Dynamically Random Graphs

Alexis Byers, Wittenberg University
Mallory Reed, Earlham College
Laura Rucci, Cabrini College
Elle VanTilburg, University of Texas-Austin
SUMSRI 2013
Miami University

July 18, 2013

In this paper, we introduce the idea of a weighted graph that has edges with associated probabilities of being available at discrete time instants. We attempt to solve problems such as the Chinese Postman Problem, finding Eulerian tours, and finding spanning trees in such graphs with the added challenge of minimizing the time spent waiting for edges to become available. We look at a related problem in which we are given a set of n tasks, each with a probability of being available and a completion time, and we provide a strategy for completing k of the n tasks with shortest expected time. Finally, we consider the computation times of our strategies and discuss avenues for further research.

1 Introduction

An *undirected graph* is a set of *vertices*, together with a set of *edges* connecting pairs of vertices. The *degree* of a vertex v, denoted $d(v)$, is the number of "edge ends" incident with that vertex. By contrast to an undirected graph, a *directed graph* is a graph in which the edges have directions associated with them from one vertex to the other. The number of edges directed toward a vertex is called the *in-degree* of the vertex, and the number of edges directed away is called the *out-degree*. In general, we can consider an undirected graph as a directed graph in which each undirected edge is really two directed edges, one directed toward one end vertex, and one directed toward the other end vertex.

We have considered various types of graphs. First, a *path graph* P_n is a graph on n vertices such that the vertices can be ordered in a list such that two vertices are adjacent if and only if they are consecutive in the list ([2]).

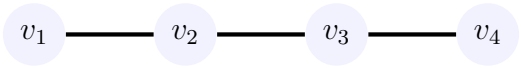


Figure 1: A path on 4 vertices, P_4

A *cycle graph* C_n on n vertices "is a graph with an equal number of vertices and edges, whose vertices can be placed around a circle so that two vertices are adjacent if and only if they appear consecutively along the circle" [5].

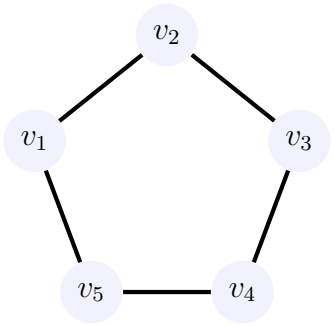


Figure 2: A cycle on 5 vertices, C_5

A *tree* is a connected graph that contains no cycles.

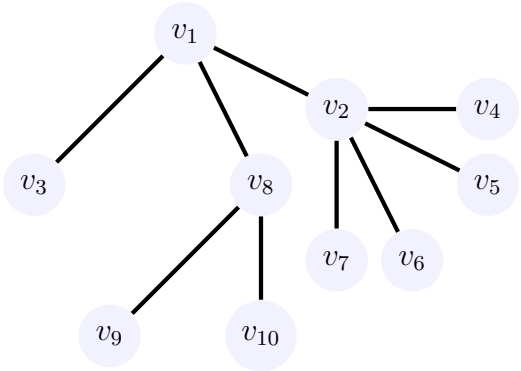


Figure 3: A tree on 10 vertices

A *complete graph* K_n is a graph on n vertices such that every vertex in the graph is adjacent to every other vertex in the graph.

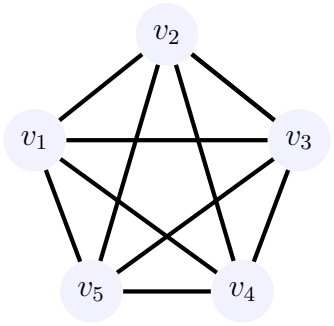


Figure 4: A complete graph on 5 vertices, K_5

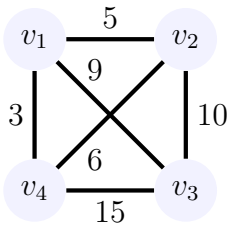
The goal of some traditional graph theory problems is to traverse graphs in certain ways. The objective of the *Chinese Postman Problem* is to begin at a designated start vertex v, travel across every edge in the graph at least once, and return to v. If instead we wish to travel from v across every edge in the graph exactly once before returning to v, then we are looking for an *Eulerian tour*, or *Eulerian circuit*. In undirected graphs, Eulerian tours exist exactly when all vertices have even degree and at most one component is nontrivial. Directed graphs have Eulerian tours exactly when each vertex has equal in-degree and out-degree and there is at most one nontrivial strong component. When G has an Eulerian tour, *Fleury's algorithm* finds one in the following way: Let S be the set of edges traversed so far, partway through the algorithm. When at vertex x, travel across any edge $\overline{xy}$ not yet traversed, so long as afterward each untraversed edge $\overline{ab}$ is such that the start vertex is reachable from b using only untraversed edges.

We consider also the objective of finding minimum spanning trees of a graph. A *spanning tree* of some graph G is a subgraph of G that is a tree including every vertex of G. When considering a graph in which each edge is assigned a weight representing some cost, a *minimum spanning tree* is a spanning tree with a minimum sum of edge weights. Two famous 'greedy algorithms' for finding minimum spanning trees are *Kruskal's algorithm* and *Prim's algorithm*. The difference between the two is the order in which we are allowed to choose edges to be included in the spanning tree. Using *Prim's algorithm*, given a start vertex, we choose an edge of least weight among those incident to that vertex. Then we choose the next edge in our spanning tree among those adjacent to the first edge chosen. We continue constructing the spanning tree by choosing an edge of least weight among those adjacent to at least one of the previously chosen edges, as long as that edge does not form a cycle. *Kruskal's algorithm* instead first chooses an edge of least weight and then chooses among all remaining edges of the graph an edge of least weight, as long as a cycle is not formed. Unlike Prim's algorithm, there is no stipulation that the next edge selected must be adjacent to one of the previous edges

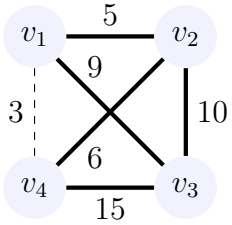
chosen. In each of these algorithms, we continue this process until all of the vertices in G are connected.

