Discrete Applied Mathematics 88 (1998) 167-180

DISCRETE APPLIED MATHEMATICS

Graph traversals, genes and matroids: an efficient case of the travelling salesman problem[☆]

Dan Gusfield[a,*], Richard Karp[b], Lusheng Wang[c], Paul Stelling[d]

[a] *Department of Computer Science, University of California at Davis, Davis, CA 95616, USA*
[b] *Department of Computer Science and Engineering, Univesity of Washington, Seattle, WA, 98195, USA*
[c] *Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong*
[d] *The Aerospace Corporation, El Segundo, CA 90245, USA*

Received 15 September 1996; received in revised form 2 December 1997; accepted 4 December 1997

Abstract

In this paper we consider graph traversal problems (Euler and Travelling Salesman traversals) that arise from a particular technology for DNA sequencing – sequencing by hybridization (SBH). We first explain the connection of the graph problems to SBH and then focus on the traversal problems. We describe a practical polynomial time solution to the Travelling Salesman Problem in a rich class of directed graphs (including edge weighted binary de Bruijn graphs), and provide bounded-error approximation algorithms for the maximum weight TSP in a superset of those directed graphs. We also establish the existence of a matroid structure defined on the set of Euler and Hamilton paths in the restricted class of graphs. 1998 Published by Elsevier Science B.V. All rights reserved.

Keywords: Travelling salesman problem; Euler tours; DNA sequencing; De Bruijn graphs; Approximation algorithms; Graph algorithms

1. Problem statement and main results

The results presented in this paper can be described either in terms of the TSP problem, or in terms a particular weighted Euler path problem. The Euler version is closer to the original motivation, so we first discuss the Euler problem and then translate it to the Travelling Salesman problem.

1.1. Definitions

An Euler *path* in a directed graph G is a directed path that traverses every edge in G exactly once. An Euler *tour* is an Euler path that starts and ends at the same

☆ Research partially supported by Dept. of Energy grant DE-FG03-90ER60999.

* Corresponding author. E-mail: gusfield@cs.ucdavis.edu.

node. A digraph that contains an Euler path is called an Euler digraph. A *2-path* is a directed subpath consisting of two consecutive edges (containing three nodes denoted v_1, v_2, v_3). Node v_2 is called the *midpoint* of the 2-path. Every 2-path in the graph has a non-negative *weight* given as input to the problem. Any given Euler path of m edges contains $m-1$ consecutive 2-paths, and the *weight* of an Euler path is the sum of the weights of those $m-1$ different 2-paths. An *optimal* Euler path (or tour) is an Euler path (tour) whose weight is maximum over all Euler paths.

1.2. Main results

The main result of the paper is that an optimal Euler path (or Euler tour if there is one) can be found in nearly linear time in any Euler digraph where the in-degree and out-degree of every node is bounded by two. Moreover, the set of Euler paths (tours) in these graphs has a matroid structure which can be used to reveal additional properties of the Euler paths (tours) in the graph. Translated to the TSP, the result is that the Travelling Salesman Problem can be solved in nearly linear time in any line-digraph where the in-degree and out-degree of any node is bounded by two. Again, in such graphs, the set of Hamilton paths (tours) has a matroid structure that can be used to reveal additional properties of these paths (tours). We also establish that finding an optimal Euler path is NP-hard when the in-degree is permitted to be as large as four. However, we provide approximation algorithms that are guaranteed to achieve a weight of one-fourth the weight of the optimal Euler path in any Euler digraph, and a weight of one-third if the in- and out-degrees are bounded by three. The conference version of this paper appears in [5].

2. The biological context of the originating problem

The work reported here grew out of a computational problem that arises in a technology called *DNA sequencing by hybridization* (SBH). In sequencing by hybridization, one attempts to learn the entire sequence of a long DNA string S by first determining which k-length substrings occur in S, where k is a fairly small number (in current proposals, around ten). Exploiting the overlap patterns of the k-length substrings, one tries to reconstruct the original string S, or determine some less precise features of S. Present technology can only report whether any particular k-length substring occurs in S and cannot tell how many times it occurs. So unique occurrence is generally assumed, and will be assumed in this paper. However, since the length of S is known, any string that violates this assumption is easily identified. We let $\mathscr{L}$ be the list of k-length substrings that occur in S.

Definition. The *SBH problem* is to determine as much as possible about the original DNA string S from list $\mathscr{L}$. In particular, if possible, uniquely determine the original string S from list $\mathscr{L}$.

Clearly, S is the shortest common superstring of $\mathscr{L}$, but the set $\mathscr{L}$ has more structure than an arbitrary instance of the superstring problem, because any two consecutive k-length substrings in S overlap by $k-1$ characters. That structure can be exploited to reduce the SBH problem to questions about Euler paths in a directed graph. That reduction was developed and explored first by Pavel Pevzner [9, 10].

