Linear Approximation of Shortest Common Superstrings

Sarah Spall and Scott Bauer

December 15, 2014

Introduction

A superstring of a set of strings S, is a string w, such that each string $s \in S$ is a substring of w. The goal of shortest common superstring is to find the shortest possible w, such that each string $s \in S$ is a substring of w. This problem has found application in both DNA sequencing and data compression. DNA is made up of four neucleotides, A, C, G, and T. Sequencing molecules, determining a "string" over the alphabet of neucleotides, is/was an important part of biology. To create the sequence of a molecule, fragments (pieces of the molecule), are used to find the shortest common superstring of the fragments; this seems to work well in practice. [3]

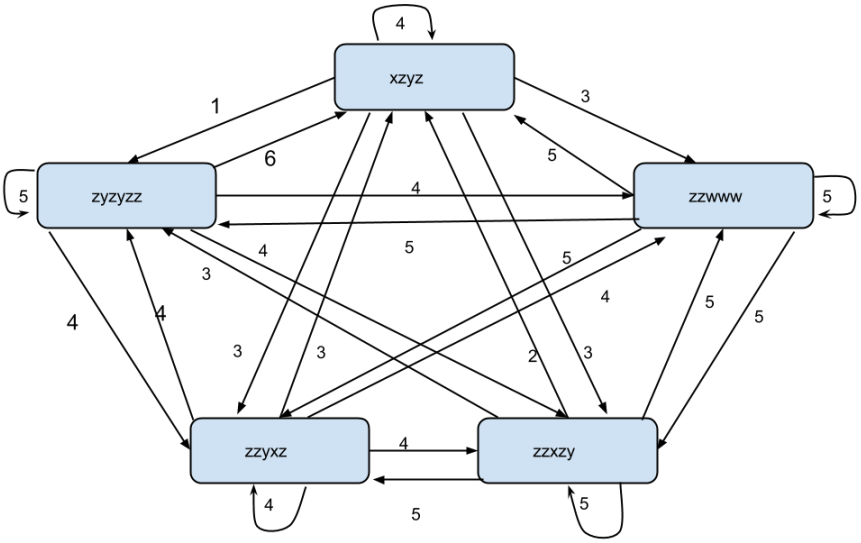
The optimizaton version of this problem has been shown to be NP-hard [1] [**?**], so research has focused on finding approximation algorithms. The first approximation algorithm seems to be a greedy algorithm discussed in Gallant (1982). This is now a commonly used greedy algorithm, addressed in newer papers. If we have a set S, the algorithm works by finding two strings s and t in S, that have maximum overlap. The overlap string o is formed from s and t, and o is placed back in S. Strings s and t are then removed from S. This process is repeated until S contains one string, or there is no non-empty overlap between strings in S. This algorithm is later addressed in [7], [6], [3], and [1]. [6] developed an algorithm that implemented the greedy approximation, and conjectured that the length of the superstring produced by their algorithm is less than or equal to $2 * len(OPT)$. They were unable to prove this however, and it remained an open problem. [7] was also unable to prove the conjectured $2 * len(OPT)$ upper bound of the greedy approximation. The first proven bound

