Kürzeste überdeckende Wörter - Shortest Superstrings

Sebastian Bauer

1. Oktober 2003

Inhaltsverzeichnis

1 Einführung

Für die Sequenzierung von Genomen wird häufig die sogenannte shotgun-Technik benutzt. Dabei werden die mehrfach vorhandenen DNA-Stränge auf biochemischen Weg an verschieden Stellen aufgetrennt. Die Trennstellen sind dabei jedoch unbekannt. Diese kurze DNA-Fragmente lassen sich nun sequenzieren. Der nächste Schritt besteht darin, diese Fragmente zu der Sequenz des ursprünglichen DNA-Strangs zusammenzusetzen. Dabei nimmt man an, dass das kürzeste Wort, das alle diese Segmente als Teilwörter enthält, eine gute Annährung des orginalen DNA-Strangs darstellt. Das Finden des kürzesten Wortes wird als das Problem der *kürzesten überdeckenden Wörter* bezeichnet (engl. *shortest superstring* Problem, kurz SSP).

Definition 1. Gegeben sei ein endliches Alphabet Σ und eine Menge von n Wörtern, $S = \{s_1, s_2, \ldots, s_n\} \subseteq \Sigma^+$. Ein *überdeckendes Wort* s (engl. *superstring*) ist ein Wort, das jedes s_i als ein Teilwort enthält. Ist die Länge von s minimal, so heißt s ein *kürzestes überdeckendes Wort*.

Beispiel 1. Sei S={abra,brac,cabr,brad,bra}. Man sieht sofort, dass bra ein Teilwort von z.B. abra ist und man es somit ignorieren kann. Es ist jetzt nicht schwer, aus S ein zugehörigen Superstring zu finden. Indem man die restlichen Wörter aus S konkateniert, erhält man den *naiven* Superstring s=abrabraccabrbrad mit $|s|$=16. Ein kürzerer Superstring wäre s=abracabrad mit $|s|$=10. Da es zu S keinen Superstring mit kürzerer Länge gibt, ist s also ein kürzester Superstring.

Wir sind nun darin interessiert, ein mögliches effizientes Verfahren zur Suche nach einem kürzesten Superstring zu finden, und setzen im folgenden immer voraus, dass S nur Wörter enthält, die keine Teilwörter von andern Wörtern in S sind. Ansonsten kann man diese Wörter leicht in Polynomialzeit aus S entfernen. Bezüglich der Komplexität des Problem, kommt man aber schnell auf ein ernüchterndes Resultat, das im nächsten Satz festgehalten wird.

Satz 2. *SSP ist NP-vollständig.*

Beweis. Wir benutzen die Entscheidungsvariante des SSP Problems. D.h. zu gegebener Menge S soll geprüft werden, ob ein Superstring der Länge k existiert. Es ist leicht einzusehen, dass SSP$\in$NP, da man in polynomieller Zeit überprüfen kann, ob ein Wort s Superstring der Wörter von S ist.

Um zu zeigen, dass SSP NP-hart ist, reduzieren wir das NP-harte DHP Problem[1] auf SSP. Zu zeigen: DHP $\leq_p$ SSP, gesucht ist also eine Funktion $f(x)$, die in polynomieller Zeit berechenbar ist und folgende Eigenschaften erfüllt:

$$\forall x \in \Sigma^* : x \in \text{DHP} \Leftrightarrow f(x) \in \text{SSP}$$

Sei $d(v)$ die Anzahl ausgehender Kanten von $v \in V$. Dann arbeitet $f(x)$ wie folgt:

1. Teste, ob $x = (V, E)$, sonst gib ({abc},1) zurück.
2. Sei n die Anzahl der Knoten und m die Anzahl der Kanten im Graphen. Wir bilden im folgenden Wörter über einem Alphabet der Form $\{v_0, \ldots, v_{n-1}, v'_0, \ldots, v'_{n-1}, \#\}$ mit $v_i \in V$.
3. Bilde Menge $C = \{v\#v' | v \in V\}$ der *Konnektoren* von V.
4. Für jedes $v \in V$ nehme die Kanten $(v, w_0), (v, w_1), \ldots, (v, w_{d-1}) \in E$ mit $d = d(v)$. Sei $p_i(v) = v'w_{i-1}v'w_i$ [2] *gepaarte Kantenkodierung* und $p(v) = \{p_i(v) | 0 \leq i < d\}$ die Menge aller gepaarten Kantenkodierungen für v. Bilde $P = \bigcup_{v \in V} p(v)$.
5. Gib $(C \cup P, 3n + 2m + 1)$ zurück.

Die Funktion lässt sich offensichtlich in Polynomialzeit berechnen. Es bleibt jetzt noch, die Äquivalenz zu zeigen.