2.1. SBH and Euler paths

Given list $\mathscr{L}$ of k-length substrings obtained from a DNA string S, the directed graph $G(\mathscr{L})$ is constructed as follows: Create 4^{k-1} nodes, each labeled with a distinct $k-1$-length DNA string. Then for each string ϕ in $\mathscr{L}$, direct an edge from the node labeled with the leftmost $k-1$ characters of ϕ to the node labeled with the rightmost $k-1$ characters of ϕ. That edge is labeled with the rightmost character of ϕ. Any node of $G(\mathscr{L})$ that does not touch any edges can be removed. Graph $G(\mathscr{L})$ is a subgraph of the well-known de Bruijn graph with alphabet size four and tuple size $k-1$. For complete examples, see [4, 5, 9, 10].

Every $k-1$ tuple of $\mathscr{L}$, other than the two at the left and right ends of the original string S, is the intersection of two adjacent k-tuples in S. The $k-1$ tuple at the start of S is the left end of one k-tuple, and the $k-1$ tuple at the right end of S is the right end of one k-tuple. Hence, either all nodes in $G(\mathscr{L})$ have an in-degree equal to their out-degree, or all but two nodes do. Therefore, $G(\mathscr{L})$ necessarily has an Euler path, and may have an Euler tour.

Conversely, an Euler path in $G(\mathscr{L})$ specifies a string S' in the following way. String S' begins with the label of the first node on the path and follows thereafter with the concatenation, in order, of the labels on the edges it traverses. For example, a path that traverses edges labeled $\{AC, CA, AC, CG, GC, CA, AA, AC, CT, TT, TA, AA, AA\}$ specifies the string $S' = ACACGCAACTTAAA$.

Any Euler path in $G(\mathscr{L})$ creates a string S' that has the same set of k-length substrings as the original string S used to create $G(\mathscr{L})$. Hence, S can be uniquely reconstructed from $\mathscr{L}$ if and only if there is a unique Euler path in $G(\mathscr{L})$. The realistic situation is that $G(\mathscr{L})$ contains more than one Euler path. Classic theorems about de Bruijn graphs (for example see [1]) establish that for any k there is a string of length roughly 4^k over a four-letter alphabet, whose graph $G(\mathscr{L})$ has $24^{(4^{(k-1)})}/4^k$ Euler paths.

The results in this paper originate from the goal of distinguishing one "more promising" Euler path, and its resulting string S', from the others. The general criteria we use is to evaluate S' by the substrings it contains of length *greater* than k. This approach is attractive because there is often some partial, indirect or *a priori* information, in addition to the observed k-length substrings, about what the original string S might be, and that information can be used to establish weights (based on likelihoods for example) that particular substrings of length longer than k are contained in S. For example [7], experimental methods have been developed that give the *rough* location of each k-tuple found in S. The weight of any 2-path (corresponding to two overlapping k-tuples) could then be a function of how close those two k-tuples are (roughly) determined to be. As another example, *pooled* information about the $k+1$ tuples in S may be available. That information indicates whether one or more of a *set* of $k+1$ tuples appears in S, but does not specify which particular $k+1$ tuples appear. A third example, based on protein database search, is detailed in [6].

For concreteness and flexibility of the model, and for tractability, we evaluate any particular string S' by the $k+1$-length substrings that it contains. A $k+1$-length substring corresponds to a 2-path in $G(\mathscr{L})$, motivating the purely graph theoretic problem of finding an *optimal Euler path* in a digraph.

3. The TSP version of the problem

We can convert the problem of finding an optimal Euler path (or tour) in a digraph G into the problem of finding an optimal Travelling Salesman path (or tour) in a directed graph $L(G)$.

Definition. The line digraph $L(G)$ is derived from a directed graph G by creating a node in $L(G)$ for each edge in G. $L(G)$ contains a directed edge from node v to node v' if the edge that defined v followed by the edge that defined v' forms a 2-path in G.

Definition. A *Hamilton* path in a graph is a path in the graph that visits each node exactly once. A Hamilton tour is a Hamilton path followed by a single edge back to the node that starts the path.

It is well known that a line digraph $L(G)$ has a Hamilton path (or tour) if and only if G has a Euler path (tour). Hence the problem of finding an Euler path (or tour) in G is equivalent to the problem of finding a Hamilton path (or tour) in $L(G)$. Further, if each edge of $L(G)$ is weighted by the weight of its corresponding 2-path in G, then an optimal Euler path (or tour) in G corresponds to a maximum weight Hamilton path (or tour) in $L(G)$. So the problem of finding an optimal Euler path (tour) in G maps into a Travelling Salesman problem in $L(G)$.