Notational preliminaries

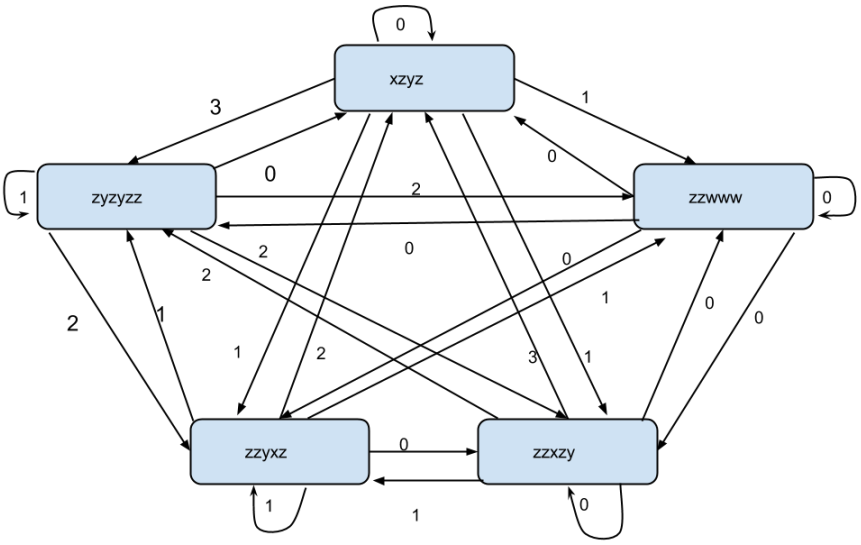
Let $S = s_1 \ldots, s_m$ be a set of m strings over the alphabet $\sum$. The paper uses $OPT(S)$ which denotes the length of the shortest common superstring for the set S. If given two strings s and t the term $ov(s,t)$ denotes the amount of overlap between s and t. For example, if $s = uv$ and $t = vw$ then $ov(s,t) = |v|$. The paper uses the term $pref(s,t)$ to denote the prefix of s with respect to t, that is $pref(s,t) = u$. Lastly, $d(s,t)$ denotes the distance from s to t. That is, $d(s,t) = |u|$. Finally, let SCS denote the shortest common superstring for some algorithm. For a set of strings $S = s_1, ..., s_m$ where each string is unique, and any $s_i \in S$ is a sub-string of any other $s_k in S$ then the SCS is trivially the concatenation of strings $SCS = s_1 s_2 ... s_m$. There are these notions $first(SCS)$ and $last(SCS)$ which denote the first string in SCS is the first string in the set S, and $last(SCS)$ is the last string in SCS. That is $first(SCS) = s_1$ and $last(SCS) = s_m$.

Throughout the analysis we talk about constructing graphs, unless otherwise specified we are constructing a overlap graph, or a distance graph:

Distance graph The graph $G = (V, E, c)$ is an edge-weighted, complete directed graph.
The set of vertices V comes from the strings from S. That is each vertex in V is one of the strings $s_1, ..., s_m \in S$. The costs c between edges is denoted $c(v_i, v_j)$ where the cost is really just the prefix between strings in S, that is $c(v_i, v_j) = pref(s_i, s_j)$. In English the weights of edges is the distance between strings s_i and s_j.
Lastly, $E = V^2$.



Overlap graph The graph $G = (V, E, c)$ is an edge-weighted, complete directed graph.
Like above, $V \in S$ and $E = V^2$.
The cost c between edges is denoted like above, but the true cost is $c(v_i, v_j) = ov(s_i, s_j)$. In English, the weights of the edges is the overlaps between strings s_i and s_j



0.1 Related Work

Work on the Shortest common superstring problem has been around since 1977. In 1980 Gallant et al. in *On Finding Minimal length superstrings*[2] set the basis for work on the shortest common superstring problem.

In Gallant, the researchers prove that given a set of strings S and a positive length K as well as an unbounded alphabet $\sum$ over those strings, that determining if S has a superstring of length K is NP-Complete. The authors do this by reducing to the *Hamiltonian path problem.* The Hamiltonian path problem is the problem where one determines in a directed or undirected graph whether there exists a path that visits each vertex only once. The Hamiltonian path problem was known to be NP-Complete so a reduction ultimately proves the completeness of the other problem [4]. Throughout all newer papers on the SCS problem each paper reduces the problem to the Hamiltonian path problem while using different tricks to reduce the SCS.