1.1 Example of Traditional Kruskal's Algorithm

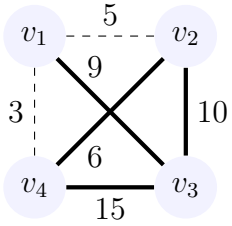
Let's look at how Kruskal's algorithm would construct a minimum spanning tree on the following edge-weighted graph, K_4:



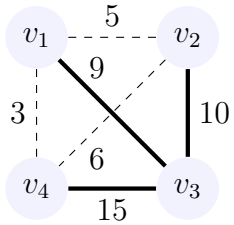
By Kruskal's algorithm, we would select edge $\overline{v_1v_4}$ first since it has the least weight among all edges of K_4. Then, we must choose our next edge among the solid edges left in the following graph:



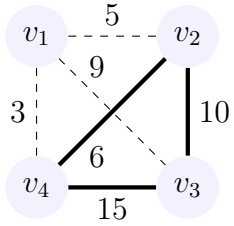
Since $\overline{v_1v_2}$ has the least weight among the edges left, we choose $\overline{v_1v_2}$ next. We are left with the following subgraph (comprised of the solid edges) of K_4 of unchosen edges:



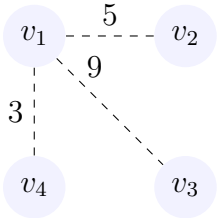
The next solid edge of least weight is $\overline{v_4v_2}$. However, if we select $\overline{v_4v_2}$, we will have a cycle formed among our chosen edges as below:



Since our spanning tree cannot contain a cycle, we must disregard $\overline{v_4v_2}$. The next edge of least weight in our graph happens to be $\overline{v_1v_3}$. Since $\overline{v_1v_3}$ forms no cycle with the edges selected so far, we include it:



Since the dashed edges now span the entire graph K_4, that is, every vertex of K_4 is in the dashed subgraph of K_4, then the dashed edges form a spanning tree of K_4, and we are done. Here is our spanning tree of K_4 with minimum possible weight of 17:



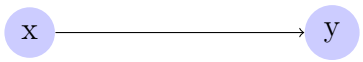
So constructing a minimum spanning tree of a connected graph is straightforward.

Unlike in traditional problems such as the example above, we study graphs in which each edge e is assigned a weight t and also a probability p. The probability p corresponds to the likelihood that, at any given moment in our problem, e is available to select.

The weight t of e represents a cost (in time, distance, currency, etc.) associated with selecting e (for instance, traveling across e). For our purposes, we often discuss the edge weights as if they were time costs (though they may be other things). The goal of our modified problems is to find a strategy that minimizes the expected time to complete our problem's objective, over all strategies for completing the objective.

1.2 Expected Value Function

Suppose at time $t = 0$ that we are waiting at x for a bus that departs along edge $\overrightarrow{xy}$. Further suppose that at each time t the edge is available with probability .07. Here t is any of the times $t = 1, 2, 3, \ldots,$ k, where the departure events are independent.



How much waiting time do we expect for the first bus to depart? Let A be the random variable denoting the amount of time until the bus departs. Consider the probability that time t is when the first available bus departs:

$$\begin{aligned} P(t=1) &= (.07) \\ P(t=2) &= (.07)(1-.07) \\ P(t=3) &= (.07)(1-.07)^2 \\ \text{And, generally: } P(t=k) &= (.07)(1-.07)^{k-1} \end{aligned}$$

Observe that $P(t = 2)$ is the probability that the bus departs at time 2 but not at time 1, $P(t = 3)$ is the probability the bus departs at time 3 but neither at time 1 nor at time 2, $\ldots$, and thus, $P(t = k)$ is the probability that the bus first departs at time k. Using the expression $E(A)$ to denote the expected total time we have to wait, we have:

$$E(A) = \sum_{k=1}^{\infty} kP(t=k) \tag{1}$$

$$= \sum_{k=1}^{\infty} k(.07)(.93)^{k-1} \tag{2}$$

$$= (.07)\sum_{k=1}^{\infty} k(.93)^{k-1} \tag{3}$$

Recall the geometric series, $\sum_{k=0}^{\infty} ar^k = \dfrac{a}{1-r}$ for $|r| < 1$. Notice that equation (3) resembles the derivative of the geometric series where $a = 1$:

$$\frac{d}{dr}(\sum_{k=0}^{\infty} r^k) = \frac{d}{dr}(\frac{1}{1-r}) \tag{4}$$

$$\text{So,} \quad \sum_{k=1}^{\infty} kr^{k-1} = \frac{1}{(1-r)^2} \tag{5}$$

Applying this to equation (3) yields:

$$(.07)\sum_{k=1}^{\infty} k(.93)^{k-1} = (.07)\frac{1}{(.07)^2} = \frac{1}{.07}. \tag{6}$$

Generalizing, we define A to be the amount of time until the bus departs when the edge has probability p of becoming available. Then the expected waiting time for A is:

$$E(A) = \tfrac{1}{p}.$$

We reach the same conclusion by means of a renewal argument: $E(A)$ is a weighted sum of two events. With probability p edge xy will be available at the first time point, yielding a waiting time of 1. With probability $1-p$, xy will not be available at the first time point. Since the system is memoryless, the expected time until A is available from that time is $E(A)$. So, the total waiting time is

$$E(A) = p(1) + (1-p)(E(A)+1)$$

which simpliifies to $E(A) = \frac{1}{p}$ as before.
(Note: In this system, even if xy has probability 1 of being available and weight 0, we would arrive at y no earlier than at time $t = 1$.)

1.3 Modifying Kruskal's Algorithm for Dynamically Random Graphs

Let's reconsider Kruskal's algorithm on K_4 with all edges having equal associated probability p, each having edge weight 0. Note that in a dynamically random graph such as this one, since not all edges are always available, time spent waiting for edges to become available must also be considered as a cost.

