Praktikumsverlauf

Hypergraph Matching

Supervisor: Prof. Dr. Christian Schulz
Co-Supervisor: Prof. Dr. Felix Joos

Mika Rother

February 8, 2021

Woche 01 (7. Dezember 2020 - 13. Dezember 2020)

In der ersten Woche ging es hauptsächlich darum erstmal in das ganze System von Hypergraphen einzutauchen und sich mit verschiedenen Frameworks auseinanderzusetzen. Dabei wurden sich vor allem KaHyPar[1] und KaMIS[2] angeschaut.

Ziel der Woche war es Hypergraphen aus einem Datenset[3] mit KaHyPar einzulesen und in einer Datei zu speichern, die von KaMIS eingelesen werden kann und anschließend verschiedene Funktionen von KaMIS zu testen. Zudem sollte der Code von Bora Ucar et al.[4] getestet werden, der mittels *Karp-Sipser* Algorithmen, Matchings für Hypergraphen berechnet. Als letztes sollte dann noch ein GitHub Repository angelegt werden, auf dem sich sowohl die beiden Frameworks KaHyPar und KaMIS, sowie zukünftiger Code befinden soll.

Nachdem der Code und die Datensets heruntergeladen wurden, wurde zunächst versucht die den Code von KaHyPar und KaMIS zum laufen zu bringen. Das Ganze musste erstmal mit Hilfe von *cmake* aufgebaut werden, was anfangs auch zu ein paar Problemen führte, aber letztendlich funktionierte. Dann wurde das Repository **HyperMatch**[5] erstellt. Nach einigen Schwierigkeiten gelang es erfolgreich allen Code auf HyperMatch abzulegen und so schon mal eines der Ziele abzuschließen. Anschließend wurde dann lange mit dem Code von KaHyPar herum getestet, um verschiedene Funktionen auszuprobieren und ein Gefühl dafür zu bekommen, was mit diesem Framework alles möglich ist.

Nun ging es darum einen Hypergraphen durch KaHyPar in ein geeignetes Format für KaMIS zu bringen. Als Test-Hypergraph wurde dafür die Datei `astro-ph.mtx.hgr` aus

[1] `https://github.com/kahypar/kahypar`
[2] `https://github.com/KarlsruheMIS/KaMIS`
[3] `https://bwdatadiss.kit.edu/dataset/190#headingFileList`
[4] `http://perso.ens-lyon.fr/bora.ucar/codes.html`
[5] `https://github.com/suomika/HyperMatch`

[3] verwendet. Diese wurde dann mit Hilfe von KaHyPar zu einer Datei `astro-ph.graph` konvertiert, die sich nun ebenfalls auf HyperMatch befindet. Mit dieser Datei gelang es dann auch die Funktionen von KaMIS zu testen.

Anschließend wurde auch noch der Code aus [4] getestet, indem die ebenfalls in [4] mitgelieferten Beispiele benutzt wurden, um Matchings zu finden. Leider gelang es nicht Hypergraphen aus [3] mit dem Code zum Laufen zu bringen, da keine Funktion gefunden wurde mit der man die Hypergraphen in ein geeignetes Format bringen kann.

Ganz am Ende der Woche wurde dann noch dieses LaTeX-Dokument angelegt, um den Arbeitsverlauf des Praktikums für jede Woche zusammenzufassen.

Woche 02

(14. Dezember 2020 - 20. Dezember 2020)

In dieser Woche ging es darum, ein Programm zu schreiben, dass in der Lage ist einen Hypergraphen in einen Graphen zu konvertieren, wobei die *Hyperedges* des Hypergraphen im neuen Graphen durch Knoten repräsentiert werden sollen. Die Kanten des resultierenden Graphen sollten aus den überlappenden *Hyperedges* gewonnen werden.

In der letzten Woche wurde bereits ein Hypergraph konvertiert, allerdings zu einem bipartiten Graphen und nicht zu einem Graphen mit dem man ein Matching im Hypergraphen finden kann.

