1 Berechnung kantendisjunkter Spannbäume

Gegeben sei ein ungerichteter Graph $G = (V, E)$. Wir wollen die Kantenmenge E in zwei kantendisjunkte Spannbäume zerlegen. O.B.d.A. besteht die Kantenmenge E aus genügend vielen Kanten (d.h. es gibt auch zwei disjunkten Spannbäume). Das hier vorgestellte Verfahren funktioniert aber auch für eine beliebige Kantenmenge und darüberhinaus für beliebig viele disjunkte Spannbäume.

Wir nennen eine Teilmenge F der Kantenmenge E *unabhängig*, falls F in 2 Wälder partitioniert werden kann. Anderenfalls nennen wir F *abhängig*. Damit definieren wir

$$\mathcal{I} := \{F \mid F \text{ ist unabhängige Teilmenge von } E\} \ .$$

Dann ist $M = (E, \mathcal{I})$ ein Matroid und wir können den Greedy-Algorithmus anwenden, um 2 disjunkte Spannbäume zu finden.

Algorithmus 1 : Berechnung einer maximalen unabhängigen Menge in $G = (V, E)$

```
begin
    F := ∅
    foreach e ∈ E do
        if F ∪ {e} ist unabhängig then
            F := F ∪ {e}
    return F
end
```

Besteht der Graph $G = (V, E)$ aus mindestens zwei disjunkten Spannbäumen, dann findet der Algorithmus 1 eine Kantenmenge F, die sich in zwei disjunkte Spannbäume zerlegen lässt. Es sind nun zwei Probleme zu lösen. Die erste Schwierigkeit ist es, die Unabhängigkeit von $F \cup \{e\}$ zu testen. Weiterhin sind wir natürlich an einer Partition von F in zwei Spannbäume interessiert. Letzteres Problem lösen wir, indem wir uns immer eine Partition von F in 2 kantendisjunkte Wälder (F_1 und F_2) merken und diese in jedem Schritt aktualisieren. Den Test auf Unabhängigkeit lösen wir, indem wir versuchen, eine sogenannte *augmentierende Sequenz* zu berechnen.

Zunächst einige Definitionen:

- Sei F ein beliebiger Wald und $e = \{v, w\}$ eine Kante, so dass sich v und w im selben Baum T_F von F befinden. Wir bezeichnen mit $\mathcal{C}(F, e)$ den eindeutigen Pfad von v nach w in T_F.
- Es seien F_1 und F_2 zwei kantendisjunkte Wälder und $e_0, e_\ell \in E$. Eine *Tauschsequenz von* e_0 *nach* e_ℓ ist eine Kantenfolge $e_0, e_1, \ldots, e_\ell$, so dass $e_{i+1} \in \mathcal{C}(F_{(i \bmod 2)+1}, e_i)$ für alle $i \in [0, \ldots, \ell - 1]$ gilt.
- Eine Tauschsequenz heißt *augmentierend*, falls e_0 in keinem Wald F_1, F_2 vorkommt, die Endknoten von e_ℓ in zwei verschiedenen Bäumen von $F_{(\ell \bmod 2)+1}$ liegen und die Tauschfolge minimale Länge hat, d.h. es gibt keine zwei Kanten e_i und e_j, so dass $j > i + 1$ und $e_j \in \mathcal{C}(F_{(i \bmod 2)+1}, e_i)$ gilt.

Geben sei eine beliebige Partition von F in zwei kantendisjunkte Wälder F_1 und F_2. Weiterhin sei eine augmentierende Sequenz von e_0 nach e_ℓ gegeben. Wir fügen nun e_0 wie folgt zu F (bzw. F_1 und F_2) hinzu: für $i \in [0, \dots, \ell - 1]$ ersetzen wir $F_{(i \bmod 2)+1}$ durch $(F_{(i \bmod 2)+1} \setminus \{e_{i+1}\}) \cup \{e_i\}$. Außerdem ersetzen wir $F_{(\ell \bmod 2)+1}$ durch $F_{(\ell \bmod 2)+1} \cup \{e_\ell\}$.

Satz 1 ([1]) *Nach einer Augmentierung sind F_1 und F_2 zwei kantendisjunkte Wälder.*

Das Ziel ist also, eine augmentierende Sequenz (falls vorhanden) bezüglich einer Kante e_0 zu berechnen. Dies erfolgt ähnlich einer Breitensuche und ist im Algorithmus 2 dargestellt.

Algorithmus 2 : Labeling-Algorithmus($F = F_1 \dot\cup F_2, e_0$)

begin
  label[e]=null für alle $e \in E$
  initialisiere leere Queue Q
  Q.insert(e_0)
  while *!Q.isEmpty* **do**
    $e = (v, w) := Q$.first
    i :=Index des Baumes zu dem e gehört ($= 0$ falls $e = e_0$)
    if *v und w liegen im selben Baum bzgl. $F_{(i \bmod 2)+1}$* **then**
      Label jede ungelabelte Kante e' von $\mathcal{C}(F_{(i \bmod 2)+1}, e)$ mit „e“ und füge e' in Q ein
    else
      STOP, augmentierende Tauschsequenz gefunden
end

Falls der Algorithmus eine augmentierende Tauschsequenz entdeckt, können wir diese an Hand der Labels zurückverfolgen und F wie oben beschrieben augmentieren. Falls der Algorithmus mit einer leeren Queue stoppt, dann ist $F \cup \{e_0\}$ abhängig.

