The Greedy Algorithm for Shortest Common Superstrings

Course "Discrete Biological Models" (Modelli Biologici Discreti)

Zsuzsanna Lipták

Laurea Triennale in Bioinformatica
a.a. 2014/15, fall term

Problem: Shortest Common Superstring

Recall the definition

Shortest Common Superstring (SCS)

Input: A collection $\mathcal{F}$ of strings.
Output: A shortest possible string S s.t. for every $f \in \mathcal{F}$, S is a superstring of f.

N.B.
The problem is NP-hard (= "very difficult" for now), therefore we will not be able to find an algorithm which

1. always finds an optimal solution (here: a *shortest* superstring), and
2. is efficient, i.e. runs in polynomial time.

The greedy algorithm for SCS finds a superstring which is not necessarily shortest, but has at most 4 times the optimal length.

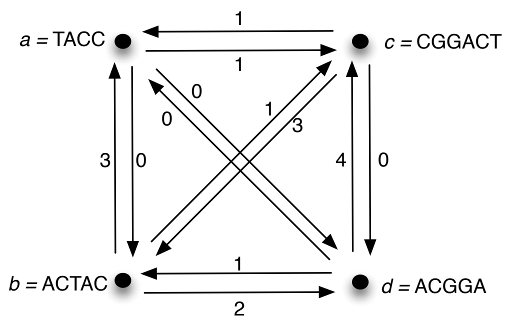
Substring-freeness

N.B.
We will assume from here on that $\mathcal{F}$ is substring-free, i.e. there are no $f \neq f' \in \mathcal{F}$ s.t. f is a substring of f'.

If $\mathcal{F}$ is not substring-free, then make it substring-free: define $\mathcal{F}' := \mathcal{F} \setminus \{f : \exists f' \in \mathcal{F}, f' \neq f, f \text{ substring of } f'\}$. Then $\mathcal{F}'$ is substring-free and has the same superstrings as $\mathcal{F}$ (why?). So we can replace $\mathcal{F}$ by $\mathcal{F}'$ and receive the same solutions.

Overlap graphs

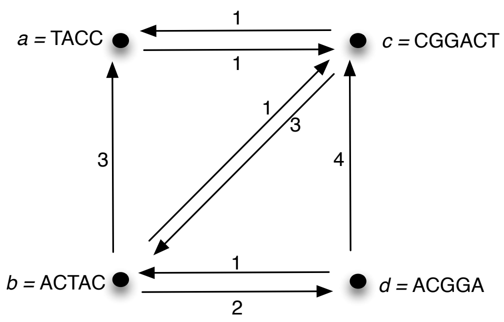
Definition
Given $\mathcal{F}$, the overlap graph $OG(\mathcal{F}) = (V, E)$ is a weighted directed graph, where $V = \mathcal{F}$, $E = \{(u, v) : u \neq v \in V\}$, and $w : E \mapsto \mathbb{R}$ is a weight function, with $w(uv) = \max\{|t| : t \text{ suffix of } u, t \text{ prefix of } v\}$.



Overlap graphs

Definition

Given $\mathcal{F}$, the overlap graph $OG(\mathcal{F}) = (V, E)$ is a weighted directed graph, where $V = \mathcal{F}$, $E = \{(u, v) : u \neq v \in V\}$, and $w : E \mapsto \mathbb{R}$ is a weight function, with $w(uv) = \max\{|t| : t \text{ suffix of } u, t \text{ prefix of } v\}$.



In the drawing, we omit edges with 0 weight.

Hamiltonian paths

We are looking for paths in $OG(\mathcal{F})$ which use every vertex exactly once. Such paths are called Hamiltonian paths.

Examples

E.g. $P_1 = (b, d, c, a)$, $P_2 = (d, c, b, a)$, $P_3 = (a, c, b, d)$.

Definition

For a path P in $OG(\mathcal{F})$, let $S(P)$ be the string defined by P, e.g. $S(P_1) = ACTACGGACTACC$, $S(P_2) = ACGGAACTACC$, $S(P_3) = TACCGGACTACGGA$.