We propose two strategies for building a spanning tree in minimum expected completion time. At each time instant, a decision must be made about which available edge, if any, should be taken. Initially, we will take any edge e_1 among those first available. After the first edge is taken, we now divide the remaining edges into two "classes." "Class A" consists of edges that share one endpoint with e_1, while "Class B" consists of edges that share neither vertex with e_1.

For a first strategy, we propose that if any single edge becomes available, it should be taken, and if edges from both classes become available, edges belonging to "Class B" are preferred.

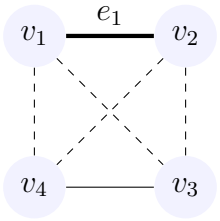


Figure 5: Class A edges are dashed while Class B edges are solid.

Once an edge e_2 from either class is taken, only one more edge, e_3, is needed to complete the spanning tree. Selecting e_2 from Class B provides four options for e_3, whereas selecting e_2 from Class A provides only three (see Figure 6). In either case, to complete the tree we simply take the first appropriate edge that becomes available.

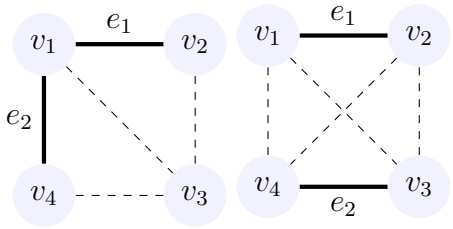


Figure 6: Acceptable edges in step 3 where e_2 is in Class A (left) or Class B (right).

Here is a second strategy: when selecting e_2, we reject a Class A edge if it becomes available first, in order to take a Class B edge, and therefore maximize future options.

We calculate the expected time to complete the spanning tree for each of these strategies. The expected time for the strategy which waits for a Class B edge for e_2 is denoted E_W ("Wait") and the expected time for our the strategy where Class B edges are preferred but Class A edges are accepted is denoted as E_{JG} ("Just Go"). Note that we use the standard notation $q = 1 - p$. Then

$$E_W = \frac{1}{1-q^6} + \frac{1}{1-q} + \frac{1}{1-q^4}$$

and

$$E_{JG} = \frac{1}{1-q^6} + \frac{1}{1-q^5} + (\frac{p}{1-q^5})(\frac{1}{1-q^4}) + (\frac{(1-q^4)(q)}{1-q^5})(\frac{1}{1-q^3}).$$

We show that $E_{JG} < E_W$ (i.e. that our first strategy has the smaller completion time). Since we may assume $p > 0$,

$$\frac{1}{1-q^6} + \frac{1}{1-q^5} + (\frac{p}{1-q^5})(\frac{1}{1-q^4}) + (\frac{(1-q^4)(q)}{1-q^5})(\frac{1}{1-q^3}) < \frac{1}{1-q^6} + \frac{1}{1-q} + \frac{1}{1-q^4}$$

$\Leftrightarrow 0 < q + q^2 + q^3 - q^4 - q^6 - q^7$
It suffices to show that the above polynomial is strictly positive on the interval $0 < q < 1$. (Note: We may assume that $q \neq 0$ since otherwise $p = 1$ and $E_{JG} = E_W$.)

To do this, we rewrite the polynomial by regrouping the terms as $(q - q^4) + (q^2 - q^6) + (q^3 - q^7)$. Since $0 < q < 1$, if $1 \leq m < n$, then $q^n < q^m$. So, $q - q^4 > 0$, $q^2 - q^6 > 0$ and $q^3 - q^7 > 0$. Thus, the polynomial is positive for $0 < q < 0$.

Therefore, we conclude that the first strategy, the one which for e_2 prefers Class B edges but also accepts Class A edges, yields the lower expected completion time regardless of the value of p.

2 TASK Problems

Sometimes, instead of working with graphs, we consider a list of tasks to be completed. This differs from previous problems because there is no interdependence between the tasks; no task depends on any other. We thus define the (k, n)-TASK problem in which we have n tasks, $T_1, T_2, ..., T_n$, each of which has an associated task time t_i and a probability of being available for completion p_i. Without loss of generality, we order the tasks such that $t_1 \leq t_2 \leq ... \leq t_n$. Our goal is to determine a strategy which completes k of the n tasks in the shortest expected amount of time. We denote such a problem by $(k, \{T_\alpha\}_{\alpha \in \mathcal{A}})$, where $\mathcal{A}$ is an indexing set of size n and $T_\alpha = (p_\alpha, t_\alpha)$ for each $\alpha \in \mathcal{A}$.

Definition 2.1. *In the $(1, n)$-TASK problem $\Gamma = (1, \{T_\alpha\}_{\alpha \in \mathcal{A}})$, a strategy S_A (corresponding to $A \subseteq \mathcal{A}$) possibly chooses task T_i if and only if $i \in A$. The expected time to complete the problem Γ following strategy S_A is denoted by E_A.*

Definition 2.2. *In the $(1, n)$-TASK problem $\Gamma = (1, \{T_\alpha\}_{\alpha \in \mathcal{A}})$, a strategy S_A is consecutive if $A = \{1, 2, ..., j\} \subseteq \mathcal{A}$ for some $j \leq n$. If this is the case, we denote S_A by S_j and E_A by E_j.*

Definition 2.3. *In the (k, n)-TASK problem $\Gamma = (k, \{T_\alpha\}_{\alpha \in \mathcal{A}})$, the expected time remaining to complete Γ following some optimal strategy after having already chosen the set of tasks $\{T_\beta\}_{\beta \in B}$ is denoted $ERT_\Gamma(B)$. So $ERT_\Gamma(B)$ is the expected time to complete*

the $\left(k-|B|, \{T_\alpha\}_{\alpha\in\mathcal{A}\setminus B}\right)$*-TASK problem under optimal strategy.*

Lemma 2.4. *Consider the* $(1,n)$*-TASK problem* $\Gamma = (1, \{T_\alpha\}_{\alpha\in\mathcal{A}})$ *and strategy* S_A*, where* $n = max\{A\}$*. Then* $E_A = E_{A\setminus\{n\}}$ *if and only if* $t_n = E_A$*.*