$x \in$ DHP: Sei die Folge $(v_0, v_1), (v_1, v_2), \ldots, (v_{n-2}, v_{n-1})$ einer der möglichen Pfade. Betrachten wir zunächst den Konnektor eines Knoten $v \in V$ und die zugehörigen gepaarten Kantenkodierungen $p(v)$. Für jedes $0 \leq i < d(v)$ ist $s(v, i) = v\#v'w_{i-1}v'w_i \ldots v'w_{i-1}$[3] ein Wort, dass den Konnektor von v und $p(v)$ überdeckt. Wir stellen fest, dass $|s(v, i)| = 4 + 2 \cdot d(v)$. Sei nun $j < n - 1$. Mit $s(v_j)$ bezeichnen wir das Wort, das entsteht, wenn wir für $s(v_j, i)$ i so wählen, dass $w_{i-1} = v_{j+1}$. Dieses i existiert, da die Kante (v_j, v_{j+1}) vorhanden ist. Setze $s(v_{n-1}) = s(v_{n-1}, i)$ mit i beliebig. Offensichtlich überlappen sich $s(v_j)$ und $s(v_{j+1})$ um genau ein Zeichen (nämlich um v_{j+1}). Man erhält s, indem man an $s(v_0)$ nacheinander für alle $j < n - 1$ die $s(v_{j+1})$'s in dem einen Zeichen überlappend aneinanderhängt. Klar, s ist Superstring aller $s(v_j)$'s und damit aller Kodierungen für Knoten und gepaarten Kanten. Es gilt:

$$|s| = |s(v_0)| + \sum_{j=1}^{n-1}(|s(v_j)| - 1) = \sum_{j=0}^{n-1}(|s(v_j)| - 1) + 1 = \sum_{j=0}^{n-1}(3 + 2d(v_j)) + 1 =$$

[1] Directed Hamiltonian Path, prüft für gerichtete Graphen, ob ein Pfad existiert, der durch alle Knoten geht. Wir verwenden hier eine Variante, bei der der Ausgangsgrad eines Knotens immer größer 0 ist.

[2] Indexarithmetik für w immer modulo $d(v)$

[3] zyklische Anordnung

$$3n + 2\sum_{j=0}^{n-1} d(v_j) + 1 = 3n + 2m + 1.$$

und damit $f(x) \in$SSP.

$x \notin$ DHP: Klar, wenn $x \neq (V, E) \Rightarrow f(x) \notin$SSP.

Angenommen, es gibt einen Superstring s, so das $f(x) \in$ SSP. Es gilt dann: $|s| \leq 3n+2m+1$. Wir nehmen an, die Knoten bekommen durch das Auftreten ihres Konnektors ihre Indizierung, d.h. der Knoten für den ersten Konnektor sei v_0, für den zweiten v_1 usw.

Zunächst ist klar, dass sich in s genau $3n$ Zeichen befinden, die die Konnektoren der Form $v_k \# v'_k$ überdecken, da sich die Konnektoren untereinander nicht überlappen. Weiterhin erkennt man, dass sich $p_i(v_k)$ und $p_j(v_l)$ für $k \neq l$ und beliebiges i und j nicht überlappen können. Also können sich die Superstrings von $p(v_k)$ und $p(v_l)$ nicht überlappen. Die Länge eines kürzesten Superstring für ein $p(v_k)$ beträgt $2 + 2d(v_k)$. In s finden wir demnach $\sum_{k=0}^{n-1}(2 + 2d(v_k)) = 2n + 2m$ Zeichen, die die Superstrings von $p(v_k)$ für alle $v_k \in V$ überdecken. Wir halten weiterhin fest, dass wir in s tatsächlich die kürzesten Superstrings von $p(v_k)$ als Teilwörter finden, andernfalls wäre s länger. Weiterhin können sich die Konnektoren ja nur mit diesen Superstrings überlappen. Wir können also davon ausgehen, dass sich Konnektoren und die Superstrings von $p(v_k)$ in s abwechseln.

Wir nehmen nun an, dass der Superstring mit einem Konnektor beginnt. Ein Konnektor $v_k \# v'_k$ kann sich mit genau einem $p_i(v_k) = v'_k w_{i-1} v'_k w_i$ in dem Zeichen v'_k überlappen. Da es n Knoten gibt und für jedes v_k ein $p_i(v_k)$ existiert, sparen wir so n Zeichen ein. Wir müssen jedoch weitere $n-1$ Zeichen einspaaren, um $|s| \leq 3n + 2m + 1$ zu erreichen.