We will discuss the problem of finding the *maximum* weight Travelling Salesman *tour* in $L(G)$, leaving the other cases to the reader. We will show that a polynomial time solution exists when the in- and the out-degree of each node in $L(G)$ is bounded by two. To simplify the exposition, we assume that the in-and out-degree of each node in $L(G)$ is exactly two, and again leave the other cases to the reader. Note that if G is a digraph where the in- and out-degree of each node is exactly two, then the in- and out-degree of each node in its line-digraph $L(G)$ is also exactly two.

Definition. We assume that the in- and out-degree of each node v in G is exactly two. The four edges incident with v are associated with four nodes and four edges in $L(G)$, called a *quad* (see Fig. 1). The four edges of the quad partition in a unique way into

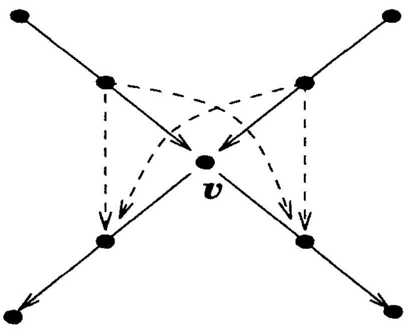

Fig. 1. Node v and the dark edges are in graph G. The quad in $L(G)$ defined by the edges incident with v is shown with dashed edges.

two pairs of edges, such that the edges in each pair are incident with all four nodes of the quad. In this partition, the pair of edges with higher weight is called the *high pair* and the other pair of edges is called the *low pair*. Define the *loss* of the quad as the summed edge weight of the high pair minus the summed edge weight of the low pair.

The following observation, whose proof is immediate, is one of the keys to the efficient TSP solution when the in- and out-degree of each node in $L(G)$ is exactly two.

Lemma 3.1. *Let $\mathscr{T}$ be an arbitrary Hamilton tour in $L(G)$. In any quad q of $L(G)$, either both the edges in the high pair of q are used in $\mathscr{T}$, and neither of the low pair edges are used, or both the edges in the low pair of q are used in $\mathscr{T}$, and neither of the high pair edges are used.*

Corollary 3.1. *The maximum weight TSP tour has weight at most the sum, over all quads, of the weights of the edges in high pairs. Similarly, the minimum weight TSP tour has weight at least the sum, over all quads, of the weights of the edges in the low pairs.*

3.1. The TSP theorem and algorithm

Theorem 3.1. *If a digraph $L(G)$ is a line digraph of some graph G, and each node in G has in-degree exactly two and out-degree exactly two, then the (maximum or minimum) Travelling Salesman tour in $L(G)$ can be found in polynomial time.*

We establish Theorem 3.1 through the following algorithm that finds a maximum weight Travelling Salesman tour. A minor change will find a minimum weight Travelling Salesman tour.

Let H be the subgraph of $L(G)$ consisting of all the nodes of $L(G)$, but only containing the edges in the high pairs of $L(G)$. At each node w of $L(G)$ there is exactly one edge from a high pair into w and one edge from a high pair out of w. Therefore, H consists of one or more node disjoint cycles. If H consists of a single cycle then, by Corollary 3.1, H is a maximum weight Hamilton tour in $L(G)$. Otherwise, contract each cycle c_i of H into a single node (denoted n_i), and extend an undirected edge between every two nodes n_i and n_j if and only if there is a quad in $L(G)$ containing one edge of cycle c_i and one edge of cycle c_j. The weight of edge (n_i, n_j) is set to the smallest *loss* of any of the quads containing one edge in c_i and one edge in c_j. Let H' denote the resulting undirected graph.

Next, compute a *minimum* spanning tree of H', and form the proposed Hamilton tour $\mathscr{T}$ in $L(G)$ as follows: If q is a quad in $L(G)$ corresponding to an edge in the minimum spanning tree, include the edges of its low pair into the proposed Hamilton tour $\mathscr{T}$; If q does not correspond to an edge in the minimum spanning tree, include the edges of its high pair into $\mathscr{T}$.

Lemma 3.2. *The set of edges $\mathscr{T}$ specified above forms a maximum weight Hamilton tour of $L(G)$.*

Proof. We already noted that if H consists of a single cycle, then it is a maximum weight Hamilton tour. So assume that H is not a single cycle and therefore every Hamilton tour of $L(G)$ contains some low pair(s) of edges.