(Defined inductively on the length of the path: for a path of length 0, $P = (f)$, $S(P) = f$. Let $P = (f_0, \ldots, f_{k+1})$ and let $S = S(P')$, where $P' = (f_0, \ldots, f_k)$, be already constructed. Then $S(P) = Sv$ where v is the suffix of f_{k+1} of length $|f_{k+1}| - w(f_k, f_{k+1})$.)

Hamiltonian paths and superstrings

Question

Does every Hamiltonian path correspond to a superstring of $\mathcal{F}$ (i.e. one that is a superstring of all $f \in \mathcal{F}$)?

Answer

Yes, since it traverses every vertex f, so by construction $S(P)$ is a superstring of f.

Question

Does every superstring of $\mathcal{F}$ correspond to a Hamiltonian path?

Answer

No, e.g. $\mathcal{F} = \{a, b\}$ where $a = ACAC, b = CACT$. Then $w(a, b) = 3$ and $w(b, a) = 0$, and there are only two Hamiltonian paths: $P_1 = (a, b)$ and $P_2 = (b, a)$, with $S(P_1) = ACACT$ and $S(P_2) = CACTACAC$. So the superstrings $ACACGGCACT$ and $ACACACT$ do not correspond to any Hamiltonian paths. (First has extra characters, second less than maximum overlap.)

Hamiltonian paths and superstrings

Minimality

A superstring S of $\mathcal{F}$ is called minimal if no proper subsequence of S is a superstring of $\mathcal{F}$. (I.e. if you remove some characters, it is no longer a superstring).

Shortest superstrings (= there is no superstring which is shorter) are also minimal: otherwise there would be a shorter one which is also a superstring.

N.B.

All shortest superstrings correspond to Hamiltonian paths.

Weights of Hamiltonian paths and superstrings

Weight of paths

For a path $P = (f_0, \ldots, f_k)$, let $w(P) = \sum_{i=0}^{k-1} w(f_i, f_{i+1})$.

Examples

E.g. $w(P_1) = 7$, $w(P_2) = 10$, $w(P_3) = 6$.

Lemma

Let P be a Hamiltonian path in $OG(\mathcal{F})$. Then

$$|S(P)| = ||\mathcal{F}|| - w(P),$$

where $||\mathcal{F}|| = \sum_{f \in \mathcal{F}} |f|$ is the total length of strings in $\mathcal{F}$.

Therefore what we are looking for are heaviest paths in $OG(\mathcal{F})$ (heavier path $\leftrightarrow$ shorter superstring).

Greedy algorithm

Warning

Remember, we will not be able to find an algorithm that is both efficient and always produces a heaviest path.

Our greedy algorithm is conceptually simple, efficient, and approximates the optimal solution (however, does not always solve the problem exactly, i.e. may not always produce a heaviest path = shortest superstring).

Greedy algorithm

The algorithm builds up a Hamiltonian path by selecting edges one by one.

Ideas

1. always try to take heaviest edge so far not selected
2. every node must have no more than one incoming and one outgoing selected edge
3. avoid cycles

Greedy algorithm

We are building up a partial path by adding edges one by one.

Avoiding cycles

When do we obtain a cycle by adding an edge (u, v) to a partial path? If and only if there was already a path (directed or undirected) between u and v. I.e. if and only if u and v belonged to the same connnected component in the partial path.

Connected components

Connected graphs

An undirected graph $G = (V, E)$ is called connected if for every $u, v \in V$ there exists a path between u and v.

Connected components

Connected graphs

An undirected graph $G = (V, E)$ is called connected if for every $u, v \in V$ there exists a path between u and v.

Connected digraphs

A directed graph is called connected if the underlying undirected graph (i.e. take away the orientation from the edges) is connected.

Connected components

Connected graphs

An undirected graph $G = (V, E)$ is called connected if for every $u, v \in V$ there exists a path between u and v.

Connected digraphs

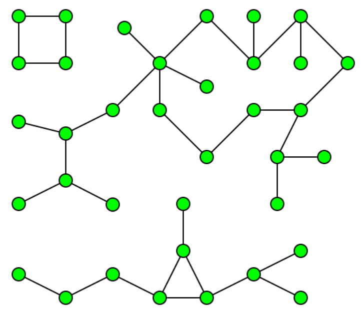
A directed graph is called connected if the underlying undirected graph (i.e. take away the orientation from the edges) is connected.