Nun, es gibt noch die Möglichkeit der Überlappung eines $p_i(v_k)$ mit einem Konnektor $v_{k+1} \# v'_{k+1}$ beim Zeichen v_{k+1}. Diese Möglichkeit gibt es, wenn in G eine Kante vom Knoten v_k nach v_{k+1} existiert. Da keine weiteren Überlappungsmöglichkeiten vorhanden sind, existieren wenigstens $n-1$ solcher Überlappungen, beginnend bei $k = 0$. Das bedeutet nicht anderes, als das G ein Pfad $0 \to 1 \to \ldots \to n-1$ besitzt. Dies steht im Widerspruch, das G keinen gerichteten Hamiltonpfad hat.

Beginnt der Superstring nicht mit einem Konnektor, so muss er mit einem Konnektor enden. Da es dann nur $n-1$ Möglickeiten gibt, dass sich ein v'_k Zeichen eines Konnektors mit den Superstring von $p(v_k)$ überlappen kann, muss es n Überlappungen der v_k Zeichen (dem Zeichen vor dem #) geben. Folglich besitzt G einen gerichteten Hamiltonkreis und somit einen gerichteten Hamiltonpfad, was ein Widerspruch ist.

□

Wir haben nun festgestellt, dass das Problem NP-vollständig ist. Dies bedeutet, dass momentan keine Algorithmen bekannt sind, die das Problem in polynomieller Zeit exakt lösen. Wir können allerdings versuchen, Polynomialzeitalgorithmen zu entwickeln, die zumindest eine angenäherte Lösung finden. Im weiteren werden nun einige solcher *Approximationsalgorithmen* vorgestellt.

2 Faktor 4 Approximation

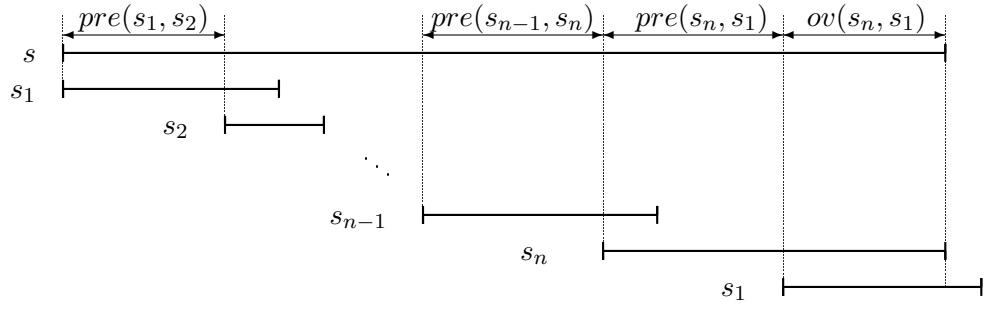
Für die Länge des optimalen Superstring für eine Menge von Strings schreiben wir OPT. Es wird nun ein Superstring Algorithmus vorgestellt, dessen Ausgabe nicht länger als 4·OPT ist. Wie kann man jedoch die Richtigkeit einer solchen Aussage zeigen? Das Bestimmen der optimalen Lösung selbst ist ja NP-hart.

Die Idee ist, eine möglichst gute untere Schranke für OPT zu finden, die leicht in Polynomialzeit bestimmbar ist. Die untere Schranke sei OPT_l. Es gilt also $\mathrm{OPT}_\mathrm{l} \leq \mathrm{OPT}$. Lässt sich zeigen, dass die Ausgabe des Algorithmus innerhalb eines Faktors der unteren Schranke liegt, dann liegt sich auch innerhalb des selben Faktors der optimalen Lösung, denn aus $|s| \leq k \cdot \mathrm{OPT}_\mathrm{l}$ folgt $|s| \leq k \cdot \mathrm{OPT}$. Wir beginnen zunächst damit, einige wichtige Bezeichnungen für unser Problem einzuführen.

Mit $\mathit{overlap}(s_i, s_j)$ bezeichnen wir die Überlappung von s_i und s_j, also den längsten Suffix von s_i, der ein Präfix von s_j ist und mit $\mathit{prefix}(s_i, s_j)$ den Präfix von s_i, der durch Entfernung der Überlappung mit s_j entsteht.

Definition 3. Sei $X = \{x_1, \ldots, x_n\}$ eine Menge von Wörtern. Ein zugehöriger *Präfix-graph I(X)* ist ein gerichteter Graph, der zu jedem Wort x_i einen entprechenden Knoten v_i und Kanten (v_i, v_j) mit Kosten $|\mathit{prefix}(x_i, x_j)|$ für jedes $1 \leq i, j \leq n$ mit $i \neq j$ besitzt. Eigenschleifen (v_i, v_i) haben die Kosten $|x_i|$.

Wir nehmen nun an, dass $s_1, s_2, \ldots, s_n$ in dieser Reihenfolge im kürzesten Superstring s auftreten und wollen die Anordnung genauer untersuchen.

