Gottfried Wilhelm Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Theoretische Informatik

Historische Entwicklung der Matching-Algorithmen

Historical Development of Matching-Algorithms

Bachelorarbeit

im Studiengang Informatik

von

Anthimos Kouroutsidis
Matrikelnummer: 10005088

Prüfer: Prof. Dr. Heribert Vollmer
Zweitprüfer: Dr. Arne Meier
Betreuer: Dr. Arne Meier

Hannover, den 12. Juni 2020

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Bachelorarbeit/Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 12. Juni 2020

_____

Vorname Nachname

Inhaltsverzeichnis

1 Einleitung

1.1 Einführung

„Matching is a powerful piece of algorithmic magic“ [Ski08]. In dieser Arbeit soll dem Leser ein Einblick in die Entwicklung der „Matching-Algorithmen“ gegeben werden. Matching-Algorithmen sind dabei Algorithmen, die für einen gegebenen Graphen möglichst viele disjunkte Paare, sogenannte Matches, finden. Dabei hängt die Schwierigkeit und somit die Laufzeit stark von der Graphklasse ab. Unterschiedliche Ansätze und Verbesserungen haben im Laufe der Zeit diese Algorithmen immer effizienter gemacht, beginnend mit der Arbeit von Jack Edmonds [Edm65] im Jahre 1965, der die damals existierende Frage, ob das Matching-Problem in der Komplexitätsklasse P oder NP liegt, mit seinem ersten effizienten Polynomialzeitalgorithmus für allgemeine Graphen beantwortete. Allerdings ist heutzutage ein Polynomialzeitalgorithmus für das Matching-Problem nicht mehr „effizient“ genug; unterschiedliche Ansätze versuchen, das Problem effizienter zu lösen. Linearzeitalgorithmen existieren nur für Approximationsalgorithmen oder sehr spezielle Graphklassen. Um die Komplexität von Problemen besser unterscheiden zu können, gibt es eine feinere Kategorisierung der bekannteren Komplexitätsklassen:

- NP: Algorithmen, die auf einer nichtdeterministischen Turingmaschine mit polynomieller Laufzeit ausgeführt werden können; mit deterministischen Turingmaschinen kann eine Lösung in polynomieller Laufzeit auf ihre Korrektheit überprüft werden, eine korrekte Lösung kann bisher deterministisch nur in exponentieller Laufzeit berechnet werden
- P: Algorithmen mit polynomieller Laufzeit
- NC: Nick's Class; parallel effiziente Entscheidungsalgorithmen
- RNC: Randomized Nick's Class; nichtdeterministisch parallel effiziente Entscheidungsalgorithmen
- quasi-NC: Algorithmen, die von der Laufzeit her NC ähneln
- SPL: Stoic Probabilistic Logspace; enthält Matching-Entscheidungsprobleme, die randomisiert gelöst werden
- L: Algorithmen mit logarithmischem Speicherbedarf

Von der Hierarchie dieser Komplexitätsklassen ist klar, dass $\mathsf{L} \subseteq \mathsf{P} \subseteq \mathsf{NP}$. Für die anderen Komplexitätsklassen existiert keine einheitliche Abstufung, da die Klasse NC

nochmals in mehrere unterschiedlich stark mächtige Klassen unterteilt wird. Allerdings ist die randomisierte Version einer Komplexitätsklasse stets mächtiger als die nicht-randomisierte Version. Somit gilt $\mathsf{NC} \subseteq \mathsf{RNC}$.

Im Hinblick auf die Komplexitätstheorie ist das Matching-Problem von Bedeutung, da es in andere Probleme, darunter das Finden eines minimalen Spannbaumes, überführt werden kann [NR59]. Für andere kombinatorische Probleme wie das Bestimmen der Isomorphie von zwei Graphen oder die Bestimmung der chromatischen Zahl eines Graphen [HMU07] existieren hingegen keine effizienten (Polynomialzeit-) Algorithmen. Im Vergleich dazu gibt es für das Matching-Problem bereits Algorithmen, die das Matching-Problem in quasi-NC berechnen. Für speziellere Graphklassen wie bipartite oder planare Graphen sind noch bessere Algorithmen möglich. Dies hängt damit zusammen, dass beschränktere Graphklassen es erlauben, Annahmen über den Graphen zu erzielen und somit die Berechnung zu erleichtern. Auf der anderen Seite gibt es die Möglichkeit, dass die Kanten der Graphen gerichtet oder gewichtet sind. Ist letzteres der Fall, so wird nicht nur ein möglichst großes Matching gesucht, sondern gleichzeitig auch das mit der geringsten Summe der Kantengewichte (Man kann sich das wie die Suche nach einem minimalen Spannbaum vorstellen). Eine detaillierte Beschreibung der verwendeten Graphklassen befindet sich im Anschluss an die Grundlagen im Abschnitt 2.3.

Die Aufgabe dieser Arbeit besteht darin, die Historie der Matching-Algorithmen aufzuzeigen und zu diskutieren sowie zusätzlich einen Einblick in die neuste Entwicklung zu geben. Die dafür berücksichtigten 14 Arbeiten haben ein Publikationsjahr zwischen 1965 und 2020. Im Hinblick auf die mit den zu den Algorithmen kompatiblen Graphklassen wird ein besonderer Fokus auf die Neuerungen und die daraus resultierenden Verbesserungen der Laufzeiten der vorgestellten Algorithmen gelegt.

1.2 Aufbau

Die Arbeit ist folgenderweise aufgebaut: Zunächst werden Grundlagen der Graphentheorie wiederholt und darauf basierend die wichtigsten Definitionen und Sätze im Bereich der Matching-Probleme erläutert (Abschnitt 2). Anschließend werden im Abschnitt 3 in chronologischer Reihenfolge unterschiedliche Arbeiten vorgestellt, welche den Algorithmus im historischen Verlauf verbessert haben oder auf neuen Ansätzen basieren, um das Problem zu lösen. Schließlich wird im Abschnitt 4 der zeitliche Verlauf der Matching-Algorithmen kompakt dargestellt. Zuletzt findet im Abschnitt 5 ein kurzer Ausblick über die noch offenen Fragen statt.

Die Arbeiten im Abschnitt 3 werden in einer steckbriefartigen Form zusammengefasst. Diese Form beinhaltet folgende Punkte:

- Graphklasse, auf die der Algorithmus angewendet werden kann
- Laufzeit des Algorithmus, ggf. Speicher
- Neuerungen (Verbesserungen, Ansätze)
- Algorithmus: Oberflächliche Beschreibung des Algorithmus

- Bedeutung: Für das Matching-Problem oder die Komplexitätstheorie relevante Informationen

2 Grundlagen

Für das Verständnis der Algorithmen werden hier erforderliche Grundlagen erläutert. Weitere spezielle Definitionen und Sätze werden später bei den Arbeiten eingeführt.

2.1 Allgemeine Graphentheorie

Hier wird zunächst nochmal die für die Definitionen im Teil „Matching“ erforderliche Graphentheorie wiederholt. Diese richten sich nach dem Standardwerk „Graphentheorie“ von Reinhard Diestel [Die06].

Definition 2.1.1 (Graph)**.** *Sei V eine Menge von* Knoten *sowie E eine Menge von* Kanten *zwischen je zwei Knoten aus V. Ein Graph G wird definiert als $G = (V, E)$.*

Viele Algorithmen können für allgemeine Graphen verwendet werden. Allerdings gibt es auch weitere Graphklassen, welche Besonderheiten aufweisen, die nur von bestimmten Algorithmen berücksichtigt werden.

Definition 2.1.2 (gerichteter Graph)**.** *Sei $G = (V, E)$ ein Graph. Falls die Kantenmenge E geordnete Tupel sind, wird G* gerichtet *genannt.*

Definition 2.1.3 (gewichteter Graph)**.** *Sei $G = (V, E)$ ein Graph. Falls das Traversieren von Kanten einen unterschiedlichen Aufwand darstellt, so existiert eine Gewichtungsfunktion $w\colon E \to \mathbb{Q}$, welche jeder Kante $e \in E$ ein Gewicht $w(e)$ zuweist. Dann nennen wir (G, w) einen* gewichteten Graphen.

Bei den beiden vorherigen Definitionen ging es um die Art der Kanten, aus denen diese bestehen. Eine weitere Besonderheit sind die bipartiten Graphen, zu denen es im Vergleich zu normalen Graphen deutlich einfachere Algorithmen mit einer schnelleren Laufzeit gibt.

Definition 2.1.4 (bipartiter Graph)**.** *Sei $G = (V, E)$ ein Graph. Falls V in zwei Mengen $V_1 \uplus V_2$ aufgeteilt werden kann, sodass für alle Kanten aus E gilt, dass diese ausschließlich zwischen Knoten von V_1 und V_2 verlaufen, so wird G* bipartit *genannt.*

Bei den bipartiten Graphen sei noch erwähnt, dass diese ebenfalls gerichtet oder gewichtet sein können.

Definition 2.1.5 (planarer Graph)**.** *Sei $G = (V, E)$ ein Graph. Wir nennen G* planar*, wenn G auf einer Ebene so gezeichnet werden kann, dass sich keine Kanten schneiden. Flächen der Ebene, die durch die Kanten von G abgegrenzt werden, bezeichnen wir als* Facetten*.*

2.2 Matching

Die Algorithmen berechnen alle ein Matching. Mithilfe der folgenden Erläuterungen soll es für den Leser verständlich werden, was die Eigenschaften und Unterschiede dieser sind.

Definition 2.2.1 (Match)**.** *Sei $G = (V, E)$ ein Graph. Sind zwei Knoten $v_1, v_2 \in V$ durch eine Kante verbunden und sonst kein Teil weiterer Kanten, so wird (v_1, v_2) als ein* Match *definiert.*

Definition 2.2.2 (Matching)**.** *Sei $G = (V, E)$ ein Graph. Ein* Matching *$M \subseteq E$ ist eine Teilmenge der Kanten von G, für die gilt, dass alle Kanten $e \in M$ Teil eines Matches sind. Knoten, die inzident zu einer Matchingkante sind, werden* gematcht *genannt, Knoten, die zu keiner Matchingkante inzident sind,* frei*.*

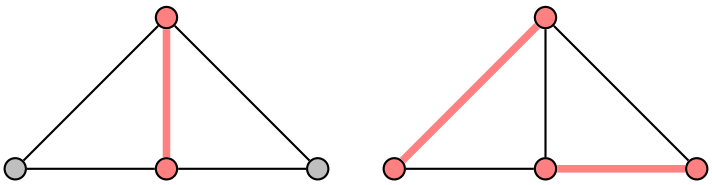


Abbildung 2.1: Zwei Beispiele für ein Matching. Knoten und Kanten, die ein Match bilden, sind rot gefärbt. Ungematchte Knoten sind grau gefärbt.

Ein triviales aber dennoch gültiges Matching ist beispielsweise ein Graph, aus dem eine beliebige Kante gewählt wird. Dementsprechend existieren weitere Kriterien, mit denen Matchings nach ihrer Mächtigkeit kategorisiert werden können.

Definition 2.2.3 (Nicht erweiterbares Matching)**.** *Sei $G = (V, E)$ ein Graph, $M \subseteq E$ ein Matching von G. M ist ein* nicht erweiterbares Matching *von G genau dann, wenn es in $E \setminus M$ keine weiteren Matches gibt.*

Um ein nicht erweiterbares Matching zu erhalten, kann man also einfach so lange eine beliebige Kante zweier ungematchter Knoten zum Matching hinzufügen, bis es keine solche Kante mehr gibt. Sowohl das linke als auch das rechte Matching aus Abbildung 2.1 sind nicht erweiterbare Matchings.

Allerdings besteht die Möglichkeit, dass abhängig von der Auswahl der Kanten für das Matching unterschiedlich viele Matches in einem nicht erweiterbaren Matching enthalten sind (wie in Abbildung 2.1). Dies gilt es zu differenzieren:

Definition 2.2.4 (Matching-Kardinalität)**.** *Sei $G = (V, E)$ ein Graph, $M \subseteq E$ ein Matching von G. Die Kardinalität $|M|$ eines Matchings entspricht der Anzahl an Matches in M.*

Für jeden Graphen gibt es ein Maximum an Matches, die ein Matching haben könnte. Angenommen, ein Graph $G = (V, E)$ hätte eine gerade Anzahl von Knoten, bei denen jeweils zwei verbunden sind, so ergibt sich eine maximal mögliche

Matching-Kardinalität von $\frac{|V|}{2}$ (siehe Abbildung 2.2 links). Sind hingegen alle Knoten $v_1, v_2, ..., v_n \in V$ ausschließlich mit $v_0 \in V$ verbunden, so beträgt die maximal mögliche Matching-Kardinalität 1 (siehe Abbildung 2.2 rechts). Folglich hängt diese von dem Graphen ab und kann nicht trivial bestimmt werden.

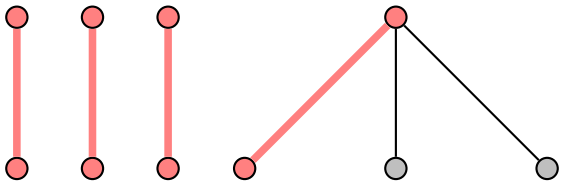


Abbildung 2.2: Die Anzahl der Matches ist abhängig von dem Graphen. Links: Alle Knoten und Kanten sind Teil des maximalen Matchings. Rechts: Nur zwei Knoten und eine Kante sind Teil des maximalen Matchings.

