Problemas de Decisión Vs Problemas de Optimización

Los problemas de decisión son aquellos admiten solamente 2 respuestas posibles: si “Sí” o un “No”. En contraste, los de optimización exigen una solución óptima si existe alguna (en el caso que no puede haber pasado alguno de los siguientes escenarios: (i) no hay ninguna solución (ii) siempre hay una solución mejor). En general, se puede formular un problema de optimización a su versión de decisión a pedir que la solución tenga cierta condición de calidad (en el caso de problema de minimización, pide que solución cuyo valor sea a lo sumo un valor k y en el caso de problema de maximización que sea por lo menos k dado un valor de k determinado).

Un algoritmo aproximado se desarrolla para brindar a un problema de optimización, la generación de solución de buena calidad relativa a la óptima o una cota buena (inferior si es de minimización, superior si es maximización).

¿En qué circunstancias convienen/hay usar algoritmos aproximados?

¿Cómo encontrar buenas cotas?

¿Cuándo un algoritmo aproximado es mejor que otro para un mismo problema de optimización?

Algoritmos Aproximados

Se utiliza ***OPT*** para denotar el valor de la solución óptima. Un algoritmo ***H*** es de factor ***α*** si para cualquier instancia ***I*** del problema de optimización ***π*** en cuestión, $\alpha \times signo(\pi) \geq \frac{valor(H(I))}{OPT} \times signo(\pi)$ donde $H(I)$ es solución dada por ***H***, $valor$ es la función que evaluar el valor de la solución y

$$signo(\pi) = \begin{cases} 1 \; si \; \pi \; es \; de \; minimización \\ -1 \; caso \; contrario \end{cases}$$

Claramente, cuando ***α*** es más cerca de 1 mejor. ¡Los algoritmos exactos hacen eso!

Cuando ***α*** es un valor constante se dice que el problema ***π*** es aproximable. Un problema de optimización es inaproximable cuando no puede existir un algoritmo aproximado de factor constante para resolverlo salvo P=NP.

Existen varios problemas de estas características:

- Coloreo de vértices en grafos
- Conjunto dominante en grafos bipartitos
- Set cover

Set Cover

Sea $\boldsymbol{U}$ un universo de elementos y $\mathbf{2^U}$ todos los subconjuntos posibles de $\boldsymbol{U}$. Consideramos $\boldsymbol{F} \subseteq \mathbf{2}^{\boldsymbol{U}}$ una colección de subconjuntos de $\boldsymbol{U}$ y cada $s_i \in U$ tiene un valor $\omega(s_i)$ que es el peso de ese subconjunto. El problema de set-cover es determinar una subcolección $\boldsymbol{F'} \subseteq \boldsymbol{F}$ de manera tal que minimiza $\sum_{s \epsilon F'} \omega(s)$ y $\bigcup_{s \epsilon F'} S = U$

Algorithm 2.2 (Greedy set cover algorithm)

1. $C \leftarrow \emptyset$
2. While $C \neq U$ do
   Find the most cost-effective set in the current iteration, say S.
   Let $\alpha = \frac{\text{cost}(S)}{|S-C|}$, i.e., the cost-effectiveness of S.
   Pick S, and for each $e \in S - C$, set $\text{price}(e) = \alpha$.
   $C \leftarrow C \cup S$.
3. Output the picked sets.

Set Cover

Supongamos que $\boldsymbol{U}$ tiene n elementos y son e_1, e_2,…, e_n ordenados de acuerdo fueron cómo fue incluido en C. Los elementos incorporados en una misma iteración pueden estar ordenados de cualquier forma.

Lemma 2.3 *For each* $k \in \{1, \ldots, n\}$, $\text{price}(e_k) \leq \text{OPT}/(n-k+1)$.

Proof: In any iteration, the leftover sets of the optimal solution can cover the remaining elements at a cost of at most OPT. Therefore, among these sets, there must be one having cost-effectiveness of at most $\text{OPT}/|\overline{C}|$. In the iteration in which element e_k was covered, $\overline{C}$ contained at least $n-k+1$ elements. Since e_k was covered by the most cost-effective set in this iteration, it follows that

