented or original) in the new graph have the property that  $\text{order}(v) = \text{order}(u) + 1$ .

**Claim 5**  $G'$  is a layer graph and is polynomial in size w.r.t the original graph  $G$ .

**Proof:** After the topological ordering and augmentation, all edges in the graph from  $u$  to  $v$  have the property that  $\text{order}(u) + 1 = \text{order}(v)$ . Assign nodes with the same order to the same layer; the resulting graph satisfies the conditions for being a layer graph.

At the worst case, an edge will cause the addition of  $|V| - 1$  intermediary nodes, meaning there can at most be  $|V|^3$  additional nodes. ■

### 3.2 Layered Graph Probabilities

In this section we show how the probability of success for each path is computed for a layered graph in polynomial time for a fixed number of paths  $k$ . This will be used to derive a “succinct certificate” necessary to show that 2-ALLOCATION  $\in$  NP.

Consider a layered graph  $G = (V, E)$  with  $K$  layers, where each edge  $(u, v)$  is labeled with a probability value  $\phi(u, v)$ . Also, there is a set of  $l$  paths.

Let  $\vec{p}_j = (p_{1,j}, p_{2,j}, \dots, p_{l,j})$  be a random vector of indicator variables where  $p_{i,j} = 1$  if path  $i$  is reachable to layer  $j$  and is equal to zero otherwise. Let  $\vec{e}_i$  be a random vector of indicator variables of the working/failed status of the edges in layer  $i$ . Let  $T(\vec{p}_i, \vec{e}_i)$  be a transfer function which takes a vector  $\vec{p}_i$  of the paths which reach layer  $i$  and an assignment  $\vec{e}_i$  of failure/successes and maps the paths reachable to layer  $i$  in  $\vec{p}_i$  to paths reachable to layer  $i + 1$  given  $E_i$ . Let  $T^{-1}(\vec{p}_{i+1}, \vec{e}_i)$  be the inverse function of  $T$  given the paths in the next layer and a set of working/failed layer  $i$  edges. See Figure 2 for an example.

**Claim 6**

$$\mathbb{P}(\vec{p}_j = \vec{x} | \vec{e}_j) = \sum_{\vec{p} \in T^{-1}(\vec{x}, \vec{e}_j)} \mathbb{P}(\vec{p}_{j-1} = \vec{p}) \quad (1)$$

**Proof:** We will show this by showing that the set of events denoted by the two sides of the equation are identical.

The sample space of events consists of assignments of working/failed status on the edges up to layer  $j$ . Let  $A$  be the event that  $\vec{p}_j = \vec{x} | \vec{e}_j$  and  $B$  be the event that  $\bigcup \vec{p}_{j-1} = \vec{p}$ .

$A \subseteq B$ : Let  $a \in A$  be an assignment of working/failed status to the edges up to layer  $j$ . Consider the set of paths  $\vec{m}$  reaching layer  $j$  with the working/failed edges of  $a$ .  $a \in A$  means that  $T(\vec{m}, \vec{e}_j) = \vec{p}_j$  because the effect of the edge status  $\vec{e}_j$  in layer  $j$  generates  $\vec{p}_j$  successful paths in layer  $j$  and therefore  $a \in B$ .

$B \subseteq A$ : Consider  $b \in B$ . Then for the set of successful paths  $\vec{m}$  formed by  $b$ ,  $T(\vec{m}, \vec{e}_j) = \vec{p}_j$ . Since  $T$  maps successful paths from one level to the next given the status of the edge on that level, we see that  $b$  is a possible assignment to the outcome of the edges given  $\vec{e}_j$ . ■