In 1988 Jorma Tarhio and Esko Ukkonen, authored *A Greedy Approximation Algorithm for Constructing Common Superstrings* [6]. Their paper developed a polynomial-time approximation algorithm that would construct a superstring within some ϵ of the optimal solution. Their approxmiation algorithm is based off a greedy heuristic. At a high level their algorithm is given a set S of strings over the alphabet $\sum$ select two strings which have the greatest mutual overlap. From there the algorithm merges the two strings and replaces the single combined string in S. An example would be the set $S = aabb, bbbaaa, aaaa$ The algorithm would select $bbbaaa$ and $aaaa$, merging and replacing so the set looks like $S = aabb, bbbaaaa$. The algorithm at finer detail is implemented using, once again, Hamiltonian paths.The algorithm constructs a longest Hamiltonian path for weighted directed graph. The weight of the edges are the overlap between strings. The greedy heuristic holds on this graph as it chooses the edges with the maximum weight. Although their algorithm constructs a solution they do not prove the lower

bound of the algorithm and leave it open for future work.

Aside from compression of strings in computation, the SCS problem has relevance in the biology field. In *Towards a DNA Sequencing Theory*, Ming Li explains the relevance of SCS in terms of DNA sequencing (in 1990). The idea of sequencing DNA is an important one to understand the basis of life. Cures for diseases, viruses, among other things all come from information that is derived from sequencing DNA. At a high level DNA strands can be encoded from a set of nucleotides A, T, C, G. This is important information. Using this reduction scientists can reduce DNA to a string which can then be used for the SCS problem. Using the SCS problem on this DNA string was very relevant in the early 90's due to the limited technology of sequencind DNA. Only around 500 base pairs could be sequenced at a time. If A scientist can get the most out of those 500 pairs then their research can continue. The paper provides an algorithm in which they argue it constructs a superstring of length $O(nlogn)$ where n is the smallest possible substring, ie the optimal soltuion. The algorithm provided in the paper takes the base greedy approximation algorithm, but instead of greedily choosing two strings which have the greatest overlap, the algorithm chooses two strings such that their merge will make it so every other group of strings has some overlap. That is he tries to maximize the number of strings in which the merge will overlap with. They call this technique group merging.

0.2 Related Work/Prior work

Prior to Blum et. al. work on the shortest common superstring problem was focused on finding approximation algorithms because the shortest common superstring was shown to be NP-hard by Gallant et al. 1980. TALK ABOUT GALLANT

Tarhio and Ukkonen (1988) developed a "greedy" approximation algorithm based on an idea first mentioned in Gallant 1982; Find the two strings in a set of strings, with the largest overlap, and merge these two strings together to form a new overlapped string. Then add new overlapped string to the set of strings, and remove two original strings. Continue to do this until there is only a single string in the set. The main result of this paper dealt with the quality of the approximation, when using "compression" as the measure of quality. They show that if n is the sum of the length of every string in set S, and k_{min} is length of the shortest common superstring for S, and k_{greedy} is the length of the string produced by their greedy approximation algorithm, then $(n - k_{greedy}) \geq \frac{1}{2} * (n - k_{min})$. There is another way to compare the quality of an approximation algorithm for the shortest common superstring problem, compare the length of the common superstring created by the approximation for a set of strings S to the length of the shortest common superstring for the set of strings S. Tarhio and Ukkonen were unable to proven an upper bound for this quality measure, but conjectured that their approximation algorithm produced common

superstrings of length no more than 2 times the length of the shortest common superstring; $k_{greedy} \leq 2 * k_{min}$.

All work was considered on a reduced set of strings, meaning no string in a set S is a substring of another string in S, because a reduced set of strings will have the same substring as the non-reduced set.
The shortest common superstring (SCS) problem can be viewed as a special case of the longest hamiltonian path problem on a weighted directed graph, and an approximation solution for SCS was approached as finding the longest hamiltonian path in a weighted directed graph by more than one paper [6] [7]. An overlap between two strings s_i and s_j, where $s_i \neq s_j$, $s_i = u * v$ and $s_j = v * w$, is v; v can be the empty string. Let $ov(s_i, s_j)$, be the length of the maximal overlap between s_i and s_j. The definition of overlap is assymetric, an overlap between s_i and s_j does not mean there is an overlap between s_j and s_i. An overlap is maximal if it is as large as possible.