By Lemma 3.1, the weight of any Hamilton tour is the weight of the edges in H minus the total loss specified by the quads whose low pairs of edges are in the tour. Further, every Hamilton tour of $L(G)$ forms a connected subgraph in the underlying undirected graph of $L(G)$ and, due to Lemma 3.1, it defines a spanning tree of H'. It follows that the maximum weight Hamilton tour of $L(G)$ has weight at most equal to the weight of the edges in the proposed tour $\mathscr{T}$.

What remains is to show that $\mathscr{T}$ forms a Hamilton tour of $L(G)$. Let H have r cycles, so every spanning tree of H' contains exactly $r-1$ edges. To start, choose any of these $r-1$ edges, for example an edge corresponding to quad q. Suppose the edges of the high pair of q are contained in cycles c_i and c_j of H (see Fig. 2(a)). Remove

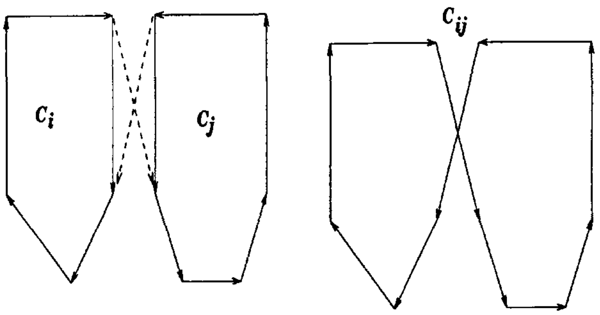

Fig. 2. (a) Two cycles c_i and c_j and the quad q whose high edges are contained in c_i and c_j. (b) The single cycle C_{ij} created by the merge of c_i and c_j.

those edges of q from H, and insert the edge of the low pair of q. This creates a new merged cycle formed from c_i and c_j (see Fig. 2(b)), and results in a directed graph $\overline{H}$ containing exactly $r-1$ cycles. Since the minimum spanning tree of H' contains no cycles, and since quads are edge disjoint, each of the remaining $r-2$ edges of the tree continues to specify a quad in H whose high-pair edges are in two different cycles in $\overline{H}$. Therefore, each of the $r-1$ remaining edges in the minimum spanning tree specifies a quad of H that can be used to merge two cycles of $\overline{H}$. Continuing in this way for $r-1$ merges, we conclude that the edges of $\mathcal{T}$ form a Hamilton tour of $L(G)$. □

A practical time bound for this algorithm is $O(n \log n)$: $L(G)$ has only $O(n)$ edges, so the creation of H' requires $O(n)$ time, and the time for the whole algorithm is dominated by the time to find a minimum spanning tree of H'. That can be done in theory in $o(n \log n)$ time, but the precise theoretical bound is not a concern in this paper.

Corollary 3.2. *An optimal Euler tour can be found in polynomial time in a digraph where every node has in-degree exactly two and out-degree exactly two.*

A very different, $O(n^2)$-time, algorithm for the problem was developed in [6], and was partly detailed in [5].

3.2. The matroid structure

The proof of Theorem 3.1 establishes a matroid structure involving the set of Hamilton paths in $L(G)$ and the set of Euler paths in G.

Let the set of quads define the ground set of a matroid. A subset of quads Q is defined to be independent if there exists a Hamilton tour $\mathcal{T}$ that contains the low pair of edges from each quad in Q. (Any remaining edges of $\mathcal{T}$ come from quads not in Q and need not necessarily be from the low edge pairs of those quads). Then the above independent sets form a matriod. The size of the base of the matroid is the number of nodes in $L(G)$. All the general matroid theorems apply, including the following.

Theorem 3.2. *If the loss of each quad is distinct, then the maximum weight Travelling Salesman tour is unique, as is the minimum weight Travelling Salesman tour.*

3.3. Binary de Bruijn graphs: an important special case

One important special class of graphs where Theorem 3.1 applies is the class of *binary de Bruijn graphs*. A binary de Bruijn graph for parameter k contains 2^k nodes, each given a unique binary number of k bits. There is a directed edge from the node with binary number i to the node with binary number j if and only if number j can be created by shifting number i right by one bit and then adding another bit to the left end. Binary de Bruijn graphs and their associated de Bruijn sequences have been extensively studied, and are described in many books and articles. A very entertaining article about de Bruijn graphs appears in Scientific American [11], and much of an entire book [3] has been written about them. So the most easily communicated special case of Theorem 3.1 is

Theorem 3.3. *Even when the edges of a binary de Bruijn graph are given arbitrary weights, the TSP problem on binary de Bruijn graphs can be solved in* $O(n \log n)$ *time.*

It is well known that binary de Bruijn graphs (and non-binary generalizations) constructed on substrings of any length k, have both Hamilton and Euler tours. In fact, each Euler tour of the de Bruijn graph for parameter k defines a Hamilton tour of the de Bruijn graph for parameter $k+1$. However, because the number of Hamilton (or Euler) tours grows as an exponential function of the number of nodes, it should not be obvious that when arbitrary edge weights are permitted, the Travelling Salesman Problem on binary de Bruijn graphs can be solved in polynomial time.