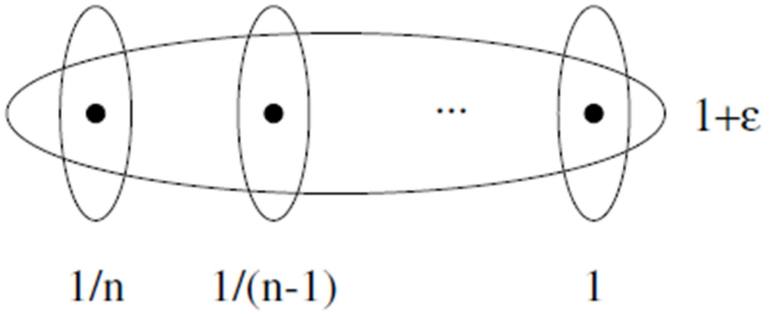
$$\text{price}(e_k) \leq \frac{\text{OPT}}{|\overline{C}|} \leq \frac{\text{OPT}}{n-k+1}.$$

□

Set Cover

Theorem 2.4 *The greedy algorithm is an H_n factor approximation algorithm for the minimum set cover problem, where $H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$.*

Proof: Since the cost of each set picked is distributed among the new elements covered, the total cost of the set cover picked is equal to $\sum_{k=1}^{n} \text{price}(e_k)$. By Lemma 2.3, this is at most $\left(1 + \frac{1}{2} + \cdots + \frac{1}{n}\right) \cdot \text{OPT}$. □



$\boldsymbol{OPT = 1 + \varepsilon}$ y el algoritmo da una solución de factor $\boldsymbol{H_n}$

¿$\boldsymbol{H_n}$ es $\theta(\log n)$?

Técnica de diseño de algoritmos x capas

Problema de ejemplo, Vertex-cover: dado un grafo G=(V,E) y una función de peso $\omega: V \to Q^+$, hallar un subconjunto V' de V tal que $\sum_{u \epsilon V'} \omega(u)$ sea mínima.

- Para $\omega: V \to \{c\}$ donde ***c*** es un valor constante, existe un algoritmo goloso trivial de factor 2.
- Para $\omega(u) = c \times \deg(u)$ donde ***c*** es un valor constante y $\deg(u)$ es el grado de u en G, existe un algoritmo de factor 2 y esta función se llama "***degree-weighted***". La solución es V.

Lemma 2.6 *Let* $w : V \to \mathbf{Q}^+$ *be a degree-weighted function. Then* $w(V) \leq 2 \cdot \mathrm{OPT}$.

Proof: Let c be the constant such that $w(v) = c \cdot \deg(v)$, and let U be an optimal vertex cover in G. Since U covers all the edges,

$$\sum_{v \in U} \deg(v) \geq |E|.$$

Therefore, $w(U) \geq c|E|$. Now, since $\sum_{v \in V} \deg(v) = 2|E|$, $w(V) = 2c|E|$. The lemma follows. □

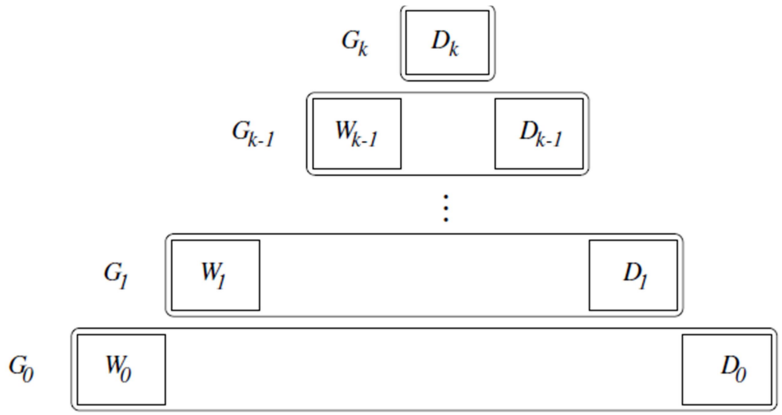
Técnica de diseño de algoritmos x capas

Observación: V es el vertex cover de mayor peso posible ya que contiene todos los vértices y OPT es el peso del vertex cover de peso mínimo. Como el mayor está entre OPT y 2×OPT. Eso implica que para cualquier par de vertex covers R y S,

$$\omega(R) \leq \omega(S) \leq 2\times\omega(R) \text{ y } \omega(S) \leq \omega(R) \leq 2\times\omega(S)$$

Técnica de diseño de algoritmos x capas

- Iteración x capa, la idea es ir reduciendo en cada capa un subgrafo inducido más pequeño hasta la última (k-ésima) que consiste un conjunto de vértices aislados. G_o=G, el grafo de capa 0 y $\omega_{-1} = \omega$. En cada iteración i, D_i es el subconjunto de vértices de grado 0, se determina un valor constante c_i=min{$\omega_{i-1}(u)/deg_{Gi}(u)$} para todo vértice u de G_i y no esté en D_i. Para los vértices u G_i y no esté en D_i se define una función *degree-weighted* $t_i(u) = c_i \times deg_{Gi}(u)$ y W_i son los vértices que además verifican $\omega_{i-1}(u) = t_i(u)$. G_{i+1} es G_i eliminando D_i y W_i.