Tarhio defines an "overlap" graph G for a set of strings $S = s_1, ..., s_n$, with vertex set $S \cup x_o, x_{n+1}$, and with directed edges (s_i, S_j), if $s_i \neq s_j$. x_o is a start node, and x_{n+1} is an end node. Each edge (s_i, s_j) has weight $ov(s_i, s_j)$. The weight of the edge from x_o to each other vertex is 0, and the weight from x_{n+1} to every other vertex is 0. So, the "overlap" graph, is a graph $G = (V, E, ov)$. From any directed hamiltonian path H on G from x_o to x_{n+1}, a common superstring can be constructed for S. The SCS for s would be constructed then from the longest hamiltonina path on G. The paper implies that finding the longest Hamiltonian path is NP-hard, but this is already well known. Since finding the longest Hamiltonian path is NP-hard an approximation algorithm is needed, and there is a well known "greedy" heuristic for longest Hamiltonian paths. To construct a Hamiltonian path, select an edge e from the remaining edges in G, the overlap graph, such that e has the largest weight and e plus the edges already selected form a subgraph that can be expanded to a Hamiltonian path. This is repeated until a Hamiltonian path has been found. Tarhio used this "greedy" approximation algorithm as part of their approximation algorithm for SCS.

Tarhio's proposed algorithm finds the maximal overlap between every string in a set of strings S, and they use the Knuth-Morris-Pratt algorithm to calculate maximum overlaps. During this step of the algorithm, if a string is found to be a substring of another string, it is removed from the set S. From this reduced set of strings S and using the calculated maximum overlaps between strings, the previously described overlap graph G can be constructed and the "greedy" approximation algorithm can be run on G.

Tarhio approached the problem of an approximation algorithm for SCS by reducing the problem to the longest Hamiltonian path problem and using a well known greedy approximation algorithm for longest hamiltonian path, to generate a common superstring. They also gave a guaranteed bound with regards to "compression". And conjectured that their approximation algorithm produces strings of length less than or equal to 2 times the length of the SCS.

Another slightly later paper Turner (1989) [7] approached an approximation algorithm as Tarhio did, thinking about it from the viewpoint of maximizing the overlap between the strings in a set S. Just as Tarhio, Turner could find no worse example than $k_{greedy} \leq 2 * k_{min}$, but was also not able to prove the upper bound. Turner also showed $(n - k_{greedy}) \geq \frac{1}{2} * (n - k_{min})$.

Turner models the problem of finding the SCS as the longest path problem, on a complete directed graph, with each edge have a non-negative length, where the weight is the overlap between a string s_i and s_j. In this specific case of the Longest Path Problem, the goal is to find a Hamiltonian path that maximized the sum of the weight of the edges of the path. At this point Turner has described almost the exact same overlap graph described in Tarhio.

Turner then presents 3 different approximation algorithms for SCS. The first approximation algorithm finds a maximum matching in the overlap graph G. Since G is complete, any matching can be extended to a path, and the maximum matching is used to find a Hamiltonian path. This is similar to the approach of Tarhio, except instead of using the "greedy" approximation algorithm for longest Hamiltonian paths, he used maximum matchings, to construct Hamiltonian paths. Turner's second approximation algorithm found a directed matching in the overlap graph G. The directed matching was also used to find a Hamiltonian path. For his third approximation algorithm Turner mentions a "greedy" approximation for the longest path problem. This algorithm is the same greedy algorithm considered in Tarhio.

Turner then relates SCS to the path version of the traveling salesman problem; This is similar to what is done in Blum et al. () This is an asymmetric case of the traveling salesman problem where the triangle inequality holds, meaning it is not NP-hard to find an approximation algorithm.