In Section 4, we will establish an NP-completeness result that makes an extension of Theorem 3.1 unlikely for graphs with high in- and out-degree. However, that result does not rule out the possibility that the TSP problem might be polynomial on de Bruijn graphs with $k>2$. That remains an open problem.

4. NP-completeness

Recall that all the results above assume that the in- and out-degrees of each node are bounded by two. What happens when that bound is raised? The situation when the bound is three remains open, but for higher degrees the problem is NP-hard. To establish that, we state the following problem that is known to be NP-hard [2].

Connected node cover in a planar graph with degree either 4 *or* 1.

Instance: A planar graph $G=(V,E)$ with degree either 4 or 1, and an integer k.

Question: Does there exist a node cover V' for G satisfying $|V'| \leqslant k$, such that the subgraph of G induced by v' is connected.

For simplicity, we call the above problem the *planar cover* problem.

Theorem 4.1. *In a directed Euler graph where all nodes have in-degree bounded by four and out-degree bounded by four, the problem of finding an optimal Euler tour is* NP-*hard.*

Proof. We reduce the planar cover problem to the Euler problem stated above.

Given a (undirected) planar graph $G=(V,E)$, where the degree of a node is either 4 or 1, we construct a directed graph $G_{\mathrm{d}}=(V,E')$. The set of nodes in G_{d} is the same as in G. Each edge in G becomes two directed edges in both directions, i.e., if $(v,u) \in E$, then both (v,u) and (u,v) are in E'. These two (directed) edges form a "circle" and hence also form two 2-paths. Such a circle is called a *basic circle*. The cost given to any 2-path in G_d is set to 1 if the 2-path defines a basic circle, otherwise, it is set to 0. For a basic circle with ends u and v, the basic circle is *cut* at u in an Euler path P, if the 2-path with midpoint u does not appear in P. Note that, any basic circle is cut at least at one end in any Euler path.

Now we show that if there is a connected node cover V' with k nodes in G, then there is an Euler tour P with cost $|E| - k + 1$ in G_d, where $|E|$ is the number of (undirected) edges in E. If any node v in V' has degree one, then remove v from V' and put the unique neighbor of v into V'. The result is another node cover of G with the same or fewer nodes. Hence, without loss of generality, we assume that the degree of every node in V' is 4. Given a connected node cover V' with k nodes, we can find a spanning tree T for V' with $k - 1$ edges. The subgraph G_T of G_d induced by T is a subgraph of G_d containing the nodes in T and the two corresponding directed edges for each edge in T. Since T is a spanning tree of G, and each edge of G corresponds to two directed edges in G_d, any depth-first traversal of G_T forms an Euler tour of G_T. Any node not in V' is adjacent to some node in V'. Hence an Euler tour of G_d can be formed from the Euler tour of G_T by splicing in every 2-path in G_d that corresponds to a basic circle and that starts at a node in V', but has a middle node not in V'. Since no node of V' has degree one, each leaf of T must have one or more neighbors not in V'. The result is an Euler tour of G_d containing exactly $|E| - k + 1$ 2-paths that follow basic circles. Hence the cost of this Euler tour is exactly $|E| - k + 1$.

We now show the converse. Suppose there is an Euler tour P of G_d with cost c. We will construct a connected node cover of size $k \leqslant |E| - c + 1$. Any node v is called a *cover* node if some basic circle is cut at v in the Euler tour P. Clearly, if the in/out-degree of a node v is 1, then v is not a cover node because the Euler tour must traverse the unique 2-path (which is a circle) that contains v. Hence only nodes of in- and out-degree 4 can be cover nodes. Let V' be the set of all cover nodes. At least one end of each basic circle must be cut in any Euler tour, and hence is a cover node. That is, V' forms a node cover of G. To see that V' is connected, note that the first node on P is always a cover node, and if P traverses the edge from u to v but does not immediately return to u, then v will be a cover node also. Applying this fact as the Euler tour P visits all nodes of G_d, it follows that V' is connected.

