Methoden der Standortplanung und Logistik mit Fallbeispielen

- Vorlesungsskript -

Dr. C. Mauckner
Konzernlogistik
Volkswagen AG, Wolfsburg

Prof. Dr. H. Noltemeier
Lehrstuhl für Informatik I
Universität Würzburg

Dipl. Inform T. Samstag
Konzernlogistik
Volkswagen AG, Wolfsburg

WS 1997/98

Inhaltsverzeichnis

Teil I

Einleitung

0.1 Logistik

- Herkunft des Begriffs „Logistik" aus dem militärischen Bereich: die Versorgung der Truppe mit Verpflegung, Munition, Abtransport Verwundeter; militärisches Nachschubwesen. In den 60er Jahren Übernahme des Begriffs auf den zivilen Bereich.

- Logistik hat lange Tradition: Beispielhaft der Transport von Steinquader auf dem Nil beim Bau der Pyramiden im alten Ägypten oder der Handelsweg Seidenstraße von China nach Europa.
  Gemeinsames Merkmal: Organisation einer Raum- und Zeitüberbrückung für diverse Rohstoffe und Handelswaren.

- Logistik ist die wissenschaftliche Lehre der Planung, Steuerung und Überwachung der Material-, Personen-, Energie- und Informationsflüsse in Systemen.
  Logistik umfaßt alle Tätigkeiten, durch die Bewegungs- und Speichervorgänge in einem Netzwerk gestaltet, gesteuert und kontrolliert werden. Durch das Zusammenspiel dieser Tätigkeiten soll ein Strom von Objekten durch das Netzwerk in Gang gesetzt werden, so daß Räume und Zeit effektiv überbrückt werden.
  Logistik als Summe aller Tätigkeiten zur Vorbereitung und Realisierung der Stoffflüsse sowie der dazugehörigen Informationsflüsse hat das Ziel, entsprechend den Bedürfnissen der Märkte

  - das richtige Gut,
  - zur richtigen Zeit,
  - am richtigen Ort,
  - in der richtigen Menge,
  - mit der richtigen Qualität,
  - mit den dazugehörigen Informationen
  - unter Berücksichtigung der Logistikkosten

  bereitzustellen.

- Die „Querschnittsfunktion" der Logistik:
  Aufgliederung der logistischen Prozesse innerhalb eines Unternehmens nach funktionellen Gesichtspunkten:

  - **Beschaffungslogistik** (Tätigkeiten im Zusammenhang mit der Beschaffung von Roh-, Hilfs- und Betriebsstoffen sowie Zukaufteilen)
  - **Produktionslogistik** (Tätigkeiten im Zusammenhang mit dem Material- und dem dazugehörigen Informationsfluß im Produktionsprozeß)
  - **Distributionslogistik** ((Tätigkeiten im Zusammenhang mit dem Warenfluß vom Hersteller bis zum Kunden)
  - **Entsorgungslogistik** (Tätigkeiten im Zusammenhang mit der Entsorgung und dem Recycling von Produkten und Verpackungen)

- Die drei Säulen der Logistik

LOGISTIK		
Technik	**Ökonomie**	**OR/Informatik**
Transport- und Lagersysteme Verladetechnik Behältersysteme/ Verpackungstechnik ...	Anlieferungs-strategien z.B. Just in Time(JIT) Qualitätsmanagement Unternehmens-strategieplanung ...	Modellierungs- und Optimierungswerk-zeuge Simulationssysteme Informationssyteme und Datenbanken Robotik (Fahrerlose Transportsysteme) ...

Abbildung 1: Die drei Säulen der Logistikforschung

- Logistische Prozeßkette
- Ziel der Logistik ist die Maximierung der Logistikleistung unter Minimierung der Logistikkosten. (um Wettbewerbsvorteile für das Gesamtunternehmen zu erlangen): Logistikkosten ($>$ 10 Prozent an den Gesamtkosten)
  - Steuerungs- (Disposition, Auftragsabwicklung, Fertigungssteuerung)und Systemkosten (Gestaltung, Planung und Kontrolle des Materialflusses)
  - Bestandskosten (Kapitalbindungskosten, Versicherungen, Abwertung, Verluste)
  - Lagerkosten (Fixer Teil: Bereithalten der Lagerkapazitäten, Variabler Teil: Ein- und Auslagerungsvorgänge)
  - Transportkosten (Interner Werksverkehr, Zwischenwerksverkehr)
  - Handlingskosten (Verpackung, Kommissionieren)

  Logistikleistung

  - Lieferzeit (Kurze Durchlaufzeiten der Kundenaufträge, gemessen vom Auftragseingang bis zur Verfügbarkeit beim Kunden. $\rightarrow$ niedrige Lagerbestände beim Kunden, kurzfristige Disposition)
  - Lieferzuverlässigkeit (Liefertreue, Termintreue, Wahrscheinlichkeit, daß die Lieferzeit eingehalten wird $\rightarrow$ sonst Störungen im Betriebsablauf beim Kunden, damit hohe Kosten)

  - Lieferflexibilität (Fähigkeit des Auslieferungssystem auf besondere Kundenwünsche einzugehen wie Modalitäten der Auftragserteilung (Abnahmemengen, Art der Auftragsübermittlung) und Liefermodalitäten(Verpackung oder Transportvarianten))
  - Lieferqualität (Liefergenauigkeit nach Art und Menge, Zustand der Lieferung $\rightarrow$ Verärgerung des Kunden, Reklamationskosten, Preisabschläge)
  - Infomationsfähigkeit (Möglichkeit, Kundenanfagen vor und nach der Auftragserteilung schnell und genau beantworten zu können)

  Trade-off zwischen Logistikkosten und Logistikleistung.

- Zielkonflikte durch isolierte Optimierung von Abteilungs- und Bereichszielen.
  - Beschaffung möchte Teile möglichst günstig einkaufen. Logistik möchte Transport von Lieferanten zum Verbrauchsort minimiert wissen.
  - Produktion möchte hohe Kapazitätsauslastung der Maschinen $\rightarrow$ In der Regel Erhöhung der Durchlaufzeit für Aufträge.

0.2 Standortplanung

- Standortplanung ist ein entscheidenter Faktor für den Erfolg eines Unternehmens:
  - Gute Standorte sichern Wettbewerbsvorteile gegenüber Konkurrenten.
  - Realisierte Entscheidungen sind irreversibel oder nur kostenintensiv und langfristig zu korrigieren.
- Teilgebiete der Standortplanung
  - Volkswirtschaftliche StP(Versucht Ansiedlung der Unternehmen und Betriebe von Wirtschaftssektoren im Raum innerhalb eines Wirtschaftsgebietes zu erklären oder zu optimieren.)
  - Betriebliche StP(Behandelt Fragen der Standortwahl einzelner Betriebe oder auch öffentlicher Einrichtungen wie Schulen, Krankenhäuser, Feuerwehrstationen, Mülldeponien, usw.)
  - Innerbetriebliche StP/ Layoutplanung (Planung der räumlichen Anordnung von Betriebsmitteln innerhalb der Grenze des Betriebsgrundstücks)
- Betriebliche Standortplanung:
  - Neugründung eines Unternehmens (Bsp. Raider, einer der größten Logistikdienstleister der USA möchte auch in Europa Fuß fassen. Wieviele Filialen soll er errichten und wo?))
  - Erweiterung eines Unternehmens (Abhängig von gegebenen Standorten)

- Ausbau/Abbau der Kapazitäten eines Unternehmens. (welcher Standort? Schließung?, Bis zur welcher Kapazität lohnt sich der Ausbau/ Wann lohnt sich Erweiterung?)

- Lösungsvorschläge
  - Deskriptive Standorttheorie
    Entwicklung eines begrifflichen Instrumentariums zur allgemeingültigen Beschreibung der Prämissen und Abläufe von Standortentscheidungsprozessen. Qualitative Bewertung von Standortfaktoren wie Wohn- und Lebensqualität, Verkehrsanbindung, qualifizierte Arbeitskräfte, ...
  - Normative, präskriptive Standorttheorie
    Entwicklung intersubjektiver nachprüfbarer Kriterien (Modelle, Lösungverfahren), mit deren Hilfe in einer konkreten Planungssituation eine Standortentscheidung getroffen werden kann. (OR-Modelle)

0.3 Operations Research

- Heutige Organisationen (Unternehmen, Armeen, Krankenhäuser) besitzen eine komplexe Struktur. $\rightarrow$ Wie sollen Ressourcen auf einzelne Aktivitäten verteilt werden, so daß es für die Organisation am effektivsten ist?

- Anfänge: Im 2.Weltkrieg haben britische und amerikanische Wissenschaftler versucht die Zusammenstellung von Schiff-Convoys auf dem Nordatlantik zu optimieren, um diese vor deutschen U-Boot-Angriffen zu schützen. Viele dieser in OR-Teams zusammengeschlossenen Spezialisten gingen nach dem Krieg in die Wirtschaft, wurden z.B. Unternehmensberater und erkannten, daß die gleichen Probleme auch in Industriebetrieben nur in einem anderen Kontext auftreten.

- Starker Aufschwung durch Entwicklung von leistungsfähigen Rechnern (technischer Sicht) und Simplexalgorithmus durch 1947 Dantzig (Lösungsverfahren)

- Weitere Bezeichnungen für OR im deutschen Sprachraum: Unternehmensforschung, mathematische Planungsrechnung, Operationsforschung, Optimierungsrechung.

- Def: Wissenschaftliche Methodik der Entscheidungsfindung, die sich mit den Operationen organisierter Systeme befaßt.

- OR nimmt einen interdisziplinären Standort ein: Mathematik, Statistik, VWL, BWL, Informatik, Technik, Physik, Verhaltenswissenschaften. Es war lange umstritten, ob OR eine eigene Wissenschaft ist oder nicht.