0.3 Main Results

Results of the paper

The papers that set up greedy before this paper conjectured that GREEDY could be $|GREEDY_{SCS}| \leq |2OPT(S)|$. We will describe how the authors proved that the upperbound string for a slightly modified GREEDY is actually $|GREEDY_{SCS}| \leq |4OPT|(S)$.

$4 \ldots OPT(S)$ upper bound for GREEDY

When we take the SCS problem and reduce it, we end up with a set of strings S which can be represented as a weighted graph G. This opens up a slew of graph algorithms that previously wouldn't have worked without the reduction. The algorithm works by finding a vertex disjoint cycle cover, where each vertex in the graph is contained in at least one cycle. In order to get a vertex disjoint cycle cover, the authors use a polynomial-time algorithm for the assignment problem. The assignment problem is the problem of finding a maximal or minimal weight matching between vertices in a weighted bipartite graph. To prove that the algorithm finds a SCS of length at most $4 \ldots OPT(S)$ they prove that the the assignment algorithm finds some optimial weight matching on graph G, and when the cycles in G are found and unraveled they are at most length 4 of the optimal solution.

To prove GREEDY's approximation they first prove that an algorithm "Concat-cycles" produces a string at most 4 of the optimal, then show that GREEDY mimics concat-cycles.

Concat-cycle algorithm

A. Given a set of strings S reduce the set of strings to a distance graph G.

B. Run a minimum weight assignment on G. The resulting set of cycles is C where $C = c_1, c_n$

C. Take the cycles from C and "unravel" them into strings s'_i. s'_i can be defined as the vertices $v_1 ... v_r ... v_1$. In English this is the string that is created by traveling through the vertices in a cycle until we've visited ever vertex in that cycle.

D. for each string $s_i, s_{i+1}, ...$ concatenate them which gives us a string SCS.

Theorem

The algorithm concat cycles produces a string of length at most $4 \ldots OPT(S)$

Explanation of the proof

The proof works by stating that since an optimal assignment on the distance graph G produces a set of cycles C, if we sum the weight of the cycles it is less than or equal to OPT(S). That is $C = c_1, ..., c_m$ then $\sum_{i=i}^{m} w(c_i) \leq OPT(S)$.
This makes sense because the weights of the edges are just $|s_i| - ov(s_i, s_j)$. That is, the weight of the edge from v_i to v_j is just the length of s_i minus the overlap it has with s_j. We will show that the sum of these is less or equal to $OPT(S)$.

In a substring free set S the distance graph will contain weights between edges of $|s_i|$. That is the distance between vertices will be the total length of the string s_i. In English, to get from vertex v_i to v_j you must travel over the total length of the string s_i. In this graph, the algorithm calculates a minimum assignment which gives us a set C, where the $\sum_{i=0}^{m} w(c_i) = \sum_{i=0}^{m} |s_i| = OPT(S)$.

When the algorithm takes the longest string l_i from the cycle c_i and merges it with all the other longest strings in each cycle we end up with overlap between strings at most $2 * \sum_{i=0}^{m} w_i$. When the algorithm finishes merging all strings together we have a resulting string with a minimal length of $\sum_{i=1}^{m} l_i - 2wi$. [TODO explain how they pull numbers out of thin air]

We will explain now how the paper shows that the modified GREEDY algorithm mimics concat-cycles and thus has the same upperbound SCS length.

Unlike Concat-cycles, the modified GREEDY takes an overlap graph G and computes the maximum overlap. The algorithm works by repeatedly selecting the maximum overlap between verticies. The algorithm continues by constructing and joining paths until disjoint cycles emerge such that all vertices in G are covered. That is it selects an edge between verticies and merges them into one vertex. The assignment of covers C is an optimal assignment, which they prove in a contrdadiciton. They say that if there exists an edge with a maximum overlap it would have to be contained in the algorithms C. Since this is an optimal assignment it mimics the concat-cycles which also creates an optimal assignment. Once the assignment is complete they simply unroll the cycles and concatenate strings together.