Proof. It suffices to show that $t_n = E_A \Leftrightarrow t_n = E_{A\setminus\{n\}}$. $E_A = E_{A\setminus\{n\}}$ then follows by commutativity. The following equations are equivalent:

$$
\begin{aligned}
t_n =& E_A \\
t_n =& \frac{1}{1-\prod\limits_{i\in A} q_i}\left(1+\sum_{k\in A}\left[p_k t_k \prod_{j\in A,\ j<k} q_l\right]\right) \\
t_n\left(1-\prod_{i\in A\setminus\{n\}} q_i\left(q_n+p_n\right)\right) =& 1+\sum_{k\in A\setminus\{n\}}\left[p_k t_k \prod_{j\in A\setminus\{n\},\ j<k} q_l\right] \\
t_n =& \frac{1}{1-\prod\limits_{i\in A\setminus\{n\}} q_i}\left(1+\sum_{k\in A\setminus\{n\}}\left[p_k t_k \prod_{j\in A\setminus\{n\},\ j<k} q_l\right]\right).
\end{aligned}
$$

That is, $t_n = E_{A\setminus\{n\}}$, as desired.

□

2.1 The Consecutive Strategy Theorem for $(1,n)$-TASK

Theorem 2.5. *(The Consecutive Strategy Theorem for* $(1,n)$*-TASK)*
In a $(1,n)$*-TASK problem, every minimal optimal strategy is a consecutive strategy.*

Proof. We proceed by induction on n. We leave the case $n = 1$ to the reader. Suppose inductively for $k = 1, 2, ..., n-1$ that in the $(1,k)$-TASK problem every minimal optimal strategy is a consecutive strategy.

Consider an arbitrary $(1,n)$-TASK problem $\Gamma = (1, \{T_\alpha\}_{\alpha\in\mathcal{A}})$ and minimal optimal strategy S_A for $A \subseteq \mathcal{A}$. Suppose S_A is not a consecutive strategy. It suffices to derive a contradiction.

Suppose task n is not accepted by S_A. Then by the induction hypothesis, S_A has to be a consecutive strategy for the $(1,n-1)$-TASK problem $\Lambda = (1, \{T_\alpha\}_{\alpha\in\mathcal{A}})$. So we may assume that task n is accepted by S_A.

Since S_A is not a consecutive strategy, there is a task $m < n$ that is not accepted

by S_A. Note that S_A is a minimal optimal strategy for Λ among tasks 1 through $m-1$ and $m+1$ through n. So by the induction hypothesis S_A is the strategy that accepts the set of tasks $A = \{1, 2, ..., n\}\backslash\{m\}$. Let S'_B be the strategy that accepts the set of tasks $B = \{1, 2, ..., n\}$. Our goal is to show that S'_B is strictly better than S_A (i.e. $E_B < E_A$) for Γ. We do this by showing that S'_B is at least as good as S_A in all situations, and strictly better than S_A in some situations. Let S'' be the set of tasks available at the time when one or more tasks first becomes available. Let j be the one task performed when implementing S_A. Note that j is the least element of S''.

Situation 1: $j \neq m$. Then the expected remaining times for S_A and S'_B are equal.

Situation 2: $j = m$ and $S''\backslash\{m\} \neq \emptyset$. Let $k = min\{S''\backslash\{m\}\}$. The expected remaining time for S_A and S'_B are t_k and t_m respectively. Since $t_k \geq t_m$, the expected remaining time of S'_B is at most the expected remaining time of S_A.

Situation 3: $S'' = \{m\}$. If only task n had been available, S_A would have taken it with expected remaining time t_n rather than the alternative of waiting another turn. Since S_A is optimal, $t_n \leq E_A$. Suppose for contradiction that $t_n = E_A$ Then $E_A = E_{A\backslash\{n\}}$ by Lemma 2.4. So $A\backslash\{n\}$ is an optimal strategy accepting fewer than $|A|$ tasks, a contradiction. So $t_n < E_A$. Thus $t_m \leq t_n < E_A$, which means that taking m (as S'_B does) is strictly better than waiting (as S_A does). So S'_B is better than S_A. □

Lemma 2.6. *If $E_{j-1} < E_j$, then $E_{j-1} < t_j$.*

Proof. Since $E_{j-1} < E_j$, we have

$$\frac{1}{1 - \prod_{i=1}^{j-1} q_i}\left(1 + \sum_{k=1}^{j-1}\left[p_k t_k \prod_{0<l<j-1} q_l\right]\right) < \frac{1}{1 - \prod_{i=1}^{j} q_i}\left(1 + \sum_{k=1}^{j}\left[p_k t_k \prod_{0<l<j} q_l\right]\right).$$

Simplification yields $E_{j-1} < t_j$ □

Proposition 2.7. *The distribution of the E_j's is unimodal.*

Proof. Let $T_1, T_2, ..., T_n$ be tasks in an $(1, n)$-TASK problem with $t_1 \leq t_2 \leq ... \leq t_n$. Suppose for the sake of contradiction that $E_J < E_{J+1}$ and $E_J \geq E_I$ for some $J < I$. From the second inequality we have

$$\frac{1}{1 - \prod_{i=1}^{J} q_i}\left(1 + \sum_{k=1}^{J}\left[p_k t_k \prod_{0<l<k} q_l\right]\right) \geq \frac{1}{1 - \prod_{i=1}^{I} q_i}\left(1 + \sum_{k=1}^{I}\left[p_k t_k \prod_{0<l<k} q_l\right]\right).$$

Simplification yields the inequality

$$\left(1+\sum_{k=1}^{J}\left[p_k t_k \prod_{0<l<k} q_l\right]\right)\left(\frac{1}{1-\prod_{i=1}^{J} q_i}-\frac{1}{1-\prod_{i=1}^{I} q_i}\right) \geq \frac{\sum_{k=J+1}^{I}\left[p_k t_k \prod_{J<l<I} q_l\right]}{1-\prod_{i=1}^{I} q_i} \geq \frac{t_{J+1}\sum_{k=J+1}^{I}\left[p_k \prod_{J<l<I} q_l\right]}{1-\prod_{i=1}^{I} q_i}.$$

since $t_{J+1} \leq t_m$ for all $m \geq J+1$. Further simplification yields

$$\frac{1}{1-\prod_{i=1}^{J} q_i}\left(1+\sum_{k=1}^{J}\left[p_k t_k \prod_{0<l<k} q_l\right]\right)\left(\frac{1-\prod_{J<l\leq I} q_l}{\sum_{k=J+1}^{I}\left[p_k \prod_{J<l<k} q_l\right]}\right) \geq t_{J+1}.$$