Satz 2 ([1]) *Terminiert der Labeling-Algorithmus mit einer leeren Queue, dann ist $F \cup \{e_0\}$ abhängig.*

Wir können die Suche nach einer augmentierenden Tauschsequenz etwas beschleunigen. Angenommen, der Labeling-Algorithmus für eine Kante e_0 terminiert mit einer leeren Queue. Sei S die Menge aller Knoten, die mit einer gelabelten Kante inzident sind. Es gilt nun (siehe Beweis von Satz 2 in [1]), dass es einen Baum T_1 in F_1 *und* einen Baum T_2 in F_2 gibt, der S aufspannt. Wir können also diese Menge „kontrahieren“, da jede weitere Kante e_0, die zwei Knoten aus S verbindet, nicht zu F hinzugefügt werden kann ($F \cup e_0$ ist dann immer abhängig). Eine Menge S, die in beiden Wäldern F_1 und F_2 aufgespannt ist, nennen wir *Klumpen*. Wir können damit unseren Algorithmus verbessern, indem wir zunächst testen, ob beide Knoten der Kante zu einem „Klumpen“ gehören. Wir initialisieren den Algorithmus zunächst mit n Klumpen. Sobald wir feststellen, dass zwei Knoten zum v und w in einem gemeinsamen Klumpen liegen, vereinigen wir die beiden entsprechenden Klumpen.

Damit finden wir bei jeder Ausführung des Labeling-Algorithmus entweder eine augmentierende Tauschsequenz oder wir können zwei Klumpen zu einem vereinigen (wodurch sich die Anzahl der Klumpen um eins verringert). Der Labeling Algorithmus wird somit höchstens $2(n-1) + n - 1 = O(n)$ mal ausgeführt.

Details zur Implementierung

Für die Implementierung der Wälder und der Klumpen verwenden wir Union-Find-Datenstrukturen (`node_partition`). Bevor wir den Labeling-Algorithmus für eine Kante $e_0 = \{v, w\}$ ausführen, überprüfen wir, ob v und w im selben Klumpen liegen. Falls ja, können wir die Kante e_0 sofort verwerfen. Anderenfalls führen wir den Labeling-Algorithmus aus. Findet dieser keine augmentierende Tauschsequenz für e_0, dann vereinigen wir die Klumpen von v und w. Bezüglich der Union-Find-Datenstruktur für die Klumpen führen wir also höchstens $2m$ find-Operationen und $n-1$ union-Operation aus. Ebenso legen wir für beide Wälder F_1 und F_2 eine Union-Find-Datenstruktur an.

Für jede Kante e speichern wir den Index des Waldes zu dem e gehört (falls e bisher zu keinem Wald gehört, setzen wir $\texttt{index}[e] := 0$). Außerdem besitzt jede Kante ein Label, das vom Labeling-Algorithmus verwendet wird.

Der Labeling-Algorithmus initialisiert zunächst alle Labels der Kanten ($\texttt{label}[e] := \mathit{null}$) und fügt die Kante $e_0 = \{v, w\}$ in die Queue ein. Für jeden Wald berechnen wir den zugehörigen Baum, der v enthält und Wurzeln diesen an v. Dazu legen wir ein Array $\texttt{p1}[v]$ und ein Array $\texttt{p2}[v]$ an. Wir setzen $\texttt{p}[v] := v$ und $\texttt{p}[u] := \mathit{null}$, falls u nicht im entsprechenden Baum ist. Dadurch können wir für jeden Knoten u einen Pfad zu v berechnen (falls vorhanden).

Labeling-Schritt. Falls die Queue leer ist, terminiert der Labeling-Algorithmus. Dann ist $F \cup \{e_0 = \{v, w\}\}$ abhängig. Anderenfalls entfernen wir die erste Kante $e = \{v_1, v_2\}$ aus der Queue und betrachten den Wald F_i mit $i = (\texttt{index}[e] \bmod 2) + 1$.

Initialisierung. Falls v_1 und v_2 im entsprechenden Wald in verschiedenen Bäumen liegen, haben wir eine augmentierende Tauschsequenz gefunden. Anderenfalls setzen wir $u := v_2$, falls $v_1 = v$ oder $\texttt{label}[\{v_1, \texttt{p}i[v_1]\}] \neq \mathit{null}$ gilt. Sonst sei $u := v_1$. Außerdem initialieren wir einen leeren (Kanten-)Stack.

Finden der ungelabelten Kanten. Wir legen nun die Kante $\{u, \texttt{p}i[u]\}$ auf den Stack und setzen $u := \texttt{p}i[u]$ bis $u = v$ oder $\texttt{label}[\{u, \texttt{p}i[u]\}] \neq \mathit{null}$ gilt.

Labeln der Kanten. Solange der Stack nicht leer ist, nehmen wir eine Kante e' vom Stack, setzen $\texttt{label}[e'] := e$ und fügen e' zur Queue hinzu.

Findet der Labeling-Algorithmus eine augmentierende Tauschsequenz, dann augmentieren wir wie oben beschrieben. Die Laufzeit des Labeling-Algorithmus ist $O(n)$. Insgesamt wird dieser $O(n)$ mal ausgeführt, d.h. die Gesamtlaufzeit ist $O(n^2)$.

Literatur

[1] J. Roskind and R. E. Tarjan. A Note on Finding Minimum-Cost Edge-Disjoint Spanning Trees. *Mathematics of Operations Research*, Volume 10, Number 4, 1985. http://www.jstor.org/stable/pdfplus/3689437.pdf (Download über TUM)