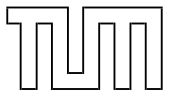
Algorithmische Bioinformatik 1

Dr. Hanjo Täubig

Lehrstuhl für Effiziente Algorithmen
(Prof. Dr. Ernst W. Mayr)
Institut für Informatik
Technische Universität München

Sommersemester 2009



Übersicht

Overlap Detection ohne Fehler

Algorithmus 20 : Suffix-Prefix-Matches(int[] S)

tree $T = T(S)$; /* verallgemeinerter Suffixbaum T_S */
stack_of_nodes $\mathrm{Stack}[k]$; /* je einen für jedes $s_i \in S$ */
int $\mathrm{level}[V(T_S)]$;
int $\mathrm{Overlap}[k, k]$;
$\mathrm{level}[\bar{\epsilon}] := 0$;
DFS(T, $\bar{\epsilon}$);

Overlap Detection ohne Fehler

Algorithmus 21 : DFS(tree T, node v)

```
forall the (i ∈ L(v)) do
    Stack[i].push(v);
if (v = s̄_j) then
    forall the (i ∈ [1 : k]) do
        if (not Stack[i].isEmpty()) then
            Overlap(s_i, s_j) := level[Stack[i].top()]};
forall the ((v, w) ∈ E(T)) do
    level[w] := level[v] + |label(v, w)|;
    DFS(w)
forall the (i ∈ L(v)) do
    Stack[i].pop();
```

Overlap Detection ohne Fehler: Laufzeit

- Für den reinen DFS-Anteil (der grüne Teil) der Prozedur benötigen wir Zeit $\mathcal{O}(kn)$.
- Die Kosten für die Pushs und Pops auf die Stacks (der schwarze Teil) betragen:

$$\sum_{v \in V(T_S)} |L(v)| =$$

$$\left|\left\{t \in \Sigma^* \,:\, \exists s \in S : \exists j \in [1 : |s|] : \, t = s_j \cdots s_{|s|}\right\}\right|$$

$$= \mathcal{O}(kn),$$

da in der zweiten Menge alle Suffixe von Wörtern aus S auftauchen.

Overlap Detection ohne Fehler: Laufzeit

- Die Aktualisierungen der Overlaps (der rote Teil) verursachen folgende Kosten.
- Da insgesamt nur k Knoten eine Sequenz aus S darstellen, wird die äußere if-Anweisung insgesamt genau k mal betreten.
- Darin wird die innere for-Schleife jeweils k mal aufgerufen.
- Da die Aktualisierung in konstanter Zeit erledigt werden kann, ist der gesamte Zeitbedarf $\mathcal{O}(k^2)$.

Overlap Detection ohne Fehler: Laufzeit

Theorem

Sei $S = \{s_1, \ldots, s_k\}$ eine Menge von Sequenzen mit $|s_i| = \Theta(n)$ für alle $i \in [1 : k]$.

Die längste Überlappung für jedes Paar (s_i, s_j) für alle $i, j \in [1 : k]$ kann in Zeit $\mathcal{O}(nk + k^2)$ berechnet werden.

Die Rechenzeit im vorherigen Theorem ist optimal, da die Eingabegröße $\Theta(kn)$ und die Ausgabegröße des Problems $\Theta(k^2)$ ist.

Layout mit Hamilton-Pfaden

- Ansatz: Layout mit Hilfe eines so genannten Overlap-Graphen
- Sei $S = \{s_1, \ldots, s_k\}$ die Menge der Fragmente und $D_{ij} = d(s_i, s_j)$ ein Score für den besten Suffix-Prefix-Overlap von s_i mit s_j.
- Im fehlerfreien Fall ist dies die Länge des Overlaps, ansonsten etwa ein Score, der sich beispielsweise aus dem semiglobalen Alignment ergibt.

Definition

Der gewichtete, gerichtete Graph (ohne Schleifen) $G_S = (V, E, \gamma)$ heißt Overlap-Graph, wenn

- $V = S$,
- $E = V \times V \setminus \{(v, v) : v \in V\}$,
- $\gamma(v, w) = D_{ij} = d(s_i, s_j)$ für $v = s_i$ und $w = s_j$.

Hamilton-Pfade

Definition

Sei $G = (V, E)$ ein gerichteter Graph.

Ein Pfad (bzw. Kreis) $(v_1, \ldots, v_\ell)$ heißt Hamilton-Pfad (-Kreis), wenn $\{v_1, \ldots, v_\ell\} = V$ und $|\{v_1, \ldots, v_\ell\}| = |V|$.