Since the probability at any time instant that at least one task of $\{T_{J+1},...,T_I\}$ is available is equal to both $\left(1-\prod_{J<l\leq I} q_l\right)$ and $\sum_{k=J+1}^{I-1}\left[p_k \prod_{J<l<k} q_l\right]$, we have that

$$\left(\frac{1-\prod_{J<l\leq I} q_l}{\sum_{k=J+1}^{I-1}\left[p_k \prod_{J<l<k} q_l\right]}\right) = 1.$$

Hence $E_J \geq t_{J+1}$, a contradiction to $E_J < E_{J+1}$ by Lemma 2.6. □

Remark 2.8. *Since every minimal optimal strategy is a consecutive strategy, the best consecutive strategy must be optimal. Thus (since the sequence of expected times of the consecutive strategies is unimodal) an optimal strategy can be found by starting with E_1 and calculating expected completion times for successive consecutive strategies until the first j is encountered such that $E_j \leq E_{j+1}$, at which point we can conclude that E_j is the unique consecutive minimal optimal strategy.*

2.2 Counterexample to a Universal Claim of Theorem 2.5

It is natural to question whether every optimal strategy to the $(1,n)$-TASK problem is a consecutive strategy. We assert that this is not the case and provide the following counterexample.
Consider the $(1,3)$-TASK problem with tasks

T_1: $p_1 = \frac{1}{2}$ $t_1 = 2$
T_2: $p_2 = \frac{1}{3}$ $t_2 = 4$
T_3: $p_2 = \frac{2}{3}$ $t_3 = 4$

The reader can verify that $E_1 = 4$,$E_2 = 4$ and $E_3 = 4$.
We conclude that E_1 is a minimal optimal strategy, since it has the least expected time of the consecutive strategies. Consider the strategy $S_{\{1,3\}}$ that accepts the tasks T_1 and T_3. Then $E_{\{1,3\}} = \dfrac{1}{1-(\frac{1}{2})(\frac{1}{3})}\left(1+(\frac{1}{2})(2)+(\frac{1}{2})(\frac{2}{3})4\right) = 4$. So $S_{\{1,3\}}$ is an optimal strategy that is not consecutive with respect to the given ordering.

The reader may question the ordering of the tasks since T_2 and T_3 have equal task times. To address this concern, we consider the similar $(1,3)$-TASK problem with tasks:

T_1: $p_1 = \frac{1}{2}$ $\quad$ $t_1 = 2$
T_2: $p_2 = \frac{2}{3}$ $\quad$ $t_2 = 4$
T_3: $p_2 = \frac{1}{3}$ $\quad$ $t_3 = 4$

Again, the reader can verify that $E_1 = 4$,$E_2 = 4$,$E_3 = 4$. We conclude that E_1 is a minimal optimal strategy. Now consider the strategy $S_{\{1,3\}}$ that accepts the tasks T_1 and T_3. Then $E_{\{1,3\}} = \dfrac{1}{1-(\frac{1}{2})(\frac{2}{3})}\left(1+(\frac{1}{2})(2)+(\frac{1}{2})(\frac{1}{3})4\right) = 4$, so $S_{\{1,3\}}$ is an optimal strategy that is not consecutive with respective to the new given ordering.

But the two orderings above are the only possible orderings such that $t_1 \leq t_2 \leq t_3$. Thus, this set of tasks provides a counterexample regardless of how we choose to order the two tasks T_2 and T_3 with equal task times. We conclude that an optimal strategy is not necessarily a consecutive strategy.

2.3 The Recursive Strategy for (k,n)-TASK

We now consider (k,n)-TASK.
Theorem 2.9. *Let $\Gamma = (k, \{T_\alpha\}_{\alpha\in\mathcal{A}})$ be a (k,n)-TASK problem and $B \subseteq \mathcal{A}$ such that $|B| \leq k-1$. Then $ERT_\Gamma(B)$ equals the expected time to complete the $\left(1, \{(p_\alpha, t_\alpha + ERT_\Gamma(B\cup\{\alpha\}))\}_{\alpha\in\mathcal{A}\setminus B}\right)$-TASK problem following optimal strategy.*

Proof. Consider $\left(k-|B|, \{(T_\alpha)\}_{\alpha\in\mathcal{A}\setminus B}\right)$-TASK problem . Suppose that in that problem task β is the first task we end up taking. From that point, under optimal strategy, after taking task β the expected time remaining $ERT_\Gamma(B) = t_\beta + ERT_\Gamma(B\cup\{\beta\})$. But in the $(1, \{(p_\alpha, t_\alpha + ERT_\Gamma(B\cup\{\beta\}))\})$-TASK problem, taking task β involves $t_\beta + ERT_\Gamma(B\cup\{\beta\})$ additional time to complete that problem. Since both problems initially have $\mathcal{A}\setminus B$ as the set of possible available tasks, their optimal strategies take the same expected time to complete. □

The Recursive Strategy on Γ, the $\left(k, \{T_\alpha\}_{\alpha\in\mathcal{A}}\right)$-TASK problem (where $|\mathcal{A}| = n$).