Da es in KaHyPar bereits einige Programme gibt, um Hypergraphen in ein anderes Format zu bringen, wurde sich zunächst daran orientiert, um ein Gefühl zu bekommen, wie man Dateien einlesen kann und eine geeignete Datenstruktur daraus aufbauen kann. Dann wurde sich überlegt (anhand eines Beispiel-Hypergraphen `example01.hgr`, zu finden in [5]), wie man aus dem eingelesenen Hypergraphen einen Graphen erstellt, mit dem KaMIS arbeiten kann.

Nachdem das Programm fertig gestellt wurde und die *cmake* Dateien geupdated wurden, wurde anhand des Beispiels geschaut, ob das Programm das tut was es soll. Anschließend wurde erneut die Datei `astro-ph.mtx.hgr` verwendet, um ein paar Tests mit KaMIS zu machen. (Die erzeugten Dateien befinden sich alle im `examples` Ordner in [5]).

Woche 03

(21. Dezember 2020 - 10. Januar 2021)

Nachdem in *Woche 02* ein Programm geschrieben wurde, dass einen Hypergraphen in einen Graphen konvertiert, sollte dieses Programm bezüglich des Feedbacks optimiert werden, so dass es auch für Hypergraphen mit gewichteten *Hyperedges* funktioniert. Dieses Vorhaben gelang und wurde anhand eines Beispiels (`example02.hgr`, zu finden in [5]) erfolgreich getestet.

Anschließend sah der Plan vor, nun verschieden Hypergraphen zu nehmen und dabei die Matchings zu vergleichen, wobei einmal das `HgrToGraph` Programm genutzt werden

sollte, (der sich daraus resultierende Graph sollte dann mit KaMIS optimal gelöst werden), und einmal der Code von Uçar et al. in [4]. Hierbei sollten die Größe der Matchings, sowie die Laufzeit verglichen werden.

Da der Uçar Code allerdings nur für d-partite, d-uniforme Hypergraphen funktioniert, mussten erstmal passende Hypergraphen gefunden werden. (Bei der Suche danach wurde auch einen Sammlung von Tensoren[6] gefunden, die eventuell später nützlich sein könnten). Dank dem Code zum generieren von d-partiten, d-uniformen Hypergraphen, konnten schließlich passende Hypergraphen kreiert werden.

Jetzt brauchte man ein Programm, dass das Format von Uçar in eine `.hgr` Datei konvertiert. Das Programm `UcarToHgr3dim` ist in der Lage einen 3-partiten, 3-uniformen Hypergraphen in das gewünschte `.hgr` Format zu konvertieren. Von da aus kann man dann Vergleiche zwischen dem `HgrToGraph` Programm und dem Uçar Code anstellen.

Als letztes sollte dann noch ein Vergleich getätigt werden. Dabei wurde eine 3-partiter, 3-uniformer Hypergraph verwendet mit 300 Knoten pro Partition und ca. 7 Millionen *Hyperedges*. Da allerdings das `HgrToGraph` Programm zu lange gebraucht hat, wurde dieser Versuch abgebrochen.

Woche 04 (11. Januar 2021 - 17. Januar 2021)

In dieser Woche ging es darum nochmal Hypergraphen im Uçar Format in das Graph Format zu konvertieren, um vergleiche bezüglich des Matchings anzustellen, eventuell auch mit kleineren Instanzen. Außerdem sollte der Matlab-Code zum generieren von Hypergraphen (auch zu finden in [4]) angeschaut werden.

Mit Hilfe einer der Generator Dateien, wurde eine Datei erstellt, die auch 3-partite, 3-uniforme Hypergraphen erzeugt. Diese wurde benutzt, um z.B. die Datei `1000_1.txt` zu erstellen (welche sich auch in [5] befindet). Diese Datei, welche sich im Uçar Format befindet, konnte dieses Mal erfolgreich sowohl ins `.hgr` als auch ins `.graph` Format konvertiert werden. Der in dieser Datei erzeugte Hypergraph besitzt 24000 Hyperedges und 1000 Knoten pro Partition. Der daraus resultierende Graph enthält dementsprechend 24000 Knoten und um die 850000 Kanten.

Die Karp-Sipser Programme haben in diesem Beispiel ein Matching von ca. 985 gefunden (es wurden mehrere Durchläufe gemacht mit verschiedenen *scaling* Iterationen). Zum Vergleich dazu haben die KaMIS Programme Matchings der Größe 993 gefunden. (Die Resultate befinden sich ebenfalls in [5]).