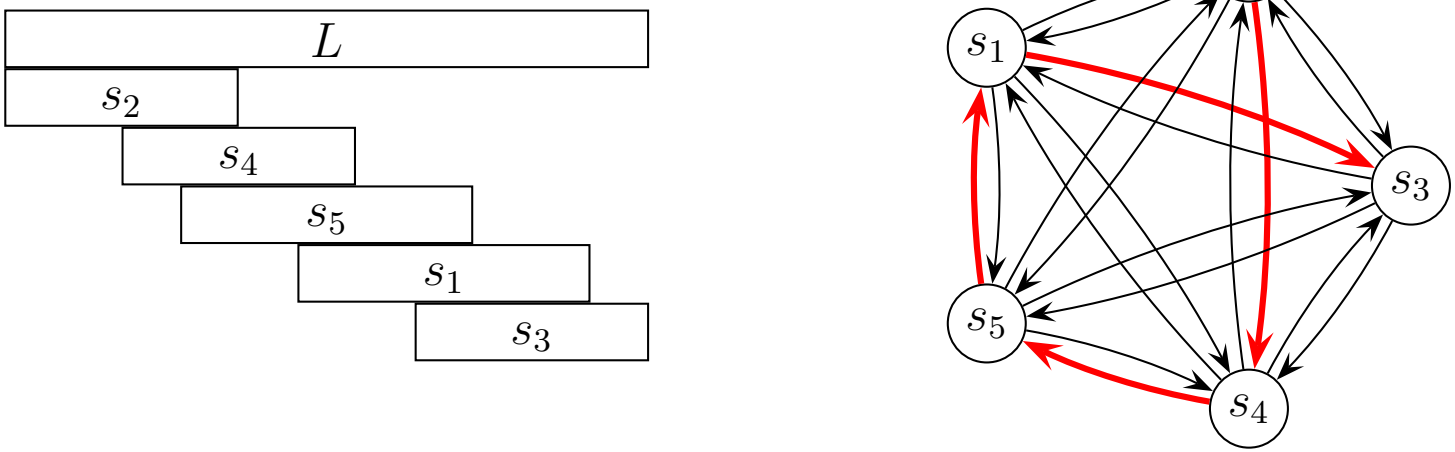
Ein Graph heißt hamiltonsch, wenn er einen Hamilton-Pfad besitzt.

Anmerkung: in der Literatur werden Graphen oft als hamiltonsch bezeichnet werden, wenn sie einen Hamilton-Kreis enthalten.

Hamilton-Pfade im Overlap-Graph

- Die Abfolge der Fragmente im Layout L entspricht offensichtlich einem Hamilton-Pfad im Overlap-Graphen.
- Berücksichtigen wir jetzt noch die Gewichte im Overlap-Graphen.
- Da wir annehmen, dass lange (und damit auch längste) Overlaps nicht durch Zufall entstehen, entspricht die Anordnung der Fragmente im optimalen (d.h. realen) Layout einem Hamilton-Pfad maximalen Gewichtes, wobei das Gewicht eines Pfades als die Summe seiner Kantengewichte definiert ist.

Hamilton-Pfade im Overlap-Graph



Skizze: Layout und Hamilton-Pfade im Overlap-Graphen

Hamilton-Pfade im Overlap-Graph

- Andererseits können wir aus einen Hamilton-Pfad immer ein Layout konstruieren, indem wir aus den Überlappungen der Fragmente ein Layout bilden.
- Bei der Berücksichtigung von Fehlern, sind an manchen Positionen des Layouts die entsprechenden Nukleotide nicht definiert (bzw. als X).
- Dies ist jedoch eine Aufgabe des dritten Schrittes, der Konstruktion des Konsens-Strings.
- Auch hier gilt, dass wir ein optimales (vermutlich kürzestes) Layout finden, wenn wir einen Hamilton-Pfad maximalen Gewichtes bestimmen.
- Leider ist das Problem der Bestimmung eines gewichtsmaximalen Hamilton-Pfades $\mathcal{NP}$-vollständig.

Layout mit Euler-Pfaden

- Alternatives Konzept zum Layout:
  Sequenzierung durch Hybridisierung
- Hierbei werden mithilfe von DNA-Microarrays alle Teilfolgen einer festen Länge ermittelt, die in der zu sequenzierenden Sequenz auftreten.