- „Streben nach Optimalität“

- Der OR-Planungsprozeß

Erkennen und Analysieren eines Problems

Bestimmen von Zielen und Handlungsmöglichkeiten

Deskriptives Modell

Mathematisches Modell

Datenbeschaffung

Lösungsfindung

Bewertung der Lösung

- **Modelle im OR**
  Ein Modell ist ein vereinfachtes (isomorphes oder homomorphes) Abbild eines realen Systems oder Problems.
  - **Entscheidungs- bzw. Optimierungsmodell** Formale Darstellung eines Entscheidungs- oder Plannungsproblems, das in seiner einfachsten Form mindestens eine Alternativenmenge und eine diese bewertende Zielfunktion enthält. (z.B. Knapsack)
  - **Simulationsmodell** Häufig komlexe Optimierungsmodelle, für die keine analytischen Lösungsverfahren existieren. Sie dienen dem Zweck, Konsequenzen einzelner Alternativen zu bestimmen (zu untersuchen, „durchzuspielen"). (Komplexe Warteschlangennetze)
  - **Beschreibungsmodell** Zur Beschreibung der Elemente und deren Beziehungen in realen Sytemen. Sie enthalten keine Hypothesen über reale Wirkungszu-

sammenhänge und erlauben daher keine Erklärung oder Prognose realer Vorgänge. (Organigramme = Strukturbeschreibung eines Unternehmens)

  - **Erklärungsmodelle** werten empirische Gesetzmäßigkeiten oder Hypothesen zur Erklärung von Sachverhalten aus. (Modelle zur Beschreibung der Funktionsweise von Volkswirtschaften)
  - **Prognosemodell** Spezielle Erklärungsmodelle zur Vorhersage von zukünftigen Entwicklungen. (Wettervorhersagemodelle, Statistische Modelle zur Bedarfsprognose)

- **Teilgebiete des OR**
  - Lineare Optimierung (z.B. Fertigungs-, Transportplanung) → Simplexalgorithmus
  - Graphentheorie (z.B. Wegoptimierung) und Netzplantechnik (z.B. Prozeß- und Projektplanung)
  - Ganzzalige (lineare) und kombinatorische Optimierung (z.B. Zuordungs-, Reihenfolge-Gruppierungs- und Auswahlprobleme)
  - Dynamische Optimierung (z.B. Losgrößenplanung)
  - Nichtlineare Optimierung(z.B. Berücksichtigung nichtlinearer Zusammenhänge)
  - Warteschlangentheorie (z.B. Untersuchung von Produktionsdurchlaufzeiten) → Leistungsbewertung verteilter Systeme
  - Simulation (Auswertung stochastischer Netzpläne) → Simulationstechnik
  - Spieletheorie (z.B. Untersuchung kompetetiver Entscheidungszenarien) → Informationsstrukturen und wissensbasierte Systeme

Teil II

Transportprobleme

Kapitel 1

Kürzeste Wege

1.1 Anwendungen

- Direkt: Suche einer kürzesten oder schnellsten Fahrstrecke zwischen zwei Orten auf einer Straßenkarte (Map&Guide, Autoroute, ...).
- Indirekt: Subproblem in vielen Netzwerkoptimierungsproblemen, z.B. bei der Bestimmung kostenminimaler Flüsse.

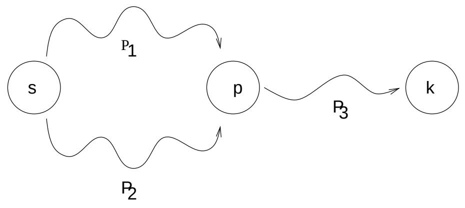
1.2 Problemdefinition

- Gegeben sein ein gerichteter Graph $G = (N, A)$ mit $n := |N|$ und $m = |A|$.
  Jede Kante $(i, j) \in A$ ist mit einer ganzzahligen Länge c_{ij} bewertet.
  $(C := \max\{c_{ij} | (i, j) \in A\})$
- $A(i)$ sei die Menge der von Knoten $i \in N$ ausgehenden Kanten.
- Die Länge eines Weges in G sei die Summe aller Kantenlängen.
- Annahmen: G enthalte keinen negativen Kreis. (Problem wird sonst NP-vollständig!)
- Problem: Bestimme die kürzesten Wege von einem ausgezeichneten Knoten s zu jedem erreichbaren Knoten aus $N - \{s\}$.

1.3 Allgemeine Eigenschaften

Baumeigenschaft:
Falls $s = i_1 - i_2 - \ldots - i_h = k$ der kürzeste Weg von s zu k ist, dann ist für jedes $q = 2, 3, \ldots, h - 1$ der Teilweg $s = i_1 - i_2 - \ldots - i_q$ ein kürzester Weg von s zu i_q. Jede kürzeste Wege-Lösung ist also ein Outtree mit Wurzel s.

Beweis:



Annahme: $P_1 - P_3$ sei kürzester Weg von s zu k, aber P_1 sei $kein$ kürzester Weg von s zu p. Dagegen sei P_2 solch ein kürzester Weg.
Daraus folgt, daß Weg $P_1 - P_3$ länger ist als $P_2 - P_3$. Widerspruch zur Annahme! $\square$

Optimalitätsbedingungen:
Sei $d(i)$ die Länge des kürzesten Weges zum Knoten i. Dann ist ein Weg P von s zu k genau dann ein kürzester Weg, falls $d(j) = d(i) + c_{ij}$ für jede Kante $(i, j) \in P$ gilt.

Beweis:
„ $\Longrightarrow$" Folgt aus Baumeigenschaft.
„ $\Longleftarrow$"
P sei ein beliebiger Pfad zwischen s und k mit Knotenfolge $s = i_1 - i_2 - \ldots - i_h = k$, wobei gilt:

$$d(k) = d(i_{h-1}) + c_{i_{h-1}, i_h} = \sum_{(i,j) \in P} c_{ij}$$

Also ist P kürzester Weg da seine Länge mit $d(k)$ übereinstimmt. $\square$

Satz: In kreisfreien gerichteten Graphen ist das kürzeste Wegeproblem in $\mathcal{O}(m)$ lösbar.
Beweisidee: Topologisches Sortieren!

1.4 Label Setting-Verfahren

- Basis: Dijkstra's Algorithmus (1959)
- Einschränkung: Keine Kantenlänge darf negativ sein. ($c_{ij} \geq 0$)
- Knotenmenge N wird in zwei disjunkte Mengen zerlegt:
  - S: Menge der permanent markierten Knoten
  - $\overline{S}$: Menge der temporär markierten Knoten
- Für alle Knoten aus $\overline{S}$ stellt $d(\cdot)$ eine obere Schranke für den kürzesten Weg dar. Für Knoten aus S ist $d(\cdot)$ gleich dem kürzesten Weg.
- In jeder Iteration wandert ein Knoten aus $\overline{S}$ nach S.
- Der Outtree aller kürzester Wege wird durch das Speichern des Vorgängerknotens verwaltet.

algorithm Dijkstra;
begin
    $S := \emptyset; \overline{S} := N$;
    $d(i) := \infty$ for each node $i \in N$;
    $d(s) := 0$ and pred$(s) := 0$;
    while $|S| < n$ **do**
    begin
        let $u \in \overline{S}$ be a node for which $d(i) = \min\{d(j) : j \in \overline{S}\}$;
        $S := S \cup i$;
        $\overline{S} := \overline{S} - i$;
        for each $(i, j) \in A(i)$ **do**
            if $d(j) > d(i) + c_{ij}$ **then** $d(j) := d(i) + c_{ij}$ and pred$(j) := i$;
    end
end;

Satz: Der Dijkstra-Algorithmus terminiert mit der korrekten Lösung.

Beweis: Induktion über $|S|$
Induktionshypothesen: $N = S \cup \overline{S}$

I, $d(i)$ optimal für alle $i \in S$.

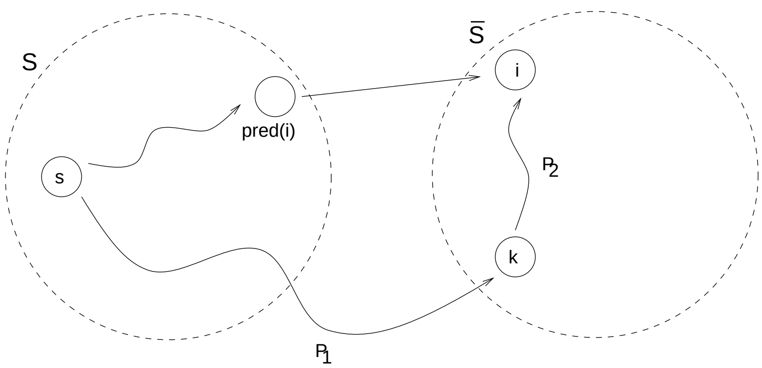
II, Für $i \in \overline{S}$ ist $d(i)$ die Länge des kürzesten Weges von s zu i, bei dem nur Knoten aus S im inneren des Weges enthalten sind.

Induktionsbegin: Nach der ersten Iteration gilt $S = \{s\}$ und $\overline{S} = N - \{s\}$.

I, $d(s) = 0$

II, erfüllt.

Induktionsschritt: $S \longrightarrow S \cup \{i\}$ wobei $d(i) = \min\{d(j) : j \in \overline{S}\}$.



I, Zu zeigen: $d(i)$ ist optimal.
$d(i)$ ist die Länge des kürzesten Weges von s nach i unter allen Wegen deren Knoten

in S liegen.
Wähle nun beliebigen Weg P von s nach i mit Knoten aus $S \cup \overline{S}$. $P = P_1 + P_2$, wobei P_1 von s zu k verläuft und nur innere Knoten aus S enthält.
Dann gilt: $d(k) \geq d(i) \Rightarrow |W_1| \geq d(k) \geq d(i)$ und da $c_{ij} \geq 0$ muß auch $|W_2| \geq 0$ erfüllt sein. Insgesamt ist also $|W| \geq d(i)$.

II, Durch die Update-Operation bei Knoten i können sich $d(j)$ für $j \in \overline{S}$ verringern. Da $pred(j) := i$ und $d(j) := d(i) + c_{ij}$ ist nach den *Optimalitätsbedingungen* auch diese Induktionshypothese erfüllt. □

Satz: Der Dijkstra-Algorithmus besitzt eine Laufzeit von $\mathcal{O}(n^2)$.