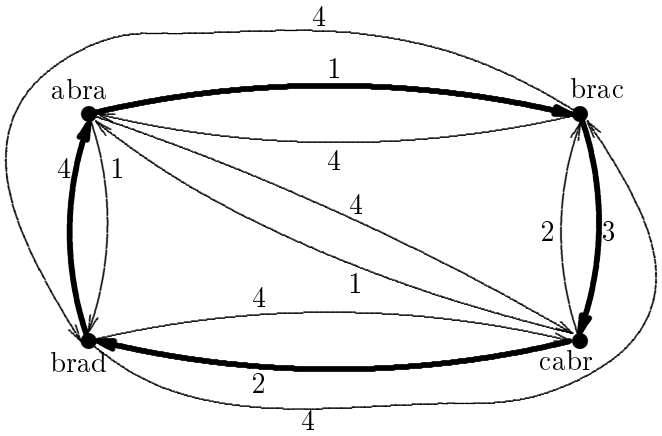


Für OPT gilt also:

$$\mathrm{OPT} = |s| = |\mathit{prefix}(s_1, s_2)| + |\mathit{prefix}(s_2, s_3)| + \ldots + |\mathit{prefix}(s_n, s_1)| + |\mathit{overlap}(s_n, s_1)|$$

Dieser Gleichung sieht man die enge Beziehung zum bekannten TSP Problem[4] sofort an. Denn die Tour $1 \to 2 \to \ldots \to n \to 1$ auf dem Präfixgraphen $I(S)$ hat Kosten von $\mathrm{TSP} = |\mathit{prefix}(s_1, s_2)| + |\mathit{prefix}(s_2, s_3)| + \ldots + |\mathit{prefix}(s_n, s_1)|$ und stellt damit eine untere Schranke für OPT dar, d.h. $\mathrm{TSP} \leq \mathrm{OPT}$. Da TSP aber ebenfalls NP-hart ist, nützt uns diese untere Schranke zunächst nicht viel.

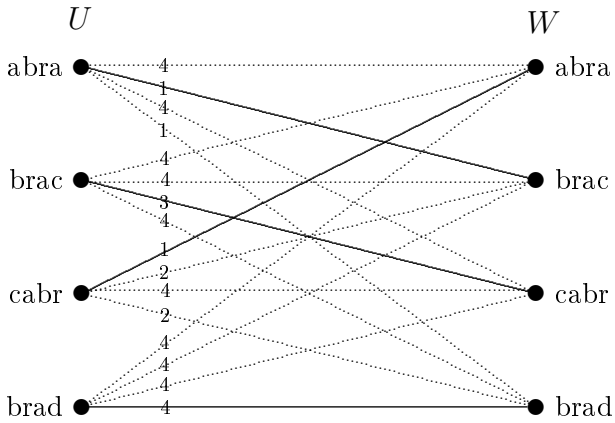
[4]Travelling Sales Person, zu gegebenen vollständigen Graphen und einer Funktion, die jeder Kante bestimmte Kosten zuordnet, ist ein hamiltonscher Kreis mit minimalen Kosten zu finden. Siehe auch Algorithmen- oder Komplexitätstheorie.



Der Präfixgraph $I(S)$ für $S = \{\text{abra,brac,cabr,brad}\}$ (ohne Eigenschleifen). Die Kanten eines Minimum Salesperson Kreises sind hervorgehoben. Beginnt man bei abra, so entsteht unser Superstring abracabrad, wenn man die Wörter der Knoten in der Reihenfolge des Kreises maximal überlappend aneinanderhängt.

Ein zum TSP Problem verwandtes Problem ist das Finden einer *Kreisüberdeckung* (engl. *cycle cover*) mit minimalen Kosten. Zunächst ist eine Kreisüberdeckung eines Graphen eine Menge C, die aus disjunkten Kreisen besteht, die zusammen alle Knoten des Graphen überdecken. Die Kosten $\mathit{weight}(c)$ eines Kreises $c \in C$ sei die Summe der Gewichte aller Kanten von c. Die Kosten der Überdeckung, bezeichnet mit $\mathit{weight}(C)$, sei die Summe der Kosten aller Kreise aus C. Bei einer Kreisüberdeckung mit minimalen Kosten wird ein C gesucht, für das gilt, dass $\mathit{weight}(C)$ minimal ist. Da jede Minimum TSP Tour $1 \to 2 \to \ldots \to n \to 1$ auch eine Kreisüberdeckung ist, sind folglich die Kosten einer minimalen Kreisüberdeckung auch nie höher. Schreiben wir jetzt für diese Kosten CC, so gilt: $\mathit{weight}(C)$=CC≤TSP≤OPT. Damit erhalten wir eine weitere untere Schranke für OPT.