Let $t_0, ..., t_{k-1}$ be the degree-weighted functions defined on graphs $G_0, ..., G_{k-1}$. The vertex cover chosen is $C = W_0 \cup ... \cup W_{k-1}$. Clearly, $V - C = D_0 \cup ... \cup D_k$.

Técnica de diseño de algoritmos x capas

Theorem 2.7 *The layer algorithm achieves an approximation guarantee of factor 2 for the vertex cover problem, assuming arbitrary vertex weights.*

Proof: We need to show that set C is a vertex cover for G and $w(C) \leq 2 \cdot \mathrm{OPT}$. Assume, for contradiction, that C is not a vertex cover for G. Then

there must be an edge (u, v) with $u \in D_i$ and $v \in D_j$, for some i, j. Assume $i \leq j$. Therefore, (u, v) is present in G_i, contradicting the fact that u is a degree zero vertex.

Let C^* be an optimal vertex cover. For proving the second part, consider a vertex $v \in C$. If $v \in W_j$, its weight can be decomposed as

$$w(v) = \sum_{i \leq j} t_i(v).$$

Next, consider a vertex $v \in V - C$. If $v \in D_j$, a lower bound on its weight is given by

$$w(v) \geq \sum_{i < j} t_i(v).$$

Técnica de diseño de algoritmos x capas

The important observation is that in each layer i, $C^* \cap G_i$ is a vertex cover for G_i, since G_i is a vertex-induced graph. Therefore, by Lemma 2.6, $t_i(C \cap G_i) \leq 2 \cdot t_i(C^* \cap G_i)$. By the decomposition of weights given above, we get

$$w(C) = \sum_{i=0}^{k-1} t_i(C \cap G_i) \leq 2 \sum_{i=0}^{k-1} t_i(C^* \cap G_i) \leq 2 \cdot w(C^*).$$

□

Example 2.8 A tight example is provided by the family of complete bipartite graphs, $K_{n,n}$, with all vertices of unit weight. The layering algorithm will pick all $2n$ vertices of $K_{n,n}$ in the cover, whereas the optimal cover picks only one side of the bipartition. □

Superstring

Problema de Superstring: dado un alfabeto finito $\sum$ y un conjunto de strings $S = \{s_1, s_2, \dots, s_n\} \subseteq \sum^+$, encontrar un string s de menor longitud tal que contiene a cada s_i. Podemos suponer sin pérdida de generalidad que ningún string s_i es substring de otro string $s_j, i \neq j$.

Motivación: DNA humano puede ser visto como un string muy largo sobre el alfabeto $\sum = \{a, c, g, t\}$. En general, se tiene muchos fragmentos de pequeños substrings de este substring donde muchos substrings tienen superposiciones. Entonces los científicos creen que un superstring mínimo de estos fragmentos es un buen candidato para DNA.

Otra aplicación: Compresión óptima. Las instancias de este problema son las mismas de Superstring. Lo que quiere es obtener un supestring σ de $S = \{s_1, s_2, \dots, s_n\} \subseteq \sum^+$ y luego codificar cada s_j por un par (p_j, l_j) donde p_j es el offset de la primera ocurrencia de s_j en σ y l_j es la longitud de s_j. El objetivo es maximizar $\|S\| - |\sigma|$ y es claro que eso ocurre si σ es superstring mínimo de S.

Superstring

- Algoritmo goloso usando máxima superposición.
- Usando algoritmo de factor H_n de set-cover
- Algoritmo de factor 4
- Algoritmo de factor ½ para el problema de máxima compresión
- Algoritmo de factor 3

Dados 2 strings $s, t \in \sum^*$, la máxima superposición de s con t es el sufijo de mayor longitud de s que es prefijo de t.

El algoritmo goloso usando máxima superposición es iterativamente, elije un par de strings s, t cuya máxima superposición es mayor entre todos pares posibles de un conjunto T de strings, inicialmente $T \coloneqq S$ donde S es la instancia del problema a resolver. En cada iteración se eliminan s, t y los reemplaza el string resultante de concadenar s y el sufijo de t eliminando el prefijo que es la máxima superposición s con t. Claramente, después $n - 1$ iteraciones, se obtiene un superstring de S. Se conjectura que es de factor 2. Con $S = \{ab^k, b^k c, b^{k+1}\}$ muestra que no puede ser mejor que 2.