### 3.3 Checking feasible solutions on a DAG

**Claim 7** *A feasible solution to  $k$ -ALLOCATION on a DAG can be checked in polynomial time.*

**Proof:** First, checking that the flow constraints have been satisfied can be done trivially in  $O(|E|)$ .

Now convert the DAG to a layer graph by the methodology outlined above. Calculation of the probabilities will be done from the first layer of the graph to the last layer of the graph. By conditional probability, we have that:

$$\mathbb{P}(\vec{p}_j = \vec{x}) = \sum_{\vec{e}_j} \mathbb{P}(\vec{p}_j = \vec{x} | \vec{e}_j)$$

by conditioning on the working/failed status of the edges of each layer. Then, we note that there are at most  $O(2^k)$  elements of  $\vec{e}_j$  and for each of the  $\vec{e}_j$ , we compute (using Claim 6) the new probability from a summation of  $O(2^k)$  elements of the previous probability. This results in  $O(2^{2k})$  multiplications and additions for each layer. By saving the value of  $\mathbb{P}(\vec{p}_j = \vec{x})$  at each layer, the final probability can be computed in  $O(|V| 4^k)$  time. ■

## 4 1-ALLOCATION is in P

We now discuss the 1-ALLOCATION problem, i.e. a simplified instance of  $k$ -ALLOCATION restricted to have only a single message to transmit (and therefore a single path to find).

In this case, we can easily calculate the probability of success for a given allocation, as it is simply the product of the probabilities of the edges that make up the (single) path. Therefore, we can find the maximal probability in polynomial time by Dijkstra's shortest path algorithm as follows:

**Claim 8** *1-ALLOCATION can be solved in time  $O(|E| \log |V|)$ .*

**Proof:** Construct a new graph  $G' = (V, E')$ , such that for each edge  $e_{uv} \in E$  with failure probability  $\phi(u, v)$  in the original graph  $G$ , we add an edge  $e'_{uv}$  to  $G'$  with weight equal to  $-\log(\phi(u, v))$ . Assuming that the operation of calculating the logarithm to a sufficient approximation is constant, then this construction is linear in the original graph  $G$ .

Note that by this construction, all edge weights are positive (since all probabilities are between 0 and 1). Therefore, the new graph cannot have any negative-weight cycles, so we can apply Dijkstra's algorithm to find the shortest path from  $s$  to  $t$  in the new graph  $G'$ . The standard heap-based Dijkstra's algorithm runs in time  $O(|E| \log |V|)$ .

Finally, by definition, the shortest path in  $G'$  is the path with the least sum of edge weights, and therefore the least sum of the negative logarithm of success probabilities, by the above construction. Therefore, this sum corresponds to the greatest product of the original probabilities, and therefore the shortest path in  $G'$  is the path with the highest probability in  $G$ . ■

As a final note, the intuition behind this solution for 1-ALLOCATION can also be used to solve the original problem with a different failure model in which message *transmissions* are independent. In other words, even if two paths share an edge, the success probabilities for each path succeeding are still independent. Performing the same transformation using negative logarithms as outlined above, then this problem is identical to the MIN-COST-MAX-FLOW problem that was given in homework set 2, and thus it can also be solved using Linear Programming in polynomial time.

## 5 2-ALLOCATION is NP-complete

We now turn to the second simplification of the problem, namely 2-ALLOCATION and prove that it is NP-complete.

### 5.1 2-ALLOCATION $\in$ NP

We first show that 2-ALLOCATION is in NP by proving the succinct certificate property for the problem. The general approach is to apply a transformation of the two path solution into a DAG and therefore into a layer graph. This allows us to apply the polynomial-time probability calculations described in Section 3.2 and therefore to derive the joint success probability for the two paths.

First, we check that the two paths given are each valid acyclic  $s$ - $t$  paths and that the flow constraints on the original graph are obeyed. These both can be done in  $O(|E|)$  time via a simple path traversal.

Now we apply a transformation to the subgraph defined by the two paths such that the resultant graph is a DAG and yet preserves the joint probability characteristics of the original two paths. To do this, we make a copy of each path in the new graph, except where they share edges. This preserves the dependency relationships on the shared edges, while avoiding cycles in the graph.