Beweis:

- In jeder Iteration wird ein Knoten aus $\overline{S}$ mit minimalem $d(i)$ gewählt. Aufwand: $n + (n-1) + \ldots + 1 = \mathcal{O}(n^2)$
- Der Update-Aufwand bei Knoten i beträgt $|A(i)|$. Gesamtaufwand aller Updates: $\sum_{i \in N} |A(i)| = \mathcal{O}(m)$.
- Gesamtaufwand für Algorithmus: $\mathcal{O}(m + n^2) = \mathcal{O}(n^2)$. □

Effiziente Implementierung

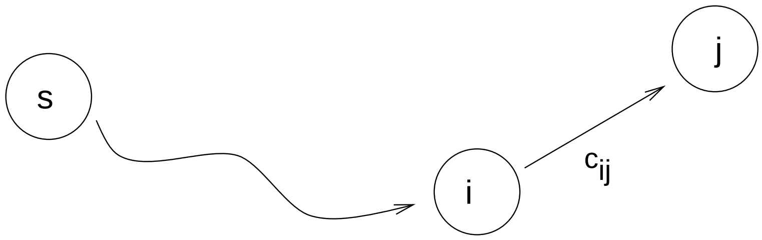
- Verwaltung der temporär markierten Knoten mit Prioritätswarteschlange(Heap) H. H speichert Objekte, die mit einer reellwertigen Zahl (Schlüssel) verknüpft sind. Folgende Operationen sind möglich:
  - *create-heap(H)*: Erstellt einen Heap.
  - *find-min(i, H)*: Liefert Objekt i mit kleinsten Schlüssel.
  - *insert(i, H)*: Fügt Objekt i mit vordefinierten Schlüssel ein.
  - *decrease-key(value, i, H)*: Ersetzt alten Schlüsselwert von i durch $value$. Dieser muß jedoch kleiner sein!
  - *delete-min(i, H)*: Löscht Element i mit minimalen Schlüssel.
- Fibonacci-Heaps: Jede Operation ist in $\mathcal{O}(1)$ amortisierter Zeit möglich. Ausnahme: *delete-min(i, H)* benötigt $\mathcal{O}(\log n)$.
- Auswahloperation benötigt somit nur $\mathcal{O}(\log(|\overline{S}|))$ Aufwand für jeden Iterationsschritt.
- Gesamtaufwand: $\mathcal{O}(n \log n + m)$. Beste streng-polynomiale Schranke für das kürzeste Wege-Problem mit nichtnegativen Kantenlängen.

1.5 Label Correcting-Verfahren

- Bellman-Ford-Algorithmus (1956/1958)
- Kantenlängen beliebig (jedoch kein negativer Kreis).
- Distanzmarken werden solange modifiziert, bis sie Optimalitätsbedingung erfüllen.

Kürzeste Wege Optimalitätsbedingungen
Die Distanzmarkierungen $d(\cdot)$ der Knoten repräsentieren genau dann die Längen der kürzesten Wege, wenn gilt: $d(j) \leq d(i) + c_{ij}$ für alle $(i, j) \in A$.

Beweis:
„ $\Longrightarrow$"



$d(\cdot)$ seinen kürzeste Wegemarkierungen und es existiere eine Kante $(i, j) \in A$
mit $d(j) > d(i) + c_{ij}$.
Die Konkatenation des kürzesten Weges von s nach i mit der Kante (i, j) wäre kürzer als $d(j)$. Widerspruch zur Voraussetzung!

„ $\Longleftarrow$"
Angenommen die Optimalitätsbedingungen wären erfüllt und $s = i_1 - i_2 - \ldots - i_k = j$ ware ein beliebiger Weg P von s zu j. Dann gilt:

$$d(j) = d(i_k) \leq d(i_{k-1}) + c_{i_{k-1},i_k} \leq \ldots \leq \sum_{(i,j)\in P} c_{ij}$$

Also ist $d(j)$ eine untere Schranke für die Länge jedes Pfades. Da $d(j)$ von mindestens einem Weg angenommen wird, ist es die Länge des kürzesten Weges. $\square$

Definition: *Reduzierte Kosten*: $c_{ij}^d = c_{ij} + d(i) - d(j)$.
Die Optimalitätsbedingungen lassen sich nun auch folgendermaßen beschreiben: $c_{ij}^d \geq 0$ für alle $(i, j) \in A$.

```
algorithm label-correcting;
begin
    d(s) := 0 and pred(s) := 0;
    d(j) := ∞ for each j ∈ N − {s};
    while some arc (i, j) satisfies d(j) > d(i) + c_ij do
    begin
        d(j) := d(i) + c_ij;
        pred(j) := i;
    end;
end;
```

Satz: Der Label Correcting-Algorithmus besitzt eine Laufzeit von $\mathcal{O}(mn^2C)$

Beweis: Jede Distanzmarke besitzt einen Wert zwischen $-(n-1)C$ und $+(n-1)C$ da jeder kürzeste Weg maximal $n-1$ Kanten entält. Jeder Update erhöht eine Distanzmarke um mindestens eins. Bei n Distanzmarken ergibt das höchstens $\mathcal{O}(n^2C)$ Iterationen. Die Suche einer, die Optimalitätsbedingungen verletzende Kante, benötigt hier $\mathcal{O}(m)$ Operationen. □

Idee: Intelligentere Kantenbestimmung

- Verwaltung einer Liste, die alle Kanten enthält, die potentiell die Kürzeste Wege Optimalitätsbedingungen verletzen könnten.
- Wenn die Liste leer wird, terminiert das Verfahren. Ansonsten entnehmen wir eine Kante (i, j) aus der Liste und falls sie eine Optimalitätsbedingung verletzt, fügen wir alle von j ausgehenden Kanten hinzu. Durch das Update von j können diese ggf. eine Optimalitätsbedingungen verletzen.
- Effizienter ist es nur die upgedateten Knoten zu speichern.
- Die Liste wird mit s initialisiert.

```
algorithm modified label-correcting;
begin
    d(s) := 0 and pred(s) := 0;
    d(j) := ∞ for each j ∈ N − {s};
    LIST := {s};
    while LIST ≠ do
    begin
        remove an element i from LIST;
        for each arc (i, j) ∈ A(i) do
        if d(j) > d(i) + c_ij then
        begin
            d(j) := d(i) + c_ij;
            pred(j) := i;
            if  j ∉ LIST then add node j to LIST;
        end;
    end;
end;
```

Satz Die Laufzeit des modifizierten Label Correcting-Algorithmus ist $\mathcal{O}(nmC)$

Beweis: Jede Marke kann höchstens $(2nC)$ mal upgedatet werden. Bei jedem Update wechselt der Knoten in die Liste und bei der späteren Entnahme wird seine Kantenliste durchlaufen. $\sum_{i \in N} |A(i)| 2nC = \mathcal{O}(nmC)$. □

Erweiterung zu einem polynomialen Verfahren

- Organisiert man die List nach dem FIFO-Prinzip, so läßt sich zeigen, daß der Algorithmus eine Laufzeit von $\mathcal{O}(nm)$ besitzt. Beweis per Induktion über die Anzahl der Durchläufe die benötigt werden, um alle Kanten, die potentiell die Kürzeste Wege Optimalitätsbedingungen verletzen können, zu untersuchen.

- Bester streng polynomialer Algorithmus zur Berechnung kürzester Wege mit bliebigen Kantenbewertungen.

- Man kann leicht die Existenz negativer Kreise entscheiden. Falls eine Distanzmarke kleiner als $-nC$ wird, so muß ein negativer Kreis vorhanden sein.

1.6 Das All-Pairs Shortest Path Problem

- Aufgabe: Bestimme die kürzesten Wege zwischen allen Knotenpaaren in G.

- Bei nichtnegativen Kanten: Wiederholte Anwendung von Dijkstra's Algorithmus. Laufzeit: $\mathcal{O}(nS(n, m, C))$. $(S(n, m, C)$ ist die Laufzeit des implementierten kürzeste Wege-Verfahrens.)

- Allgemein: Lösen des ersten einfachen kürzesten Wegeproblems mit einem Label-Correcting-Verfahren. Danach Transformation zu reduzierten Kantenkosten (diese sind nichtnegativ!). Denn für jeden Weg P von k nach l gilt:

$$\sum_{(i,j)\in P} c^d_{ij} = \sum_{(i,j)\in P} c_{ij} + d(k) - d(l)$$

  Wiederholte Anwendung von Dijkstra's Algorithmus auf reduzierte Kosten. Laufzeit: $\mathcal{O}(nm + nS(n,m,C)) = \mathcal{O}(nS(n,m,C))$

- Für dichte Graphen ist eine Generalisierung des Label Correcting-Verfahrens effizienter.

- $d[i,k]$ sei die Länge des kürzesten Weges zwischen Knoten i und k.

- pred$[i,k]$ sei der Vorgängerknoten von Knoten k im kürzesten Weg von i nach k.

All-Pairs Shortest Path Optimalitätsbedingungen:
Die Distanzmarken repräsentieren genau dann die Längen der kürzesten Wege, wenn gilt: $d[i,j] \leq d[i,k] + d[k,j]$ für alle $i,j,k \in N$.

Beweis: analog zum einfachen kürzesten Wege-Problem. □

```
algorithm all-pairs label-correcting;
begin
    set d[i,j] := ∞ for all [i,j] ∈ N × N;
    set d[i,j] := 0 for all i ∈ N;
    for each (i,j) ∈ A do d[i,j] := c_ij;
    while the network contains three nodes i, j, and k
        satisfying d[i,j] > d[i,k] + d[k,j] do d[i,j] := d[i,k] + d[k,j];
end;
```

Satz: Der All-Pairs Label-Correcting-Algorithmus terminiert innerhalb $\mathcal{O}(n^3C)$ Iterationen mit der korrekten Lösung.

Beweis: analog zum einfachen kürzesten Wege-Problem. □

- Test der All-Pairs Shortest Path Optimalitätsbedingungen benötigt $\Omega(n^3)$ Tripeloperationen.

- $d^k[i,j]$ sei die Länge des kürzesten Weges von i nach j bei dem nur die Knoten $1, 2, \ldots, k-1$ als innere Knoten vorkommen. $(d[i,j] = d^{n+1}[i,j])$

- Dynamischer Programmieransatz.

Lemma: $d^{k+1}[i,j] = \min\{d^k[i,j], d^k[i,k] + d^k[k,j]\}$

Beweis: Ein kürzester Weg von i nach j, der alle Knoten von 1 bis k benutzen darf, ist das Minimum eines kürzesten Weges ohne Knoten k und eines zusammengesetzen Weges der über Knoten k verläuft und dessen Teilstücke nur aus kürzeste Wegen ohne Knoten k bestehen. □

algorithm Floyd-Warshall;
begin
  for all node pairs $[i, j] \in N \times N$ **do**
    $d[i, j] := \infty$ and pred$[i, j] := 0$;
  for all nodes $i \in N$ **do** $d[i, i] := 0$;
  for each arc $(i, j) \in A$ **do** $d[i, j] := c_{ij}$ and pred$[i, j] := i$;
  for each $k := 1$ to n **do**
    for each $[i, j] \in N \times N$ **do**
      if $d[i, j] > d[i, k] + d[k, j]$ **then**
      begin
        $d[i, j] := d[i, k] + d[k, j]$;
        pred$[i, j]$:=pred$[k, j]$;
      end;
end;

Satz: Der Floyd-Warshall-Algorithmus berechnet die kürzesten Wege zwischen allen Knotenpaaren in $\mathcal{O}(n^3)$.

Einschub: Klassifikation von Algorithmen

- C = größter Kostenwert einer Kante in G.
  U = größter Kapazitätsschranke einer Kante bzw. größter Bedarf/Angebot eines Knotens in G.