Superstring

Dado un conjunto de strings $S = \{s_1, s_2, \dots, s_n\}$, se define $\sigma_{i,j,k}$ es el resultado de superponer los últimos k caracteres de s_i con los primeros k caracteres de s_j siempre que sea posible con ese valor de k.

Sea M, el conjunto de $\sigma_{i,j,k}$ para valores posibles de i, j, k teniendo en cuenta a S.

Podemos definir la instancia del problema de set cover correspondiente a S. En este el universo es Sy la colección de los subconjuntos a considerar corresponde a cada string $\pi \in M \cup S$, donde $set(\pi) = \{s \in S \mid s \; es \; substring \; de \; \pi\}$.

Algorithm 2.10 (Shortest superstring via set cover)

1. Use the greedy set cover algorithm to find a cover for the instance $\mathcal{S}$. Let $\text{set}(\pi_1), \dots, \text{set}(\pi_k)$ be the sets picked by this cover.
2. Concatenate the strings $\pi_1, \dots, \pi_k$, in any order.
3. Output the resulting string, say s.

Superstring

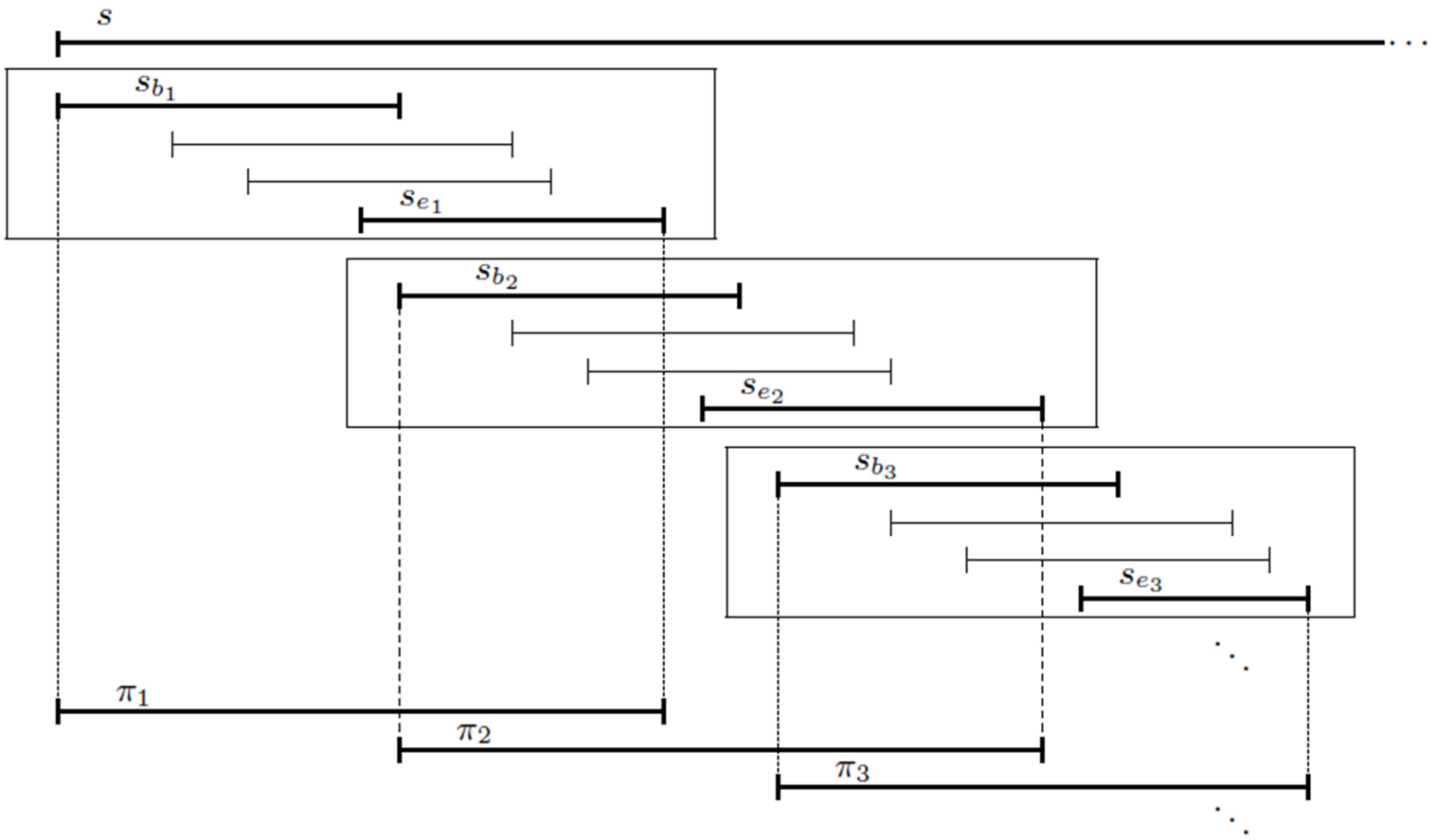
Lemma 2.11 $\mathrm{OPT} \leq \mathrm{OPT}_{\mathcal{S}} \leq 2 \cdot \mathrm{OPT}$.