Eine minimale Kreisüberdeckung in $I(S)$ zu finden, gelingt nun in Polynomialzeit. Dazu erstellen wir einen bipartiten Graphen H mit Knotenmenge $U = \{u_1, u_2, \ldots, u_n\}$ und $W = \{w_1, w_2, \ldots, w_n\}$ und Kanten (u_i, w_j) mit Kosten von $|\mathit{prefix}(s_i, s_j)|$ für $1 \leq i, j \leq n$ und $i \neq j$. Die Kanten (u_i, w_i) bekommen Kosten in Höhe von $|s_i|$. Einer Kreisüberdeckung mit minimalen Kosten von $I(S)$ entspricht nun ein perfektes Matching von H mit minimalen Kosten. Hat man also solch ein perfektes Matching (bestimmbar in polynomieller Zeit), so hat man auch eine Kreisüberdeckung minimaler Kosten von $I(S)$.



Der zum Beispiel gehörende bipartite Graph. Kanten, die zu einem minimalen perfekten Matching gehören sind hervorgehoben.

Sei $a \circ b$ Konkatenation der Wörter a und b und sei $c = (i_1 \to i_2 \to \ldots \to i_l \to i_1)$ ein Kreis im Präfixgraphen, dann ist

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

Wir setzen

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

und halten fest, dass

1. $|\alpha(c)| = \mathit{weight}(c)$ und $|\sigma(c)| = \mathit{weight}(c) + |s_{i_1}|$,
2. $\sigma(c)$ alle Wörter $s_{i_1}, \ldots, s_{i_l}$ überdeckt,
3. alle Wörter $s_{i_1}, \ldots, s_{i_l}$ Teilwörter von $\alpha(c) \circ \alpha(c) \circ \ldots \circ \alpha(c) = \alpha(c)^\infty$ sind und dass
4. $\sigma(c)$ durch das „Öffnen“ von Kreis c an einem beliebigen Wort s_{i_1} gefunden werden kann. Wir nennen s_{i_1} das *repräsentative* Wort dieses Kreises c.

Wir können jetzt den kompletten Algorithmus angeben:

Algorithmus 1 (Shortest superstring - Faktor 4).

1. Erstelle Präfixgraphen $I(S)$.
2. Finde eine Kreisüberdeckung minimaler Kosten im Präfixgraphen, $C = \{c_1, \ldots, c_k\}$.
3. Gebe $\sigma(c_1) \circ \ldots \circ \sigma(c_k)$ zurück.

Man überzeugt sich leicht, dass dieser Algorithmus tatsächlich ein überdeckendes Wort der Wörter in S liefert. Findet man in jedem Kreis ein repräsentatives Wort, dessen Länge maximal so groß ist wie das Gewichts des Kreises, so ist die Ausgabe nicht länger als 2·OPT. Was passiert aber, wenn alle Wörter eines Kreises c länger sind? Zunächst bemerken wir, dass alle diese Wörter *periodisch* sein müssen, da sie ja Teilwörter von $\alpha(c)^\infty$ sind.

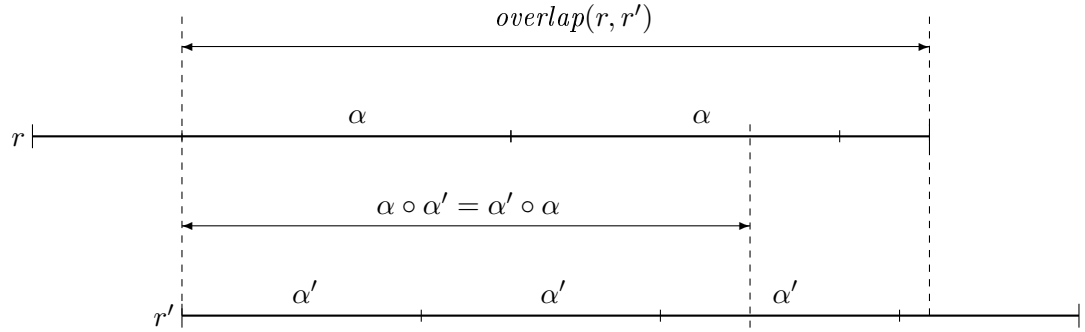
Lemma 4. *Falls jedes Wort in $S' \subseteq S$ ein Teilwort von t^∞ ist, so existiert ein Kreis im Präfixgraphen, der alle Knoten, die den Wörten aus S' entsprechen, überdeckt. Die Kosten des Kreises sind höchstens $|t|$.*

Beweis. Für jedes Wort in S' lokalisiert man die Startposition des ersten Vorkommens in t^∞. Da kein Wort Teilwort eines anderen ist, sind diese Positionen für jedes Wort verschieden und liegen in der ersten Kopie von t. Betrachtet man den Kreis im Präfixgraphen in der Folge des Vorkommens, so ist klar, dass das Gewicht des Kreises nicht höher als $|t|$ sein kann. □

Lemma 5. *Sind c und c' zwei Kreise aus C und sind r und r' zwei repräsentative Wörter dieser Kreise. Es gilt:*

$$|overlap(r, r')| < weight(c) + weight(c')$$

Beweis. Angenommen es gelte $|overlap(r, r')| \geq weight(c) + weight(c')$. Seien nun α und α' die Präfixe von $overlap(r, r')$ der Länge $weight(c)$ bzw. $weight(c')$.