Definition 2.2.5 (Maximum-Matching (Maximales Matching))**.** *Sei $G = (V, E)$ ein Graph, $M \subseteq E$ ein Matching von G. M ist ein* Maximum-Matching *von G genau dann, wenn kein weiteres Matching $M_{alt} \subseteq G$ mit $|M| < |M_{alt}|$ existiert.*

An den Beispielen aus Abbildung 2.2 ist ersichtlich, dass es Graphen gibt, die nur eine Belegung an Matches haben, die ein Maximum-Matching ist (links), oder mehrere (rechts, die Kante kann beliebig gewählt werden). Interessante Probleme, die nicht trivial gelöst werden können, sind zum Beispiel „Was ist ein Maximum-Matching für einen Graphen G?“ oder „Gibt es mehrere Maximum-Matchings für G?“.

Eine noch mächtigere Form des Matchings ist das perfekte Matching.

Definition 2.2.6 (Perfektes Matching)**.** *Sei $G = (V, E)$ ein Graph, $M \subseteq E$ ein Matching von G. M ist ein* perfektes Matching *von G genau dann, wenn für alle Knoten $v \in V$ gilt, dass diese Teil eines Matches sind.*

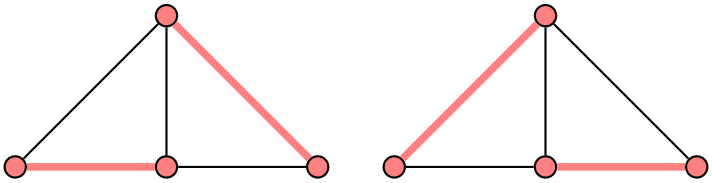


Abbildung 2.3: Der Graph aus Abbildung 2.1 hat genau zwei perfekte Matchings.

Auch hier können die Fragen für ein Maximum-Matching analog für ein perfektes Matching gestellt werden. Zusätzlich existiert noch das nicht-triviale Problem, zu entscheiden, ob es überhaupt ein perfektes Matching für G gibt. Im Falle von Abbildung 2.3 hat der Graph mehrere perfekte Matchings.

Die Arbeiten, welche in dieser Arbeit zusammengefasst werden, beschäftigen sich mit diesen Problemen und versuchen, diese möglichst effizient zu lösen. Dabei werden

unterschiedliche Ansätze verfolgt, welche unter anderem auf den folgenden Definitionen und Sätzen aufbauen:

Definition 2.2.7 (Faktor-kritischer Graph)**.** *Sei $G = (V, E)$ ein Graph. G ist ein* faktor-kritischer Graph *genau dann, wenn für jedes $v \in V$ gilt, dass es für $G' = (V \setminus \{v\}, E)$ ein perfektes Matching gibt.*

Kann in einem Graphen ein Teilgraph gefunden werden, welcher die Eigenschaften eines faktor-kritischen Graphen besitzt, so kann über diesen Teilgraphen gesagt werden, dass es in jedem Fall mehrere Maximum-Matchings gibt.

Definition 2.2.8 (Grad eines Knotens)**.** *Sei $G = (V, E)$ ein Graph. Der* Grad eines Knotens *bezeichnet die Anzahl der Kanten, welche zu diesem inzident sind: $deg \colon V \to \mathbb{N}$*

Jeder Graph G, bei dem alle Knoten den Grad 1 haben, hat ein perfektes Matching. Umgekehrt lässt sich aus der Existenz eines perfekten Matchings folgern, dass es möglich ist, Kanten aus G so zu entfernen, das alle Knoten den Grad 1 haben.

Definition 2.2.9 (Faktorisierbarkeit eines Graphen)**.** *Sei $G = (V, E)$ ein Graph. G heißt* α-faktorisierbar*, wenn ein Teilgraph $G' = (V, E')$ mit $E' \subseteq E$ existiert, sodass jeder Knoten aus V den Grad α besitzt.*

In dem Falle wäre G 1-faktorisierbar.

Wählt man einen Knoten als Startknoten und führt eine Breiten- oder Tiefensuche durch, so traversiert man den Graphen schrittweise, wie es auch in den Algorithmen der Fall ist. Dabei folgen die Algorithmen Wegen.

Definition 2.2.10 (Weg eines Graphen)**.** *Sei (G, w) mit $G = (V, E)$ ein gewichteter Graph. Ein* Weg *bezeichnet eine Sequenz $v_0, v_1, ..., v_n \in V$ von Knoten, für die gilt, dass für jedes Paar an benachbarter Knoten eine Kante $(v_i, v_{i+1}) \in E$ existiert. Die* Länge eines Weges *ist dann n. Das* Gewicht eines Weges *besteht aus der Summe der Gewichte der Kanten, welche dieser enthält: $\sum_{i=0}^{n-1} w(v_i, v_{i+1})$*

Problematisch bei Wegen ist jedoch, dass es möglich ist, dass Knoten mehrfach besucht werden. Dies soll ausgeschlossen werden, weshalb die Algorithmen anstelle von Wegen eine mächtigere Form, Pfade, verwenden.

Definition 2.2.11 (Pfad eines Graphen)**.** *Sei $G = (V, E)$ ein Graph. Ein* Pfad *ist ein Weg, für den zusätzlich gilt, dass keine Knoten mehrfach in diesem vorkommen.*

Analog zum Weg gelten auch hier die Definitionen der Länge und des Gewichtes von Pfaden.

Nun soll das Konzept des Pfades erweitert werden, um Matches in einem Graphen finden zu können. Ein Hilfskonstrukt dafür sind alternierende Pfade.

Definition 2.2.12 (Alternierender Pfad)**.** *Sei $G = (V, E)$ ein Graph und $M \subseteq E$ ein Matching. Ein Pfad P heißt* alternierend*, wenn dieser abwechselnd Kanten aus M und $E \setminus M$ enthält.*

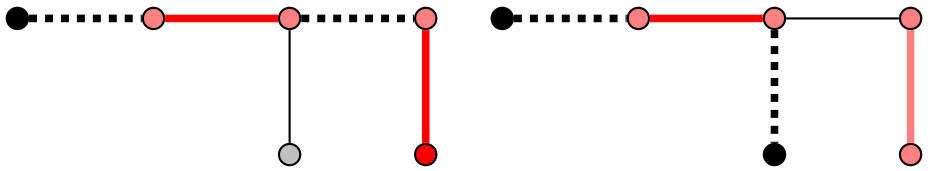


Abbildung 2.4: Links: Beispiel für einen alternierenden Pfad. Rechts: Beispiel für einen augmentierenden Pfad. Die Kanten eines augmentierenden / alternierenden Pfades sind farblich hervorgehoben.

Wenn man einen alternierenden Pfad zu einem augmentierenden Pfad erweitert, so hat man die Möglichkeit, die Kardinalität des Matchings zu erhöhen.

Definition 2.2.13 (Augmentierender Pfad)**.** *Sei $G = (V, E)$ ein Graph, $M \subseteq E$ ein Matching sowie P ein alternierender Pfad. Ein Pfad $P' \supset P$ heißt* augmentierend*, wenn er den alternierenden Pfad P dadurch erweitert, dass dieser zusätzlich noch an einem ungematchten Knoten anfängt sowie endet.*

Einfach formuliert ist ein augmentierender Pfad ein Pfad, der zwei ungematchte Knoten, die durch eine Folge von Matches getrennt sind, miteinander verbindet.

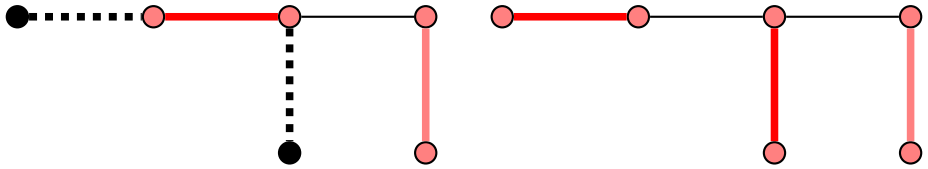


Abbildung 2.5: Die Kanten aus dem augmentierenden Pfad aus (links) werden umgekehrt. Das Ergbnis ist (rechts), ein Matching mit einer um 1 erhöhten Kardinalität.

Hat man einen augmentierenden Pfad gefunden, so kann man die Kardinalität des Matchings um 1 erhöhen, indem man die Kanten der Matchings „umkehrt“. Damit ist gemeint, dass die Kanten des augmentierenden Pfades, die bisher matchende Kanten waren, aufgelöst werden und die bisher ungematchten Kanten in matchende umgewandelt werden.

Satz 2.2.1 (Satz von Berge [Ber57])**.** *Sei $G = (V, E)$ ein Graph und $M \subseteq E$ ein Matching. M ist ein maximales Matching genau dann, wenn es keinen augmentierenden Pfad gibt.*

Mit dem Satz von Berge ist es möglich, ein Maximum-Matching zu finden, indem ein zunächst beliebiges Matching schrittweise „verbessert“ wird. Solange es augmentierende Pfade gibt, wird es aktualisiert. Gibt es keine augmentierenden Pfade mehr, so ist das Matching zu einem Maximum-Matching geworden.

Die Matching-Probleme sind in viele weitere Graphenprobleme überführbar, zum Beispiel das Finden einer kleinstmöglichen Knotenüberdeckung in einem bipartiten Graph.

Definition 2.2.14 (Knotenüberdeckung eines Graphen)**.** *Sei $G = (V, E)$ ein Graph. Eine* Knotenüberdeckung *ist eine Menge von Knoten $V' \subseteq V$, für die gilt, dass alle Kanten $e \in E$ mindestens einen Knoten $v' \in V'$ enthalten. Eine Knotenüberdeckung heißt* kleinstmöglich *genau dann, wenn es für den Graphen keine weitere Knotenüberdeckung mit weniger Kanten gibt.*

Mithilfe des Satzes von König lässt sich die Kardinalität eines Maximum-Matchings in einem bipartiten Graphen bestimmen.

Satz 2.2.2 (Satz von König [Wag70])**.** *Sei $G = (V, E)$ ein bipartiter Graph. Ein maximales Matching enthält genau so viele Kanten wie die Anzahl der Knoten in einer minimalen Knotenüberdeckung.*

Als nächstes geht es um das Problem, zu entscheiden, ob ein beliebiger Graph ein perfektes Matching hat. Dafür wird das Konzept der Zusammenhangskomponente benutzt.

Definition 2.2.15 (Zusammenhangskompenente eines Graphen)**.** *Sei $G = (V, E)$ ein Graph. Eine* Zusammenhangskomponente *wird als eine inklusionsmaximale Teilmenge $V' \subseteq V$ definiert, für die gilt, dass es für jedes $v_i \in V'$ einen Pfad zu $v_0, v_1, ..., v_{|V'|-1}$ gibt.*

Der Satz von Tutte liefert eine Möglichkeit, die Existenz eines perfekten Matchings in einem Graphen zu belegen oder zu widerlegen.

Satz 2.2.3 (Satz von Tutte [Tut47])**.** *Sei $G = (V, E)$ ein Graph. G hat ein perfektes Matching genau dann, wenn für jede Teilmenge $V' \subseteq V$ gilt, dass für den Graphen $G' = (V \setminus V', E')$ die Anzahl der Zusammenhangskomponenten mit einer ungeraden Anzahl von Knoten höchstens $|V'|$ beträgt.*

Der Satz von Berge ist auf beliebige Graphen anwendbar, allerdings lässt sich die Laufzeit zum Finden von augmentierenden Pfaden in allgemeinen Graphen erheblich verbessern.

Definition 2.2.16 (Kreis eines Graphen)**.** *Sei $G = (V, E)$ ein Graph. Ein Pfad mit einer Länge größer als 2, bei dem Start- und Endknoten identisch sind, wird als* Kreis *definiert. Gilt zusätzlich, dass ausschließlich der Start- und der Endknoten identisch sind und sonst keine weiteren Knoten im Kreis mehrfach vorkommen, so wird dieser als* Zykel *bezeichnet.*

Existieren Blüten in dem Graphen, so können diese „geschrumpft“ werden, womit das Finden von augmentierenden Pfaden erleichtert wird.

Definition 2.2.17 (Blossom (Blüte))**.** *Sei $G = (V, E)$ ein Graph und $M \subseteq E$ ein Matching. Eine* Blüte *B wird definiert als ein alternierender Pfad gerader Länge, wobei der letzte Knoten einen Kreis mit ungerader Länge bildet, in dem jede zweite Kante Teil von M ist.*

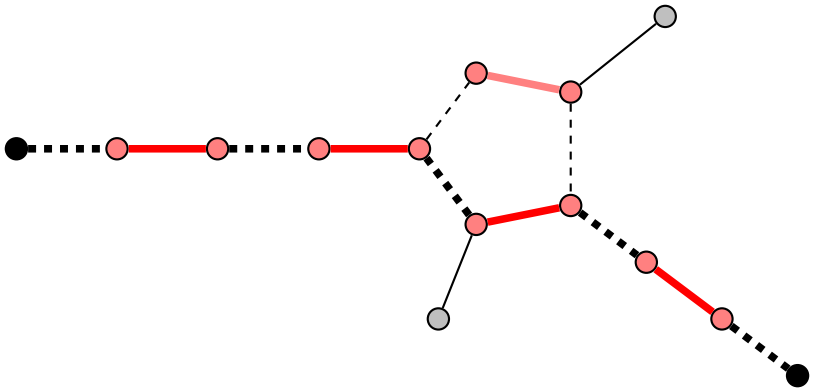


Abbildung 2.6: Ein augmentierender Pfad durch eine Blüte. Die Knoten aus dem Kreis können für das Finden eines augmentierenden Pfades zu einem virtuellen Knoten kontrahiert werden (siehe 3.1).