- *streng polynomialer Algorithmus*: Worstcase-Komplexität ist durch eine polynomiale Funktion der Problemparameter $n, m, \log C, \log U$ beschränkt.
  Beispiele: $\mathcal{O}(n^2), \mathcal{O}(m + n \log C)$.

- *exponentieller Algorithmus*: Laufzeit nicht polynomial beschränkbar durch $n, m, \log C, \log U$.
  Beispiele: $\mathcal{O}(nC), \mathcal{O}(2^n), \mathcal{O}(n!)$.

- *pseudo-polynomialer Algorithmus*: Laufzeit beschränkbar durch polynomialer Funktion der Problemparameter n, m, C, U.
  Falls $C, U \in \mathcal{O}(n^k) \to$ Streng polynomial.

Kapitel 2

Maximale Flüsse

2.1 Anwendungen

- Oft Teilproblem in komplizierteren Netzwerkproblemen (z.B. Kostenminimale Flüsse).
- Ableitung elementarer kombinatorischer und graphentheoretischer Aussagen.

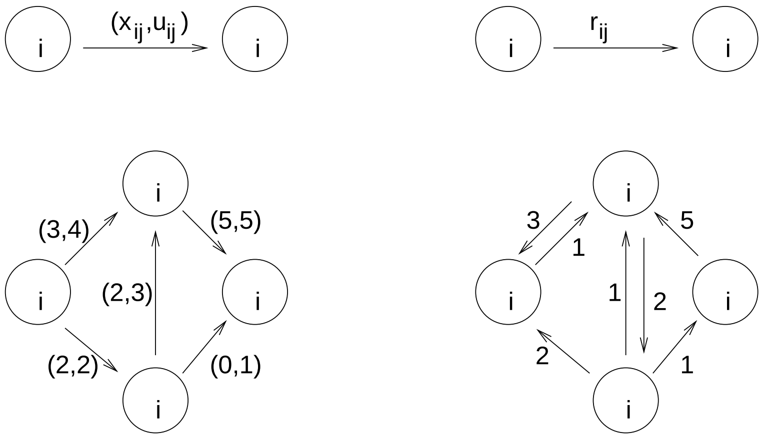
2.2 Problemdefinition

- Gegeben: gerichtetes Netzwerk $G = (N, A)$ mit nichtnegativen, ganzzahligen Kapaziäten u_{ij} an den Kanten.
- $U = \max\{u_{ij} : (i,j) \in A\}$
- Gesucht ist ein maximaler Fluß von Quelle s zu Senke t.
- Fluß x ist zulässig, falls die Balance- und Kapazitätsbedingungen bei jedem Knoten erfüllt sind.
- Maximiere v
  wobei
  $$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = \begin{cases} v & \text{für } i = s, \\ 0 & \text{für alle } i \in N - \{s \text{ und } t\}, \\ -v & \text{für } i = t \end{cases}$$
  $0 \le x_{ij} \le u_{ij}$ für alle $(i,j) \in A$

Definitionen:

- **Inkrementgraph** $G(x)$
  Gegeben sei ein Fluß x. Weise jeder Kante (i,j) eine neue Kapazität r_{ij} zu, die den

maximal zusätzlichen Fluß beschreibt den man von i nach j transferieren kann, indem man nur die Kanten (i,j) und (j,i) benutzt. $r_{ij} := u_{ij} - x_{ij} + x_{ji}$. Beispiel:



Ausgangsgraph **Inkrementgraph**

- ***s-t* Cut**
  Ein s-t Cut $[S,\overline{S}]$ ist eine disjunkte Partition der Knotenmenge N in zwei Teilmengen S und $\overline{S}$, wobei $s \in S$ und $t \in \overline{S}$. Eine Vorwärtskante im Cut $[S,\overline{S}]$ ist eine Kante die von einem Knoten aus S zu einem Knoten aus $\overline{S}$ führt. Die Kapazität eines s-t Cuts ist die Summe der Kapazitäten aller Vorwärtskanten: $u[S,\overline{S}] = \sum_{(i,j)\in(S,\overline{S})} u_{ij}$.

2.3 Flußvergrößernde Wege

- Idee: Suche einen Weg von s zu t im Inkrementgraphen der nur Kanten mit $r_{ij} > 0$ enthält.

- Die Menge an Einheiten die man über einen solchen Weg transportieren kann entspricht dem Minimum aller Kapazitäten r_{ij} entlang des Wegs. Da alle r_{ij} positiv sind, wird in jeder Interation der Gesamtfluß um mindestens eine Einheit erhöht (solange ein solcher Weg existiert).

algorithm augmenting path;
begin
  $x := 0$;
  while $G(x)$ contains a directed path from node s to node t **do**
  begin
    identify an augmenting path P from node s to node t;
    $\delta := \min\{r_{ij} : (i,j) \in P\}$;
    augment δ units of flow along P and update $G(x)$;
  end;
end;

- Der Labeling Algorithmus benutzt Suchtechnik, um flußvergrößernde Wege zu finden.
- Ausgehend von s werden alle im Inkrementgraphen über positive Kanten erreichten Knoten markiert bis t erreicht wird und somit ein flußvergrößernder Weg gefunden ist.
- Der Algorithmus terminiert, wenn t nicht mehr erreicht werden kann.

```
algorithm labeling;
begin
    label node t;
    while t is labeled do
    begin
        unlabel all nodes;
        set pred(j) := 0 for each j ∈ N;
        label node s and set LIST := {s};
        while LIST ≠ ∅ or t is unlabeled do
        begin
            remove a node i from LIST;
            for each arc (i, j) in the residual network emanating from node i do
                if r_ij > 0 and node j is unlabeled then
                    set pred(j) := i, label node j and add j to LIST;
        end;
        if t is labeled then augment;
    end;
end;

procedure augment;
begin
    use the predecessor labels to trace back from the sink to the source to
        obtain an augmenting path P from node s to node t;
    δ := min{r_ij : (i, j) ∈ P};
    augment δ units of flow along P and update the residual capacities;
end;
```

Satz: Der Labeling-Algorithmus terminiert mit einem maximalen Fluß von s nach t.
Beweis: Wenn der Algorithmus abbricht, dann ist N in die Menge S der markierten und $\overline{S}$ der unmarkierten Knoten geteilt. Es gilt: $s \in S$ und $t \in \overline{S}$. Da kein Knoten mehr in $\overline{S}$ erreicht werden kann, muß für jede Kante (i, j) aus Cut $[S, \overline{S}]$ gelten: $r_{ij} = 0$.
Aus $r_{ij} := u_{ij} - x_{ij} + x_{ji}$ folgt somit $x_{ij} = u_{ij}$ und $x_{ji} = 0$, da $x_{ij} \leq u_{ij}$ und $x_{ij} \geq 0$. Es gilt also:

$$v = \sum_{(i,j)\in(S,\overline{S})} x_{ij} - \sum_{(j,i)\in(\overline{S},S)} x_{ij} = \sum_{(i,j)\in(S,\overline{S})} u_{ij} = u[S, \overline{S}]$$

Da der Fluß v kleiner oder gleich der Kapazität jedes s-t Cuts sein muß, ist $[S, \overline{S}]$ ein minimaler Cut. □

Daraus folgt sofort:
Max-Flow-Min-Cut-Theorem: Der maximale Fluß von einer Quelle s zu einer Senke t in einem kapazitierten Netzwerk ist gleich der Kapazität eines minimalen s-t Cuts.

Bemerkung: Bei ganzzahligen Kapazitäten existiert immer auch ein ganzzahliger maximaler Fluß. Aber nicht jeder optimale Fluß muß ganzzahlig sein!

Satz: Der Labeling-Algorithmus löst das Maximale Flußproblem in $\mathcal{O}(nmU)$.
Beweis:

1. In jeder Iteration wird eine Augmentierung ausgeführt. Aufwand: $\mathcal{O}(m)$, da jede Kante bzw. jeder Knoten höchstens einmal untersucht wird.
2. Die Kapazität des Cuts $(\{s\}, N - \{s\})$ ist höchstens nU.
3. In jeder Iteration wird aus Gründen der Ganzzahligkeit mindestens eine Einheit augmentiert. $\square$

Nachteile des Labeling-Algorithmus

- Es gibt Instanzen mit exponentieller Laufzeit.
- Keine zwingende Terminierung und Konvergenz bei irrationalen Kantenkapazitäten. Aber Max-Flow-Min-Cut-Theorem gilt weiterhin.
- Verfahren nutzt nicht die Labelinginformation der vorherigen Iteration.

Ableitung graphentheoretische Aussagen (Menger's Theoreme)

Satz: Die maximale Anzahl kantendisjunkter Wege von Knoten s zu Knoten t entspricht der minimalen Zahl von Kanten deren Löschung notwendig ist, um alle Wege von s zu t zu unterbrechen.

Beweis: Weise jeder Kanten im Netzwerk die Kapazität 1 zu und löse das maximale Flußproblem. Angenommen wir können maximal v Einheiten von s zu t transportieren, dann zerfällt der Fluß in Wegflüsse von s zu t und Kreisflüsse. Die Wegflüsse summieren sich zu v und sind kantendisjunkt, da jede Kante nur die Kapazität 1 besitzt.
Betrachten wir alle s-t Cuts im Netzwerk. Jeder Cut besitzt eine Kapazität die der Anzahl der Vorwärtskanten entspricht. Jeder Weg von s zu t enthält eine Kante aus jedem Cut. Daher unterbricht eine Löschung aller Kanten in einem Cut alle möglichen Wege von s zu t. Aus dem Max-Flow-Min-Cut-Theorem folgt der Satz. $\square$

Satz: Die maximale Anzahl knotendisjunkter Wege von Knoten s zu Knoten t entspricht der minimalen Zahl von Knoten deren Löschung notwendig ist, um alle Wege von s zu t zu unterbrechen.