An edge (u,v) in G is *double* cut if the corresponding basic circle is cut at both u and v. Let NT be a set of double cut edges in E. Then NT is a network connecting the nodes in V'. Eliminating some of the edges in NT, we can form a spanning tree T for V'. Without decreasing the cost, we can modify P to P' such that only the basic circles corresponding to an edge in T are double cut. Since T has $|V'| - 1$ edges, the obtained Euler tour has cost $c' = |E| - |V'| + 1$. Thus, $c \leqslant c' = |E| - |V'| + 1$. That is, $|V'| \leqslant |E| - c + 1$. Therefore, given an Euler tour P with cost c, we can find a connected node cover of size $k \leqslant |E| - c + 1$. □

DNA can be considered as a string over an alphabet of four characters, so the Theorem 4.1 is disappointing. However, Corollary 3.2 may still apply to the SBH problem even when the alphabet is not binary, since the theorem only requires that the in and out degrees be bounded by two, not that the underlying problem come from a binary alphabet. Moreover, in enumerative or branch and bound algorithms for an optimal Euler problem with higher degrees, when the successive enumerated choices reduce the remaining graph to one with binary degree bounds, then an optimal algorithm can be applied. This may be quite effective in practice. For example, if there are n nodes with in-degree and out-degree of three each, then one can naively enumerate 6^n choices to find an optimal Euler path. But 3^n choices suffice when the algorithm for optimal Euler paths is also employed.

5. Approximation algorithms

We first consider Euler digraphs without any degree bounds, and present an algorithm that is guaranteed to get within *one-quarter* of the weight of an optimal (maximum weight) Euler Path. The algorithm is a simple greedy algorithm that successively examines 2-paths in decreasing order of their weight. At each step, the algorithm examines a particular 2-path and determines whether there is an Euler path containing that particular 2-path and all the previously fixed 2-paths. If so, the new 2-path is fixed along with the previously fixed 2-paths. To fix a 2-path, replace it by a single edge from the start node to the end node of the 2-path. Since the graph has an Euler path, this method will find one; we call it the *greedy path*, and denote it P^g. To efficiently test whether a paraticular 2-path can be added to the previously fixed 2-paths, simply replace the 2-path with a single edge (as if it were to be fixed) and apply the well-known neccessary and sufficient conditions for a directed graph to have an Euler path.

To analyze the goodness of the greedy path, we need to the following definition and key lemma.

Definition. Any Euler path (or a proposed path) can be described as a set of 2-paths. Given an Euler path P, a *switch* of P creates a new set of 2-paths by removing some 2-paths in the description of P, and including an equal number of other 2-paths. An *Euler switch* of P is a switch where the resulting set of 2- paths defines an Euler path.

Lemma 5.1. *Let e_1, e_2 be any arbitrary 2-path. Let P be any Euler path that does not contain the 2-path e_1, e_2. If there exists an Euler path that* does *contain the 2-path e_1, e_2, then there is an Euler switch of P that removes at most four 2-paths in P, and that results in an Euler path containing the 2-path e_1, e_2.*

Proof. Suppose that the Euler path P does not contain the 2-path e_1, e_2, but another Euler path P' does contain it. Then there are two edges e_3 and e_4 in the graph such that

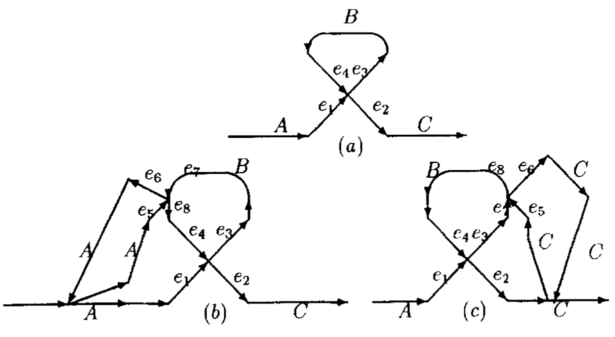

Fig. 3. Case 1.

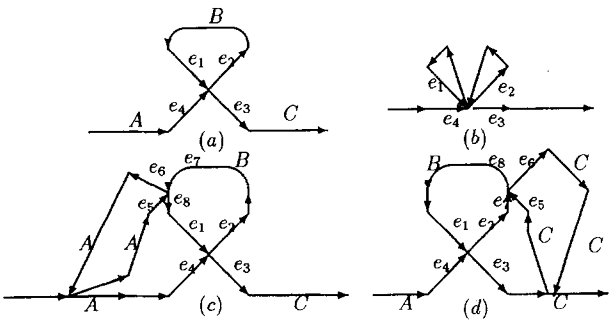

Fig. 4. Case 2.

the 2-paths e_1, e_3 and e_4, e_2 are contained in P (see Fig. 3(a)). Let v be the common node for the four edges e_1, e_2, e_3 and e_4. There are two cases.