2.3 Abkürzungsverzeichnis für Graphklassen

Die Algorithmen basieren auf unterschiedlichen Graphtypen, die jeweils zusätzlich weitere Eigenschaften oder Einschränkungen haben können. Aus Gründen der Übersichtlichkeit werden folgende Informationen kompakt abgekürzt:

- Graphklassen:
  - $\mathcal{ALL}$: beliebiger ungerichteter gewichtsloser Graph
  - $\mathcal{BP}$: bipartiter ungerichteter gewichtsloser Graph
  - $\mathcal{PL}$: planarer ungerichteter gewichtsloser Graph
- Grapheigenschaften:
  - $\mathcal{G}_{bg}$: Graph hat bounded Genus
  - $\mathcal{G}^{\rightarrow}$: Graph hat gerichtete Kanten
  - $\mathcal{G}_{w}$: Graph hat gewichtete Kanten

Einen Graphen, der bipartit ist und gerichtete sowie gewichtete Graphen mit bounded Genus hat, schreiben wir nach dieser Notation als $\mathcal{BP}^{\rightarrow}_{w,bg}$.

3 Matching-Algorithmen

3.1 Erster Polynomialzeitalgorithmus für allgemeine Graphen

Graphklasse	Laufzeit	Speicher	Neuerungen
$\mathcal{ALL}$	$O(\|V\|^4)$	$O(\|V\|^2)$	Satz von Berge (1957)

1965 veröffentlichte Jack Edmonds in „Path, trees and Flowers“ [Edm65] den ersten Polynomialzeitalgorithmus, der das Matching-Problem auch für nicht-bipartite Graphen effizient löst.

Nötige Grundlagen

Definition 3.1.1 (Baum, Wald)**.** *Wir bezeichnen einen zusammenhängenden Graphen $G = (V, E)$ als* Baum*, wenn es für jeden Knoten $v \in V$ jeweils genau einen Pfad zu allen anderen Knoten $v' \in V$ gibt.*

Einen Graphen G, der nicht zusammenhängend ist, bezeichnen wir als Wald*, wenn alle zusammenhängenden Teilgraphen $G' \in G$ Bäume sind.*

Definition 3.1.2 (Alternierender Baum, Wald)**.** *Sei $G = (E, V)$ ein Baum. Wir erzeugen aus G einen* alternierenden Baum*, indem wir einen Knoten $r \in V$ wählen, ihn als gerade markieren und rekursiv für jeden markierten Knoten alle unmarkierten Nachbarknoten als ungerade markieren, wenn dieser gerade markiert ist, oder als gerade markieren, wenn dieser ungerade ist, und dabei die Kanten mit der Farbe des Knotens markieren.*

Einen Wald W *bezeichnen wir als* alternierend*, wenn für alle Bäume $B \in W$ gilt, dass diese alternierend sind.*

Algorithmus

Sei $G = (V, E)$ ein Graph, $T \in E$ ein Matching aus G sowie r ein ungematchter Knoten aus G.

Von r ausgehend bilde einen alternierenden Baum. Tritt ein Kreis von ungerader Länge auf, so handelt es sich um eine Blüte. Diese wird zu einem Knoten geschrumpft. Dieser Vorgang wird so weit fortgesetzt, bis ein weiterer ungematchter Knoten gefunden wurde oder keine weiteren Knoten mehr in den augmentierenden Baum aufgenommen werden können. Tritt ersteres ein, so wurde ein augmentierender Pfad gefunden. Blüten, die geschrumpft wurden, werden wieder expandiert. Bisherige Matches des

augmentierenden Pfades werden entfernt und durch Matches, beginnend mit dem ersten Knoten, ersetzt.

Der Algorithmus terminiert, sobald keine augmentierenden Pfade gefunden werden können. Ist der Graph nicht zusammenhängend, so wird für jeden zusammenhängenden Graphen ein alternierender Baum erzeugt, also ein alternierender Wald.

Bedeutung

Das Matching-Problem ist äquivalent zum Problem der Berechnung eines minimalen Spannbaumes eines Graphen [NR59].

In seiner Publikation macht Edmonds auf die Bedeutung eines „effizienten Algorithmus“ (gemeint sind Polynomialzeitalgorithmen) aufmerksam und vernachlässigt einfache Anpassungen des Algorithmus, durch welche die Laufzeit des Algorithmus weiter verbessert werden könnte.

Analog zur Church'schen These stellt Edmonds die Frage, ob es eine Möglichkeit gibt, eine untere Schranke für die tatsächliche Laufzeit eines Problems vorherzusagen. Idealerweise gäbe es für jede Komplexitätsklasse einen Algorithmus, der für alle Probleme dieser Klasse anwendbar ist.

Seit dem mittleren 19. Jahrhundert konnte diese Frage, die als P-NP-Problem bekannt ist, noch nicht beantwortet werden. Für NP-vollständige Probleme ist aktuell deswegen nicht ausgeschlossen, dass diese mittels eines Polynomialzeitalgorithmus gelöst werden können. Es reicht, dass ein einziger Polynomialzeitalgorithmus für ein NP-vollständiges Problem gefunden wird, um zu zeigen, dass alle Probleme aus NP in polynomieller Laufzeit gelöst werden können. Dies folgt daraus, dass auf jedes Problem, das in NP liegt, von allen anderen NP-schweren Problemen mittels Polynomialzeitreduktion reduziert werden kann.

3.2 Der bedeutsamste Matching-Algorithmus für bipartite Graphen

Graphklasse	Laufzeit	Neuerungen
$\mathcal{BP}$	$O((\|E\| + \|V\|) \cdot \sqrt{\|V\|})$	Laufzeitverbesserung aus $O(\|V\| \cdot \|E\|)$

1973 erschien unter dem Titel „An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs“ [HK73] ein Algorithmus von John E. Hopcroft und Richard M. Karp.

Nötige Grundlagen

Der Hopcroft-Karp-Algorithmus baut auf den Satz von Berge auf und nutzt die Eigenschaften von bipartiten Graphen aus (keine Zykel ungerader Länge, keine Kanten zwischen Knoten einer Partition), wodurch dieser einfach und effektiv wird.

Algorithmus

Der Algorithmus sucht wiederholt eine größtmögliche Menge an minimal-großen augmentierenden Pfaden, die untereinander knotendisjunkt sind. Diese Pfade werden dann dafür genutzt, um das Matching iterativ zu verbessern, indem die bisher gematchten Kanten in ungematchte und die ungematchten in gematchte umgetauscht werden. Der Algorithmus terminiert, wenn es keinen augmentierenden Pfad mehr gibt:

Eingabe: Ein bipartiter Graph $G = (V, E)$ sowie ein Matching $M \subseteq E$, wobei zu Beginn $M = \varnothing$.
Ausgabe: M als ein maximales Matching für G.

while M enthält mindestens einen augmentierenden Pfad. **do**
  Setze $Q = \varnothing$, $l =$ minimale Länge eines augmentierenden Pfades.
  Fülle die Menge Q mit augmentierenden Pfaden $Q_1, Q_2, ..., Q_t$, wobei für jeden augmentierenden Pfad Q_i gilt:
    Q_i hat die Länge l.
    Q_i ist knotendisjunkt mit allen anderen augmentierenden Pfaden in Q.
  Augmentiere M nacheinander mit allen augmentierenden Pfaden aus Q.
end while
return M

Jede Phase benötigt höchstens $O(|E| + |V|)$ Schritte, und da es höchstens $\sqrt{|V|}$ Phasen geben kann, beträgt die Laufzeit $O((|E| + |V|) \cdot \sqrt{|V|})$ oder $O(|V|^{5/2})$.

Bedeutung

Das Matching-Problem speziell für bipartite Graphen ist ein besonderer Spezialfall des Matching-Problems für allgemeine Graphen. Folgende Anwendungsgebiete wurden durch den Hopcroft-Karp-Algorithmus beeinflusst:

- Die Bestimmung eines maximalen Flusses in einem Flussnetzwerk mit beschränkten Kapazitäten mittels des Algorithmus von Ford und Fulkerson. [FF62]
- Der 1955 publizierte Kuhn-Munkres-Algorithmus, auch Ungarische Methode genannt; gesucht wird dabei ein perfektes Matching für einen bipartiten gewichteten Graphen, sodass die Summe der Gewichte der Kanten des Matchings minimal ist. [Kuh55]
- Das Lösen des Transportproblems von Hitchcock, bei dem es darum geht, für eine Menge an Gütern aus unterschiedlichen Quellen und einer Menge an Zielen mit bestimmter Nachfrage nach den Gütern, wobei der Transport von Gütern von einer Quelle zu einem Ziel mit Kosten behaftet ist, eine Aufteilung der Güter von den Quellen zu den Zielen so zu finden, dass die Summe der Kosten der Transporte minimal ist. [Hit41]

- Das Entscheidungsproblem, ob ein Baum zu einem Teilbaum eines anderen Baumes isomorph ist. [Mat78]

Bedingt dadurch, dass es eine Vielzahl an Anwendungsbeispielen gibt, welche äquivalent sind zum maximalen Matching in bipartiten Graphen, ist die Komplexitätsklasse dieses Problems von Bedeutung.

3.3 Verbesserung des Algorithmus von Edmonds

Graphklasse	Laufzeit	Neuerungen
$\mathcal{ALL}$	$O(\sqrt{\|V\|} \cdot \|E\|)$	Verbesserte Behandlung von Blüten

Im Jahre 1980 verbesserten Silvio Micalii und Vijay V. Vazirani in „An $O(\sqrt{|v|} \cdot |E|)$ algorithm for finding maximum matching in general graphs“ [MV80] den von Jack Edmonds entwickelten Algorithmus, indem Blüten anders behandelt werden.

Nötige Grundlagen

Definition 3.3.1 (evenlevel, oddlevel eines Knotens)**.** *Das* evenlevel eines Knotens v *ist die Länge eines alternierenden Pfades mit minimaler gerade Länge von v zu einem freien Knoten. Wenn es keinen alternierenden Pfad gerader Länge gibt, so ist das evenlevel unendlich.*

Das oddlevel eines Knotens v *ist die Länge eines alternierenden Pfades mit minimaler ungerade Länge von v zu einem freien Knoten. Wenn es keinen alternierenden Pfad ungerader Länge gibt, so ist das oddlevel unendlich.*

Definition 3.3.2 (level eines Knotens)**.** *Das* level eines Knotens v *bezeichnet das Minimum von evenlevel(v) und oddlevel(v).*

Das level eines Knotens v kann somit auch als die Länge eines alternierenden Pfades minimaler Länge von v zu einem freien Knoten gesehen werden.

Definition 3.3.3 (outer, inner eines Knotens)**.** *Ein Knoten v wird genau dann* outer *genannt, wenn level(v) gerade ist.*

Ein Knoten v wird genau dann inner *genannt, wenn level(v) ungerade ist.*

Definition 3.3.4 (bridge)**.** *Eine Kante (u, v) wird* bridge *genannt, wenn entweder evenlevel(u) und evenlevel(v) endlich sind oder oddlevel(u) und oddlevel(v) endlich sind.*

Sei P nun ein augmentierender Pfad. Da P eine ungerade Länge hat, ist jede Kante $(u, v) \in P$ eine bridge. Existiert eine bridge, so gibt es mindestens zwei Knoten u und v mit einem endlichen evenlevel und oddlevel.

Definition 3.3.5 (tenacity)**.** *Die Funktion* tenacity *wird folgendermaßen definiert:*

$$tenacity(u,v) = \begin{cases} evenlevel(u) + evenlevel(v) + 1, & falls\ (u,v) \notin M \\ oddlevel(u) + oddlevel(u) + 1, & falls\ (u,v) \in M \end{cases}$$

Die tenacity einer Brücke ist äquivalent zur minimalen Länge eines alternierenden Weges zwischen zwei freien Knoten. Ist der Weg hingegen ein Pfad, so ist dieser augmentierend.

Algorithmus

An allen freien Knoten wird ein BFS-Suchalgorithmus mit in jeder Phase inkrementierter Tiefe, beginnend mit 0, ausgeführt, wobei das level dieses Knotens gesucht wird.

Findet der Suchalgorithmus eine bridge-Kante (u, v), so wird eine Subroutine ausgeführt, die nach einem augmentierenden Pfad der Länge $tenacity(u, v)$ sucht.

Wird kein augmentierender Pfad gefunden, so wird ein Blüte B erzeugt. Dabei werden weitere Kanten gefunden, die ebenfalls bridges sind und deren tenacity berechnet wird.

Bevor die augmentierenden Pfade verwendet werden, um die Kardinalität des vorhandenen Matchings zu verbessern, wird überprüft, ob diese disjunkt sind. Mit der Anwendung dieser endet die Phase.

Da eine Phase $O(|E|)$ Schritte beinhaltet sind insgesamt $O(\sqrt{|v|} \cdot |E|)$ Schritte notwendig für das Finden des maximalen Matchings.

Bedeutung

Silvio Micalii und Vijay V. Vazirani waren nicht die einzigen, die den Algorithmus von Edmonds optimierten. Weitere Arbeiten von Gabow [Gab76], Kameda und Munro [KM74] und Even und Kariv [EK75] hatten sich ebenfalls mit diesem Thema beschäftigt.