Woche 05 (18. Januar 2021 - 24. Januar 2021)

Das Ziel dieser Woche war einige Tensoren aus [6] zu nehmen und die Codes von Uçar gegen KaMIS antreten zu lassen, im Bezug auf die Größe des Matchings. Außerdem

[6] `http://frostt.io`

sollten weitere Dateien mit den Generator Programmen erzeugt werden, mit variierenden Größen und Dimensionen, welche dann auch verglichen werden sollten.

Um die Tensoren zu vergleichen, mussten erst einige Programme geschrieben werden, welche einen Tensor im `.tns` Format, sowohl ins `.hgr` Format als auch ins Uçar Format konvertieren können. (Diese Programme befinden sich ebenfalls auf [5]).

Nachdem die Tensoren nell-2, uber und nips also passend ins `.hgr` Format übertragen wurden, wurde schnell klar, dass es nicht möglich war sie ins `.graph` Format zu übertragen. Die Vermutung, warum das nicht ging, bestand darin, dass es Knoten in diesen Hypergraphen gibt, die in fast allen, bzw. sehr vielen Hyperedges enthalten sind, (was auch im Paper von Uçar et al.[7] erwähnt wurde). Das würde dazu führen, dass man tausende von GBytes an Speicher bräuchte, um sie ins `.graph` Format zu überführen.

Da die Tensoren also nicht geeignet waren, um Vergleiche anzustellen, wurden stattdessen einige weitere synthetische Hypergraphen erzeugt. Diese waren entweder 3-partite, 3-uniforme oder 9-partite, 9-uniforme Hypergraphen, und hatten entweder 1000 Knoten pro Partition oder 10000 Knoten pro Partition. Mit diesen Dateien klappte die Konvertierung ins `.graph` Format, (auch wenn das sehr viel Speicher in Anspruch nahm: Z.B. eine `.hgr` Datei, die 38.1 MB groß war, wurde nach dem Konvertieren zu einer `.graph` Datei 3.19 GB groß). Deshalb wurden nur wenige Dateien erzeugt.

Alle Vergleiche, die dann mit den Dateien gemacht wurden, lieferten dasselbe Ergebnis: Die Karp-Sipser Algorithmen lieferten zwar ein schnelleres Ergebnis, als die KaMIS Programme, jedoch war die Größe des Matchings bei KaMIS stets höher.

Woche 06 (24. Januar 2021 - 07. Februar 2021)

Da das große Problem mit dem Konvertieren der Tensoren vom Hypergraph in das Graph-Format darin besteht, dass einige Knoten (*pins*) in sehr vielen Hyperedges vorhanden sind, war das Ziel dieser Woche diese *pins* zu entfernen bevor das Programm über alle Hyperkanten iteriert.

Nach einigen Versuchen das Problem zu lösen, konnte schließlich eine Lösung gefunden werden, die sich der Datenstrukturen von KaHyPar bedient. Mit Hilfe dieser Lösung lief das Programm erstmals soweit, dass es in eine `.graph`-Datei geschrieben hat. Leider war der Speicher des Mac's auf dem gerechnet wurde nicht mehr groß genug, um eine > 30 GB Datei zu speichern, weshalb diese Datei nicht gespeichert wurde.

Um das Problem mit dem Speicher zu lösen, wurde ein Server der Universität Heidelberg eingerichtet auf dem in Zukunft alle Rechnungen erfolgen sollen. Dementsprechend bestand der nächste Schritt darin, die Materialien (HyperMatch, Hypergraph-Instanzen, etc.) vom Mac auf den Uni Server zu kopieren, was nach einigen Startschwierigkeiten auch gelang.

Anschließend war gedacht dann auf den Uni Servern, die beträchtlich schneller sind, die Graph Instanzen der Tensoren zu erzeugen und auszuwerten, um anschließend Ver-

[7] `https://hal.archives-ouvertes.fr/hal-01924180v3/document`

gleiche mit Uçar et al. vorzunehmen. Dabei trat allerdings das Problem auf, dass die `exec` Dateien nicht auf dem Betriebssystem der Server ausführbar waren und zusätzlich auch `cmake` Befehle nicht funktionierten, aufgrund der unterschiedlichen Dateipfade der Materialien.