$3 \ldots OPT(S)$ **upper bound for TGREEDY**

In the modified GREEDY algorithm the final step was to simply concatenate strings until only one string remained, the SCS. The authors propose that simply concatenating the strings is what is limiting the algorithm. They propose instead of simply concatenating strings together you run the original GREEDY algorithm on the strings.

Theorem

TGREEDY produces a superstring of at most $3 \ldots OPT(S)$

Explanation of the proof

We take the set of strings O that have a self overlap that the modified GREEDY algorithm produced and the matching C.

The superstring problem is MAX SNP-hard

As the final result of the paper the authors show that the superstring problem is MAX SNP-hard, by reducing TSP(1,2) to the superstring problem; TSP(1,2) was shown to be MAX SNP-hard in [**?**]. Which, although published after Blum et al. was cited by Blum et al. that TSP(1,2) had been shown to be MAX SNP-hard. MAX SNP is a class of optimization problems, and it is known that every problem in this class can be approximated within some constant factor.

TSP(1,2) is the special case of the traveling salesman problem where the distances are either 1 or 2. The instance of TSP(1,2) considered in this paper is the instance where each edge in the graph H has distance 1, and each edge not in the graph has distance 2. There is the additional restriction that H have a bounded degree D (the precise bound is not important). [**?**] showed TSp(1,2) to be MAX SNP-hard even for this restricted version.

0.4 sarah

Blum et al. (1991) approached approximation algorithms for SCS from a different perspective than previous papers, by dealing with a minimization problem on the "distance" graph. Blum et al. defines a graph $G_s = (V, E, d)$ that has m verticies V, and m^2 edges $E = (i, j) : 1 \leq i, j \leq m$. And the weight function is $d(,)$: edge (i, j) has weight $d(i, j) = d(s_i, s_j)$. This is the distance graph, and is similar to the Turner one mentioned in related work.

Using this graph, Blum proves a $4*OPT(S)$ bound, for a modified greedy algorithm, and then a $3*OPT(S)$ bound for another modified greedy algorithm. The previous best proof of an approximation guarantee with respect to length was Li and a bound of $n \log n$, where n is the length of the shortest common superstring.

H is a directed graph, hardness holds for both directed and undirected, with a bounded degree D specifying an instance of TSP(1,2). For every vertex v in H there are two characters, v and v', and an additional character #, so for each vertex v there is the string $v\#v'$; This string is called the "connector" for v. For each vertex v in H, loop through the edges leaving v in an arbitrary cyclic order: $(v, w_o), ..., (v, w_{D-1})(*)$. For the ith edge out of v there is string $p_i(v) = v'w_{i-1}v'$. A graph G_S, as described earlier in the paper, is a distance-weighted graph, and is constructed for the set of strings p. If the subgraph of G_S, G_2 with only edges of weight (2), there is a component for each vertex v in H. This is because and two strings $p_i(v)$, will have distance of 2 from eachother, and therefore the vertices from $p_i(v)$ will form a cycle in in their component in G_2. In each component in G_2, there is also a "connector" vertex with an edge to each vertex $p_i(v)$.

Theorem

GREEDY produces a string of length at most $4 * OPT(S)$

Explanation of proof

If we sort the edges of the overlap grpah G in descending order, GREEDY operates by iterating over the list and determining whether the slected edge should be included in the SCS. We can label edges in this list with the terms "better" and "worse". An edge is determined to be better if it has a larger overlap than all others. An edge is determined to be worse has a smaller overlap than all others. Obviously, a better edge will have $\geq$ overlap to a worse edge. The paper also uses the term "dominate" to mean an edge f has more overlap than an edge e where f shares a head or tail with e. GREEDY can choose not to include an edge under two conditions. Condition one is it will disclude an edge if we have previously included an edge that dominates the edge it's trying to merge. Lastly, it will not include an edge that creates a cycle. In this proof they use another term "bad back edges" which means an edge that would create a cycle if we were to choose it. You can think of a bad back edge as a string with more overlap with itself than it does with the string GREEDY is attempting to merge it with.