3.4 Nutzung linearer Optimierung zum Lösen des Matching-Problems

Graphklasse	Laufzeit	Neuerungen
$\mathcal{ALL}_w$	unklar	Lineare Optimierung

1984 präsentierten M. Grötschel und O. Holland ihren Ansatz, lineare Optimierung für das Berechnen eines maximalen Matchings unter Berücksichtigung von Kantengewichten zu verwenden, in ihrer „Solving matching problems with linear programming“ genannten Arbeit. [GH85]

Nötige Grundlagen

- Lineare Optimierungsalgorithmen
- Simplex-Methode
- Schnittebenenverfahren

Algorithmus

Grundlegend wird ein heuristisch ermitteltes Matching um weitere Kanten erweitert und ein linearer Optimierungsalgorithmus verwendet, der für den gegebenen Teilgraphen über mehrere Iterationen ein Optimum berechnet.

Wähle eine Menge an Kanten, wobei für jeden Knoten die Anzahl der Kanten beschränkt wird. Finde ein beliebiges Matching für diese Menge. Erweitere dies zu einem perfekten Matching, indem für ungematchte Knoten die Kanten hinzugefügt werden. Formuliere nun ein lineares Optimierungsproblem, um das Matching mit den geringsten Kosten zu finden. Verwende das Schnittebenenverfahren (cutting plane recognition), versuche dabei zunächst, Heuristiken anzuwenden, und wenn keine cuts gefunden wurden, verwende eine modifizierte Padberg-Rao Prozedur. Überprüfe, ob es sich um ein optimales perfektes Matching handelt oder es Kanten gibt, die zu einer Verbesserung führen, und wiederhole die Prozedur mit den hinzugefügten Kanten falls notwendig.

Die Performanz des Algorithmus übersteigt die Performanz kombinatorischer Algorithmen aufgrund des Overheads für das Starten des Optimierungsalgorithmus erst bei einer größeren Anzahl an Knoten; die Ausführung des Algorithmus sei am effektivsten bei vollständigen Graphen und mehr als 500 Knoten.

Bedeutung

Die Autoren berichten, dass lineare Optimierungsalgorithmen neben dem Matching-Problem ebenfalls weitere, zu dem Matching-Problemen unverwandte, Probleme lösen können und so flexibel sind. Das Schnittebenenverfahren wurde bereits für weitere NP-vollständige kombinatorische Probleme wie das Botenproblem (travelling salesperson problem) [CP80] oder das lineare Sortierproblem (linar ordering problem) [GJR84] eingesetzt.

2018 wurde gezeigt, dass bestimmte Algorithmen der Simplex Methode NP-mächtig sind, also jedes Problem der Komplexitätsklasse NP mit polynomiellem Overhead implizit lösen können [DS18].

3.5 Deterministischer paralleler Algorithmus für perfekte Matchings in bipartiten planaren Graphen

Graphklasse	Komplexitätsklasse	Speicher	Neuerungen
$\mathcal{BP} \cap \mathcal{PL}$	SPL	$O(\log \lvert V \rvert)$	Deterministische Gewichtungsfunktion für einzigartiges perfektes Matching

2008 präsentierten Samir Datta, Raghav Kulkarni und Sambuddha Roy im später unter dem Titel „Deterministically Isolating a Perfect Matching in Bipartite Planar Graphs“ [DKR10] veröffentlichten Artikel einen Algorithmus, der deterministisch und parallel ein perfektes Matching für einen bipartiten und planaren Graphen berechnet.

Nötige Grundlagen

Definition 3.5.1 (Isolationslemma [MVV87])**.** *Sei M die Menge $1, 2, ..., n$ sowie F eine Familie der Untermengen von M. Wenn jedem Element $m \in M$ zufällig ein Gewicht $g \in 1, 2, ..., 2n$ zugewiesen wird, so wird mit einer hohen Wahrscheinlichkeit ($> 50\%$) die minimale Summe der Gewichte aus F einzigartig.*

Definition 3.5.2 (Zirkulation)**.** *Sei G ein Graph sowie C ein Kreis in G. Die Zirkulation von C in G ist die Summe der vorzeichenbehafteten Gewichte der Kanten: $\sum_{e \in C} w'(e)$.*

Definition 3.5.3 (Lemma)**.** *Wenn die Zirkulationen für alle Zykel in einem bipartiten Graphen ungleich 0 sind, dann ist das perfekte Matching mit minimalem Gewicht einzigartig.*

Definition 3.5.4 (Arten von Problemen)**.** *Gegeben sei ein Graph G. Folgende Probleme werden definiert:*

- *Problem der Entscheidung: Existiert ein perfektes Matching für G?*
- *Suchproblem: Finde ein perfektes Matching in G, falls eins existiert.*
- *Problem der Einzigartigkeit: Hat G genau ein perfektes Matching?*

In dieser Arbeit gilt zusätzlich, dass G bipartit und planar ist.

Neuerungen

Anstelle des Isolationslemmas wird durch Verwendung eines nach [BTV09] ermittelten polynomiell-langen advice strings eine deterministische Logspace-Prozedur verwendet, welche Gewichte an die Kanten hinzufügt, sodass das Gewicht des perfekten Matchings mit minimalem Gewicht einzigartig wird.

Algorithmus

Beginnend mit dem Graphen als Eingabe, führe folgende Schritte aus:

1. Transformiere den Graphen in einen eulerschen bipartiten planaren Graphen
2. Markiere benachbarte Facetten des planaren Graphen mit unterschiedlichen Vorzeichen (+ / −)
3. Gewichte die Kanten einer Facette so, dass diese immer in derselben Richtung traversiert werden

Führe anschließend folgende Schritte aus, um ein einzigartiges Matching zu extrahieren:

1. Führe den Graphen über in eine $n \times n$ Matrix
2. Bilde die Determinante der Matrix
3. Anhand eines Koeffizienten dieser lässt sich ablesen, welches minimale Gewicht das perfekte Matching hat
4. Überprüfe für jede Kante, ob das Entfernen dieser das minimale Gewicht des perfekten Matchings erhöht. Ist dies der Fall, so ist diese Kante Teil des perfekten Matchings, andernfalls entferne diese komplett.

Bedeutung

SPL (stoic probabilistic logarithmic space) ist eine Komplexitätsklasse, welche von Allender, Reinhardt und Zhou eingeführt wurde [ARZ99]. Diese enthält alle Probleme, die auf die Berechnung einer Determinante reduziert werden können unter der Voraussetzung, dass diese entweder 0 oder 1 betragen kann.

Für bipartite planare Graphen war bereits vorher bekannt, dass das Existenzproblem (1) „Gibt es ein perfektes Matching?“ und das Suchproblem (2) „Finde ein perfektes Matching“ in NC liegen. Für allgemeine planare Graphen liegt allerdings nur Frage (1) in NC; für Frage (2) sind ausschließlich Algorithmen in P bekannt.

Die Autoren sehen Ihre Arbeit als Schritt in die Richtung, um die Frage zu beantworten, ob es einen Algorithmus für allgemeine (planare) Graphen gibt, indem sie nach einer Möglichkeit suchen, den Gewichtungsalgorithmus so anzupassen, dass er auch für allgemeinere Graphentypen anwendbar ist.

Außerdem stellen die Autoren die zentrale Frage, ob das Matching-Problem in NC lösbar ist. Diese Frage wird bis heute versucht für allgemeine Graphen zu beantworten.

Die Autoren weisen darauf hin, dass das Problem der Erreichbarkeit eines Knotens in einem gerichteten Graphen auf bipartites perfektes Matching reduziert werden kann [KUW86]. Außerdem gilt, dass Erreichbarkeit für planare Graphen auf perfektes Matching für bipartite planare Graphen reduziert werden kann [DKLM10].

3.6 Vereinfachung und ausführlicher Beweis des allgemeinen Matching-Algorithmus

Graphklasse	Laufzeit	Neuerungen
$\mathcal{ALL}$	$O(\sqrt{\|V\|} \cdot \|E\|)$	Vereinfachter Algorithmus (aus [MV80])

2012 veröffentlichte Vijay V. Vazirani eine Überarbeitung des im Jahre 1980 vorgestellten Matching-Algorithmus für allgemeine Graphen (siehe Abschnitt 3.3). Unter dem Titel „A Simplification of the MV Matching Algorithm and its Proof“ [Vaz12] beweist der Autor eine vereinfachte Version des Algorithmus.

Nötige Grundlagen

Definition 3.6.1 (inverse Ackermannfunktion [CLRS09])**.** *Die inverse Ackermannfunktion wird als*

$$\alpha(m,n) = \min\{i \geq 1 : A(i, \lfloor m/n \rfloor) \geq \log_2 n\}$$

definiert, wobei für A gilt:

$$\begin{array}{lcl} \mathrm{A}(0,n) & = & n+1 \\ \mathrm{A}(m+1,0) & = & \mathrm{A}(m,1) \\ \mathrm{A}(m+1,n+1) & = & \mathrm{A}(m,A(m+1,n)) \end{array}$$

Algorithmus

Die Implementierung des Algorithmus hat sich nicht verändert. Der Leser wird auf den Abschnitt „Algorithmus“ im Abschnitt 3.3 hingewiesen.

In dem Pointer-Modell beträgt die Laufzeit $O(|E| \cdot \sqrt{|V|} \cdot \alpha(|E|,|V|))$, wobei α die inverse Ackermannfunktion ist. Im RAM-Modell hat der Algorithmus eine Laufzeit von $O(|E| \cdot \sqrt{|V|})$.

Bedeutung

Auch nach mehreren Jahrzehnten wurde kein Matching-Algorithmus für allgemeine Graphen gefunden, der eine deutlich bessere Laufzeit hat. Der Hauptgedanke besteht weiterhin darin, augmentierende Pfade mit minimaler Länge in einem bereits vorhandenem Matching zu finden und dieses schrittweise zu verbessern. Dabei unterscheiden sich die Algorithmen für allgemeine und für bipartite Graphen nur in der Weise, wie diese augmentierenden Pfade gefunden werden.

Der Autor erklärt, dass die Suche nach besseren Matching-Algorithmen zur Entwicklung von Verbesserungen für andere Probleme und Paradigmen der Komplexitätstheorie beigetragen hat.

Als Beispiele für Verbesserungen für andere Probleme nennt der Autor unter anderem das Isolationslemma (siehe Abschnitt 3.5, Nötige Grundlagen). Für Paradigmen der Komplexitätstheorie gibt der Autor als Beispiel die Definitionen der Komplexitätsklassen P [Edm65] und #P [Val79] oder die Äquivalenz von zufälliger Generation und approximierendes Abzählen für selbstreduzierende Probleme [JVV86].

Zusätzlich merkt der Autor an, dass Shapley 2012 den Nobelpreis für den Gale-Shapley-Algorithmus [GS62], auch Stabiles-Matching-Algorithmus genannt, erhalten hat. Der Algorithmus wurde dafür verwendet, um Ärzte Krankenhäusern zuzuweisen, wobei die Ärzte eine Präferenz bezüglich der Ärzte sowie die Krankenhäuser eine Präferenz bezüglich der Krankenhäuser nennen konnten [oS20].

3.7 Approximationsalgorithmen für Matching in gewichteten Graphen

Graphklasse	Laufzeit	Neuerungen
$\mathcal{ALL}_w$	$O(\|E\| \cdot \epsilon^{-1} \cdot \log \epsilon^{-1})$	Linearer Approximierungsalgorithmus

2014 präsentierten R. Duan und S. Pettie in „Linear-Time Approximation for Maximum Weight Matching“ [DP14] einen linearen Approximationsalgorithmus für das Matching-Problem in gewichteten Graphen.

Nötige Grundlagen

Die Autoren definieren MWM (maximum weight matching) als das Matching-Problem für gewichtete Graphen, analog dazu MWPM (maximum weight perfect matching) als das perfekte Matching-Problem für gewichtete Graphen sowie MCM (maximum cardinality matching) das Matching-Problem für ungewichtete Graphen. Bei den Algorithmen für gewichtete Graphen wird nach einem (perfekten) Matching gesucht, sodass die Summe der Gewichte der Kanten minimal ist.

Definition 3.7.1 (Wurzelblüte (Root Blossom))**.** *Eine Wurzelblüte ist eine Blüte, die nicht in anderen Blüten enthalten ist.*

Definition 3.7.2 (Infrage kommende Kanten (Eligible edges)**.** *Sei $G = (V, E)$ ein Graph. Es existiert eine Teilmenge $E_{elig} \subseteq E$ an Kanten, die als* infrage kommend *markiert sind. Diese werden anhand bestimmter Kriterien gewählt. Der entstehende Teilgraph, der nur infrage kommende Kanten enthält, wird $G_{elig} = (V, E_{elig})$ genannt.*

Algorithmus

Für jeden Knoten wird eine zusätzliche Variable y eingeführt. Abhängig davon besteht das Vorgehen darin, eine Gleichung mittels linearer Optimierung zu minimieren. In jedem Durchlauf werden dabei folgende Schritte durchgeführt:

1. Suche eine maximale Menge an knotendisjunkten augmentierenden Pfaden in G_{elig} und erweitere durch diese das Matching
2. Suche Blüten, die von freien Knoten in alternierenden Pfaden gerader Länge erreichbar sind, und schrumpfe diese. Aktualisiere G_{elig}.
3. Passe die Werte der linearen Optimierung an
4. Löse Wurzelblüten auf, die nicht mehr relevant sind

Der (1ϵ)-MWM-Algorithmus verwendet eine Tiefensuche zur Ermittlung augmentierender Pfade und hat insgesamt eine Laufzeit von $O(|E| \cdot \epsilon^{-1} \cdot \log \epsilon^{-1})$ für allgemeine Graphen. Sind die Gewichte der Kanten ganze Zahlen, so verbessert sich die Laufzeit auf $O(|E| \cdot \epsilon^{-1} \cdot \min\{\log \epsilon^{-1}, \log N\})$, wobei N das größte Kantengewicht aus G ist.