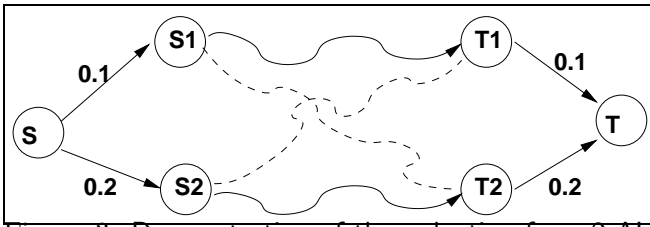
**Claim 9** *Given two  $s$ - $t$  paths in a graph  $G$ , one can construct two new paths in a DAG  $G'$  that have the same joint success probability distribution as the original two.*

**Proof:** To begin, for each edge  $(u, v)$  in  $P_1$ , we add an edge  $(u_1, v_1)$  to the graph  $G'$ . Similarly, for each edge  $(u, v)$  in  $P_2$ , we add an edge  $(u_2, v_2)$  to  $G'$ . Now, for each shared edge  $(u, v)$ , modify the new graph to merge nodes  $u_1$  with  $u_2$  and  $v_1$  with  $v_2$ .

The probability of success for each path is defined as the probability that each edge along the path succeeds, that is  $\mathbb{P}(P_i = 1) = \mathbb{P}(E_1 = 1, E_1 = 1, \dots, E_n = 1)$ , where  $E_i$  is an indicator random variable for whether or not the  $i$ 'th edge of the path succeeds in the original graph. By the above construction, each edge is either independent (i.e. not shared in the two paths) or dependent (i.e. shared). As the independent edges remain independent in the new graph, and the dependent edges remain dependent, the joint probability is unmodified by the graph transformation.

Neither path by itself introduces a cycle in the new graph by the above construction, since the paths are copied when added to the new graph. Therefore, the only way a cycle could occur is if it involves at least one shared edge. Suppose there is a cycle; there are two cases:

The first case is where there is exactly one shared edge  $(u, v)$  on the cycle. For there to be a cycle, there



	$\mathbb{P}(S = 0)$	$\mathbb{P}(S = 1)$	$\mathbb{P}(S = 2)$
parallel	0.9504	0.0492	0.0004
cross	0.9604	0.0392	0.0004

Figure 3: Demonstration of the reduction from 2-ALLOCATION to 2-DISJOINT-PATH. By finding two paths on the given graph (where all edge capacities are 1) with sufficiently high probability of  $\mathbb{P}(S \geq 1)$ , we therefore must find an edge-disjoint path from  $s_1$  to  $t_1$  and from  $s_2$  to  $t_2$ .

must be a path from  $v$  to  $u$ . Since there are no shared edges on this cycle, then each vertex must not be shared as well. Therefore, there must be a cycle in one of the two paths, which is a contradiction.

The second case is if there are  $k > 1$  shared edges along the cycle. By definition, there must be zero or more unshared edges along *both* paths connecting the  $i$ 'th shared edge to the  $(i + 1)$ 'th shared edge. For there to be a cycle, there must also be a path from the  $(k - 1)$ 'th shared edge to the first shared edge along at least one path. However for this to occur, there must be a cycle in one of the paths alone, again a contradiction. Therefore  $G'$  is a DAG. ■

**Theorem 10** *2-ALLOCATION is in NP.*

**Proof:** By Claim 9 and Claim 7 we can compute the joint probability of success for two paths in an arbitrary graph  $G$ . This suffices for the succinct certificate property. ■

## 5.2 2-ALLOCATION is NP-hard

In this section we prove that 2-ALLOCATION is NP-hard by using a reduction from the following NP-complete problem [1, 5, 6, 8]:

**Definition 11** *The 2-DISJOINT-PATH problem:*

*Given:* A directed graph  $G = (V, E)$  and vertices  $s_1, s_2, t_1, t_2 \in V$ .

*Find:* Two paths in  $G$ , one from  $s_1$  to  $t_1$  and one from  $s_2$  to  $t_2$  that do not share a edge.