Layout mit Euler-Pfaden

T	G	A	C	G	A	C	A	G	A	C	T
T	G	A	C								
	G	A	C	G							
		A	C	G	A						
			C	G	A	C					
				G	A	C	A				
					A	C	A	G			
						C	A	G	A		
							A	G	A	C	
								G	A	C	T

Beispiel: Teilsequenzen der Länge 4, die bei SBH ermittelt werden

- In unserem Beispiel erhalten wir also die folgende Menge an (sehr kurzen, wie für SBH charakteristisch) so genannten Oligos:

$$\{\mathrm{ACGA}, \mathrm{ACAG}, \mathrm{AGAC}, \mathrm{CAGA}, \mathrm{CGAC}, \mathrm{GACA}, \mathrm{GACG}, \mathrm{GACT}, \mathrm{TGAC}\}.$$

Layout mit Euler-Pfaden

- Auch hier müssen wir wieder ein Layout für diese Menge konstruieren.
- Allerdings würden wir mehrfach vorkommende Teilsequenzen nicht feststellen.
- Diese Information erhalten wir über unser Experiment erst einmal nicht, so dass wir eine etwas andere Modellierung finden müssen.
- Außerdem versuchen wir die Zusatzinformation auszunutzen, dass (bei Nichtberücksichtigung von Fehlern) an jeder Position des DNS-Strangs ein Oligo der betrachteten Länge bekannt ist.
- Die Modellierung mit Hamilton-Pfaden ergab ein schwieriges Problem.
- Ähnliches aber algorithmisch einfacheres Problem: Auffinden eines Euler-Pfads/-Kreises

Euler-Pfade

Definition

Sei $G = (V, E)$ ein gerichteter Graph. Ein Pfad bzw. ein Kreis $(v_1, \ldots, v_\ell)$ heißt Euler-Pfad/Kreis, wenn
$\{(v_{i-1}, v_i) : i \in [2 : \ell]\} = E$ und $|\{(v_{i-1}, v_i) : i \in [2 : \ell]\}| = |E|$
bzw.
$\{(v_{i-1}, v_i), (v_\ell, v_1) : i \in [2 : \ell]\} = E$ und
$|\{(v_{i-1}, v_i), (v_\ell, v_1) : i \in [2 : \ell]\}| = |E|$.

Wir nennen einen Graph eulersch, wenn er einen Euler-Pfad besitzt.

Wir weisen hier darauf hin, dass wir aus gegebenem Anlass den Begriff eulerscher Graph anders definieren als in der Literatur üblich. Dort wird ein Graph als eulersch definiert, wenn er einen eulerschen Kreis besitzt.

Euler-Kreis und Knotengrade

Lemma

Sei $G = (V, E)$ ein gerichteter Graph.
Der Graph G ist genau dann eulersch, wenn es zwei Knoten $u, w \in V$ gibt, so dass folgendes gilt:

- $d^-(v) = d^+(v)$ *für alle* $v \in V \setminus \{u, w\}$,
- $d^-(u) + 1 = d^+(u)$ *und*
- $d^-(w) = d^+(w) + 1$.

Ein Euler-Pfad in G kann in Zeit $\mathcal{O}(|V| + |E|)$ ermittelt werden, sofern ein solcher existiert.

Oligo-Graph

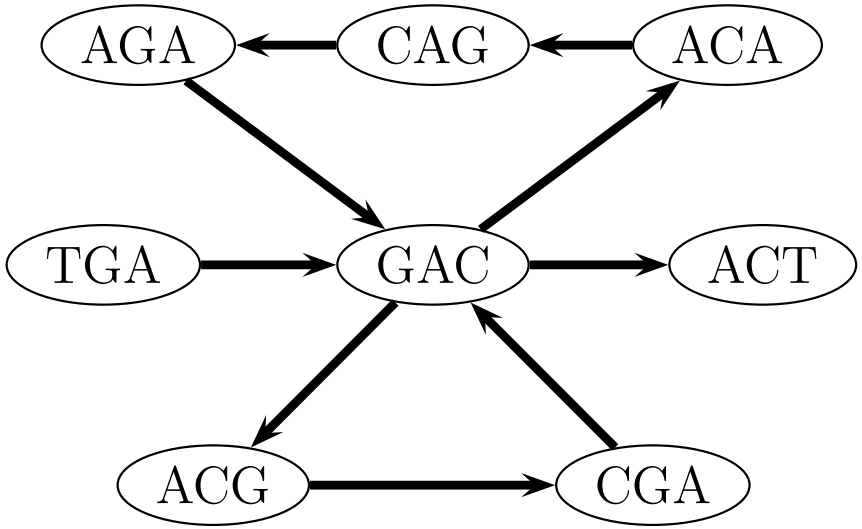
Definition

Sei $S = \{s_1, \ldots, s_k\}$ eine Menge von ℓ-Oligos über Σ, d.h. $|s_i| = \ell$ für alle $i \in [1 : k]$.
Der gerichtete Graph $G_S = (V, E)$ heißt Oligo-Graph, wobei