Klar ist, dass $overlap(r, r')$ ein Präfix von α^∞ und ein Präfix von $(\alpha')^\infty$ ist. Es ist α dann aber auch ein Präfix von $(\alpha')^\infty$ und α' ein Präfix von α^∞. Wegen $|overlap(r, r')| \geq |\alpha| + |\alpha'|$ gilt nun $\alpha \circ \alpha' = \alpha' \circ \alpha$. Für jedes $k > 0$ folgt daraus, dass $\alpha^k \circ (\alpha')^k = (\alpha')^k \circ \alpha^k$ und schliesslich $\alpha^\infty = (\alpha')^\infty$. Da alle Wörter in c' Teilwörter von $(\alpha')^\infty$ sind, sind sie also auch Teilwörter von α^∞. Nach Lemma 4 gibt es nun einen Kreis, dessen Gewicht höchstens $|\alpha| = weight(c)$ ist und alle Wörter in c und c' überdeckt. Dies steht im Widerspruch, dass C eine minimale Kreisüberdeckung war. □

Satz 6. *Algorithmus 1 erreicht einen Approximationsfaktor von 4 für das Shortest Superstring Problem.*

Beweis. Sei $weight(C) = \sum_{i=1}^{k} weight(c_i)$. Die Ausgabe des Algorithmus hat die Länge:

$$\sum_{i=i}^{k} |\sigma(c_i)| = \sum_{i=1}^{k} weight(c_i) + \sum_{i=1}^{k} |r_i| = weight(C) + \sum_{i=1}^{k} |r_i| \qquad (1)$$

Dabei bezeichnet r_i wieder das repräsentative Wort des Kreises c_i. Bereits erarbeitet wurde, dass $weight(C) \leq \text{OPT}$. Angenommen $r_1, r_2, \ldots, r_k$ sind wie ihr Auftreten im

kürzesten Superstring von S indiziert. Zunächst bestimmen wir für OPT eine weitere untere Schranke:

$$\sum_{i=1}^{k} |r_i| - \sum_{i=1}^{k-1} |overlap(r_i, r_{i+1})| =$$

$$|prefix(r_1, r_2)| + \ldots |prefix(r_k, r_1)| + |overlap(r_k, r_1)| \leq \text{OPT}.$$

Nun wenden wir Lemma 5 an und bekommen für OPT:

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

Umgestellt nach $\sum_{i=1}^{k} |r_i|$ ergibt sich:

$$\sum_{i=1}^{k} |r_i| \leq \text{OPT} + 2\underbrace{\sum_{i=1}^{k} weight(c_i)}_{\leq \text{OPT}} \leq 3\text{OPT}.$$

Eingesetzt in (1) erhalten wir schliesslich:

$$\sum_{i=1}^{k} |\sigma(c_i)| = weight(C) + \sum_{i=1}^{k} |r_i| \leq 4\text{OPT}.$$

□

3 Kompression

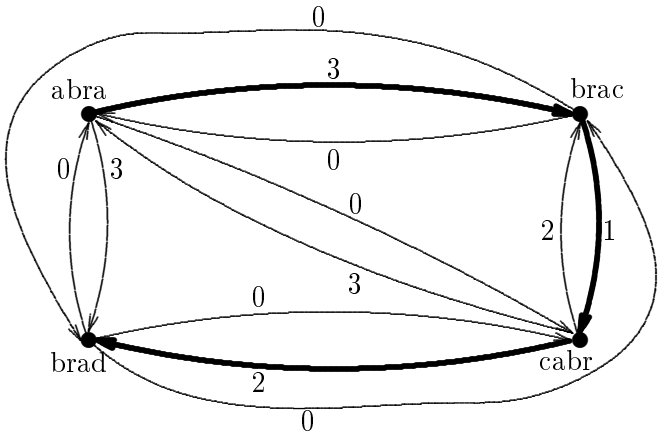
Ein weiteres wichtiges Anwendungebiet für das Finden möglichst kurzer überdeckender Wörter besteht in der Datenkompression. Sind die Positionen und die Längen aller Wörter im zugehörigen Superstring bekannt, so lässt sich leicht die ursprüngliche Wortmenge wiederherstellen.