Beweis: Splitte jeden Knoten i aus $N - \{s, t\}$ in zwei Knoten i' und i'' und füge die Kante (i', i") mit Kapazität 1 hinzu. Alle eingehenden Pfeile werden i' und alle ausgehenden Pfeile

i'' zugewiesen. Diese Pfeile bekommen die Kapazität ∞.
Jeder kantendisjunkter Weg in unserem neuen Netzwerk entspricht einem knotendisjunkten Weg im Ausgangsntetzwerk. Jeder Cut mit endlicher Kapazität enthält nur neu hinzugefügte Kanten. Direkte Übertragung der Ergebnisse vom vorherigen Satz! $\square$

Maximale Flüsse mit unteren Kapazitätsschranken

Maximiere v
wobei

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = \begin{cases} v & \text{für } i = s, \\ 0 & \text{für alle } i \in N - \{s \text{ und } t\}, \\ -v & \text{für } i = t \end{cases}$$

$l_{ij} \leq x_{ij} \leq u_{ij}$ für alle $(i,j) \in A$

- Problem: Es muß keine zulässige Lösung existieren
- Zweiphasiger Ansatz (jeweils Lösung eines maximalen Flußproblems):
  - Bestimmen eines zulässigen Flusses
    Transformation in ein Zirkulationsproblem: Hinzunahme der Kante (t,s) mit unbeschränkter Kapazität. Eine zulässige Zirkulation impliziert einen zulässigen Fluß. Durch die Ersetzung $x_{ij} = x'_{ij} + l_{ij}$ erhält man:

    $\sum_{j:(i,j)\in A} x'_{ij} - \sum_{j:(j,i)\in A} x'_{ji} = b(i)$ für alle $i \in N$,
    $0 \leq x'_{ij} \leq u_{ij} - l_{ij}$ für alle $(i,j) \in A$.
    wobei
    $b(i) := \sum_{j:(j,i)\in A} l_{ji} - \sum_{j:(i,j)\in A} l_{ij}$.

    Da $\sum_{i\in N} b(i) = 0$, hat man nun ein zulässigen Fluß ohne untere Schranken zu bestimmen, was sich leicht auf ein maximales Flußproblem reduzieren läßt. Danach, falls eine Lösung existiert, erfolgt eine Rücktransformation $x_{ij} = x'_{ij} + l_{ij}$.
  - Konvertieren in einen maximalen Fluß
    Gegeben sei zulässiger Fluß x bezüglich unterer Kapazitätsschranken. Erweiterung der Kapazität im Inkrementgraphen:

    $$r_{ij} = (u_{ij} - x_{ij}) + (x_{ji} - l_{ji})$$

    Mit dieser Erweiterung bleiben alle flußvergrößernden Wegeverfahren korrekt. Bei einer Erweiterung der Kapazität eines s-t Cuts zu

    $$u[S,\overline{S}] = \sum_{(i,j)\in(S,\overline{S})} u_{ij} - \sum_{(i,j)\in(\overline{S},S)} l_{ij}$$

    bleibt auch das Max-Flow-Min-Cut-Theorem weiterhin gültig.

Ohne Beweis:
Satz: Eine zulässige Zirkulation bei einem Netzwerk mit nichtnegativen unteren und oberen Kapazitätsschranken existiert genau dann, wenn für jede Menge S gilt:

$$\sum_{(i,j)\in(\overline{S},S)} l_{ij} \leq \sum_{(i,j)\in(S,\overline{S})} u_{ij}$$

Verbesserung des Labeling-Verfahrens

Algorithmusidee

- Wir suchen uns in jeder Iteration einen flußvergrößernden Weg, der uns den Gesamtfluß ausreichend erhöht.
- Δ-*Inkrementgraph* $G(x,\Delta)$
  $G(x,\Delta)$ ist ein Subgraph von $G(x)$, der alle Kanten (i,j) mit $r_{ij} \geq \Delta$ enthält. $G(x,1) = G(x)$.
- Wir beginnen mit $\Delta = 2^{\lfloor \log U \rfloor}$ und halbieren Delta nach jeder Δ-Phase.
- In jeder Δ-Phase werden mindestens Δ-Einheiten auf jedem flußvergrößernden Weg augmentiert.

```
algorithm capacity scaling;
begin
    x := 0;
    Δ := 2^⌊log U⌋;
    while Δ ≥ 1 do
    begin
        while G(x, Δ) contains a path from node s to node t do
        begin
            identify a path P in G(x, Δ);
            δ := min{r_ij : (i, j) ∈ P};
            augment δ units of flow along P and update G(x, Δ);
        end
        Δ := Δ/2 ;
    end;
end;
```

Satz: Der Capacity-Scaling-Algorithmus löst das Maximale Flußproblem in $\mathcal{O}(m^2 \log U)$.

Beweis: Insgesamt gibt es $\mathcal{O}(\log U)$ Phasen. Da in der letzen Phase $G(x)$ betrachtet wird, ist der Algorithmus korrekt.

Am Ende einer Δ-Phase sei x' der Fluß und v' der Flußwert.Seien in S alle Knoten die in $G(x',\Delta)$ von s erreichbar sind. Da t nicht mehr in $G(x',\Delta)$ erreichbar ist, stellt $[S,\overline{S}]$ einen s-t Cut dar. Jede Vorwärtskante muß kleiner sein als Δ und somit die Gesamtkapazität des Cuts $[S,\overline{S}]$ im Inkrementgraphen $G(x')$ höchstens $m\Delta$. Sei v^* der maximale Fluß, dann gilt: $v^* - v' \leq m\Delta$. In der nächsten Phase mit mindestens $\Delta/2$ großen flußvergrößernden Wegen gibt es somit maximal $2m$ Interationen. In jeder Δ-Phase kann es also höchstens nur $2m$ Augmentierungen geben. Jede Augmentierung und jeder Update des Δ-Inkrementgraph kostet höchstens $\mathcal{O}(m)$ Zeit. □

2.4 Der Preflow-Push Algorithmus

Definitionen

- *Preflow:*
  Ein Fluß bei dem man erlaubt, daß bei einem Koten mehr Einheiten hinein- als herausfließen. Ein Preflow ist ein Funktion $x : A \to |R$ mit folgender Eigenschaft:
  $$\sum_{j:(j,i)\in A} x_{ji} - \sum_{j:(ij)\in A} x_{ij} \geq 0 \text{ für alle } i \in N - \{s,t\}.$$
- *Überschuß oder Exzeß:*
  $$e(i) := \sum_{j:(j,i)\in A} x_{ji} - \sum_{j:(ij)\in A} x_{ij} \text{ für alle } i \in N$$
  Somit ist s der einzige Knoten mit negativen Exzeß.
- Ein Knoten mit positiven Exzeß sei ein *aktiver* Knoten. (Außer s und t).

Algorithmusidee

- Der Algorithmus iteriert solange es aktive Knoten gibt. Falls keine aktiven Knoten mehr existieren, ist eine zulässige, maximale Strömung erreicht.
- In jedem Schritt wird ein aktiver Knoten gewählt und dessen Exzeß ggf. an seine Nachbarknoten transferiert. Dabei werden Knoten die näher an der Senke liegen bevorzugt.
- Die Nähe zur Senke wird mittels Distanzmarken beschrieben.
- Preflow-Push Algorithmen sind sowohl theoretisch wie auch praktisch den Flußvergrößernden Wege-Algorithmen überlegen.

Definitionen

- *Distanzmarken*
  Jedem Knoten $i \in N$ ist eine nichtnegative Distanzmarke $d(i)$ zugeordnet. Diese Distanzmarken seien dann gültig, falls
  $$d(t) = 0; \text{ und } d(i) \leq d(j) + 1 \text{ für alle Kanten } (i,j) \text{ im Inkrementgraphen } G(x)$$
  erfüllt sind. Sie lassen sich einfach mit Dijkstra's Kürzeste Wege Verfahren bestimmen.

- *Zulässige Kanten*
  Eine Kante (i, j) heißt zulässig, falls gilt: $d(i) = d(j) + 1$;

algorithm preflow-push;
begin
  preprocess;
  while the network contains an active node **do**
  begin
    select an active node i;
    push/relabel(i);
  end;
end;

procedure preprocess;
begin
  $x := 0$;
  compute the exact distance labels $d(i)$;
  $x_{sj} := u_{sj}$ for each arc $(s, j) \in A(s)$;
  $d(s) := n$;
end;

procedure push/relabel(i);
begin
  if the network contains an admissible arc (i, j) **then**
    push $\delta = \min\{e(i), r_{ij}\}$ units of flow from node i to node j
  else replace $d(i)$ by $\min\{d(j) + 1 : (i, j) \in A(i)$ and $r_{ij} > 0\}$;
end;

Anschauliche Beschreibung als physikalisches Netzwerk

- Kanten beschreiben Wasserleitungen, Knoten Verbindungen und die Distanzmarken die Höhe der Verbindungen gemessen zum Boden.
- Man will möglichst viel Wasser von der Quelle zur Senke transportieren.
- Zu Beginn wird die Quelle weit nach oben gehoben und das Wasser fließt zu allen Nachbarknoten und von dort zu tiefer liegenden Knoten.
- Kann an einer Verbindung das Wasser nicht abfließen, so wird es der Knotenpunkt eine Einheit nach oben gehieft. Das Wasser kann nun wieder abwärts zur Senke fließen, oder falls die Senke nicht mehr erreichen kann, strömt der Rest wieder Richtung Quelle.

Definitionen

- *Saturierte und unsaturierte Push-Operation*:
  Eine Push-Operation von Knoten i nach j mit δ Einheiten verringert $e(i)$ und r_{ij}

bzw. erhöht $e(j)$ und r_{ji} um δ Einheiten. Eine Push-Operation heißt saturiert, falls $\delta = r_{ij}$. Andernfalls unsaturiert.

Lemma: Zu jedem Zeitpunkt gilt: Jeder Knoten i mit positiven Exzeß ist mit der Quelle s im Inkrementgraphen durch einen gerichteten Weg von i nach s verbunden.

Beweis:
Nach dem Dekompositionstheorem zerfällt der Preflow in drei Teile:

1. Fluß von s zu t auf gerichteten Wegen.
2. Fluß von s zu aktiven Knoten auf gerichteten Wegen.
3. Fluß auf gerichteten Kreisen im Netzwerk

Sei i ein aktiver Knoten bzgl Fluß x, dann muß es nach Dekomposition von x einen Pfad P von s zu i geben, da Teile (1) und (3) zum Exzeß bei i nichts beitragen. Der Inkrementgraph enthält genau den entgegengesetzten Weg. □

Lemma: Für jeden Knoten $i \in N$ gilt: $d(i) < 2n$.

Beweis:
Bei der letzten Relabel-Operation hatte Knoten i positiven Exzeß, somit enthielt der Inkrementgraph einen Weg P von i nach s mit maximal $n-2$ Kanten.
Mit $d(s) = n$ und $d(k) \leq d(l) + 1$ für jede Kante (k, l) aus P gilt: $d(i) \leq d(s) + |P| < 2n$.
□

Lemma: Jede Distanzmarke wächst höchstens $2n$ mal. Die Anzahl aller Relabel-Operationen liegt bei höchstens $2n^2$.

Beweis:
Jede Relabel-Operation erhöht eine Markierung um mindestens eine Einheit. □

Lemma: Falls eine Relabel-Operation k mal bei einem Knoten i ausgeführt wird, führt der Knoten höchstens $km/2$ saturierte Push-Operationen aus.