**Theorem 12** *2-ALLOCATION is NP-hard.*

**Proof:** From the given graph  $G$ , select any four vertices  $s_1, s_2, t_1, t_2$ . For each edge  $(u, v)$  in the graph, assign the success probability  $\phi(u, v) = 1.0$  and the capacity  $C(u, v) = 1$ . Add a super source and super sink  $s$  and  $t$  and four additional edges with success probabilities  $\phi(s, s_1) = 0.1$ ,  $\phi(s, s_2) = 0.2$ ,  $\phi(t_1, t) = 0.1$ ,  $\phi(t_2, t) = 0.2$ . The augmented graph is shown in Figure 3. Run the 2-ALLOCATION decision problem on this augmented graph to find two paths from  $s$  to  $t$  such that  $\mathbb{P}(S \geq 1) \geq 0.495$ .

There is a solution to this 2-ALLOCATION problem  $\iff$  there is a solution to 2-DISJOINT-PATH.

$\implies$  Suppose there is a solution to 2-ALLOCATION. First, there can be no shared edges due to the flow constraints. Second, the probability set on the edges is such that any path which includes edge  $(s, s_1)$  must also use edge  $(t_1, t)$  and similarly with  $(s, s_2)$  and  $(t_2, t)$ . This can be verified by calculation, the results of which are demonstrated in Figure 3.

$\impliedby$  The solution of 2-DISJOINT-PATH maps directly onto the specialized case of 2-ALLOCATION and satisfies the required probabilities. ■

### 5.3 2-ALLOCATION is NP-complete

By Theorem 10 and Theorem 12, we know that 2-ALLOCATION is NP-complete.

## 6 k-ALLOCATION is NP-hard

Generalizing the reduction from the previous section, we now show the final result, that the full k-ALLOCATION problem is NP-hard. We use the more general disjoint paths problem:

**Definition 13** *The DISJOINT-PATH problem:*

*Given:* A directed graph  $G = (V, E)$  and a target collection  $T = \{(s_1, t_1), (s_2, t_2), \dots, (s_k, t_k)\}$  of pairs of vertices in  $V$ .

*Find:*  $k$  paths in  $G$  connecting each pair of vertices  $(s_i, t_i)$  that do not share any edges.

The general technique used by Theorem 12 is again used here to show that k-ALLOCATION can reduce to DISJOINT-PATH. We adopt a similar construction in which we augment the graph with a supersource  $s$  and a supersink  $t$ . Choose a set of distinct probability values  $\{p_1 \dots p_k\}$ . For each pair  $(s_i, t_i)$  in the target collection, add a *source edge* from  $s$  to  $s_i$  and a *sink edge* from  $t_i$  to  $t$ , both with probability  $p_i$ . Set all other edge probabilities to one and all edge capacities to one.

With this construction, it suffices to show that the maximal probability of receiving one or more messages out of  $k$  transmissions is achieved in the “parallel” case. In other words, that the probability of at least one success is maximized on paths  $\{(s, s_1, \dots, t_1, t), (s, s_2, \dots, t_2, t), \dots, (s, s_k, \dots, t_k, t)\}$ . Any “crossed” path that leads from  $s_i$  to some other  $t_j$  will result in a reduced overall probability. This means that if there was a satisfying k-ALLOCATION using the maximal probability, then it must be the case that the paths returned by k-ALLOCATION are valid paths in the original problem. Since  $k$  is a constant, we can calculate this probability in constant time. Therefore, we can construct an instance of k-ALLOCATION to solve the general DISJOINT-PATH problem.