Definition 7. Gegeben ist eine Menge von Wörtern bezeichnet mit X. Mit $\|X\|$ kennzeichnen wir die Summe der Längen aller Wörter aus $\|X\|$. Die *Kompression*, die von einen Superstring s erreicht wird, ist die Differenz aus der Summe der Längen aller Eingabewörter und $|s|$, also $\|S\| - |s|$.

Offensichtlich wird die maximale Kompression nur von einem kürzesten Superstring erreicht.

Es gibt nun Algorithmen, von denen bekannt ist, dass sie wenigstens die halbe der maximal möglichen Kompression erreichen. Wir werden später diese Eigenschaft zur Verbesserung von Algorithmus 1 verwenden. Zunächst wollen wir jedoch ein Algorithmus mit einer solchen Eigenschaft vorstellen.

Definition 8. Sei $X = \{x_1, \ldots, x_n\}$ eine Menge von Wörtern. Ein zugehöriger *Überlappungsgraph H(X)* ist ein gerichteter Graph, der zu jedem Wort x_i einen entprechenden Knoten v_i und Kanten (v_i, v_j) mit Gewicht $|overlap(x_i, x_j)|$ für jedes $i \neq j, 1 \leq i, j \leq n$ besitzt ($H(X)$ hat keine Eigenschleifen).



Der Überlappungsgraph $H(S)$ für $S = \{$abra,brac,cabr,brad$\}$. Die Kanten eines Maximum Salesperson Pfades sind hervorgehoben. Beginnt man bei abra, so entsteht unser Superstring abracabrad, wenn man die Wörter der Knoten in der Reihenfolge des Kreises maximal überlappend aneinanderhängt.

Wir nehmen an, dass die zu komprimierenden Wörter $s_1, \ldots, s_n$ in der Reihenfolge auftreten, wie sie auch im kürzesten Superstring zu finden sind. Die maximale Kompression ist dann bestimmt durch

$$\sum_{i=1}^{n-1} |overlap(s_i, s_{i+1})| .$$

Dies sind die Kosten des Maximum Traveling Salesperson Pfads $1 \to 2 \to \ldots \to n$ im Überlappungsgraphen $H(S)$ der Wörter $s_1, \ldots, s_n$. Diese Kosten werden durch die Kosten der maximalen TSP Tour begrenzt, die wiederrum durch eine Kreisüberdeckung mit maximalen Kosten beschränkt wird. Diese kann wieder ähnlich der minimalen Kreisüberdeckung mittels eines Matchings bestimmt werden.

Da $H(S)$ keine Eigenschleifen besitzt, hat jeder Kreis eine Länge von wenigstens 2. Entferne die Kante mit geringsten Kosten von jedem Kreis, um eine Menge von disjunkten Pfaden zu erhalten. Die Summe der Gewichte entlang der Pfade ist wenigstens halb so groß, wie optimale Kompression. Überlappe die Wörter $s_1, \ldots, s_n$ entlang den Kanten der Pfade und konkateniere die resultierenden Wörter. Die Kompression des daraus entstehenden Superstring ist wenigstens halb so hoch, wie die optimale Kompression.

Ein weiterer Algorithmus, der einen Superstring findet, der wenigstens halbe Kompression erreicht, ist der sogenannte *Greedy Superstring Algorithmus* (ohne Beweis). Er fügt nacheinander zwei Wörter mit maximaler Überlappung aus S zusammen, bis nur noch ein Wort übrig ist. Dieses Wort ist ein Superstring s.

4 Faktor 3 Approximation

Der Faktor 4 Algorithmus lässt sich auf recht einfache Weise noch weiter verbessern. Denn jeder Superstring der Wörter $\sigma(c_i), i = 1, \ldots, k$ ist auch ein Superstring aller Wörter aus S. Anstatt die $\sigma(c_i)$ einfach aneinanderzuhängen, können wir nun versuchen, sie so gut wie möglich überlappen zu lassen.

Algorithmus 2 (Shortest superstring - Faktor 3).

1. Erstelle Präfixgraphen $I(S)$.
2. Finde eine Kreisüberdeckung minimaler Kosten im Präfixgraphen, $C = \{c_1, \ldots, c_k\}$.
3. Führe Kompressionsalgorithmus aus Abschnitt 3 auf $\{\sigma(c_1), \ldots, \sigma(c_k)\}$ aus. Das Resultat sei τ.
4. Gebe τ zurück

Mit OPT_σ bezeichnen wir die Länge des kürzesten überdeckenden Wortes für die Wörter in $S_\sigma = \{\sigma(c_1), \ldots, \sigma(c_k)\}$. Sei r_i das repräsentative Wort von c_i.

Lemma 9. $|\tau| \leq \mathrm{OPT}_\sigma + \mathit{weight}(C)$.