Bedeutung

Lineare Approximationsalgorithmen sind in Bereichen sinnvoll, wo eine schnelle Laufzeit wichtiger ist als eine optimale Lösung. Dazu gehört das Routen von Paketen in Netzwerken mittels Switches. Pakete an mehreren Eingängen müssen in jeder Phase mehreren Ausgängen zugeordnet werden. $\frac{1}{2}$-MCM-Algorithmen, die dieses bipartite Matching lösen, sind der iSLIP-Algorithmus [McK99] sowie der PIM-Algorithmus [AOST92].

Die Autoren behaupten, dass die Algorithmen von Hopcroft und Karp [HK73] und Micali und Vazirani [MV80] $(1-\epsilon)$-MCM-Algorithmen sind, welche auf den Beobachtungen basieren, dass eine maximale Menge von augmentierenden Pfaden minimaler Länge in Linearzeit gefunden werden können, das Anwenden dieser die Länge der augmentierender Pfade minimaler Länge erhöht und dass nach k Durchgängen das Ergebnis ein $(1 - \frac{1}{k+1})$-MCM ist.

Die Arbeit enthält außerdem eine tabellarische Darstellung unterschiedlicher Algorithmen im zeitlichen Verlauf für das MCM-Problem in allgemeinen Graphen, das MWM-Problem in bipartiten und in nicht-bipartiten Graphen sowie Algorithmen, die das MCM-Problem und das MWM-Problem approximierend lösen.

3.8 Eingabereduktion in Linearzeit für effizientere Matching-Algorithmen

Graphklasse	Laufzeit	Neuerungen
$\mathcal{ALL}$	$O(k \cdot (\|V\| + \|E\|))$	Problemkern-Reduktion

Im Jahr 2017 stellten George B. Mertzios, André Nichterlein und Rolf Niedermeier in „The Power of Linear-Time Data Reduction for Maximum Matching“ [MNN17] eine Problemkern-Reduktion in linearer Laufzeit für das Matching-Problem vor.

Nötige Grundlagen

Definition 3.8.1 (Problemkern-Reduktion)**.** *Ein parametrisierbares Problem ist eine Menge an Instanzen (I, k), wobei $I \in \Sigma^*$ für ein endliches Alphabet Σ und $k \in \mathbb{N}$ der Parameter ist. Wir bezeichnen zwei Instanzen (I, k) und (I', k') parametrisierbarer Probleme P und P' als äquivalent, wenn (I, k) genau dann eine Ja-Instanz aus P ist, wenn (I', k') eine Ja-Instanz aus P' ist. Als* Problemkern-Reduktion *bezeichnen wir einen Algorithmus, der für jede Instanz (I, k) aus P in polynomieller Zeit eine äquivalente Instanz (I', k') aus P berechnet, sodass $|I'| + k' \leq f(k)$ für eine berechenbare Funktion f gilt. Wir sagen, dass f die* Größe *des Problemkerns misst, und wenn $f(k) \in k^{O(1)}$ gilt, dann erlaubt P einen polynomiellen Problemkern. Oftmals wird ein Problemkern dadurch erreicht, indem in Polynomialzeit ausführbare Datenreduktionsregeln angewendet werden. Wir nennen eine Datenreduktionsregel $\mathcal{R}$* korrekt*, wenn die Instanz (I', k'), die durch Anwendung von $\mathcal{R}$ auf (I, k) entstanden ist, äquivalent zu (I, k) ist. Eine Instanz, bei der das Anwenden von Datenreduktionsregeln keine Veränderung bringt, nennen wir* reduziert*.*

Algorithmus

Der Algorithmus nimmt als Eingabe einen ungerichteten Graphen $G = (V, E)$ sowie eine natürliche Zahl s und beantwortet die Frage, ob es ein Matching $M \subseteq E$ mit Kardinalität $|M| = s$ gibt.

Der Algorithmus für allgemeine Graphen ist folgendermaßen aufgebaut:

1. Nutze in konstanter Linearzeit ausführbare Anweisungen, um eine Knotenmenge X zu berechnen, sodass das Matching in (G ohne X) trivial in linearer Zeit lösbar ist.

2. Berechne in Linearzeit ein beliebiges nicht erweiterbares Matching M in (G ohne X).

3. Starte mit M als initiales Matching in G und vergrößere dessen Kardinalität höchstens $|X| = k$ mal, um in einer Laufzeit von $O(k \cdot (|V| + |E[])$ ein maximales Matching für G zu erhalten.

Die erste Phase wird dabei durch einfache Datenreduktionsregeln umgesetzt:

Reduktionsregel 2.1 Sei $v \in V$. Wenn $deg(v) = 0$, dann lösche v. Wenn $deg(v) = 1$, dann lösche v und den Nachbarknoten von v und verringere die Lösungsmenge s um 1 (v wird mit seinem Nachbarknoten gematcht).

Reduktionsregel 2.3 Sei v ein Knoten mit $deg(v) = 2$ sowie u und w die Nachbarknoten von v, für die $deg(u) \leq 2$ und $deg(w) \leq 2$ gilt. Dann entferne v, verschmelze u und w und verringere s um 1.

Mithilfe der Problemkerne lässt sich das Matching mit Laufzeit $O(k \cdot (|V| + |E[])$ berechnen, wobei k einer der folgenden Parameter ist: feedback vertex number, feedback edge number, vertex cover number

Für bipartite Graphen lässt sich das Matching-Problem mittels Problemkern-Reduktion in $O(k^{7.5} + |V| + |E[)$ berechnen.

Bedeutung

Die Autoren erwähnen, dass Matching-Algorithmen in Linearzeit für spezielle Graphklassen existieren, darunter für konvex bipartite Graphen [SY96], für stark chordale Graphen [DK98], für chordal bipartite Graphen [Cha96] sowie für cocomparability Graphen [MNN17].

Der Ansatz der Parametrisierung für in Polynomialzeit berechenbare Probleme wird FPT (fixed parameter tractable) in P oder FPTP genannt.

In der Problemkern-Reduktion geht es darum, beweisbar effektiv und effizient Daten zu reduzieren. Das Ziel besteht darin, die Problemkern-Reduktion in einer niedrigeren Komplexität als den eigentlichen Algorithmus durchzuführen. Somit wird ein linearer Algorithmus bei Problemen, die in polynomieller Zeit berechnet werden können, gesucht. Wenn $\mathsf{P} \neq \mathsf{NP}$ gilt, dann ist für ein Problem, das NP-schwer ist, auch ein Polynomialzeitalgorithmus für die Problemkern-Reduktion sinnvoll.

Die Autoren ergänzen, dass um einen Problemkern-Reduktions-Algorithmus in Linearzeit zu erhalten speziell angepasste Datenstrukturen und Datenreduktionsregeln gefunden werden müssen, welche die Möglichkeiten in Linearzeit voll ausschöpfen.

3.9 Algorithmus für perfektes Matching in quasi-NC für allgemeine Graphen

Graphklasse	Laufzeit	Neuerungen
$\mathcal{ALL}$	$O(\log^3 \|V\|)$ bei $n^{O(\log^2 n)}$ Prozessoren	Derandomisierung des Isolationslemmas (Abschnitt 3.5, Nötige Grundlagen)

Im Jahr 2017 zeigten Ola Svensson und Jakub Tarnawski in „The Matching Problem in General Graphs is in Quasi-NC“ [ST17], dass das perfekte Matching-Problem für allgemeine Graphen mit einer Laufzeit in quasi-NC berechnet werden kann.

Nötige Grundlagen

Definition 3.9.1 (Tutte-Matrix)**.** *Sei $G = (V, E)$ ein Graph. Wir bezeichnen eine $|V| \times |V|$-Matrix als Tutte-Matrix T, welche folgendermaßen definiert ist:*

$$T(G)_{u,v} = \begin{cases} X_{(u,v)} & \text{falls } (u,v) \in E \text{ und } u < v \\ -X_{(u,v)} & \text{falls } (u,v) \in E \text{ und } u > v \\ 0 & \text{falls } (u,v) \notin E \end{cases}$$

wobei $X_{(u,v)}$ für $(u, v) \in E$ Variablen sind. Der Satz von Tutte (siehe 2.2.3) besagt, dass $\det T(G) \neq 0$ *genau dann, wenn G ein perfektes Matching hat.*

Relevant hier ist noch zu erwähnen. dass das Berechnen einer Determinante in NC liegt [Csa76, Ber84] und sich somit zur Parallelisierung eignet.

Algorithmus

Der Algorithmus weist jeder Kante aus dem Graphen ein Gewicht so zu, dass die Zirkulation aller Kreise in dem Graphen ungleich 0 sind. Als Zirkulation wird die Summe der Gewichte der Kanten eines Kreises bezeichnet. Gilt diese Bedingung, so wird die Gewichtungsfunktion w als *isolierend* bezeichnet und es existiert genau ein perfektes Matching mit minimaler Summe der Kantengewichte.

Der Algorithmus baut auf den für bipartite Graphen entrandomisierenden Algorithmus aus [FGT16] auf und hat eine Laufzeit von $O(\log^3 |V|)$ bei $n^{O(\log^2 n)}$ Prozessoren.

Bedeutung

Die Autoren bemerken, dass selbst nach einem halben Jahrhundert seit dem Beginn der Entwicklung von Matching-Algorithmen die deterministische parallele Komplexität des perfekten Matching-Problems noch nicht vollständig verstanden wurde.

Ein Problem bezeichnen die Autoren als „parallel effizient lösbar“, wenn es einen Algorithmus gibt, der in polylogarithmischer Zeit mit polynomisch vielen Prozessoren dieses löst. Ein Problem ist somit in der Klasse NC enthalten, wenn es uniforme Schaltkreise mit polynomieller Größe und polylogarithmischer Tiefe besitzt. Wird Randomisierung erlaubt, so erhält man die Komplexitätsklasse RNC.

Die Entrandomisierung des Isolationslemmas wird als zentrales Problem dargestellt. Sowohl für die Entscheidungsversion („Existiert ein perfektes Matching?“) [Lov79] als auch für die Suchversion („Finde ein perfektes Matching, falls eines existiert.“) [KUW86, MVV87] wurde bisher gezeigt, dass diese in RNC liegen. Die Autoren stellen die Frage, ob Randomisierung für das perfekte Matching überhaupt notwendig ist, um die unterschiedlichen Matchings isolieren zu können. Eine Entrandomisierung des Isolationslemmas impliziert die Existenx von Algorithmen mit einer Laufzeit in quasi-NC.

Als nächstes Ziel benennen die Autoren den Schritt von quasi-NC in NC für das perfekte Matching-Problem. Dafür müsste die Anzahl der Gewichtungsfunktionen von $\log n$ auf eine Konstante reduziert werden.

Schließlich erwähnen die Autoren, dass ein verwandtes Problem, welches bisher nicht entrandomisiert werden konnte, das exakte Matching-Problem [PY82] ist. Im Vergleich zum regulären Matching-Problem ist eine Teilmenge der Kanten des Graphen gefärbt. Gesucht wird ein perfektes Matching so, dass es genau k gefärbte Kanten hat. Von dem Problem ist bekannt, dass es in RNC liegt [MVV87]. Allerdings ist nicht bekannt, ob es überhaupt in Polynomialzeit lösbar ist.

3.10 Skalierender Algorithmus für das gewichtete perfekte Matching-Problem

Graphklasse	Laufzeit	Neuerungen
$\mathcal{ALL}_w$	$O(\lvert E\rvert \cdot \sqrt{\lvert V\rvert} \cdot \log(\lvert V\rvert \cdot N))$	Neuer skalierender Algorithmus

Ran Duan, Seth Pettie und Hsin-Hao Su stellten 2018 in ihrer Arbeit „Scaling Algorithms for Weighted Matching in General Graphs“ [DPS18] einen skalierenden Algorithmus für das perfekte Matching-Problem in gewichteten Graphen vor.

Nötige Grundlagen

Definition 3.10.1 (große und kleine Blüten)**.** *Sei B eine Blüte. Gegeben $\tau \in \mathbb{N}$ wird B als* große Blüte *bezeichnet, wenn sie mindestens τ Knoten enthält. Hat B weniger als τ Knoten, so wird B als* kleine Blüte *bezeichnet.*

Algorithmus

Der Algorithmus ist eine Weiterentwicklung von [Gab85] und [GT91], zerlegt im Vergleich dazu alte Blüten auf eine neue Weise und erfordert ein weniger schweres Ziel in jeder Skalierung.

Kantengewichte werden nur bitweise berücksichtigt. In jeder Skalierung wird versucht, ein optimales perfektes Matching unter Verwendung der i höchstwertigen Bits zu finden, wobei i die Iteration der Skalierung angibt. Im Vergleich zu den vorherigen Algorithmen wird hier bei jeder Skalierung nicht ein approximierend optimales perfekte Matching, sondern ein approximierend optimales fast-perfektes Matching berechnet. Dies bietet den Vorteil, dass Blüten in der nächsten Skalierung schneller zerlegt werden können. Für diesen Prozess werden zwei Algorithmen, „Liquidationist“ und „Hybrid“, vorgestellt.

Die Zerlegung großer Blüten verläuft dabei bei beiden Algorithmen identisch ab und ist im Vergleich zur Zerlegung kleiner Blüten viel effizienter. Es müssen nur die Gewichte und Parameter der anderen großen Blüten und des Graphen aktualisiert werden. Die Zerlegung kleiner Blüten ist unterschiedlich implementiert und erfordert die Nutzung weiterer Suchalgorithmen.