If we take two bad back edges they are either disjoint or one is contained in the other. If the two intervals are contained in each other we denote the minimal innermost string of this containment as a "culprit string". Between two successive culprit strings is a path, which is referred as a "weak link". If we remove weak links we end up with disjoint "blocks" which are just paths. Each block has a nonempty cuplprit as the middle segment along with a left and right extension, which may or may not exist.

Given a set S_m where S_m consits of the greedy edges together with the back edge of the culprit. When we apply the algorithm modified GREEDY to S_m it will construct us an assignment C_m that is an optimal assignment for S_m. We now create two graphs G_l which is the left/middle portion of a block and G_r which is the right/middle portion. l_l and l_r is denoted as the sum of the left and right extensions and l_c to be the sum of the centers, and o_w to be the sum of overlaps. They find optimal assignments on these two graphs, M_l and M_r. With these two assignments they show that they can bound overlap but during the calculation some edges are double counted. In order to stop this double count and prove the upper bound they figure out how to get adjacent edges to dominate each other which reduces the double counting. Ultimately the show some math which we omit because we cant really explain it without copy pasting it here which proves the upper bound $4 * OPT(S)$.

0.5 Conclusion

Improvements/Followups.

There have been a large amount of improvements and followups on the SCS problem. The paper proved that their algorithm had an upper bound minimum string length of $3n$, where n is the optimal length. Successive papers throughout the years improved the upper bounds: Teng and Yao improved the bound to $2\frac{8}{9}n$; Czumaj et al. by $2\frac{5}{6}n$; Kosaraju, Stien and Park by $2\frac{50}{63}n$; Stein and Armen by $2\frac{3}{4}n$, then by $2\frac{2}{3}n$; Breslauer et al. by 2.596 and lastly $2.5n$ by Sweedyk [5]. We will quickly describe what Sweedyk does.

The previous algorithms, including the one in our paper constructed a directed graph G where each edge in the graph has a positive weight. The weight denotes the length. The algorithms would then construct a cycle cover on the graph. The cycle cover would be a minimum-length cycle cover. The set of cycle-covers were the strings S that would be concatenated together to form the superstring.

The algorithm described in the $2 * \frac{1}{2}$ paper worked by constructing a graph G, like above and constructing a minimal-length cycle-cover. Where it differs from previous algorithms is the algorithm then takes the set of cycle-covers and attempts to merge each cycle with another cycle. Once complete they follow above and remove each cycle and unravel it to create a string. Once all cycles are unraveled from the cycles they concatenate and have a superstring. The authors argue that their algorithm can be improved upon up to $2OPT$ and hint at working on a proof for showing that a modified version will acheive $2\frac{1}{3}OPT$

References

[1] Blum, A., Jiang, T., Li, M., Tromp, J., and Yannakakis, M. Linear approximation of shortest superstrings. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing* (1991), ACM, pp. 328–336.

[2] Gallant, J., Maier, D., and Astorer, J. On finding minimal length superstrings. *Journal of Computer and System Sciences 20*, 1 (1980), 50–58.

[3] Li, M. Towards a dna sequencing theory (learning a string). In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on* (1990), IEEE, pp. 125–134.

[4] Michael, R. G., and David, S. J. Computers and intractability: a guide to the theory of np-completeness. *WH Freeman & Co., San Francisco* (1979).

[5] Sweedyk, Z. A 2.5-approximation algorithm for shortest superstring. *SIAM Journal on Computing 29*, 3 (2000), 954–986.

[6] Tarhio, J., and Ukkonen, E. A greedy approximation algorithm for constructing shortest common superstrings. *Theoretical Computer Science 57*, 1 (1988), 131–145.

[7] Turner, J. S. Approximation algorithms for the shortest common superstring problem. *Information and computation 83*, 1 (1989), 1–20.