- $V = \{s_1 \cdots s_{\ell-1}, s_2 \cdots s_\ell \,:\, s \in S\} \subseteq \Sigma^{\ell-1}$,
- $E = \{(v, w) \,:\, \exists s \in S :\, v = s_1 \cdots s_{\ell-2} \wedge w = s_2 \cdots s_{\ell-1}\}$.

- Als Knotenmenge nehmen wir alle $(\ell-1)$-Tupel aus $\Sigma^{\ell-1}$ her.
- Damit die Knotenmenge im Zweifelsfall nicht zu groß wird, beschränken wir uns auf alle solchen $(\ell-1)$-Tupel, die ein Präfix oder Suffix eines Oligos sind.
- Kanten zwischen zwei solcher $(\ell-1)$-Tupel führen wir von einem Präfix zu einem Suffix desselben Oligos.

Oligo-Graph



Beispiel: Oligo-Graph für die Oligos
$\{\mathrm{ACGA}, \mathrm{ACAG}, \mathrm{AGAC}, \mathrm{CAGA}, \mathrm{CGAC}, \mathrm{GACA}, \mathrm{GACG}, \mathrm{GACT}, \mathrm{TGAC}\}$
(aus der Sequenz $\mathrm{TGACGACAGACT}$)

- Es kann mehrere eulersche Pfade im Oligo-Graphen geben.
- Einer davon entspricht der ursprünglichen Sequenz.

Layout mit Euler-Pfaden

- Probleme hierbei stellen natürlich Sequenzierfehler dar, die den gesuchten eulerschen Pfad zerstören können.
- Ebenso können lange Repeats (größer gleich ℓ) zu Problemen führen.
- Wäre im obigen Beispiel das letzte Zeichen der Sequenz ein A, so gäbe es ein Repeat der Länge 4, nämlich GACA.
- Im Oligo-Graphen würde das dazu führen, dass die Knoten ACT und ACA verschmelzen würden.
- Der Graph hätte dann ebenfalls keinen Euler-Pfad mehr (außer wir würden Mehrfachkanten erlauben, hier eine Doppelkante zwischen GAC nach ACA).

Layout mit Euler-Pfaden

- Idee: Wir kennen ja Sequenzen der Länge 500.
- Diese teilen wir in überlappende Oligos der Länge ℓ (in der Praxis wählt man $\ell \approx 20$) wie folgt ein.
- Sei $s = s_1 \cdots s_n$ ein Fragment, dann erhalten wir daraus $n - \ell + 1$ ℓ-Oligos durch $s^{(i,\ell)} = s_i \cdots s_{i+\ell-1}$ für $i \in [1 : n - \ell + 1]$.
- Diese Idee geht auf Idury und Waterman zurück und funktioniert, wenn es keine Sequenzierfehler und nur kurze Repeats gibt.
- Voraussetzung: die zu sequenzierende Sequenz ist gut überdeckt, dass heißt jedes Nukleotid wird durch mindestens ℓ verschiedene Oligos überdeckt.

Layout mit Euler-Pfaden

- Vorteil: man kann versuchen, die Fehler zu reduzieren.
- Ein Sequenzierfehler erzeugt genau ℓ fehlerhafte Oligos (außer der Fehler taucht am Rand des Fragments auf, dann entsprechend weniger).
- Hierbei nutzt man aus, dass eine Position ja von vielen Fragmenten und somit auch Oligos an derselben Position überdeckt wird (in der Praxis etwa 10) und dass pro Oligo aufgrund deren Kürze (in der Praxis etwa 20) nur wenige Sequenzierfehler (möglichst einer) vorliegen.

Layout mit Euler-Pfaden

Definition

Ein Oligo heißt solide, wenn es in einer bestimmten Mindestanzahl der vorliegenden Fragmente vorkommt (beispielsweise mindestens in der Hälfte der Fragmente, die eine Position überdecken sollen).