Beweis: Angenommen ein saturierte Push-Operationen von i nach j wird durchgeführt. Da Kante (i, j) zulässig ist, muß gelten: $d(i) = d(j) + 1$. Bevor der Algorithmus auf der Kante wieder eine saturierte Push-Operation ausführt, müssen Einheiten wieder von j nach i zurückgeflossen sein. Also es galt: $d'(j) = d'(i) + 1$. Beim folgenden Push gilt: $d''(i) = d''(j) + 1$. Zusammen:

$$d''(i) = d''(j) + 1 \geq d'(j) + 1 = d'(i) + 2 \geq d(i) + 2$$

Also muß zwischen zwei saturierten Push-Operationen eine Distanzmarkierung mindestens um zwei Einheiten angewachsen sein. □

Lemma: Der generische Preflow-Push-Algorithmus führt höchstens nm saturierte Push-Operationen aus.

Beweis:
Folgt aus den beiden vorhergehenden Lemmata. □

Lemma: Der generische Preflow-Push-Algorithmus führt $\mathcal{O}(n^2m)$ nichtsaturierte Push-Operationen aus.

Beweis:
Sei I die Menge der aktiven Knoten und Potentialfunktion $\Phi := \sum_{i \in I} d(i)$. Da $|I| < n$ und $d(i) < 2n$ gilt $\Phi < 2n^2$. Bei der Terminierung des Verfahrens gilt: $\Phi = 0$. Bei jeder Relabel/Push-Operation tritt einer der beiden Fälle auf:

- **Fall 1:** Es gibt keine zulässige Kante, also Relabel-Operation.
  Die Knotenmarkierung wird um $\epsilon \geq 1$ Einheiten erhöht und somit steigt Φ um ϵ Einheiten. Da die gesamte Erhöhung einer Marke durch $2n$ beschränkt ist, ist die Erhöhung von Φ durch $2n^2$ beschränkt.

- **Fall 1:** Eine zulässige Kante existiert und es wird eine saturierte oder unsaturierte Push-Operation durchgeführt.
  a, Saturierter Fall
  Ggf. Erzeugung eines neuen aktiven Knotens j und somit Erhöhung Φ um $d(j)$ Einheiten. Da $d(j) < 2n$ und es maximal nm saturierte Push-Operationen geben kann, gilt eine maximale Erhöhung von Φ um $2n^2m$ durch saturierte Push-Operationen.

  b, Unsaturierter Fall
  Gleichzeitige Erniedrigung von Φ um $d(i)$ Einheiten und Erhöhung um $d(j) = d(i) - 1$ Eiheiten, falls j durch die Push-Operationen aktiv wird. Also mindestens Verringerung von Φ um eine Einheit pro unsaturierter Push-Operation.

Da sich also Φ um maximal $2n^2 + 2n^2m$ erhöhen kann und niemals negativ wird, kann es höchstens $\mathcal{O}(n^2m)$ unsaturierte Push-Operationen geben. □

Satz: Der generische Preflow-Push-Algorithmus besitzt eine Laufzeit von $\mathcal{O}(n^2m)$.

Beweis:
Angenommen der Algorithmus verwaltet eine Liste LIST für alle aktiven Knoten. Knoten werden hinzugefügt, wenn sie nach einer Push-Operation aktiv werden und noch nicht in LIST enthalten sind. Knoten werden gelöscht, wenn sie durch eine unsaturierte Push-Operation inaktiv werden. Alle Operationen auf LIST lassen sich durch geeignete Datenstrukturen (z.B. doppel-verkettete Liste) in konstanter Zeit implementieren. □

Effizient Implementierungen

- *FIFO Preflow-Push-Algorithmus*
  Untersuchung der aktiven Knoten in first-in, first-out Reihenfolge. Laufzeit: $\mathcal{O}(n^3)$

- *Highest-label Preflow-Push-Algorithmus*
  Wählt den aktiven Knoten mit höchster Distanzmarkierung. Laufzeit: $\mathcal{O}(n^2\sqrt{m})$

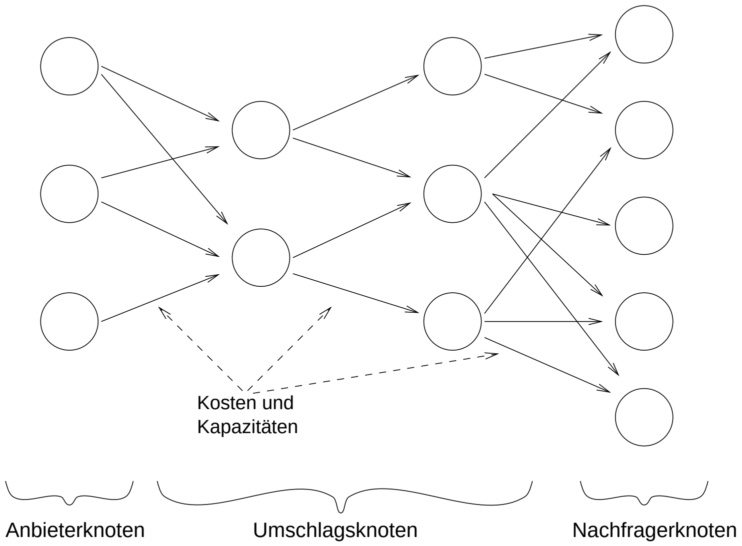
- *Excess Scaling-Algorithmus*
  Transferiert Einheiten von Knoten mit ausreichend hohen Exzeß zu Knoten mit ausreichend niedrigen Exzeß. Laufzeit: $\mathcal{O}(nm + n^2 \log U)$

Kapitel 3

Kostenminimale Flüsse

3.1 Anwendungen

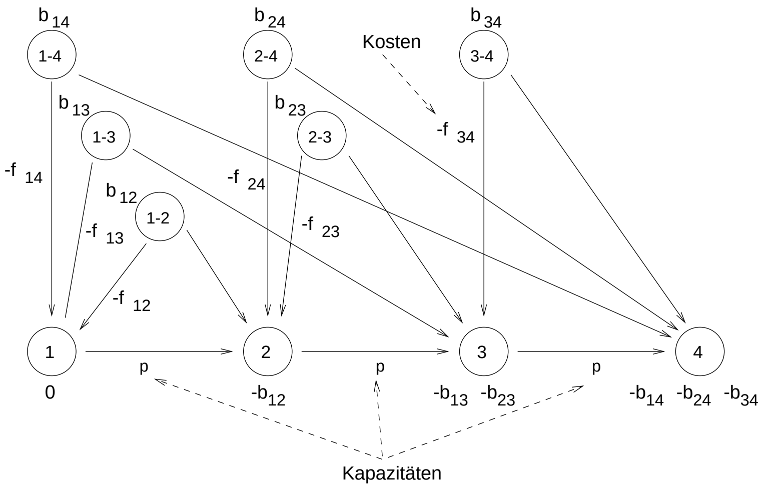
- Direkte Anwendung in Beschaffungs- und Distributionsproblemen:
  Gegeben sei eine Menge von Anbietern und eine Menge von Nachfragern eines bestimmten Gutes. Die Kosten und Kapazitätsschranken der Distributionskanäle seien bekannt. Gesucht ist ein kostenminimaler Fluß der Güter von Anbieter- zu Nachfragerknoten, so daß die Gesamtnachfrage befriedigt wird.



- „Hopping Airplane“
  Eine Fluggesellschaft benutzt ein Flugzeug, mit einer Kapazität von maximal p Flugpassagieren, um eine feste Sequenz von n Städten abzufliegen. In jeder Stadt können Reisende zu- und aussteigen. Sei b_{ij} die Anzahl Passagiere, die von Stadt i zu Stadt j fliegen möchten und f_{ij} der entsprechende Flugpreis.

  Die Fluggesellschaft möchte nun die Anzahl von Passagieren wissen, die bedient werden können, so daß die Kapazitätsschranke zu keinem Zeitpunkt überschritten wird.

Lösung: Formulierung als MCFP.



Alle anderen Kosten seinen null und alle anderen Kapazitäten unbeschränkt.

3.2 Problemdefinition

- Gegeben sei ein gerichtetes Netzwerk $G = (N, A)$ mit $|N| = n, |A| = m$.
- Für jede Kante $(i, j) \in A$ entstehen für den Transport einer Einheit auf dieser Kante c_{ij} Kosten. Linearitätsannahme: Kosten steigen linear zur transportierten Menge.
- Sei x_{ij} die auf Kante (i,j) transportierte Menge. Dann muß diese zwischen zwei vorgegebenen Kapazitätsschranken l_{ij} und u_{ij} liegen:

$$l_{ij} \leq x_{ij} \leq u_{ij}$$

- Für jeden Knoten $i \in N$ gibt es ein festes Angebot/Nachfrage b(i) eines Gutes.

$$\begin{aligned} b(i) > 0 &: i \text{ ist ein Angebotsknoten} \\ b(i) < 0 &: i \text{ ist ein Bedarfsknoten} \\ b(i) = 0 &: i \text{ ist ein Umladeknoten} \end{aligned}$$

  Es gilt das globale Gleichgewicht: $\sum_{i=1}^{n} b(i) = 0$.
  U sei das Maximum vom größten Betrag aller $b(i)$ und aller Kantenkapazitäten

- Problem: Transportiere unter Beachtung der Kanten-Kapazitätsbeschränkungen alle Güter von den Angebotsknoten zu den Bedarfsknoten, so daß der Bedarf an den Senken gedeckt wird und die Gesamttransportkosten minimal werden.

- Lineares Programm:
$$\min \sum_{(i,j)\in A} c_{ij} x_{ij}$$
unter Beachtung
$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = b(i) \text{ für alle } i \in N \text{ lokales Gleichgewicht.}$$
$$l_{ij} \le x_{ij} \le u_{ij} \text{ für alle } (i,j) \in A$$

- Matrixschreibweise:
$\min \mathbf{cx}$
$\mathbf{Nx} = \mathbf{b}$
$\mathbf{l} \le \mathbf{x} \le \mathbf{u}$

  x, c, l, u : m-dimensionale Vektoren
  b : n-dimensionaler Vektor
  N : $n \times m$-Matrix (Inzidenzmatrix)

- Wunsch x_{ij} ganzzahlig: LP $\to$ ILP(Integer Lineares Programm)
Aber: Falls Input-Parameter(b_{ij}, c_{ij}, u_{ij}) ganzzahlig, dann ist die Lösung ebenfalls ganzzahlig. (Tieferer Grund: Unimodularität der Inzidenzmatrix).