Case 1: P traverses the 2-path e_1, e_3 before it traverses the 2-path e_4, e_2 (see Fig. 3(a)). Let A be the subpath of P from the starting node to v (visits v via e_1), B be the subpath of P starting with the edge e_3 and ending at edge e_4, and C be the rest of P following A and B (see Fig. 3(a)). Let V_A, V_B and V_C be the set of nodes on the subpaths A, B, and C, respectively. A and B are *disjoint* if $V_A \cap V_B = \{v\}$, and e_1 is the only edge in A that is incident to v. Similarly, B and C are *disjoint* if $V_B \cap V_C = \{v\}$, and e_2 is the only edge in C that is incident to v. If A and B are not disjoint, then there is an Euler switch of P that removes the four 2-paths e_5, e_6; e_7, e_8; e_1, e_3; and e_4, e_2 in P, and that results in an Euler path containing the 2-path e_1, e_2 (see Fig. 3(b)). The case that C and B are not disjoint is similar (see Fig. 3(c)). If V_A and V_B are disjoint and V_B and V_C are disjoint, then the 2- path e_1, e_2 is not in any Euler path. The reason is that in any Euler path, there is a subpath which starts with edge e_1, enters the nodes in V_B, then leaves the nodes in V_B via e_2 and never comes back to V_B. This contradicts the assumption that there exists an Euler path containing the 2-path e_1, e_2.

Case 2: P traverses (e_4, e_2) before it traverses (e_3, e_1).

Let A be the subpath of P from the starting node to v (visits v via e_4), B be the subpath of P starting from the edge e_2 and ending at edge e_1, and C be the rest of P

after A and B (see Fig. 4(a)). Let V_A, V_B and V_C be the set of nodes on the subpaths A,B, and C, respectively. If A visits some nodes in $V_B - \{v\}$ before it visits e_2, then there is an Euler switch that removes the four 2-paths e_5,e_6; e_7,e_8; e_1,e_3; and e_4,e_2 and that results in an Euler path containing the 2-path e_1,e_2 (see Fig. 4(c)). Similarly, if C visits some nodes in $V_B - \{v\}$ after the starting point v of C, then there is an Euler switch that removes four 2-paths and that results in an Euler path containing the 2-path e_1,e_2 (see Fig. 4(d)). Otherwise, e_2 and e_1 are the starting and ending edges for the subpath B. If B visits v via edges not in $\{e_1,e_2\}$, then there is an Euler switch that removes three 2-paths and that results in an Euler path containing the 2-path e_1,e_2 (see Fig. 4(c)). If B does not visit v except via edges e_1 and e_2, then no Euler path contains the 2-path e_1,e_2. The reason is that any Euler tour/path enters the nodes in V_B via edge e_2, leaves the nodes in V_b via e_1 and never comes back to the nodes in V_B. □

Using Lemma 5.1 we can prove

Theorem 5.1. *The greedy path P^g has weight within one-quarter that of an optimal Euler path.*

Proof. Let P^* be an optimal Euler path. Using Lemma 5.1, we will transform P^* to the greedy path P^g, and at each step loose at most 3/4 the weight of the 2-paths deleted from P^*.

To begin, consider the set of 2-paths in P^*, and identify and fix any of those 2-paths that are also in P^g. To fix a 2-path, replace its two edges with a single edge, so that it will remain in all the subsequent Euler paths. Color red the remaining 2-paths of P^*, i.e., the 2-paths in $P^* - P^g$. Now Let e_1,e_2 be the greatest weight 2-path in $P^g - P^*$.

By Lemma 5.1, there is an Euler switch of P^* that brings in e_1,e_2 and three other 2-paths, while removing only four 2-paths of P^*. Call the resulting Euler path P'. At this point, fix all the entering 2-paths (including e_1,e_2) that are contained in P^g. Once fixed, they can never be removed. We want to compare the weight of the 2-paths brought into P' to the weight of the 2-paths removed.

We claim there cannot be a red 2-path with greater weight than e_1,e_2. If there were, let p be the red 2-path with the largest weight, and note that p would be contained in an Euler path where all the higher weight 2-paths are both in P^* and in P^g. Therefore, p should have been chosen when the greedy algorithm examined it. Since there are no red 2-paths with weight greater than e_1,e_2, every 2-path in P^* of weight greater than e_1,e_2 is fixed. Therefore the four 2-paths that are removed during the Euler switch each have weight less than or equal to the weight of e_1,e_2. It follows that the weight of the entering 2-paths is at least one-quarter that of the removed two-paths.