Zwei Oligos heißen benachbart, wenn sie durch eine Substitution ineinander überführt werden können.

Ein Oligo heißt Waise, wenn es nicht solide ist, und es zu genau einem anderen soliden Oligo benachbart ist.

Layout mit Euler-Pfaden

- Beim Korrekturvorgang suchen wir nach Waisen und ersetzen diese in den Fragmenten durch ihren soliden Nachbarn.
- Mithilfe dieser Prozedur kann die Anzahl der Fehler deutlich reduziert werden.
- Anmerkung: Fehler sind hier nicht bezüglich der korrekten Sequenz gemeint, sondern es werden Fehler reduziert, die im zugehörigen Oligo-Graphen eulersche Pfade eliminieren.

Layout mit Euler-Pfaden

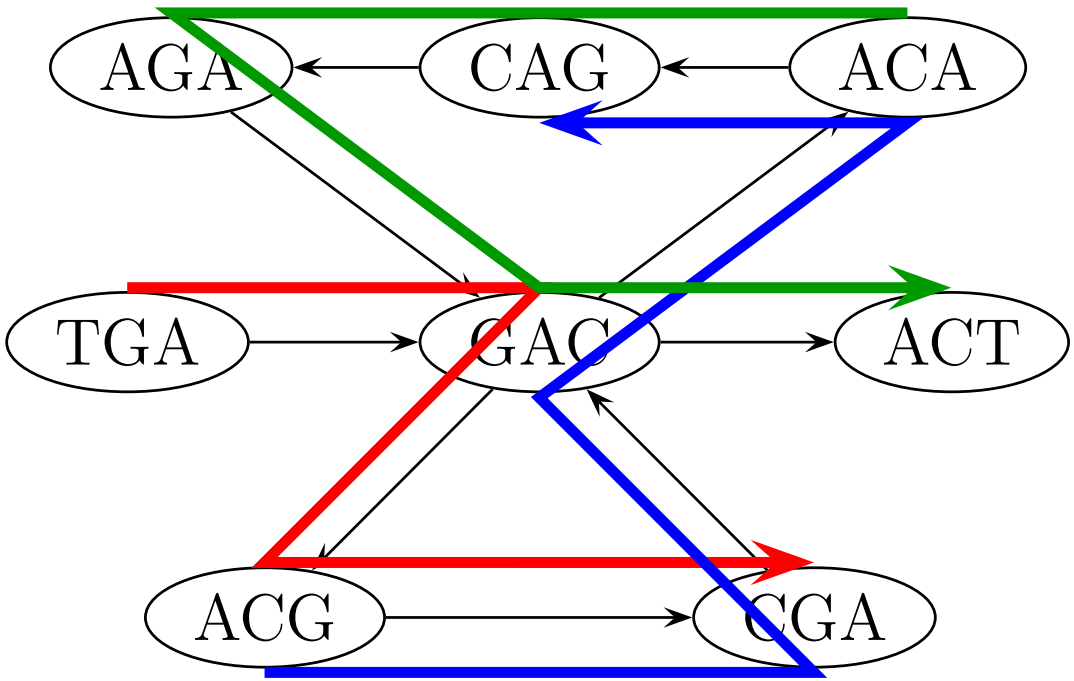
- Wie wir schon vorher kurz angemerkt haben, können Repeats ebenfalls eulersche Pfade eliminieren.
- Um dies möglichst gering zu halten, erlauben wir in unserem Graphen mehrfache Kanten.

- Außerdem haben wir in unserem Oligo-Graphen ja noch eine wichtige Zusatzinformation.
- Die Oligos sind ja nicht durch Hybridisierungsexperimente entstanden, sondern wir haben sie aus den Sequenzinformationen der Fragmente abgelesen.

Layout mit Euler-Pfaden

- Ein Fragment der Länge n induziert daher nicht nur $n - \ell + 1$ Oligos, sondern wir kennen ja auch die Reihenfolge dieser Oligos.
- Das heißt nichts anderes, als dass jedes Fragment einen Pfad der Länge $n - \ell$ auf den $n - \ell + 1$ Oligos induziert (siehe auch Abbildung für die Fragmente TGACGA, ACGACAG, ACAGACT).
- Somit suchen wir jetzt nach einem Euler-Pfad im Oligo-Graphen, der diese Pfade nach Möglichkeit respektiert.
- Dies macht die Aufgabe in der Hinsicht leichter, dass bei mehreren möglichen eulerschen Pfaden leichter ersichtlich ist, welche Variante zu wählen ist.
- Bei langen Repeats bleibt dieses prinzipielle Problem jedoch weiterhin bestehen.