Proof: Consider an optimal set cover, say $\{\mathrm{set}(\pi_i) | 1 \leq i \leq l\}$, and obtain a string, say s, by concatenating the strings π_i, $1 \leq i \leq l$, in any order. Clearly, $|s| = \mathrm{OPT}_{\mathcal{S}}$. Since each string of S is a substring of some π_i, $1 \leq i \leq l$, it is also a substring of s. Hence $\mathrm{OPT}_{\mathcal{S}} = |s| \geq \mathrm{OPT}$.

To prove the second inequality, let s be a shortest superstring of $s_1, \ldots, s_n$, $|s| = \mathrm{OPT}$. It suffices to produce *some* set cover of cost at most $2 \cdot \mathrm{OPT}$.

Consider the leftmost occurrence of the strings $s_1, \ldots, s_n$ in string s. Since no string among $s_1, \ldots, s_n$ is a substring of another, these n leftmost occurrences start at distinct places in s. For the same reason, they also end at distinct places. Renumber the n strings in the order in which their leftmost occurrences start. Again, since no string is a substring of another, this is also the order in which they end.

Superstring



Superstring

For each pair of strings (s_{b_i}, s_{e_i}), let $k_i > 0$ be the length of the overlap between their leftmost occurrences in s (this may be different from their maximum overlap). Let $\pi_i = \sigma_{b_i e_i k_i}$. Clearly, $\{\text{set}(\pi_i) | 1 \leq i \leq t\}$ is a solution for $\mathcal{S}$, of cost $\sum_i |\pi_i|$.

The critical observation is that π_i does not overlap π_{i+2}. We will prove this claim for $i = 1$; the same argument applies to an arbitrary i. Assume, for contradiction, that π_1 overlaps π_3. Then the occurrence of s_{b_3} in s overlaps the occurrence of s_{e_1}. However, s_{b_3} does not overlap s_{b_2} (otherwise, s_{b_3} would have been put in the second group). This implies that s_{e_1} ends later than s_{b_2}, contradicting the property of endings of strings established earlier.

Because of this observation, each symbol of s is covered by at most two of the π_i's. Hence $\text{OPT}_{\mathcal{S}} \leq \sum_i |\pi_i| \leq 2 \cdot \text{OPT}$. □

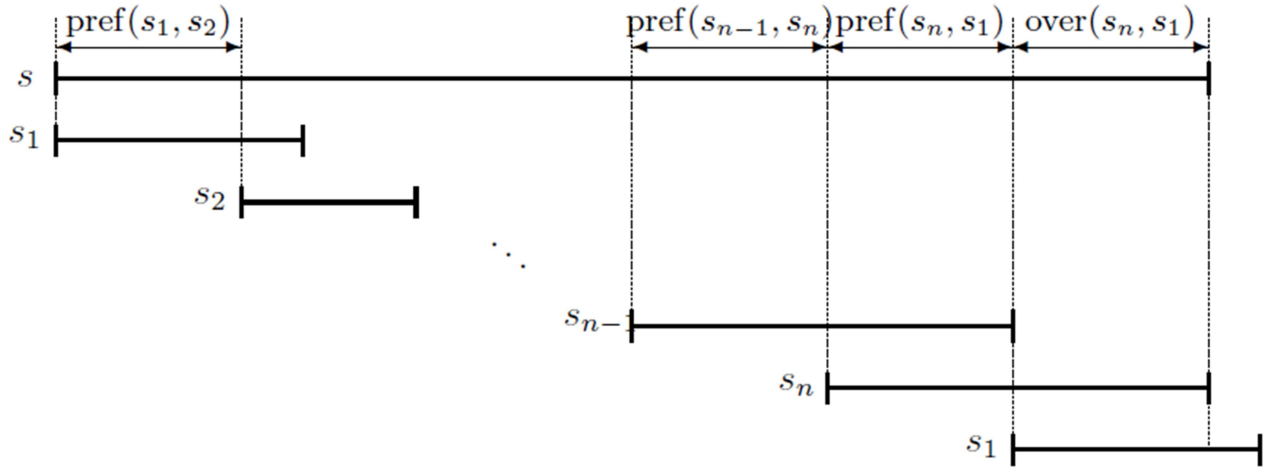
The size of the universal set in the set cover instance $\mathcal{S}$ is n, the number of strings in the given shortest superstring instance. This fact, Lemma 2.11, and Theorem 2.4 immediately give the following theorem.