Erst nach der letzten Skalierung wird ein perfektes Matching erzeugt, falls vorher Knoten frei geblieben waren.

Die Laufzeit des Algorithmus beträgt $O(\lvert E\rvert \cdot \sqrt{\lvert V\rvert} \cdot \log(\lvert V\rvert \cdot N))$, wobei N für das Maximum der Kantengewichte des Graphen steht. Diese ergibt sich daraus, dass $O(\log(\lvert V\rvert \cdot N))$ Skalierungen notwendig sind. Beim Verwenden des „Liquidationist“-Algorithmus beträgt die Laufzeit einer Skalierung $O(Edm)$, wobei Edm für die Laufzeit einer Edmonds-Suche (siehe [Gab85]) steht. Mit dem „Hybrid“-Algorithmus beträgt die Laufzeit einer Skalierung hingegen $O(\lvert E\rvert \cdot \sqrt{\lvert V\rvert})$.

Bedeutung

Das Matching-Problem für gewichtete Graphen scheint leichter berechenbar als das perfekte Matching-Problem für gewichtete Graphen [DS12, HK12, Pet12].

Die Autoren erwähnen, dass es zwei Barrieren gibt, die eine bessere Laufzeit von Matching-Algorithmen für gewichtete Graphen verhindern.

1. Die Laufzeit für das Berechnen der maximalen Kardinalität eines Matchings beträgt $O(|E| \cdot \sqrt{|V|})$, obwohl die Berechnung dieser einfacher erscheint als die einer Skalierung.

2. Für bipartite Graphen, wo Blüten weniger komplex sind als in allgemeinen Graphen, wurde bisher kein Algorithmus mit besserer Laufzeit gefunden. Eine Ausnahme bildet [CMSV17], welche allerdings nur für spärliche bipartite Graphen gilt.

3.11 Gewichtetes perfektes Matching in NC für planare Graphen

Graphklasse	Komplexitätsklasse	Neuerungen
$\mathcal{PL}_w$	NC	Gewichtsberechnung nach Kasteleyn

2018 stellte Piotr Sankowski in seiner mit „NC Algorithms for Weighted Planar Perfect Matching and Related Problems“ [San18] betitelten Arbeit Matching-Algorithmen in der Komplexitätsklasse NC für planare Graphen mit polynomiell beschränkten Kantengewichten vor.

Nötige Grundlagen

Definition 3.11.1 (Dual eines planaren Graphen)**.** *Sei $G = (V, E)$ ein planarer Graph. Um den* dualen Graphen *G' von G zu erhalten, gehe folgendermaßen vor:*

1. *Erzeuge für jede Facette aus G einen Knoten in G'.*

2. *Für jede Kante e aus G erzeuge eine Kante e' in G' so, dass diese e schneidet.*

G' ist ebenfalls planar und der duale Graph von G' ist wieder G.

Definition 3.11.2 (Vereinfachter Graph)**.** *Sei $G = (V, E)$ ein planarer Graph. Wir bezeichnen G als* vereinfachten Graphen*, wenn dieser keine Knoten mit Grad 1 hat und wenn keine zwei Knoten mit Grad 2 inzident zueinander sind.*

Definition 3.11.3 (k-Zusammenhang eines Graphen)**.** *Sei G ein Graph. G ist* k-zusammenhängend*, wenn durch das Löschen beliebiger $k - 1$ Kanten kein Graph G' erzeugt werden kann, der nicht zusammenhängend ist.*

Algorithmus

Sei G ein planarer Graph. Der Algorithmus für ein perfektes Matching ist folgendermaßen aufgebaut:

1. Entferne nicht erlaubte Kanten und vereinfache G, indem Knoten mit Grad 1 entfernt werden und die dazu inzidente Kanten ins Matching aufgenommen werden. Kürze Pfade, die aus Knoten mit Grad 2 bestehen, ab. Suche für jede Zusammenhangskomponente (diese ist 2-zusammenhängend) ein perfektes Matching.

2. Suche knotendisjunkte Kreise mit gerader Länge. Gewichte diese und entferne nicht erlaubte Kanten. Suche Blüten im kritischen Dual jeder Zusammenhangskomponente.

3. Konstruiere ein perfektes Matching, das alle Blüten aus dem Dual berücksichtigt. Führe dabei den Algorithmus rekursiv für Teilgraphen aus.

Durch das Anpassen der Kriterien für erlaubte Kanten kann der Algorithmus speziell das perfekte Matching mit minimaler Summe der Kantengewichte finden.

Der Algorithmus verwendet Routinen, von denen bekannt ist, dass diese in NC liegen, und wird damit deterministisch und parallel ausgeführt. Dazu gehören die Berechnung von Zusammenhangskomponenten und von Spannbäumen in einem ungerichteten Graphen [CL95, SV82], die Ermittlung von Pfaden in gerichteten Graphen, das Finden von maximal unabhängigen Mengen in Graphen [Lub86] sowie das Zählen von der Anzahl der Knoten in Teilbäumen [MR89, TV84].

Bedeutung

Da das Maximalgewicht Matching-Problem für bipartite planare Graphen effizient auf das gewichtete Matching-Problem in allgemeinen planaren Graphen reduziert werden kann, ist der vorgestellte Algorithmus gleichzeitig auch ein NC-Algorithmus für das Maximalgewicht Matching-Problem. Für allgemeine planare Graphen wurde aber noch kein Algorithmus gefunden, der das Maximalgewicht Matching-Problem in NC löst.

Ein NC-Algorithmus für planare Graphen existierte vorher nur für das perfekte Matching in Graphen, die bipartit sind [MV00, Mil86]. Dabei seien perfekte Matchings in allgemeinen Graphen genauso mächtig wie Matchings mit maximaler Kardinalität [San07]. Das maximales-Matching-Problem konnte bisher in NC nur für bipartite Graphen gelöst werden [Kar86].

Der Algorithmus baut auf die Arbeit von Kasteleyn [Kas63] auf, in der gezeigt wurde, dass das Problem der Anzahl an perfekten Matchings in planaren Graphen auf Determinantenberechnungen reduzierbar ist.

Ist die Anzahl an perfekten Matchings polynomiell beschränkt so existiert ein Algorithmus, der in NC ein perfektes und gewichtetes perfektes Matching konstruieren kann [AHT07].

3.12 Ein neuer parametrisierter Algorithmus für maximales Matching

Graphklasse	Laufzeit	Neuerungen
$\mathcal{ALL}$	$O(k^4 \cdot \|V\| + \|E\|)$	Rekursives Teilen von G in Module

In der Arbeit „Fully Polynomial FPT Algorithms for Some Classes of Bounded Clique-width Graphs“ [CDP19] untersuchten 2019 David Coudert, Guillaume Ducoffe und Alexandru Popa die P-Vollständigkeit von Graphenproblemen. Dabei präsentierten sie einen Matching-Algorithmus mit einer Laufzeit von $O(k^4 \cdot |V| + |E|)$, wobei k entweder modular-width [CO00] oder die P_4-Spärlichkeit des Graphen ist.

Nötige Grundlagen

Definition 3.12.1 (Join, Clique-width, Split)**.** *Sei G ein Graph.*

Ein Join *ist eine Menge an Kanten, welche einen vollständigen bipartiten Teilgrap induzieren.*

Clique-width *[CO00] ist ein Maß dafür, wie einfach es ist, einen Graphen zu rekonstruieren, indem Joins zwischen Knotenmengen hinzugefügt werden.*

Ein Split *ist ein Join, der zusätzlich G schneidet.*

Definition 3.12.2 (Modul)**.** *Sei (A, B) eine Bipartion einer Knotenmenge, welche durch das Entfernen eines Splits erreicht wurde. Wenn jeder Knoten aus A inzident zu einigen Kanten des Splits ist, dann ist A ein* Modul *von G. Somit gilt: Für jeden Knoten $v \in B$ gilt, dass v entweder zu keinem oder zu allen Knoten aus A inzident ist.*

Definition 3.12.3 (Quotient Graph)**.** *Sei G ein Graph M die Menge der Module von G. Wir erzeugen einen* Quotient Graph *G' für G, indem wir aus jedem Modul $m \in M$ genau einen Knoten wählen.*

Definition 3.12.4 (P_4-Spärlichkeit [BO99])**.** *Die P_4-Spärlichkeit von G ist als das minimale $q \geq 7$ definiert, sodass G ein $(q, q-3)$-Graph ist, also höchstens $q \cdot |V| - (q-3)$ Kanten hat.*

Algorithmus

Sei G ein Graph. Um ein maximales Matching M zu finden, gehe folgendermaßen vor:

1. Berechne für G den Quotient Graphen G'.
2. Für jedes Modul aus G' bestimme ein maximales Matching. Rufe den Algorithmus rekursiv mit G' auf, wenn nicht einer der „einfachen“ Fälle auftritt:
   a) G' enthält keine Kanten
   b) G' ist ein vollständiger Graph

c) G' ist ein trivialer Graph (besteht nur aus Modulen mit einzelnen Knoten)

3. Entferne aus jedem Modul alle ungematchten Kanten aus dem Graphen.

4. Suche nach augmentierenden Pfaden zwischen Modulen und erzeuge so ein maximales Matching.

Je nach der Struktur der Graphen werden dabei unterschiedliche Methoden verwendet, um ein maximales Matching zu erzielen.

Die Laufzeit des Algorithmus beträgt $O(q(G)^4 \cdot |V| + |E|)$, wobei $q(G)$ die P_4-Spärlichkeit von G ist. Die Aufteilung von G kann in einer Laufzeit von $O(|V| + |E|)$ berechnet werden [TCHP08].

Bedeutung

Analog zu dem Konzept der NP-Härte und NP-Vollständigkeit von Problemen wird in dieser Arbeit nach [Wil15] die Kategorisierung von Problemen in P-schwer und P-vollständig untersucht. In [WW10] wurde gezeigt, dass viele graphentheoretische Probleme äquivalent sind bei Reduktionen mit unter-quadratischer Zeit. Gäbe es für eines dieser Probleme einen Algorithmus mit unter-quadratischer Zeit, dann wäre dies für alle anderen Probleme ebenfalls der Fall.

Anhand komplexitätstheoretischer Annahmen scheint es für viele Graphenprobleme unwahrscheinlich zu sein, dass diese schneller als in quadratischer Zeit berechnet werden können. Linearzeitalgorithmen sind dagegen für viele Probleme möglich, wenn die Graphen bounded Clique-width sind (siehe Satz von Courcelle [Cou90]).

Die Autoren zitieren die Autoren aus Abschnitt 3.8, dass maximalex Matching die „Drosophila“ der Studie der FPT-Algorithmen in P bekommen könnte. In [FLS+18] wurde ein randomisierter parametrisierter Algorithmus mit polynomieller Abhängigkeit von der Baumweite für maximales Matching vorgestellt.

Mithilfe von Approximationsalgorithmen sowie FPT-Algorithmen wird wie im NP-Fall nach Methoden gesucht, um die Existenz von FPT-Polynomialzeitalgorithmen zu belegen oder zu widerlegen.

Zur Beurteilung der parametrisierbarer Komplexität unterschiedlicher Graphenprobleme werden Eigenschaften wie Clique-width mit der oberen Schranke modular-width [CO00], split-width [Rao08], neighbourhood diversity [Lam10] und P_4-Spärlichkeit [BO99] verwendet.

Die Autoren sind der Meinung, dass die Entwicklung neuer Typeigenschaften notwendig ist, um verbesserte Algorithmen für Graphen mit bounded modular-width zu erhalten.

3.13 Schnelleres gewichtetes perfektes Matching in bipartiten planaren Graphen in $\widetilde{O}(|V|^{4/3})$

Graphklasse	Laufzeit	Neuerungen
$(\mathcal{BP} \cap \mathcal{PL})_w$	$O(\lvert V\rvert^{4/3} \cdot \log^2(\lvert V\rvert) \cdot \log(\lvert V\rvert \cdot C))$	Verbesserte Laufzeit

2019 stellten Mudabir Kabir Asathulla, Sanjeev Khanna, Nathaniel Lahn und Sharath Raghvendra in „A Faster Algorithm for Minimum-cost Bipartite Perfect Matching in Planar Graphs“ [AKLR19] einen schnelleren Algorithmus für das gewichtete perfekte Matching Problem in bipartiten planaren Graphen vor.

Nötige Grundlagen

Definition 3.13.1 (r-Division)**.** *Eine* r-Division *eines planaren Graphen* G *ist eine Menge* $R = R_1, ..., R_l$ *von* $l = O(n/r)$ *Teilgraphen, welche zusammen alle Kanten von* G *enthalten. Jeder Teilgraph wird* Partition *genannt. Jede Kante ist in mindestens einer Partition enthalten. Ein Knoten, der zu Kanten von mehr als einer Partition inzident ist, wird* grenzbildender Knoten *genannt und sonst* interner Knoten*.*

Es gelten weitere Eigenschaften für r*-Divisionen, die hier ausgelassen wurden.*

Algorithmus

Der Algorithmus für das gewichtete perfekte Matching Problem baut auf zwei vorhandene Algorithmen auf und kombiniert diese:

1. Der Teile-und-herrsche Algorithmus von Lipton und Tarjan [LT80], der auf dem Satz des planaren Separators basiert
2. Der zweite Algorithmus von Gabow und Tarjan [GT89] basiert auf dem Bit-Skalierung Paradigma. Jede Skalierung korrespondiert zu einem Bit des Kantengewichtes.