- In den folgenden Abschnitten wird $l = 0$ vorausgesetzt, da sich der Fall mit nichttrivialen unteren Schranken ähnlich wie beim Maximalen Flußproblem auf den einfachen Fall reduzieren läßt.

3.3 Optimalitätsbedingungen

Inkrementgraph $G(x)$:
Jede Kante $(i,j) \in A$ wird in $G(x)$ durch zwei Kanten (i,j) und (j,i) ersetzt. Kante (i,j) besitzt die Kostenbewertung c_{ij} und die Kapazität $r_{ij} = u_{ij} - x_{ij}$. Kante (j,i) besitzt die Kostenbewertung $c_{ji} = -c_{ij}$ und die Kapazität $r_{ji} = x_{ij}$. In $G(x)$ seinen nur Kanten mit $r_{ij} > 0$.

Satz (Augmenting-Cycle-Eigenschaft): Seien x und x^0 zwei beliebige zulässige Strömungen im Graphen G. Dann läßt sich x als die Summme von x^0 und dem Fluß auf höchstens m Kreisen $G(x^0)$ schreiben. Weiter sind die Kosten von x gleich den Kosten von x^0 plus den Kosten der Flüsse auf den m Kreisen.

Satz (Negative-Cycle-Optimalitätsbedingungen): Eine zulässige Strömung x^* ist genau dann eine optimale Lösung des MCFP, wenn sie die folgende Bedingung erfüllt:
$G(x^*)$ enthält keinen (gerichteten) Kreis mit negativen Kosten.

Beweis: Angenommen x sei eine zulässige Strömung und $G(x)$ enthalte einen Kreis mit negativen Kosten. Dann kann x nicht kostenminimal sein, da eine Erhöhung des Flusses

entlang des Kreises die Kosten verringert.
Sei nun x^* eine zulässige Strömung und $G(x^*)$ enthalte keinen Kreis mit negativen Kosten. Sei x^0 ein kostenminimaler Fluß und $x^0 \neq x^*$. Wegen der Augmenting-Cycle-Eigenschaft können wir $x^0 - x^*$ in höchstens m strömungsvergrößernde Kreise bezüglich der Strömung x^* zerlegen und die Summe der Kosten der Strömungen auf diesen Kreisen ist gleich $cx^0 - cx^*$. Da alle Kreise in $G(x^*)$ nichtnegative Kosten haben, muß $cx^0 - cx^* \geq 0$ bzw. $cx^0 \geq cx^*$ gelten. Da x^0 jedoch ein kostenminimaler Fluß ist, muß auch $cx^0 \leq cx^*$ gelten. Damit folgt $cx^0 = cx^*$ und x^* ist somit ein kostenminimaler Fluß. □

Defintion (Reduzierte Kosten): Jedem Knoten $i \in N$ wird eine reelle Zahl (Potential) π zugeordnet. Bei einem gegebenen Knotenpotential $\pi = (\pi(1), \ldots, \pi(n))$ definieren wir zu jeder gerichteten Kante (i,j) die *reduzierten Kosten* $c^\pi_{ij} = c_{ij} - \pi(i) + \pi(j)$.

Satz (Reduced-Cost-Optimalitätsbedingungen): Eine zulässige Strömung x^* in G ist genau dann eine optimale Lösung des Problems kostenminimaler Flüsse, wenn ein Knotenpotential π existiert, die die folgenden Bedingungen erfüllt:

$$c^\pi_{ij} \geq 0 \text{ für jeden Pfeil } (i,j) \text{ in } G(x^*).$$

Beweis: Man zeigt, daß diese Bedingung äquivalent zu der Negative-Cycle-Optimalitätsbedingung ist.
Erfülle x^* also die Reduced-Cost-Optimalitätsbedingungen. Dann gilt $\sum_{(i,j)\in W} c_{ij} = \sum_{(i,j)\in W} c^\pi_{ij} \geq 0$ für jeden Kreis W in $G(x^*)$. Damit erfüllt x^* auch die Negative-Cycle-Optimalitätsbedingung.
Sei nun x^* eine Strömung, so daß $G(x^*)$ keine Kreise mit negativen Kosten enthalte. Sei $d(\cdot)$ die Länge der kürzesten Wege von Knoten 1 zu allen anderen Knoten in $G(x^*)$ bezüglich Kosten c. Da $G(x^*)$ keine negativen Kreise enthält, gilt $d(j) \leq d(i) + c_{ij}$ für alle $(i,j) \in G(x^*)$, oder auch $c_{ij} - (-d(i)) + (-d(j)) \geq 0$. Definieren wir $\pi := -d$, so erhalten wir $c^\pi_{ij} \geq 0$, und x^* erfüllt somit die Reduced Cost Optimalitätsbedingungen. □

3.4 Der Cycle-Canceling Algorithmus

Idee:

- Starte mit einer zulässigen Lösung.
- Eliminiere sukzessive negative Kreise im Inkrementgraphen. In jeder Iteration erhält man eine verbesserte zulässige Lösung.
- Abbruch, falls kein negativer Kreis mehr existiert, da Optimalität erreicht.

```
algorithm cycle-canceling;
begin
    establish a feasible flow x in the network;
    while G(x) contains a negative cycle do
    begin
        use some algorithm to identify a negative cycle W;
        δ := min{r_ij : (i,j) ∈ W};
        augment δ units of flow in the cycle W and update G(x);
    end;
end;
```

Satz: Der Cycle-Canceling Algorithmus besitzt eine Laufzeit von $\mathcal{O}(nm^2CU)$ wenn die negative Kreisbestimmung in $\mathcal{O}(nm)$ erfolgt.

Beweis: mCU ist eine obere Schranke für die Kosten des initialen zulässigen Flusses. In jeder Iteration werden die Flußkosten um mindestens eine Einheit gesenkt. Daher durchläuft das Verfahren höchstens $\mathcal{O}(mCU)$ Iterationen. □

Bemerkungen:

- Verbesserung zu einem streng-polynomialen Verfahren möglich. Sukzessive Elimination des Kreises mit minimalen mittleren Kosten. Die mittleren Kosten eines Kreises sind seine Kosten dividiert durch die Anzahl seiner Kanten. Laufzeit: $\mathcal{O}(n^2m^3 \log n)$

3.5 Der Successive Shortest Path Algorithmus

Definition Pseudoflow: Ein Fluß x, der die Kapazitäts- und Nichtnegativitätsbedingungen erfüllt, jedoch nicht zwingend die Gleichgewichtsbedingungen.
Das Ungleichgleichgewicht eines Knotens i sei:

$$e(i) = b(i) + \sum_{j:(j,i)\in A} x_{ji} - \sum_{j:(i,j)\in A} x_{ij} \text{ für alle } i \in N.$$

Wir nennen $e(i)$ den Exzeß eines Knotens, falls $e(i) > 0$. Andernfalls sei $-e(i)$ das Defizit, falls $e(i) < 0$.

Idee:

- Starte mit einer Lösung(Pseudoflow), die die Optimalitätskriterien erfüllt.
- Schicke sukzessive Einheiten von Exzeß- zu Defizitknoten auf kürzesten Wegen im Inkrementgraphen. In jeder Iteration erhält man eine Lösung, die den Optimalitätskriterien genügt und sich zunehmend einer zulässigen Lösung annähert.
- Abbruch, falls zulässige Lösung erreicht.

```
algorithm successive shortest path;
begin
    x := 0 and π := 0;
    e(i) := b(i) for all i ∈ N;
    initialize the sets E := {i : e(i) > 0} and D := {i : e(i) < 0};
    while E ≠ ∅ do
    begin
        select a node k ∈ E and a node l ∈ D;
        determine shortest path distances d(j) from node s to all other
            nodes in G(x) with respect to the reduced costs c^π_ij;
        let P denote a shortest path from node k to node l;
        update π := π − d;
        δ := min [e(k), −e(l), min{r_ij : (i, j) ∈ P}];
        augment δ units of flow along the path P;
        update x, G(x), E, D and the reduced costs;
    end;
end;
```

Satz: Angenommen, ein Pseudoflow x erfüllt die Reduced-Cost-Optimalitätsbedingungen bezüglich eines Potentials π. Repräsentiere d den Vektor der Länge der kürzesten Wege von einem Knoten s zu allen anderen Knoten im Inkrementgraphen $G(x)$, wobei c_{ij}^{π} die Länge des Pfeils (i, j) sei. Dann gelten die folgenden Eigenschaften:

1. Der Pseudflow x erfüllt ebenfalls die Reduced-Cost-Optimalitätsbedingungen bzgl. des Knotenpotentials $\pi' = \pi - d$.

2. Die reduzierten Kosten $c_{ij}^{\pi'}$ sind 0 für alle Pfeile (i, j) in einem kürzesten Weg von s zu jedem anderen Knoten.

Beweis: Nach Definition von d gilt:

$$d(j) \leq d(i) + c_{ij}^{\pi} \text{ für alle } (i, j) \text{ in } G(x).$$

Einsetzen von $c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j)$ ergibt $d(j) \leq d(i) + c_{ij} - \pi(i) + \pi(j)$, oder $c_{ij} - (\pi(i) - d(i)) + (\pi(j) - d(j)) \geq 0$. Also $c_{ij}^{\pi'} \geq 0$ und somit erster Teil bewiesen.
Für jeden Pfeil (i, j) auf einem kürzesten Weg von s zu einem Knoten l gilt: $d(j) = d(i) + c_{ij}^{\pi}$.
Daher folgt analog zu Teil 1: $c_{ij}^{\pi'} = 0$ □

Satz: Angenommen der Pseudoflow x erfülle die Reduced-Cost-Optimalitätsbedingungen und wir erhalten die Strömung x' aus x durch Erhöhen des Flusses auf einem kürzesten Weg vom Knoten s zu einem anderen Knoten k. Dann erfüllt auch x' die Reduced-Cost-Optimalitätsbedingungen.

Beweis: Definiere π und π' wie im vorhergehenden Satz. Dann gilt $c_{ij}^{\pi'} = 0$ für jedes (i, j) in einem kürzesten Weg von s nach k. Durch Erhöhen des Flusses auf diesem Weg kommt höchstens der Pfeil (j, i) zum Inkrementgraphen hinzu. Da aber $c_{ij}^{\pi'} = 0$ gilt, gilt auch $c_{ji}^{\pi'} = 0$ und die Reduced-Cost-Optimalitätsbedingungen bleiben erfüllt. □

Satz Der Successive Shortest Path Algorithmus terminiert mit der korrekten Lösung und besitzt eine Laufzeit von $\mathcal{O}(nUS(n,m,C)$, wobei $S(n,n,C)$ für die Laufzeit eines kürzesten Wege-Verfahrens steht.