Connected components

In a graph $G = (V, E)$, a connected component is a maximal subset C of V s.t. for every $u \neq v \in C$, there is a path between u and v. Equivalently, the induced subgraph $(C, E(C))$ is maximally connected[1]. Thus, V can be uniquely partitioned into connected components: $V = C_1 \cup \ldots \cup C_k$, where k is the number of connected components. G is connected iff $k = 1$.

[1] maximal = if you add one element, then the property no longer holds. Here: If you add an element, then no longer connected.

Connected components



A graph with 3 connected components[2].

[2] source: Wikipedia

Greedy algorithm

Algorithm *Greedy algorithm*
Input: weighted directed graph $OG(\mathcal{F})$ with n vertices
Output: Hamiltonian path in $OG(\mathcal{F})$

for $i \leftarrow 1$ to n
  do $in[i] \leftarrow 0$; $out[i] \leftarrow 0$; $Conn(i) \leftarrow \{i\}$ — Initialize
Sort edges by weight, heaviest first
for each edge (f, g) in sorted order — Process edges
  do if $in[g] = 0$ and $out[f] = 0$ and $Conn(f) \neq Conn(g)$ — Test
    then select (f, g);
      $in[g] \leftarrow 1$; $out[f] \leftarrow 1$; — Update
      $Union(Conn(f), Conn(g))$
   if there is only one component — Terminate
    then break
Return selected edges

Greedy algorithm: data structures

We need the following data structures:

1. arrays $in[], out[]$ of length n
2. sets which maintain the connected components of the partial path being constructed and a function *Conn* which, for every element i, identifies its connected component

Union-Find

The second can be done efficiently with a Union-Find data structure. Given a ground set X, this maintains disjoint subsets of X and supports these basic operations:

1. MakeSet(x) – generates a singleton set $\{a\}$ $(x \in X)$
2. FindSet(x) – identifies which set x is in
3. Union(S, T) – makes the union of two sets S and T

Union-Find

The second can be done efficiently with a Union-Find data structure. Given a ground set X, this maintains disjoint subsets of X and supports these basic operations:

1. MakeSet(x) – generates a singleton set $\{a\}$ $(x \in X)$
2. FindSet(x) – identifies which set x is in
3. Union(S, T) – makes the union of two sets S and T

For $|X| = n$, a series of $m > n$ union and/or find operations can be done in time practically linear[3] in m.

[3] More precisely, in $O(m\alpha(m, n))$, where $\alpha(m, n)$, the *inverse Ackermann function*, grows so slowly that it can be considered a constant.

Analysis of Greedy algorithm

1. Initialization (lines 1,2): $3n$ constant time operations, $O(n)$ time
2. Sorting edges (line 3): n^2 edges[4], comparison constant-time, so $O(n^2 \log n)$ time
3. Processing edges (lines 4-10): for every edge, 2 lookups ($in[g]$ and $out[f]$, line 5) and 2 find-operations ($Conn(f)$, $Conn(g)$, line 5), 2 updates ($in[g]$ and $out[f]$, line 7) and 1 union-operation (line 8), and 1 more lookup (line 9, no. of components); so for each edge, 3 union/find operations and 5 constant-time operations (lookups, updates); altogether there are n^2 edges (not all are necessarily processed but may be); so in total at most $3n^2$ union/find operations and $5n^2$ constant-time operations $= O(n^2)$ time
4. Return edges: $n - 1$ edges $= O(n)$ time

Total time: $O(n) + O(n^2 \log n) + O(n^2) + O(n) = O(n^2 \log n)$.

[4] In actual fact, there are $n(n-1) \leq n^2$ edges, but the analysis is simpler with n^2.

Greedy algorithm

Note that this algorithm always returns a Hamiltonian path if the input graph is an overlap graph, since these are complete graphs. It would not work on any directed weighted graph (why?) Even on an overlap graph, the algorithm does not necessarily return a Hamiltonian path with maximum weight.

However, it is efficient, since it runs in $O(n^2 \log n)$ time on a fragment collection $\mathcal{F}$ with $|\mathcal{F}| = n$ (n different fragments).