**Definition 14** *A parallel path is a path from  $s$  to  $t$  in the constructed graph  $G$  that includes the source edge  $(s, s_i)$  and the sink edge  $(t_i, t)$  for some pair  $(s_i, t_i) \in T$ , where  $T$  is the target collection from the DISJOINT-PATH problem statement.*

**Definition 15** *A cross path is a path from  $s$  to  $t$  in the constructed graph  $G$  that includes a source edge  $(s, s_i)$  and a sink edge  $(t_j, t)$  for some pair  $(s_i, t_k) \notin T$ .*

**Claim 16** *Let  $x$  and  $y$  be probability of edge failure for the two sets of edges, and let  $x > y$ . Then  $\mathbb{P}(S_{\parallel} > 0) > \mathbb{P}(S_{\times} > 0)$ , where  $S_{\parallel}$  ( $S_{\times}$ ) is the random variable representing the number of successes on the parallel (cross) path.*

**Proof:** Given any two pairs  $(s_i, t_i), (s_j, t_j) \in T$ , let  $x$  be the probability on the source edge  $(s, s_i)$  and the sink edge  $(t_i, t)$ , and let  $y$  be the corresponding probability on  $(s, s_j)$  and  $(t_j, t)$ . Then we see that:

$$\mathbb{P}(S_{\parallel} > 0) = \mathbb{P}(\text{both succeed}) + \mathbb{P}(\text{one succeeds}) \tag{2}$$

$$= x^2y^2 + 2((1-x)xy^2 + (1-y)yx^2) + (1-x)^2y^2 + (1-y)^2x^2 \tag{3}$$

$$\mathbb{P}(S_{\times} > 0) = x^2y^2 + 2((1-x)xy^2 + (1-y)yx^2) + 2(1-x)(1-y)xy \tag{4}$$

Now comparing  $\mathbb{P}(S_{\parallel} > 0)$  and  $\mathbb{P}(S_{\times} > 0)$ , we see that

$$x > y \implies (x-y)^2 > 0 \implies x^2 - 2xy + y^2 > 0 \tag{5}$$

Working out the terms that are different between  $\mathbb{P}(S_{\parallel} > 0)$  and  $\mathbb{P}(S_{\times} > 0)$ , we see that:

$$\mathbb{P}(S_{\parallel} > 0) - \mathbb{P}(S_{\times} > 0) \tag{6}$$

$$= (1-x)^2y^2 + (1-y)^2x^2 - 2(1-x)(1-y)xy \tag{7}$$

$$= (2x^2y^2 - 2xy^2 - 2yx^2 + x^2 + y^2) - (2xy - 2x^2y - 2y^2x + 2x^2y^2) \tag{8}$$

$$= x^2 + y^2 - 2xy \tag{9}$$

$$> 0 \tag{10}$$

■

**Claim 17** *The set of paths with the highest probability of succeeding at least once is the set of paths which take the parallel pairs of edges.*

**Proof:** Let  $S$  be a random variable equal to the number of successful paths for a given set of paths.  $S = P_1 + P_2 + \dots + P_k$ , where  $P_i$  is the indicator variable for whether or not path  $i$  succeeds. Assume there exists at least one crossed path, and let  $P_a$  be one such path that has the highest probability on its source edge. Let  $P_b$  be the path which uses  $P_a$ 's paired sink edge. Now consider  $S' = P'_a + P'_b + \dots + P_k$  where  $P'_a$  has been paired with its sink edge (i.e. made into a parallel path) and  $P'_b$  now has the sink edge that belonged to  $P_a$ .

By Claim 16, we see that  $\mathbb{P}(P'_a + P'_b) > \mathbb{P}(P_a + P_b)$ . Since the rest of the paths were not changed, and all paths are independent (since they do not share any edges), this means that  $\mathbb{P}(S' > 0) > \mathbb{P}(S > 0)$ . This “fixup” can be done repeatedly until all of the paths use parallel paired edges, and therefore the set of parallel paired edges has the highest probability of at least one success. ■

**Theorem 18** *k-ALLOCATION is NP-hard.*

**Proof:** Do the construction described at the beginning of this section. Calculate the probability  $p'$  of at least one message success given that there is a set of parallel paths. Run the k-ALLOCATION decision problem on the constructed graph with  $n = 1$  to find a set of paths such that the probability of success is greater or equal to  $p'$ . Since the probability obtained by the parallel paths is the highest possible probability allocation, there is no other solution than the one which indicates the parallel paths. This solution suffices to solve the DISJOINT-PATH problem, therefore k-ALLOCATION is NP-hard. ■

## 7 Related work and conclusion

In the course of this investigation, we have found some connections to other work that are related to the k-ALLOCATION problem.