- Consider every subset $B \subseteq \mathcal{A}$ such that $|B| = k - 1$. For each B, calculate the expected time $ERT_\Gamma(B)$.
- Now suppose that, for some $j \leq k - 1$, $ERT_\Gamma(B)$ has been calculated for all $B \subseteq \mathcal{A}$ with $|B| = j$. Let $C \subseteq \mathcal{A}$ with $|C| = j - 1$ be arbitrary. Then by Theorem 2.9, $ERT_\Gamma(C)$ is equal to the expected time of a best strategy to complete the $\left(1, \{(p_\alpha, t_\alpha + ERT_\Gamma(C \cup \{\alpha\}))\}_{\alpha \in \mathcal{A} \setminus C}\right)$-TASK problem. Calculate $ERT_\Gamma(C)$ for each such C. In this manner, we eventually calculate $ERT_\Gamma(\varnothing)$, which is the expected time to complete Γ under optimal strategy.
- At any given point in the problem having taken the tasks in B, we follow the minimal optimal (consecutive?) strategy for completing the $\left(1, \{(p_\alpha, t_\alpha + ERT_\Gamma(B \cup \{\alpha\}))\}_{\alpha \in \mathcal{A} \setminus B}\right)$-TASK problem until we choose a task.

2.4 Computation Time

The following algorithm can be run to find the consecutive minimal optimal strategy, as well as the expected time of the $(1, \{p_i, t_i\}_{i \in \mathcal{A}})$-TASK problem with $|\mathcal{A}| = n$ tasks. Prior to running the algorithm, the tasks must be ordered and indexed by increasing task time. The output $I - 1$ indicates that the consecutive strategy accepting tasks up to $I - 1$ is our minimal optimal strategy. The final value of E is the expected time to complete the problem.

$Q \leftarrow 1$
$S \leftarrow 1$
$E \leftarrow \infty$
$I \leftarrow 1$
while $I \leq n$ AND $E > t_I$ **do**
  $S \leftarrow S + p_I t_I Q$
  $Q \leftarrow Q(1 - p_I)$
  $E \leftarrow \dfrac{S}{1 - Q}$
  $I \leftarrow I + 1$
end while
return I-1, E

Each time the loop is executed, four multiplications and divisions are completed. The maximum number of loop executions is n. Thus the algorithm has $O(n)$ with regard to multiplications and divisions and $O(n \ \ln(n))$ with regard to ordering [1]. Storage requirements for this algorithm are small, with only four stored variables in addition to the problem specifics. In the $(k, \{p, t\})$-TASK problem, computation time is determined by the number of times the algorithm for the $(1, \{p, t\})$-TASK problem is run. The algorithm is run for each subset of $\mathcal{A}$ with cardinality greater than $n - k$. Since there are at most 2^n subsets of $\mathcal{A}$, multiplications and divisions are at most on order $n2^n$. However,

when k is held constant with respect to n, a tighter bound can be found by considering that the number of subsets is at most on the order of $\binom{n}{k}$, which simplifies to the order of n^{k-1}. Thus, the order for multiplications and divisions is n^k. Ordering is completed once after the expected remaining times for each subset within a group of subsets with the same cardinality has been computed. Since k such orderings occur, each with order $n\ \ln(n)$, the overall order with respect to ordering is $kn\ \ln(n)$.

3 Eulerian Tours and CPP

We now return to graphs and consider strategies for completing Eulerian tours and the CPP on graphs with probabilities as well as edge weights. Our goal is to find strategies which complete these problems while minimizing expected time of completion.

3.1 Cut-Edge Lemma

Suppose in some directed graph G that $V(G)$ partitions into subsets V_1, V_2. Let $G_1 = G[V_1], G_2 = G[V_2]$, and suppose $\overrightarrow{xy}$ is the only edge from G_1 to G_2 and $\overrightarrow{yx}$ is the only edge from G_2 to G_1. Our objective is to complete the Chinese Postman Problem, that is, starting at some vertex v in G_1, to traverse each edge in G and return to v in the minimum amount of time in G.

Lemma 3.1. *If we are located at vertex y, we will not return to G_1 until all of the edges in G_2 have been traversed.*

Proof. Let $n_1(t)$ be the number of untraversed edges in the edge set $E(G_1) \cup \overrightarrow{yx}$ and $n_2(t)$ be the number of untraversed edges in $E(G_2)$ at some time t. Suppose inductively that $n_1(t_0) + n_2(t_0) < k$. Then from vertex y, every optimal strategy will not accept $\overrightarrow{yx}$ until $n_2 = 0$.

Suppose $n_1(t_0) + n_2(t_0) = k$. Let S be an optimal strategy, and suppose we are at vertex y. Let e be the first edge traversed when following S from y at this stage.

Case 1: $e \in E(G_2)$. Since S is an optimal strategy, it will only accept edges that take us to locations with lower expected remaining time than our current location. So, after taking edge e, S will not return to y until some untraversed edge in G_2 has been taken. Therefore, when we return to y at time t_1, $n_1(t_1) + n_2(t_1) < k$. So by the induction hypothesis, S will not accept $\overrightarrow{yx}$ until $n_2(t) = 0$. So, overall, S will not accept $\overrightarrow{yx}$ until $n_2(t) = 0$.

Case 2: Suppose that $e = \overrightarrow{yx}$. If $n_2(t^*) = 0$ then we're done. If $n_2(t^*) > 0$, then after traversing e we must take $\overrightarrow{xy}$ at some point to traverse the remaining edges in G_2 , say this occurs at time t^*. Let T_1 be the actions taken by strategy S from point x before taking $\overrightarrow{xy}$ again. So T_1 is a random variable because the availability of edges in G_1 is random. Then let $E(T_1)$ be the expected time to complete T_1. So at time t^* we find ourselves at y with $n_1(t^*) + n_2(t^*) < k$, again because the optimality of S ensures that we take some untraversed edge in G_2 before returning to y. Note that if $n_2(t_0) = 0$, no optimal strategy would have $e = \overrightarrow{yx}$ since it could not traverse a new edge before returning to y. By the Induction Hypothesis, S will not accept $\overrightarrow{yx}$ again until $n_2(t) = 0$. Let $E(G_2)$ be the expected remaining time to traverse the untraversed edges in G_2 and return to y. Let $E(T_1^*)$ be the expected time spent traversing the remaining untraversed edges in G_1 and returning to v. So the expected remaining time, given that T_1 time units were spent in G_1 before next crossing $\overrightarrow{xy}$, to complete the problem following S is:

$$t(\overrightarrow{yx}) + E(T_1) + t(\overrightarrow{xy}) + E(G_2) + \frac{1}{p(\overrightarrow{yx})} + t(\overrightarrow{yx}) + E(T_1^*).$$

Consider the strategy S' that does not accept $\overrightarrow{yx}$ until $n_2(t) = 0$. Then the expected remaining time to complete the problem is:

$$E(G_2) + \frac{1}{p(\overrightarrow{yx})} + t(\overrightarrow{yx}) + E(G_1)$$

Where $E(G_1)$ is the expected time to traverse the untraversed edges in G_1 and return to v following an optimal strategy to do so.
Since $E(G_1)$ and $E(T_1) + E(T_1^*)$ are both expected time to traverse the untraversed edges in G_1 (at t_0) and since $E(G_1)$ is an optimal time/ under an optimal strategy for doing so, $E(G_1) \leq E(T_1) + E(T_1^*)$.