Beweis. O.B.d.A erscheinen die $\sigma(c_1), \ldots, \sigma(c_k)$ in dieser Anordnung im kürzesten Superstring. Da jedes $\sigma(c_i)$ r_i als Präfix und Suffix hat, gilt nach Lemma 5

$$|\mathit{overlap}(\sigma(c_i), \sigma(c_{i+1}))| \leq \mathit{weight}(c_i) + \mathit{weight}(c_{i+1})$$

Für die maximale Kompression, die für S_σ erreicht werden kann ist, gilt dann

$$\|S_\sigma\| - \mathrm{OPT}_\sigma = \sum_{i=1}^{k-1} |\mathit{overlap}(\sigma(c_i), \sigma(c_{i+1}))| \leq 2 \cdot \mathit{weight}(C)$$

Die Kompression, die von unserem Kompressionalgorithmus auf S_σ erzielt werden kann, ist wengistens halb so groß, wie die maximale Kompression. Also gilt

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

und damit

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

Es folgt

$$|\tau| \leq \mathrm{OPT}_\sigma + \mathit{weight}(C)$$

□

Lemma 10. $\mathrm{OPT}_\sigma \leq \mathrm{OPT} + \mathit{weight}(C)$.

Beweis. Sei OPT_r die Länge des kürzesten überdeckenden Wortes der Wörter in $S_r = \{r_1, \ldots, r_k\}$. Da r_i Präfix und Suffix von $\sigma(c_i)$ ist, ist die maximale Kompression, die für S_σ erzielt werden kann wenigstens so groß wie die Kompression für S_r:

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

Für $\|S_\sigma\|$ gilt

$$\|S_\sigma\| = \sum_{i=1}^{k} |\sigma(c_i)| = \sum_{i=1}^{k} |\alpha(c_i) \circ r_i| = \mathit{weight}(C) + \|S_r\|,$$

was dann schließlich

$$\mathrm{OPT}_\sigma \leq \mathrm{OPT}_r + \mathit{weight}(C)$$

ergibt. Das Lemma folgt aus $\mathrm{OPT}_r \leq \mathrm{OPT}$. □

Wir kombinieren diese beiden Lemmas und erhalten:

Satz 11. *Algorithmus 2 erzielt einen Approximationsfaktor von 3 für das Shortest Superstring Problem.*

5 Zusammenfassung

Am Anfang haben gezeigt, dass es momentan keine exakten Polynomialzeitalgorithmen für das SSP Problem bekannt sind und dass es vermutlich auch keine gibt. Daraufhin haben wir Algorithmen aus [1] bzw. [2] vorgestellt, die das Problem bis auf Faktor 3 approximieren. Geht es auch besser? Nun, zunächst wurde in [1] gezeigt, dass SSP auch ein MAX SNP hartes Problem ist. Dies bedeutet, dass, sofern P≠NP gilt, man keine Approximationsalgorithmen entwickeln kann, die das Problem beliebig genau annähern können. Solche Probleme können nur bis auf einen bestimmten konstanten Faktor genau bestimmt werden. Mittlerweile wurden Algorithmen entwickelt, die unser SSP Problem bis auf Faktor $2\frac{1}{2}$ approximieren können. Folgende Tabelle listet Autoren einiger solcher Algorithmen auf.

Autoren	Jahr	Faktor
Blum, Jiang u.a.	1991	3
S. Teng, F. Yao	1993	$2\frac{8}{9}$
A. Czumaj u.a.	1994	$2\frac{5}{6}$
R. Kosaraju, J. Park, C. Stein	1994	$2\frac{50}{63}$
C. Armen, C. Stein	1995	$2\frac{3}{4}$
C. Armen, C. Stein	1996	$2\frac{2}{3}$
D. Breslauer, T. Jiang, Z. Jiang	1997	2,596
E. Z. Sweedyk	1999	$2\frac{1}{2}$

Eine besondere Stellung nimmt der Greedy Superstring Algorithmus ein. Der genaue Approximationsfaktor konnte bis heute noch nicht bewiesen werden. Es wird vermutet, dass der Approximationsfaktor hier 2 beträgt, da noch keine Eingabe gefunden wurde, bei dem der Algorithmus ein längeres Ergebnis geliefert hat. In [3] wurde jedoch gezeigt, dass für eine beschränkte Menge von Eingaben der Greedy Superstring Algorithmus sogar eine viel bessere Approximation liefert.

Literatur

[1] Blum, Jiang, Li, Tromp und Yannakakis. *Linear Approximation of Shortest Superstrings. 1994.*

[2] V. Vazirani. *Approximation Algorithms. 2001.*

[3] M. Weinard, G. Schnitger. *On the Greedy Superstring Conjecture. 2003.*