There is similar work in the literature in the area of network reliability and . It has been shown that determining the probability of connectness given edge failures for  $k$  node pairs is #P hard.[7]

One point of note is that the construction in Section 5.1 does not work for 3 or more paths. The rationale behind this can be seen in Figure 7. In this case, we have three paths, none of which have a cycle, yet where each path shares an edge with each of the other two. In this case, there is no way to construct a DAG that preserves the probability characteristics of this graph; hence we cannot use the same construction to generate a layer graph. An interesting future exercise would be to determine if the construction would hold for planar graphs.

If we recast the 1-ALLOCATION problem to determining a path such that the probability of success is *exactly*  $k$ , then we can easily show that this recast problem is NP-hard. The reduction follows the same

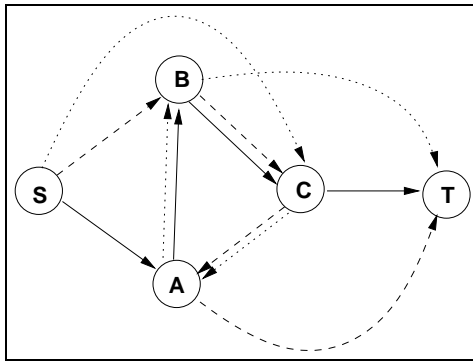


Figure 4: Counter-example to show why the construction in Section 5.1 fails for three or more paths.

logic as in that of finding disjoint paths of length  $k$  [2]. In particular, we can modify the input graph such that each vertex can be visited only once by splitting the node into an “input” side and an “output” side with a capacity 1, probability 1 edge connecting the two. Then assign all other edges in the graph probability  $p$  and capacity 1. Finding a path for one message with probability  $p^{|V|}$  is therefore identical to determining if there is a Hamiltonian path from  $s$  to  $t$ . Hence this modified allocation problem is NP-hard.

In conclusion, we are hopeful that in future work we will find a technique to prove that the  $k$ -ALLOCATION problem is in NP, and therefore is NP-complete.

### Acknowledgements

We are greatly appreciative of preliminary work and discussion with Sushant Jain as well as ongoing discussion with Michael Rosenblum and email correspondence with Brian Dean.

### References

- [1] Steven Fortune, John Hopcroft, and James Wyllie, *The Directed Subgraph Homeomorphism Problem*, in *Theoretical Computer Science*, Vol 10, (1980) pp. 111-121.
- [2] I. Itai, Y. Perl, Y. Shiloach, *The complexity of finding maximum disjoint paths with length constraints*, *Networks* 12 (1982) pp. 277-286.
- [3] Sushant Jain, Michael Demmer, Rabin Patra and Kevin Fall, *Using redundancy to cope with failures in a Delay Tolerant Network*, in *SIGCOMM 2005* (to appear)
- [4] Sushant Jain, Kevin Fall, and Rabin Patra, *Routing in a Delay Tolerant Network*, in *SIGCOMM 2004*
- [5] Richard Karp, *Complexity of Computer Computations* Plenum Press, New York, 1972.
- [6] M. Middendorf and F. Pfeiffer, *On the complexity of the disjoint paths problem* *Combinatorica*, 13:97-107, 1993.
- [7] Leslie Valiant, *The Complexity of Enumeration and Reliability Problems*, in *SIAM Journal on Computing*, Vol. 8, No. 3, August 1979, pp. 410-421.
- [8] J. Vygen. *NP-completeness of some edge-disjoint paths problems*, *Discrete Applied Mathematics*, 61:83-90, 1995.