Theorem 2.12 *Algorithm 2.10 is a $2H_n$ factor algorithm for the shortest superstring problem, where n is the number of strings in the given instance.*

Superstring – Factor 4

Algorithm 7.1 (Shortest superstring – factor 4)

1. Construct the prefix graph corresponding to strings in S.
2. Find a minimum weight cycle cover of the prefix graph, $\mathcal{C} = \{c_1, \ldots, c_k\}$.
3. Output $\sigma(c_1) \circ \ldots \circ \sigma(c_k)$.



Superstring – Factor 4

En la solución óptima s, tomemos la primera ocurrencia de cada string de *S* y los enumeramos como s_1,s_2, …, s_n. Claramente, la superposición entre s_i y s_{i+1} es máxima y la denotamos como overlap(s_i, s_{i+1}) ya que sino se podría conseguir otro superstring más corto. Y cada s_i se puede subdivir en prefix(s_i, s_{i+1}) y overlap(s_i, s_{i+1}). Además, no hay hueco en S (cualquiera posición es usada por lo menos por algún s_i.

$$\begin{aligned}\mathrm{OPT} = &|\mathrm{prefix}(s_1, s_2)| + |\mathrm{prefix}(s_2, s_3)| + \ldots + |\mathrm{prefix}(s_n, s_1)| \\ &+ |\mathrm{overlap}(s_n, s_1)|.\end{aligned}$$

Se puede definir un grafo orientado, colocando un nodo *i* para cada string s_i de *S*. Para cada par ordenado orientado $i \rightarrow j$ su peso es $|prefix(s_i, s_j)|$(*i* puede ser igual a *j*). Claramente, la longitud de circuito Hamiltoniano mínimo es cota inferior para OPT. También, la de un cycle cover mínimo (que resuelve como matching perfecto de un grafo bipartito, flujo con costos)

Superstring – Factor 4

If $c = (i_1 \to i_2 \to \ldots i_l \to i_1)$ is a cycle in the prefix graph, let

$$\alpha(c) = \text{prefix}(s_{i_1}, s_{i_2}) \circ \ldots \circ \text{prefix}(s_{i_{l-1}}, s_{i_l}) \circ \text{prefix}(s_{i_l}, s_{i_1}).$$

Notice that each string $s_{i_1}, s_{i_2}, \ldots, s_{i_l}$ is a substring of $(\alpha(c))^{\infty}$. Next, let

$$\sigma(c) = \alpha(c) \circ s_{i_1}.$$

$s_{i_1} = r$ es llamado como string representante de *c* y cualquier s_{i_j} puede haber tomado ese papel. Siempre conviene elegir uno de menor longitud. Claramente $\sigma(c)$ es superstring de $s_{i_1}, s_{i_2}, \ldots, s_{i_l}$ independientemente quien sea representante. Como la solución final es $\sigma(c_1) \circ \sigma(c_2) \circ \ldots \circ \sigma(c_k)$, entonces es superstring de todo $s_i, 1 \leq i \leq n$.

En caso que si se pudiera asegurar que cada $r_i \leq \alpha(c_i)$,$1 \leq i \leq k$ entonces el algoritmo sería de factor 2. Por lo tanto, los casos complicados son los strings de algún ciclo son todos largos.

Superstring – Factor 4

Lemma 7.2 *If each string in $S' \subseteq S$ is a substring of t^∞ for a string t, then there is a cycle of weight at most $|t|$ in the prefix graph covering all the vertices corresponding to strings in S'.*