So:

$$E(G_2) + \frac{1}{p(\overrightarrow{yx})} + t(\overrightarrow{yx}) + E(G_1) \leq E(G_2) + \frac{1}{p(\overrightarrow{yx})} + t(\overrightarrow{yx}) + E(T_1) + E(T_1^*) <$$
$$E(G_2) + \frac{1}{p(\overrightarrow{yx})} + t(\overrightarrow{yx}) + E(T_1) + E(T_1^*) + t(\overrightarrow{yx}) + t(\overrightarrow{xy})$$

Since $t(\overrightarrow{yx}), t(\overrightarrow{xy}) > 0 \Rightarrow\Leftarrow$ □

3.2 Fleury's Algorithm on Trees

Let T be a tree and consider its digraph analogue, $\overrightarrow{T}$ in which each edge from v_i to v_j is replaced by two directed edges, $\overrightarrow{v_1v_2}$ and $\overrightarrow{v_2v_1}$. We consider solving the CPP on such a graph. Because each edge in a tree is a cut edge, each pair of vertices in the digraph analogue of a tree is connected by exactly two edges, one in each direction. Moreover, removing these two edges breaks T into two components. So, by Lemma 3.1 an optimal strategy will not take any edge more than once, since doing so would mean either returning to a component in which every edge has already been traversed or returning to an earlier component before completing every edge in the current component. Since, in this type of graph, an optimal strategy will take no edge more than once, any optimal strategy will actually construct an Euler tour when solving the CPP. We consider the following modified version of Fluery's algorithm to complete the problem.Starting at a vertex $s \in V(T)$, wait for an outgoing edge to become available. Of the first outgoing edges to become available, take the edge with the lowest p value. Repeat from current location, but only take an outgoing edge to a previously visited vertex if all other edges incident to the current vertes have been traversed. The expected time using our modified Fleury's algorithm to traverse T is:

$$E_F = \frac{e}{p} + \frac{1}{1-q^{d(s)}} + \sum_{v \in V} \left(\sum_{0<i<d(v)} \frac{1}{1-q^i} \right) \tag{7}$$

where e is the number of edges in T, each of them with the same assigned probability p and where edge e_i has arbitrary weight t_i. Note that the equation above does not take into account the weights t_i. To correct this, simply add $\sum_i t_i$ to the equation.

We proceed by induction on $V(T)$. The reader can verify that our formula works for a tree on two vertices. Suppose inductively that our formula works for all trees with fewer vertices than tree T. It suffices to show that our formula works for T. We compute the expected time E_F to traverse T starting at s as follows:
From s, the expected waiting time for an edge to become available is $\dfrac{1}{1-q^{d(s)}}$. Let $\overline{sn}$ be the edge selected by Fleury's Algorithm. Let T_1 and T_2 be the components of $T - \overline{sn}$ containing s and n, respectively. From vertex n, we must traverse T_2 before crossing $\overrightarrow{ns}$ by Fleury's Algorithm.
Case 1: T_1 and T_2 are nontrivial. Our expected traversal time for T_2 is (by inductive hypothesis):

$$E_{T_2} = \frac{e(T_2)}{p} + \frac{1}{1-q^{d(n)-1}} + \sum_{\substack{v \in V(T_2) \\ v \neq n}} \left(\sum_{0<i<d(v)} \frac{1}{1-q^i} \right) + \sum_{0<i<d(n)-1} \frac{1}{1-q^i}$$

Once T_2 is traversed we must cross $\overrightarrow{ns}$, and the expected waiting time for that edge to become available is $\frac{1}{p}$. From s, our additional expected traversal time to complete T_1 (the rest of T) is:

$$E_{T_1} = \frac{e(T_1)}{p} + \frac{1}{1-q^{d(s)-1}} + \sum_{\substack{v \in V(T_1) \\ v \neq s}} \left(\sum_{0<i<d(v)} \frac{1}{1-q^i} \right) + \sum_{0<i<d(s)-1} \frac{1}{1-q^i}$$

Putting it all together, our total traversal time for all of T is:

$$\begin{aligned} E_F &= \frac{1}{1-q^{d(s)}} + \frac{e(T_2)}{p} + \frac{1}{1-q^{d(n)-1}} + \sum_{\substack{v \in V(T_2) \\ v \neq n}} \left(\sum_{0<i<d(v)} \frac{1}{1-q^i} \right) + \sum_{0<i<d(n)-1} \frac{1}{1-q^i} \\ &\quad + \frac{1}{p} + \frac{e(T_1)}{p} + \frac{1}{1-q^{d(s)-1}} + \sum_{\substack{v \in V(T_1) \\ v \neq s}} \left(\sum_{0<i<d(v)} \frac{1}{1-q^i} \right) + \sum_{0<i<d(s)-1} \frac{1}{1-q^i} \\ &= \frac{e}{p} + \frac{1}{1-q^{d(s)}} + \sum_{\substack{v \in V(T_2) \\ v \neq n}} \left(\sum_{0<i<d(v)} \frac{1}{1-q^i} \right) + \sum_{\substack{v \in V(T_1) \\ v \neq s}} \left(\sum_{0<i<d(v)} \frac{1}{1-q^i} \right) \\ &\quad + \sum_{0<i<d(n)} \frac{1}{1-q^i} + \sum_{0<i<d(s)} \frac{1}{1-q^i} \\ &= \frac{e}{p} + \frac{1}{1-q^{d(s)}} + \sum_{v \in V} \left(\sum_{0<i<d(v)} \frac{1}{1-q^i} \right), \text{ which is what we wanted to show.} \end{aligned}$$