In jeder Skalierung führt der Algorithmus folgende Schritte aus:

- Führe $O(n^{1/3})$ Iterationen des Algorithmus von Gabow und Tarjan aus
- Konstruiere einen komprimierten residual Graphen H mit $O(n^{2/3})$ Knoten und $O(n)$ Kanten durch r-Division des planaren Graphen G mit $r = n^{2/3}$.
- Für jede der $O(r)$ Partitionen der r-Division existiert eine Kante zwischen zwei Knoten von H genau dann, wenn sie durch einen direkten Pfad in der Partition verbunden sind.
- Mithilfe existierender effizienter shortest-path Datenstrukturen werden die übrigen $O(n^{2/3})$ Kanten gematcht, indem wiederholt ein augmentierender Pfad mit minimalen Kosten berechnet wird.

Jede Skalierung hat eine Laufzeit von $\widetilde{O}(|V|^{4/3})$. Die totale Laufzeit beträgt $O(|V|^{4/3} \cdot \log^2(|V|) \cdot \log(|V| \cdot C))$, wobei C das maximale Gewicht einer Kante ist.

Bedeutung

Das perfekte Matching in bipartiten Graphen ist ein Spezialfall des Flussmaximierungsproblems mit mehreren Quellen und Senken, wobei alle Quellen und alle Senken dieselbe Menge produzieren / konsumieren. Bei allgemeinen Graphen kann das Problem durch Hinzufügen einer Gesamtquelle und Gesamtsenke auf das Flussmaximierungsproblem mit nur einer Quelle und einer Senke reduziert werden. Allerdings ist diese Reduktion bei planaren Graphen nicht möglich. Aus diesem Grund unterscheiden sich die Algorithmen für das Flussmaximierungsproblem mit mehreren Quellen und Senken für planare Graphen von denen für den Fall mit einer einzelnen Quelle und Senke.

Für das Flussmaximierungsproblem mit mehreren Quellen und Senken in ungewichteten planaren Graphen gibt es in dem Fall, dass bekannt ist, welche Menge an jeder Quelle und Senke produziert / konsumiert wird, einen Algorithmus, der einen gültigen Fluss in $O(n \cdot \log^1 n / \log\log n)$ berechnet [MW10]. Perfektes Matching in bipartiten Graphen ist davon ein Spezialfall, sodass dieser Algorithmus auch auf das perfekte Matching-Problem in bipartiten planaren Graphen angewedet werden kann.

Ein Algorithmus für das Flussmaximierungsproblem in planaren Graphen mit mehreren Quellen und Senken [BKM+17] kann auf das maximale Matching-Problem in bipartiten planaren Graphen angewendet werden.

Auf das gewichtete perfekte Matching-Problem für planare Graphen kann diese Reduktion jedoch nicht angewendet werden, da alle Algorithmen, die das Flussmaximierungsproblem unter Ausnutzung der planaren Eigenschaften des Graphen berechnen, den Zusammenhang vom maximalen Fluss in einem planaren Graphen und dem kürzesten Pfad in dessen Dual Graphen nutzen und es dazu keinen Zusammenhang mit der Berechnung minimaler Flüsse gibt.

Maximales Matching in allgemeinen Graphen kann außerdem mit dem Gaußschen Eliminationsverfahren berechnet werden. Der Algorithmus ist randomisiert und hat eine Laufzeit von $O(|V|^{\omega})$ [MS04] bzw. $O(|V|^{\omega/2})$ für planare Graphen [MS06]. ω ist hierbei der niedrigste Exponent für das Problem der Matrixmultiplikation.

Neben dem perfekten Matching-Problem in planaren bipartiten Graphen gibt es das zweidimensionale euklidische bipartite Matching-Problem, bei dem die Kantengewichte der euklidischen Distanz zwischen Knoten entsprechen, wenn diese auf eine Ebene abgebildet werden, und ein perfektes Matching mit minimaler Summe der Kantengewichte gesucht wird.

Wenn die Entfernungen zwischen Knoten aus ganzen Zahlen bestehen, so existiert eine Reduktion auf das gewichtete perfekte Matching-Problem in bipartiten planaren Graphen [Sha13].

3.14 Ein SPL-Algorithmus für das Entscheidungsproblem des perfekten Matchings in bipartiten Graphen mit logarithmischem Genus

Graphklasse	Komplexitätsklasse	Neuerungen
$\mathcal{BP}$	SPL	Effizientere Gewichtungsfunktion

Chetan Gupta, Vimal Raj Sharma und Raghunath Tewari stellten in der 2020 veröffentlichten Arbeit „Efficient Isolation of Perfect Matching in $O(\log n)$ Genus Bipartite Graphs“ [GST20] ein Verfahren zur Bestimmung eines perfekten Matchings auf einem bipartiten Graphen mit logarithmischem Genus vor, das in SPL berechnet werden kann.

Algorithmus

Sei G ein bipartiter Graph mit Genus g. Der Algorithmus gewichtet alle Kanten von G so, dass das gewichtete perfekte Matching des Graphen einzigartig wird.

Dafür wird aus G ein gerichteter Graph G' erzeugt. Die beiden Mengen der Bipartion von G werden L und R bezeichnet. Allen Kanten von L nach R wird eine Richtung zugewiesen. Die perfekten Matches für G' werden daraufhin aufgrund eines Entscheidungskriteriums in Klassen eingeteilt, wobei Matchings in der selben Klasse zueinander topologisch äquivalent sind.

Mithilfe einer Gewichtungsfunktion wird für jede der $22 \cdot g$ Klassen ein perfektes Matching isoliert. Anschließend werden durch ein Hashverfahren k Gewichtungsfunktionen gewählt. Für ein $1 \leq i \leq k$ isoliert die i-te Gewichtungsfunktion ein perfektes Matching mit minimalem Gewicht in G', welches gleichzeitig ein einzigartiges perfektes Matching mit minimalem Gewicht in G ist.

Bedeutung

Die Frage, ob das Problem auch für allgemeine Graphen in NC gelöst werden kann, ist noch nicht beantwortet worden. Es gilt: $\mathsf{SPL} \subset \mathsf{NC}^2$ [ARZ99] und $\mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{NC}^2 \subseteq \mathsf{NC}$ [Pap94].

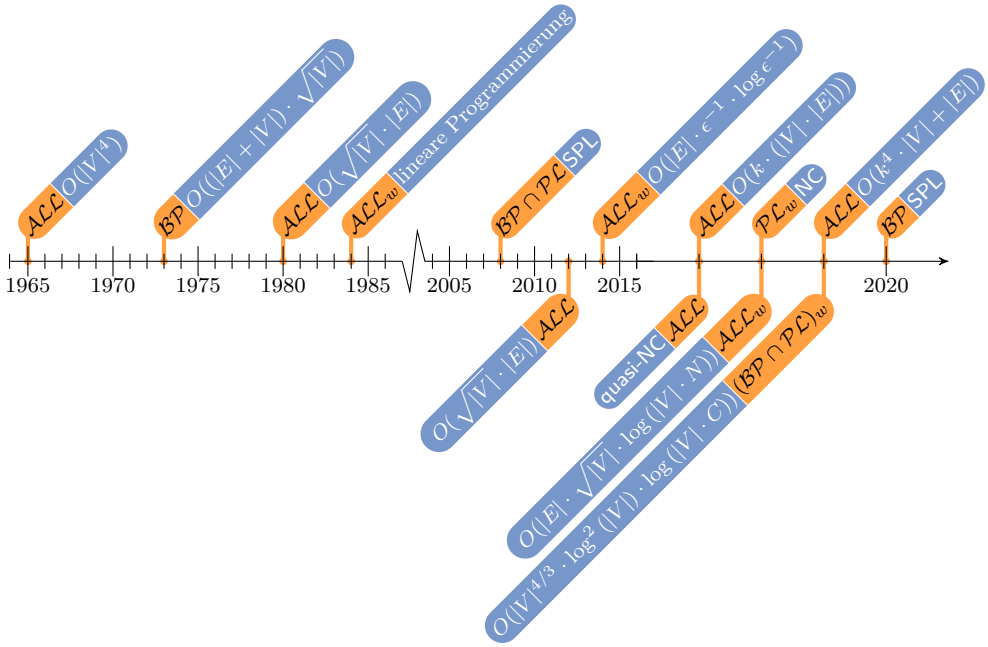
Die Konstruktion von isolierenden Gewichtungsfunktionen ist von Bedeutung für das Erreichbarkeitsproblem in gerichteten Graphen. Die logspace-Konstruktion einer polynomiell beschränkten pfadisolierenden Gewichtungsfunktion würde implizieren, dass das Erreichbarkeitsproblem in gerichteten Graphen in UL beechnet werden kann. Dies würde das NL-UL-Problem beantworten [RA00]. Die logspace-Konstruktion einer polynomiell beschränkten perfektes Matching isolierenden Gewichtungsfunktion würde beweisen, dass $\mathsf{NL} \subseteq \mathsf{SPL}$ [CSV84].

4 Zeitlicher Verlauf

Unterschiedliche Ansätze und Verbesserungen haben dazu geführt, dass sich die Laufzeit der Matching-Algorithmen von 1965 bis 2020 enorm verbessert hat:

- Für allgemeine Graphen von erstmalig $O(|V|^4)$ auf quasi-NC
- Für bipartite Graphen von $O(|V| \cdot |E|)$ auf SPL (bei $\log n$ Genus)
- Für planare Graphen gibt es einen Algorithmus in NC

Die folgende Grafik bietet eine Übersicht der hier vorgestellten Matching-Algorithmen. Diese werden in Abhängigkeit von der Zeit dargestellt. Dabei wird die Graphklasse in schwarz auf gelbem Hintergrund und die Laufzeit in weiß auf blauem Hintergrund angegeben. Der Übersichtlichkeit halber wurden die Intervalle von 2016 - 2020 länger dargestellt sowie die Zeitspanne zwischen 1985 und 2005, zu der keine Arbeiten hier vorgestellt wurden, abgekürzt.



Während zu Beginn der Entwicklung der Fokus der Algorithmen auf allgemeinen und bipartiten Graphen lag, liegt dieser bei aktuellen Algorithmen bei stärker eingeschränkten Graphklassen, wie bipartiten und vom Genus her beschränkten Graphen. Außerdem gelangen gewichtete Graphen im Laufe der Zeit verstärkt in den Fokus.

Zusammenfassend haben vor allem die folgenden Ideen Einfluss auf das Matching-Problem gehabt:

- Hopcroft-Karp-Algorithmus: Ziemlich früh erschien ein performanter Algorithmus, der die Struktur von bipartiten Graphen ausnutzt, um ein effizientes Verfahren zur Berechnung eines maximalen Matchings durch iteratives Augmentieren zu liefern.
- Algorithmus von Edmonds: Der erste Polynomialzeitalgorithmus für maximales Matching in allgemeinen Graphen, ermöglicht durch das Konzept der Blüten. Dieser Algorithmus wurde im Folgenden weiter optimiert.
- Isolationslemma: Durch das Isolationslemma wurde es möglich, das perfekte Matching in einem Graphen einzigartig zu machen, indem den Kanten unterschiedliche Gewichte zugewiesen werden und nach dem Matching mit der geringsten Summe der Kantengewichte gesucht wird. Selbst, wenn der Graph mehrere perfekte Matchings hat, beschränkt sich die Anzahl an Lösungen durch das Isolationslemma auf eins.
- Skalierende Algorithmen: Jeder Kante werden neben dem Gewicht weitere Parameter zugewiesen. Diese Parameter werden anhand bestimmter Kriterien bestimmt und in jeder Skalierung aktualisiert. Nach einer gewissen Anzahl an Skalierungen endet der Algorithmus mit einer Lösung.
- Gewichtungsfunktion: Vor allem in den neueren Algorithmen wird eine Gewichtungsfunktion verwendet, um Matching-Probleme zu lösen. Statt des ursprünglich randomisierten Isolationslemmas wurden deterministische Gewichtungsfunktionen gesucht, um die Komplexität das Problems von RNC auf quasi-NC zu verbessern.

Weitere Ansätze wie das lineare Optimierungsverfahren, das Datenreduktionsverfahren und die Aufteilung des Graphen in effizient lösbare Einheiten sind Verfahren, die zur weiteren Erkundung der Komplexitätstheorie geführt haben. Für bestimmte Graphentypen wie zum Beispiel einen Graphen, der aus zwei vollständigen Teilgraphen, die mit einem sehr langen Pfad verbunden sind, besteht, könnten diese Verfahren ein schnelleres Ergebnis liefern als die anderen Algorithmen. In der Anwendung kann es sein, dass ein Approximationsalgorithmus für das Matching-Problem ausreichend ist. In dem Fall eignet sich mitunter ein lineares Optimierungsverfahren.

5 Ausblick

An dem Matching-Problem wurde schon lange geforscht, mit Resultaten, die relevant für viele Bereiche der Komplexitätstheorie sind. Nicht umsonst wird es als die „Drosophila“ der Studie der FPT-Algorithmen in P bezeichnet [FLS+18]. Viele unterschiedlichen Nuancen des Matching-Problems, wie die Entscheidungsfrage nach einem perfekten Matching oder die Bestimmung eines gewichteten perfekten Matchings, erlauben unterschiedlich effiziente Algorithmen. Es ist zu erwarten, dass weitere abgewandelte Matching-Probleme und Graphen erfunden werden, die im Vergleich zu den herkömmlichen Problemen entscheidende Einschränkungen oder Vereinfachungen haben, sodass neue Algorithmen für diese optimiert werden können. Die Hoffnung besteht, dass sich andere (zumeist Graphen-) Probleme, die sich auf diese Probleme reduzieren lassen, schneller berechnen lassen.