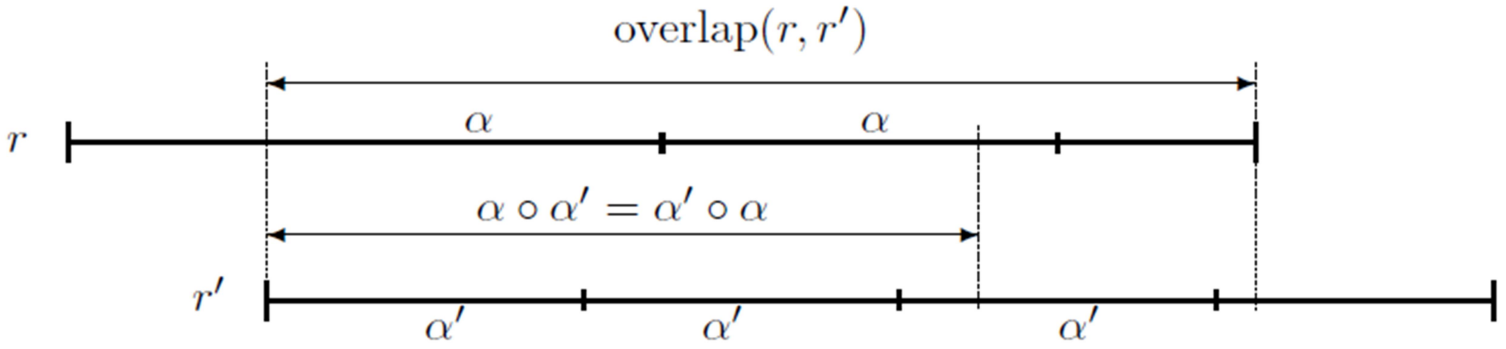
Proof: For each string in S', locate the starting point of its first occurrence in t^∞. Clearly, all these starting points will be distinct (since no string in S is a substring of another) and will lie in the first copy of t. Consider the cycle in the prefix graph visiting the corresponding vertices in this order. Clearly, the weight of this cycle is at most $|t|$. □

Lemma 7.3 *Let c and c' be two cycles in $\mathcal{C}$, and let r, r' be representative strings from these cycles. Then*

$$|\text{overlap}(r, r')| < \text{wt}(c) + \text{wt}(c').$$

Superstring – Factor 4

Proof: Suppose, for contradiction, that $|\text{overlap}(r, r')| \geq \text{wt}(c) + \text{wt}(c')$. Denote by α (α') the prefix of length $\text{wt}(c)$ ($\text{wt}(c')$, respectively) of $\text{overlap}(r, r')$.



Clearly, $\text{overlap}(r, r')$ is a prefix of both α^∞ and $(\alpha')^\infty$. In addition, α is a prefix of $(\alpha')^\infty$ and α' is a prefix of α^∞. Since $\text{overlap}(r, r') \geq |\alpha| + |\alpha'|$, it follows that α and α' commute, i.e., $\alpha \circ \alpha' = \alpha' \circ \alpha$. But then, $\alpha^\infty = (\alpha')^\infty$. This is so because for any $k > 0$,

$$\alpha^k \circ (\alpha')^k = (\alpha')^k \circ \alpha^k.$$

Superstring – Factor 4

Hence, for any $N > 0$, the prefix of length N of α^∞ is the same as that of $(\alpha')^\infty$.

Now, by Lemma 7.2, there is a cycle of weight at most $\mathrm{wt}(c)$ in the prefix graph covering all strings in c and c', contradicting the fact that $\mathcal{C}$ is a minimum weight cycle cover. □

Theorem 7.4 *Algorithm 7.1 achieves an approximation factor of 4 for the shortest superstring problem.*

Proof: Let $\mathrm{wt}(\mathcal{C}) = \sum_{i=1}^{k} \mathrm{wt}(c_i)$. The output of the algorithm has length

$$\sum_{i=1}^{k} |\sigma(c_i)| = \mathrm{wt}(\mathcal{C}) + \sum_{i=1}^{k} |r_i|,$$

where r_i denotes the representative string from cycle c_i. We have shown that $\mathrm{wt}(\mathcal{C}) \leq \mathrm{OPT}$. Next, we show that the sum of the lengths of representative strings is at most $3 \cdot \mathrm{OPT}$.

Superstring – Factor 4

Assume that $r_1, \ldots, r_k$ are numbered in order of their leftmost occurrence in the shortest superstring of S. Using Lemma 7.3, we get the following lower bound on OPT:

$$\mathrm{OPT} \geq \sum_{i=1}^{k} |r_i| - \sum_{i=1}^{k-1} |\mathrm{overlap}(r_i, r_{i+1})| \geq \sum_{i=1}^{k} |r_i| - 2\sum_{i=1}^{k} \mathrm{wt}(c_i).$$

Hence,

$$\sum_{i=1}^{k} |r_i| \leq \mathrm{OPT} + 2\sum_{i=1}^{k} \mathrm{wt}(c_i) \leq 3 \cdot \mathrm{OPT}.$$

□

Algoritmo de factor ½ para máxima compresión

We give a superstring algorithm that achieves at least half the optimal compression. Suppose that the strings to be compressed, $s_1, \cdots, s_n$, are numbered in the order in which they appear in a shortest superstring. Then, the optimal compression is given by

$$\sum_{i=1}^{n-1} |\text{overlap}(\sigma_i, \sigma_{i+1})|.$$

This is the weight of the traveling salesman path $1 \to 2 \to \ldots \to n$ in the *overlap graph*, H, of the strings $s_1, \cdots, s_n$. H is a directed graph that has a vertex v_i corresponding to each string s_i, and contains an edge $(v_i \to v_j)$ of weight $|\text{overlap}(s_i, s_j)|$ for each $i \neq j$, $1 \leq i, j \leq n$ (H has no self loops).