Case 2: When T_1 is trivial and T_2 is nontrivial.
As in Case 1, our expected waiting time for an edge to become available is $\frac{1}{1-q^{d(s)}}$, and our expected traversal time for T_2 remains the same as in Case 1. Once T_2 is traversed, our expected waiting time to cross $\overrightarrow{ns}$ is $\frac{1}{p}$. This will complete T, so the total traversal time is:

$$E_F = \frac{1}{1-q^{d(s)}} + \frac{e(T_2)}{p} + \frac{1}{1-q^{d(n)-1}} + \sum_{\substack{v \in V(T_2) \\ v \neq n}} \left(\sum_{0<i<d(v)} \frac{1}{1-q^i} \right) + \sum_{0<i<d(n)-1} \frac{1}{1-q^i} + \frac{1}{p} + 0.$$

Note that since $d(s) = 1$, $\sum_{1<i<d(s)} \frac{1}{1-q^i} = 0$. So,

$$E_F = \frac{1}{1-q^{d(s)}} + \frac{e}{p} + \sum_{\substack{v \in V(T_2) \\ v \neq n}} \left(\sum_{0<i<d(v)} \frac{1}{1-q^i} \right) + \sum_{0<i<d(n)} \frac{1}{1-q^i} + \sum_{1<i<d(s)} \frac{1}{1-q^i}$$

$$= \frac{e}{p} + \frac{1}{1-q^{d(s)}} + \sum_{v \in V} \left(\sum_{0<i<d(v)} \frac{1}{1-q^i} \right), \text{ which is what we wanted to show.}$$

Case 3: T_1 is nontrivial and T_2 is trivial. This is similar to Case 2.
Case 4: Both T_1 and T_2 are trivial. Then T is a tree on two vertices, which was our base case.

3.3 Convexity Argument for Eulerian Circuits

A digraph has at least one Eulerian circuit if and only if it has at most one non-trivial strong component and has in-degree equal to out-degree at each vertex. Note that in any such digraph, the number of Eulerian circuits is $c\prod_i (d(i)-1)!$, where c is the number of in-trees at any vertex [2]. We look at the implications of this theorem for digraph analogs of trees. Since each edge in a tree is replaced by one in-edge and one out-edge, every vertex in a digraph analog of a tree has in-degree equal to out-degree. Note that the number of in-trees at any vertex in a digraph analog of a tree is 1. Therefore, each digraph analog of a tree has $\prod_i (d(i)-1)!$ Eulerian circuits. Given that the tree has n vertices, we have that $\sum_{i=1}^{n} d(v_i) = 2(n-1)$. Consider $f(m) = (m-1)!$. Then

$$\begin{aligned} a \leq b &\Rightarrow a \leq b-1 \\ &\Rightarrow \frac{a!}{(a-1)!} \leq \frac{(b-1)!}{(b-2)!} \\ &\Rightarrow ((a+1)-1)!((b-1)-1)! \leq (a-1)!(b-1)! \\ &\Rightarrow f(a+1) + f(b-1) \leq f(a) + f(b). \end{aligned}$$

So, $f(m)$ is discretely concave up. By convexity, this means that a digraph analog of a tree has the most Eulerian circuits when the degrees of the vertices are furthest apart and the fewest when the degrees of the vertices are closest together. This means that stars have the most Eulerian circuits and paths have the fewest Eulerian circuits.

3.4 CPP on Digraph Analogues of Trees

Consider the formula for expected time to complete the CPP on the digraph analogue of a tree. Given a tree with n vertices and a start vertex with out-degree $d(s)$, what

distribution of $2(n-1)$ directed edges, each with probability $p > 0$, give a tree with lowest expected completion time? Note that the sum of the out-degrees of the vertices, $\sum_{i=1}^{n} d(v_i)$, equals $2(n-1)$. Consider the discrete function $f(m) = \sum_{0<i<m} \frac{1}{1-q^i}$. Since $0 \leq q < 1$,

$$\begin{aligned} a < b &\Rightarrow q^{a+1} > q^b \\ &\Rightarrow \frac{1}{1-q^{a+1}} > \frac{1}{1-q^b} \\ &\Rightarrow \sum_{0<i<a} \frac{1}{1-q^i} - \sum_{0<i<a+1} \frac{1}{1-q^i} < \sum_{0<i<b-1} \frac{1}{1-q^i} - \sum_{0<i<b} \frac{1}{1-q^i} \\ &\Rightarrow f(a) + f(b) < f(a+1) + f(b-1). \end{aligned}$$

So, $f(m)$ is discretely concave down. Thus, by convexity, the expected time of the CPP problem is minimized when the out-degrees of the vertices are furthest apart and maximized when the vertex out-degrees are as close together as possible. Thus, digraph analogues of stars have the lowest expected times and digraph analogues of paths have the highest expected times.

4 Conclusions

We leave the readers with several ideas for research extensions: While we were able to find strategies to solve traversal and spanning tree problems in several specific instances, we would like to find more general results. For instance, is there an optimal strategy for finding Euler tours in any graph with arbitrary p and t values in which an Euler tour exists? We looked at a Kruskal like problem in K_4, but we wonder if a strategy could be determined for all complete graphs. We also are concerned about the computational time required to execute our strategies. While the computation time for the Recursive Strategy for (k, n)-TASK is manageable for small values of k and n, as these increase, computation time increases exponentially. We question if an optimal strategy with a more efficient computation time exists.

References

[1] K. Rosen: *Discrete Mathematics and Its Applications* (Seventh Edition) McGraw-Hill Publishing Company (2011).

[2] D.B. West: *Introduction to Graph Theory* (Second Edition) Prentice Hall (2001).