Layout mit Euler-Pfaden



Beispiel: Aus Fragmenten generierter Oligo-Graph

Layout mit Spannbäumen

- Greedy-Algorithmus für Fragment Layout:
  Overlaps in der Reihenfolge nach ihrem Score einarbeiten
  (beginnend mit dem größten)
- Sind beide Sequenzen noch nicht im Layout enthalten, so werden sie mit dem aktuell betrachteten Overlap in dieses neu aufgenommen.
- Ist eine der beiden Sequenzen bereits im Layout enthalten, so wird die andere Sequenz mit dem aktuell betrachteten Overlap in das Layout aufgenommen.

Layout mit Spannbäumen

- Hierbei muss beachtet werden, wie sich die neue Sequenz in das bereits konstruierte aufnehmen lässt. Kommt es hier zu großen Widersprüchen, so wird die Sequenz mit dem betrachteten Overlap nicht aufgenommen.
- Sind bereits beide Sequenzen im Overlap enthalten und befinden sich in verschiedenen Zusammenhangskomponenten des Layouts, so wird versucht die beiden Komponenten mit dem aktuell betrachteten Overlap zusammenzufügen.
- Auch hier wird der betrachtete Overlap verworfen, wenn sich mit diesem Overlap die beiden bereits konstruierten Layouts nur mit großen Problemen zusammenfügen lassen.

Layout mit Spannbäumen

- Bei dieser Vorgehensweise gibt es wiederum insbesondere Probleme bei den Repeats (Teilsequenzen die mehrfach auftreten).
- Bei Prokaryonten ist dies eher selten, bei Eukaryonten treten jedoch sehr viele Repeats auf.
- In der Regel werden dann solche Repeats, die ja mehrfach in der Sequenz auftauchen, auf eine Stelle im Konsensus abgebildet.
- Ist die vorhergesagte Sequenz deutlich zu kurz, so deutet dies auf eine fehlerhafte Einordnung von Repeats hin.
- Versucht man nun beim Aufbau nicht darauf zu achten, ob ein Pfad entsteht, sondern nur, dass Kreise vermieden werden, erhält man einen so genannten Spannbaum oder Spannwald des Overlap-Graphen.
- Hierzu benötigen wir zuerst noch einige grundlegende Definitionen aus der Graphentheorie.

Spannbäume und Spannwälder

Sei $G = (V, E)$ ein (in unserem Fall gerichteter) Graph.

Definition

Ein Teilgraph $G' \subset G$ heißt aufspannender Teilgraph von G, wenn $V(G') = V(G)$.

Definition

Ein Teilgraph $G' \subset G$ heißt Spannwald, wenn $V(G') = V(G)$ ist und G' ein Wald (also kreisfrei) ist.

Definition

Ein Teilgraph $G' \subset G$ heißt Spannbaum, wenn G' ein aufspannender Teilgraph von G ist und G' ein Baum (also kreisfrei und zusammenhängend) ist.

Minimale und maximale Spannbäume

Definition

Sei $G = (V, E, \gamma)$ ein gewichteter gerichteter Graph und T ein Spannbaum von G.
Das *Gewicht des Spannbaumes* T von G ist definiert durch

$$\gamma(T) := \sum_{e \in E(T)} \gamma(e).$$

Ein Spannbaum T für G heißt minimaler bzw. maximaler Spannbaum, wenn er unter allen möglichen Spannbäumen für G minimales bzw. maximales Gewicht besitzt, d.h.

$$\gamma(T) = \min\left\{\gamma(T') \;:\; T' \text{ ist ein Spannbaum von } G\right\}$$

bzw.

$$\gamma(T) = \max\left\{\gamma(T') \;:\; T' \text{ ist ein Spannbaum von } G\right\}$$

Minimale und maximale Spannbäume

Fakt

Sei $G = (V, E, \gamma)$ ein gewichteter Graph und sei $\gamma(e) < B$ für alle $e \in E$.

Sei $G' = (V, E, \gamma')$ ein gewichteter Graph mit $\gamma'(e) := B - \gamma(e)$ für alle $e \in E$.

Dann ist ein Spannbaum T in G genau dann ein minimaler Spannbaum, wenn T ein maximaler Spannbaum in G' ist.