Algoritmo de factor ½ para máxima compresión

The optimal compression is upper bounded by the cost of a maximum traveling salesman tour in H, which in turn is upper bounded by the cost of a maximum cycle cover. The latter can be computed in polynomial time using matching, similar to the way we computed a minimum weight cycle cover. Since H has no self loops, each cycle has length at least 2. Remove the lightest edge from each cycle of the maximum cycle cover to obtain a set of disjoint paths. The sum of weights of edges on these paths is at least half the optimal compression. Overlap strings $s_1, \cdots, s_n$ according to the edges of these paths and concatenate the resulting strings. This gives a superstring achieving at least half the optimal compression.

Superstring – Factor 3

Algorithm 7.5 (Shortest superstring – factor 3)

1. Construct the prefix graph corresponding to strings in S.
2. Find a minimum cycle cover of the prefix graph, $\mathcal{C} = \{c_1, \ldots, c_k\}$.
3. Run the greedy superstring algorithm on $\{\sigma(c_1), \ldots, \sigma(c_k)\}$ and output the resulting string, say τ.

Let OPT_σ denote the length of the shortest superstring of the strings in $S_\sigma = \{\sigma(c_1) \ldots \sigma(c_k)\}$, and let r_i be the representative string of c_i.

Lemma 7.6 $|\tau| \leq \mathrm{OPT}_\sigma + \mathrm{wt}(\mathcal{C})$.

Superstring – Factor 3

Proof: Assume w.l.o.g. that $\sigma(c_1), \ldots, \sigma(c_k)$ appear in this order in a shortest superstring of S_σ. The maximum compression that can be achieved on S_σ is given by

$$\sum_{i=1}^{k-1} |\text{overlap}(\sigma(c_i), \sigma(c_{i+1}))|.$$

Since each string $\sigma(c_i)$ has r_i as a prefix as well as suffix, by Lemma 7.3,

$$|\text{overlap}(\sigma(c_i), \sigma(c_{i+1}))| \leq \text{wt}(c_i) + \text{wt}(c_{i+1}).$$

Hence, the maximum compression achievable on S_σ is at most $2 \cdot \text{wt}(\mathcal{C})$, i.e., $||S_\sigma|| - \text{OPT}_\sigma \leq 2 \cdot \text{wt}(\mathcal{C})$.

Superstring – Factor 3

The compression achieved by the greedy superstring algorithm on S_σ is at least half the maximum compression. Therefore,

$$||S_\sigma|| - |\tau| \geq \frac{1}{2}(||S_\sigma|| - \mathrm{OPT}_\sigma).$$

Therefore,

$$2(|\tau| - \mathrm{OPT}_\sigma) \leq ||S_\sigma|| - \mathrm{OPT}_\sigma \leq 2 \cdot \mathrm{wt}(\mathcal{C}).$$

The lemma follows. □

Superstring – Factor 3

Lemma 7.7 $\mathrm{OPT}_\sigma \leq \mathrm{OPT} + \mathrm{wt}(\mathcal{C})$.

Proof: Let OPT_r denote the length of the shortest superstring of the strings in $S_r = \{r_1, \ldots, r_k\}$. The key observation is that each $\sigma(c_i)$ begins and ends with r_i. Therefore, the maximum compression achievable on S_σ is at least as large as that achievable on S_r, i.e.,

$$||S_\sigma|| - \mathrm{OPT}_\sigma \geq ||S_r|| - \mathrm{OPT}_r.$$

Clearly, $||S_\sigma|| = ||S_r|| + \mathrm{wt}(\mathcal{C})$. This gives

$$\mathrm{OPT}_\sigma \leq \mathrm{OPT}_r + \mathrm{wt}(\mathcal{C}).$$

The lemma follows by noticing that $\mathrm{OPT}_r \leq \mathrm{OPT}$. □

Theorem 7.8 *Algorithm 7.5 achieves an approximation factor of 3 for the shortest superstring problem.*