Auf der anderen Seite ist der nächste Schritt beim Matching-Problem, neben für bipartite Graphen auch für allgemeine Graphen zu zeigen, dass das perfekte Matching-Entscheidungsproblem in NC liegt.

Allerdings scheinen die Matching-Algorithmen auf eine untere Schranke zu stoßen – eine deutliche Verbesserung der Laufzeit würde implizieren, dass einige andere Graphenprobleme wie zum Beispiel das Finden eines zweitkürzesten Pfades zwischen zwei Knoten in einem gewichteten Graphen ebenfalls eine bessere Laufzeit haben, indem diese auf das Matching-Problem reduziert werden [WW10]. Dies ist unwahrscheinlich, da das Lösen eines Problems durch Reduktion auf ein anderes Problem einen höheren Rechenaufwand darstellt als das direkte Lösen des Problems (der Reduktionsalgorithmus entfällt) und in dem Fall die effizientesten bekannten direkten Lösungsalgorithmen für alle verwandten Probleme eine schlechtere Laufzeit hätten als es bei einer Reduktion auf das Matching-Problem der Fall wäre.

Als ein besonderes Graphenproblem, das sich im Vergleich zu anderen Graphenproblemen, für die bisher ausschließlich NP-Algorithmen bekannt sind, im „unteren Teil“ von P berechnen lässt, ist Matching durchaus ein „powerful piece of algorithmic magic“ [Ski08].

Literaturverzeichnis

[AHT07] Manindra Agrawal, Thanh Minh Hoang, and Thomas Thierauf. The polynomially bounded perfect matching problem is in NC ^{2}. In Wolfgang Thomas and Pascal Weil, editors, *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*, volume 4393 of *Lecture Notes in Computer Science*, pages 489–499. Springer, 2007.

[AKLR19] Mudabir Kabir Asathulla, Sanjeev Khanna, Nathaniel Lahn, and Sharath Raghvendra. A faster algorithm for minimum-cost bipartite perfect matching in planar graphs. *ACM Trans. Algorithms*, 16(1):2:1–2:30, 2019.

[AOST92] Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P. Thacker. High speed switch scheduling for local area networks. In Barry Flahive and Richard L. Wexelblat, editors, *ASPLOS-V Proceedings - Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, Massachusetts, USA, October 12-15, 1992*, pages 98–110. ACM Press, 1992.

[ARZ99] Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting uniform and nonuniform upper bounds. *J. Comput. Syst. Sci.*, 59(2):164–181, 1999.

[Ber57] C. Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences of the United States of America*, 43(9):842–844, Sep 1957. 16590096[pmid].

[Ber84] Stuart J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Inf. Process. Lett.*, 18(3):147–150, 1984.

[BKM+17] Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *SIAM J. Comput.*, 46(4):1280–1303, 2017.

[BO99] Luitpold Babel and Stephan Olariu. On the p-connectedness of graphs - A survey. *Discret. Appl. Math.*, 95(1-3):11–33, 1999.

[BTV09] Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *TOCT*, 1(1):4:1–4:17, 2009.

[CDP19] David Coudert, Guillaume Ducoffe, and Alexandru Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. *ACM Trans. Algorithms*, 15(3):33:1–33:57, 2019.

[Cha96] Maw-Shang Chang. Algorithms for maximum matching and minimum fill-in on chordal bipartite graphs. In Tetsuo Asano, Yoshihide Igarashi, Hiroshi Nagamochi, Satoru Miyano, and Subhash Suri, editors, *Algorithms and Computation, 7th International Symposium, ISAAC '96, Osaka, Japan, December 16-18, 1996, Proceedings*, volume 1178 of *Lecture Notes in Computer Science*, pages 146–155. Springer, 1996.

[CL95] Ka Wong Chong and Tak Wah Lam. Finding connected components in o(log n log log n) time on the EREW PRAM. *J. Algorithms*, 18(3):378–402, 1995.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[CMSV17] Michael B. Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in õ ($m^{10/7}$ log W) time (extended abstract). In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 752–771. SIAM, 2017.

[CO00] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discret. Appl. Math.*, 101(1-3):77–114, 2000.

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

[CP80] Harlan Crowder and Manfred W. Padberg. Solving large-scale symmetric travelling salesman problems to optimality. *Management Science*, 26(5):495–509, 1980.

[Csa76] L. Csanky. Fast parallel matrix inversion algorithms. *SIAM J. Comput.*, 5(4):618–623, 1976.

[CSV84] Ashok K. Chandra, Larry J. Stockmeyer, and Uzi Vishkin. Constant depth reducibility. *SIAM J. Comput.*, 13(2):423–439, 1984.

[Die06] Reinhard Diestel. *Graphentheorie (3. Aufl.)*. Springer, 2006.

[DK98] Elias Dahlhaus and Marek Karpinski. Matching and multidimensional matching in chordal and strongly chordal graphs. *Discret. Appl. Math.*, 84(1-3):79–91, 1998.

[DKLM10] Samir Datta, Raghav Kulkarni, Nutan Limaye, and Meena Mahajan. Planarity, determinants, permanents, and (unique) matchings. *TOCT*, 1(3):10:1–10:20, 2010.

[DKR10] Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory Comput. Syst.*, 47(3):737–757, 2010.

[DP14] Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1):1:1–1:23, 2014.

[DPS18] Ran Duan, Seth Pettie, and Hsin-Hao Su. Scaling algorithms for weighted matching in general graphs. *ACM Trans. Algorithms*, 14(1):8:1–8:35, 2018.

[DS12] Ran Duan and Hsin-Hao Su. A scaling algorithm for maximum weight matching in bipartite graphs. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1413–1424. SIAM, 2012.

[DS18] Yann Disser and Martin Skutella. The simplex algorithm is np-mighty. *ACM Trans. Algorithms*, 15(1), November 2018.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[EK75] Shimon Even and Oded Kariv. An o(n^2.5) algorithm for maximum matching in general graphs. In *16th Annual Symposium on Foundations of Computer Science, Berkeley, California, USA, October 13-15, 1975*, pages 100–112. IEEE Computer Society, 1975.

[FF62] L. R. FORD and D. R. FULKERSON. *Flows in Networks*. Princeton University Press, 1962.

[FGT16] Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-nc. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 754–763. ACM, 2016.

[FLS+18] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Trans. Algorithms*, 14(3):34:1–34:45, 2018.

[Gab76] Harold N. Gabow. An efficient implementation of edmonds' algorithm for maximum matching on graphs. *J. ACM*, 23(2):221–234, 1976.

[Gab85] Harold N. Gabow. A scaling algorithm for weighted matching on general graphs. In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 90–100. IEEE Computer Society, 1985.

[GH85] Martin Grötschel and Olaf Holland. Solving matching problems with linear programming. *Math. Program.*, 33(3):243–259, 1985.

[GJR84] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32:1195–1220, 12 1984.

[GS62] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[GST20] Chetan Gupta, Vimal Raj Sharma, and Raghunath Tewari. Efficient isolation of perfect matching in o(log n) genus bipartite graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 27:25, 2020.

[GT89] Harold N. Gabow and Robert Endre Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989.

[GT91] Harold N. Gabow and Robert Endre Tarjan. Faster scaling algorithms for general graph-matching problems. *J. ACM*, 38(4):815–853, 1991.

[Hit41] Frank L. Hitchcock. The distribution of a product from several sources to numerous localities. *Journal of Mathematics and Physics*, 20(1-4):224–230, 1941.

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[HK12] Chien-Chung Huang and Telikepalli Kavitha. Efficient algorithms for maximum weight matchings in general graphs with small edge weights. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1400–1412. SIAM, 2012.

[HMU07] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley, 2007.

[JVV86] Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.

[Kar86] Howard J. Karloff. A las vegas RNC algorithm for maximum matching. *Combinatorica*, 6(4):387–391, 1986.

[Kas63] P. W. Kasteleyn. Dimer statistics and phase transitions. *Journal of Mathematical Physics*, 4(2):287–293, 1963.

[KM74] T. Kameda and J. Ian Munro. A o(|v|*|e|) algorithm for maximum matching of graphs. *Computing*, 12(1):91–98, 1974.

[Kuh55] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[KUW86] Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

[Lam10] Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. In Mark de Berg and Ulrich Meyer, editors, *Algorithms - ESA 2010, 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part I*, volume 6346 of *Lecture Notes in Computer Science*, pages 549–560. Springer, 2010.

[Lov79] László Lovász. On determinants, matchings, and random algorithms. In Lothar Budach, editor, *Fundamentals of Computation Theory, FCT 1979, Proceedings of the Conference on Algebraic, Arthmetic, and Categorial Methods in Computation Theory, Berlin/Wendisch-Rietz, Germany, September 17-21, 1979*, pages 565–574. Akademie-Verlag, Berlin, 1979.

[LT80] Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

[Lub86] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986.

[Mat78] David W. Matula. Subtree isomorphism in o(n5/2). In B. Alspach, P. Hell, and D.J. Miller, editors, *Algorithmic Aspects of Combinatorics*, volume 2 of *Annals of Discrete Mathematics*, pages 91 – 106. Elsevier, 1978.

[McK99] Nick McKeown. The islip scheduling algorithm for input-queued switches. *IEEE/ACM Trans. Netw.*, 7(2):188–201, 1999.

[Mil86] Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. Syst. Sci.*, 32(3):265–279, 1986.

[MNN17] George B. Mertzios, André Nichterlein, and Rolf Niedermeier. The power of linear-time data reduction for maximum matching. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, volume 83 of *LIPIcs*, pages 46:1–46:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[MR89] Gary L. Miller and John H. Reif. Parallel tree contraction part 1: Fundamentals. *Advances in Computing Research*, 5:47–72, 1989.

[MS04] Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 248–255. IEEE Computer Society, 2004.

[MS06] Marcin Mucha and Piotr Sankowski. Maximum matchings in planar graphs via gaussian elimination. *Algorithmica*, 45(1):3–20, 2006.

[MV80] S. Micali and V. V. Vazirani. An o(v|v| c |e|) algoithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*, pages 17–27, 1980.

[MV00] Meena Mahajan and Kasturi R. Varadarajan. A new nc-algorithm for finding a perfect matching in bipartite planar and small genus graphs (extended abstract). In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 351–357. ACM, 2000.

[MVV87] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[MW10] Shay Mozes and Christian Wulff-Nilsen. Shortest paths in planar graphs with real lengths in $O(n\log^2 n/\log\log n)$ time. In Mark de Berg and Ulrich Meyer, editors, *Algorithms - ESA 2010, 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part II*, volume 6347 of *Lecture Notes in Computer Science*, pages 206–217. Springer, 2010.

[NR59] R. Z. Norman and Michael O. Rabin. An algorithm for a minimum cover of a graph. 1959.

[oS20] The Royal Swedish Academy of Sciences. *Stable matching: Theory, evidence, and practical design*, 2012 (accessed Mai 28, 2020). `https://www.nobelprize.org/uploads/2018/06/popular-economicsciences2012.pdf`.

[Pap94] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[Pet12] Seth Pettie. A simple reduction from maximum weight matching to maximum cardinality matching. *Inf. Process. Lett.*, 112(23):893–898, 2012.

[PY82] Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *J. ACM*, 29(2):285–309, 1982.

[RA00] Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000.

[Rao08] Michaël Rao. Solving some np-complete problems using split decomposition. *Discret. Appl. Math.*, 156(14):2768–2780, 2008.

[San07] Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 118–126. SIAM, 2007.

[San18] Piotr Sankowski. NC algorithms for weighted planar perfect matching and related problems. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 97:1–97:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[Sha13] R. Sharathkumar. A sub-quadratic algorithm for bipartite matching of planar points with bounded integer coordinates. In Guilherme Dias da Fonseca, Thomas Lewiner, Luis Mariano Peñaranda, Timothy M. Chan, and Rolf Klein, editors, *Symposuim on Computational Geometry 2013, SoCG '13, Rio de Janeiro, Brazil, June 17-20, 2013*, pages 9–16. ACM, 2013.

[Ski08] Steven Skiena. *The Algorithm Design Manual, Second Edition*. Springer, 2008.

[ST17] Ola Svensson and Jakub Tarnawski. The matching problem in general graphs is in quasi-nc. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 696–707. IEEE Computer Society, 2017.

[SV82] Yossi Shiloach and Uzi Vishkin. An o(log n) parallel connectivity algorithm. *J. Algorithms*, 3(1):57–67, 1982.

[SY96] G. Steiner and J.S. Yeomans. A linear time algorithm for maximum matchings in convex, bipartite graphs. *Computers & Mathematics with Applications*, 31(12):91 – 96, 1996.

[TCHP08] Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645. Springer, 2008.

[Tut47] W. T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, s1-22(2):107–111, 1947.

[TV84] Robert Endre Tarjan and Uzi Vishkin. Finding biconnected components and computing tree functions in logarithmic parallel time (extended summary). In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*, pages 12–20. IEEE Computer Society, 1984.

[Val79] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[Vaz12] Vijay V. Vazirani. A simplification of the mv matching algorithm and its proof, 2012.

[Wag70] Klaus Wagner. *Graphentheorie*. Bibliographisches Institut Hochschultaschenbücher, Mannheim, 1970. Satz 9.9.

[Wil15] Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In Thore Husfeldt and Iyad A. Kanj, editors, *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, volume 43 of *LIPIcs*, pages 17–29. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[WW10] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 645–654. IEEE Computer Society, 2010.