To continue the transformation, find the largest 2-path e_2,e_3 in $P^g - P'$. By the same reasoning as before, every 2-path in P' with weight greater than that of e_2,e_3 must be in P^g. Again do a switch of P' that brings in e_2,e_3 along with three other 2-paths, and removes four 2-paths from P', each of weight no greater than e_2,e_3. Hence the weight of the entering 2-paths is at least one-quarter that of the exiting red 2-paths. Again, we fix any entering 2-paths from P^g. Continuing in this way, the final resulting Euler path contains only the 2-paths of P^g, and has weight at least one-quarter that of P^*. □

Corollary 5.1. *If a digraph $L(G)$ is a line digraph of an Euler graph G, then a Hamilton path of $L(G)$ can be found which has total weight at least one-fourth that of the maximum weight Hamilton path of $L(G)$.*

Note that for the maximization criteria of the Travelling Salesman problem, there is a simple approximation method that gets within $1/2$ of optimal, and there are more complex recent results that obtain even better bounds [8]. Those methods are not of use in the present problem because they allow the path to include (zero length) edges that are not part of the input graph. Stated differently, those results are on complete graphs. The results established in this section hold for Euler paths on the original input graph G, or Hamilton paths on $L(G)$.

The theorem above applies to any Euler digraph regardless of its in- and out-degrees. We next improve the result the special case that the in- and out-degrees are bounded by three.

Theorem 5.2. *When the in- and out-degrees are bounded by three, an Euler path can be found in polynomial time whose weight is within one-third that of P^*.*

Proof. The algorithm is again greedy but picks *three* 2-paths at a time, rather than one 2-path. At any node v there are six possible ways to choose the three 2-paths with middle node v. Our new algorithm computes the weight of each of these six ways for each node and sorts the $6n$ choices, largest first. Then, it constructs an Euler path by testing, in order of decreasing weight, whether there is an Euler path which contains (along with the 2-paths already fixed) the next choice of (three) 2-paths in the sorted list. If yes, then those three 2-paths are fixed in the Euler path under construction. Call the resulting Euler path the *greedy-three path*. We claim that it has weight within one-third that of P^*.

The argument is similar to the proof of Theorem 5.1. We transform an optimal Euler path P^* to the greedy-three path by a series of Euler switches using Lemma 5.1 again. To switch in the three chosen 2-paths that have middle node v, we will switch in one 2-path at a time. As noted before, when switching in one 2-path whose middle is v, some 2-paths at exactly one other node are changed. After switching in the first two desired 2-paths whose middle is v, the third desired 2-path will have also been switched in by default. The result is that we have switched in the desired three 2-paths whose middle is v, while affecting 2-paths at only two nodes other than v. Hence at most three sets of three 2-paths from P^* are removed in order to bring in the desired three 2-paths. With the kind of reasoning used in the proof of Theorem 5.1, the weight of the three 2-paths brought in will be no less than the weight, at each of the three affected nodes, of the three 2-paths of P^* that are removed. Therefore the weight of the 2-paths brought in is at least one-third the weight of what is removed. Continuing in this way until the greedy-three path is created, proves the theorem. □

Acknowledgements

We would like to thank Yefim Dinitz, R. Ravi and Ron Shamir for early conversations on the problem discussed here. We would also like to thank the referees for careful reading and helpful comments.

References

[1] S. Even, Graph Algorithm, Computer Science Press, Mill Valley, CA, 1979.
[2] M. Garey, D. Johnson, Computers and Intractability, Freeman, San Francisco, 1979.
[3] S. Golomb, Shift Register Sequences, Holden-Day, San Francisco, 1967.
[4] D. Gusfield, Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology, Cambridge University Press, Cambridge, 1997.
[5] D. Gusfield, R. Karp, L. Wang, P. Stelling, Graph traversals, genes and matroids: an efficient special case of the Travelling Salesman Problem, Proc. 7th Symp. on Combinatorial Pattern Matching, Lecture Notes in Computer Science, vol. 1075, Springer, Berlin, 1996, pp. 304–319.
[6] D. Gusfield, L. Wang, P. Stelling, Graph traversals, genes and matroids: An efficient special case of the Travelling Salesman Problem, CSE-96-3, Technical report, Department of Computer Science, University of California, Davis, 1996.
[7] S. Hannenhalli, W. Fellows, H. Lewis, S. Skiena, P. Pevzner, Positional sequencing by hybridization, Comput. Appl. BioSci. 12 (1996) 19–24.
[8] R. Kosaraju, J. Park, C. Stein, Long tours and short superstrings, Proc. 35th IEEE Symp. on Foundations of Computer Science, 1994, pp. 166–177.
[9] P. Pevzner, l-tuple DNA sequencing: computer analysis, J. Biomol. Struct. Dyn. 7 (1989) 63–73.
[10] P.A. Pevzner, DNA physical mapping and alternating eulerian cycles in colored graphs, Algorithmica 12 (1994) 77–105.
[11] S. Stein, The mathematician as explorer, Scientific American (May 1961) 149–163.