Beweis: Der Algorithmus wird mit dem Pseudoflow $x \equiv 0$ initialisiert. Damit ist $G(x) = G$ und mit $\pi = 0$, also $c^{\pi}_{ji} = c_{ij} \geq 0$, sind die Reduced-Cost-Optimalitätsbedingungen erfüllt. Solange ein Knoten noch nicht gesättigt ist, sind E und D nicht leer, der Algorithmus findet also zwei Knoten k und l und einen kürzesten Weg zwischen diesen. Da in jedem Schleifendurchlauf der Exzeß eines Knotens echt verringert wird und keine neuen übersättigten Knoten hinzukommen, terminiert der Algorithmus nach höchstens nU Schleifendurchläufen. $\square$

3.6 Polynomielle Verfahren mittels Capacity Scaling

- In ungünstigen Fällen führt der Successive Shortest Path Algorithmus viele Augmentierungen mit jeweils wenigen Einheiten aus.
- Idee: Skalierung der Kapazitäten. Suche erst kürzeste Wege zwischen Anbietern mit hohem Angebot und Nachfragern mit hoher Nachfrage und transportiere nur auf Kanten mit entsprechend großer freier Kapazität:
  $S(\Delta) := \{i \in N : e(i) \geq \Delta\}$
  $T(\Delta) := \{i \in N : e(i) \leq -\Delta\}$
  Der Δ-Inkrementgraph $G(x, \Delta)$ ist definiert als ein Subgraph von $G(x)$ mit Kanten, die mindestens Kapaziät Δ aufweisen.

```
algorithm capacity scaling;
begin
   x := 0 and π := 0;
   Δ := 2^⌊log U⌋;
   while Δ ≥ 1
   begin {Δ-scaling phase}
      for every arc (i, j) in the residual network G(x) do
          if r_ij ≥ Δ and c^π_ij < 0 then send r_ij units
            of flow along arc (i, j), update x and the imbalances e(·);
      S(Δ) := {i ∈ N : e(i) ≥ Δ};
      T(Δ) := {i ∈ N : e(i) ≤ −Δ};
      while S(Δ) ≠ ∅ and T(Δ) ≠ ∅ do
      begin
         select a node k ∈ S(Δ) and a node l ∈ T(Δ);
         determine shortest path distances d(·) from k to all other nodes in the
           Δ-residual network G(x, Δ) with respect to the reduced costs c^π_ij;
         let P denote shortest path from node k to node l in G(x, Δ);
         update π := π − d;
         augment Δ units of flow along the path P;
         update x, S(Δ), T(Δ), and G(x, Δ);
      end;
      Δ := Δ/2;
   end;
end;
```

Satz: Der Capacity-Scaling Algorithmus terminiert mit einer korrekten Lösung und besitzt eine Laufzeit von $\mathcal{O}(m \log U S(n, m, nC))$.

Beweis: Die 2Δ-Phase endet, wenn $S(2\Delta) = \emptyset$ oder $T(2\Delta) = \emptyset$. Dann gilt entweder $e(i) < 2\Delta$ für alle $i \in N$ oder $e(i) > -2\Delta$ für alle $i \in N$. Damit ist die Summe aller Exzesse (Defizite) von oben (unten) durch $2n\Delta$ ($-2n\Delta$) beschränkt. Am Anfang einer Δ-Phase

wird untersucht, ob jede Kante (i,j) im Δ-Inkrementgraphen die Optimalitätsbedigungen $c_{ij}^{\pi} \geq 0$ erfüllt. Falls nicht, so wird die Kante saturiert ($x_{ij} := u_{ij}$), so daß sie im Inkrementgraphen wegfällt. Die inverse Kante $c_{ji}^{\pi} = -c_{ij}^{\pi} \geq 0$ erfüllt die Optimalitätsbedigung. Die Saturierung einer Kante (i,j) ändert das Ungleichgewicht der Endknoten um maximal 2Δ, da $r_{ij} < 2\Delta$. Daher ist die Summe der Exzesse insgesant durch $2n\Delta + 2m\Delta = 2(n+m)\Delta$ beschränkt.
Jede Augmentierung in der Δ-Phase erhöht den Fluß um mindestens Δ Einheiten. Durch die initiale Setzung $\Delta := 2^{\lfloor \log U \rfloor}$ und durch die Halbierung von Δ in jeder Iteration, führt das Verfahren $\mathcal{O}(\log U)$ Δ-Phasen aus. Im letzten Schritt gilt $G(x, \Delta) = G(x)$, daher terminiert der Algorithmus korrekt.
In jeder Phase wird der Fluß um mindestens Δ Einheiten in jeder Augmentierung erhöht. Daher kann es höchstens $2(n+m)$ Augmentierungen pro Phase geben, und somit liegt die Laufzeit bei $\mathcal{O}(m \log U S(n, m, nC))$. □

Liste effizienter Verfahren

Algorithmus	Laufzeit
Capacity scaling algorithm	$\mathcal{O}((m \log U)(m + n \log n))$
Cost scaling algorithm	$\mathcal{O}(n^3 \log(nC))$
Double scaling algorithm	$\mathcal{O}(nm \log U \log(nC))$
Minimum mean cycle-canceling algorithm	$\mathcal{O}(n^2 m^3 \log n)$
Repeated capacity scaling algorithm	$\mathcal{O}((m^2 \log n)(m + n \log n))$
Enhanced capacity scaling algorithm	$\mathcal{O}((m \log n)(m + n \log n))$

3.7 Das Netzwerksimplex-Verfahren

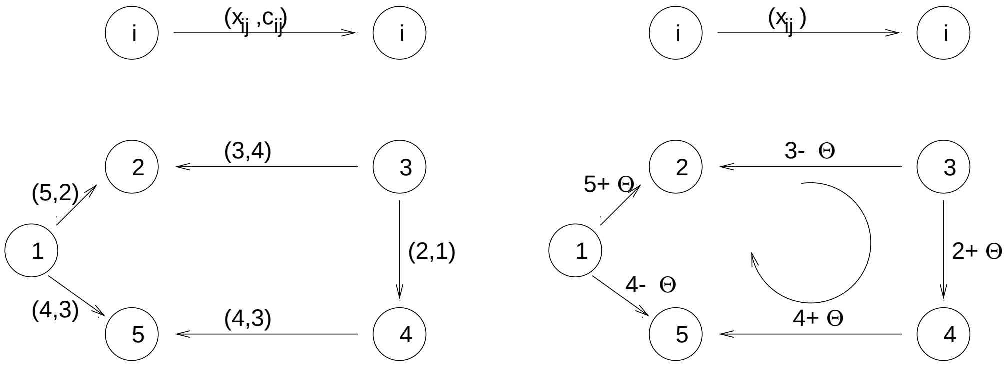
Bemerkungen

- Das MCFP läßt sich als linaeres Programm beschreiben und mittels allgemeinen Simplex-Algorithmus lösen. Diese Methode ist jedoch relativ ineffizient.
- Durch Ausnutzung der speziellen Struktur des MCFP ist es jedoch möglich den Simplex auf diese spezielle Problemstellung anzupassen. Die Arbeitsweise des Simplex wird in einem graphentheoretischen Kontext beschrieben.

Definitionen

- Für jede zulässige Lösung x, nennen wir eine Kante (i,j) eine *freie Kante*, falls $0 < x_{ij} < u_{ij}$ und *beschränkte Kante*, falls $x_{ij} = 0$ oder $x_{ij} = u_{ij}$.
- Eine zulässige Lösung heißt *zykelfreie Lösung*, falls das Netzwerk keinen Zykel enthält der ausschließlich aus freien Kanten besteht.
- Wir nennen eine zulässige Lösung verbunden mit einem spannenden Baum des Netzwerks eine *spannende Baum-Lösung*, falls jede Kante außerhalb des Baums beschränkt ist. Die Kanten im spannenden Baum dürfen frei oder beschränkt sein.

Betrachte folgende Situation (unkapazitierter Fall):



- Erhöhen wir den Fluß im Uhrzeigersinn um eine Einheit, dann reduzieren wir die Gesamtkosten um 1 Einheit. Wir können maximal $\Phi = 3$ wählen.
- Sei $c_{12} := 5$. Dann kostet uns eine Eröhung des Flusses um eine Einheit 2 Kosteneinheiten. Daher ist Φ so klein wie möglich zu wählen: $\Phi = -2$.
- In beiden Fällen erhalten wir einen Zykel der mindestens eine beschränkte Kante enthält.

Eigenschaft 1: Falls die Zielfunktion eines MCFP über einer zulässigen Region von unten beschränkt ist, dann besitzt das Problem immer eine optimale zykelfreie Lösung.

Bemerkungen:

- Eine zykelfreie Lösung läßt sich sehr einfach in eine spannende Baum-Lösung transferieren: In einer zykelfreien Lösung bilden die freien Kanten einen Wald. Bildet dieser Wald einen spannenden Baum, so haben wir eine spannende Baum-Lösung. Ansonsten nehme geeignete beschränkte Kanten hinzu. (Zu einer zykelfreie Lösung kann man im allgemeinen mehrere spannende Baum-Lösungen konstruieren.)

Eigenschaft 2: Falls die Zielfunktion eines MCFP über einer zulässigen Region von unten beschränkt ist, dann besitzt das Problem immer eine optimale spannende Baum-Lösung.

- Eine spannende Baum-Lösung teilt A in drei Untermengen:
  - $\mathbf{T}$, die Kanten des spannenden Baums.
  - $\mathbf{L}$, die Kanten außerhalb des spannenden Baums mit Fluß 0.
  - $\mathbf{U}$, die Kanten außerhalb des spannenden Baums mit saturierten Fluß.
- Das Tripel $(\mathbf{T}, \mathbf{L}, \mathbf{U})$ nennen wir die spannende Baumstruktur.
- Zu jeder spannenden Baumstruktur $(\mathbf{T}, \mathbf{L}, \mathbf{U})$ läßt sich eine spannende Baum-Lösung bestimmen (siehe unten). Eine spannenden Baumstruktur heißt zulässig (optimal), falls die dazugehörige spannende Baum-Lösung zulässig (optimal) ist.

- Falls jede Kante aus $\mathbf{T}$ frei ist, so nennen wir den spannenden Baum *nichtdegeneriert*, anderfalls *degeneriert*

Satz (MCF-Optimalitätsbedingungen): Die spannende Baumstruktur $(\mathbf{T}, \mathbf{L}, \mathbf{U})$ ist eine optimale Struktur für das MCFP, falls sie zulässig ist und für ein Potential π die reduzierten Kosten die folgenden Bedingungen erfüllen:

(a) $c_{ij}^{\pi} = 0$ for all $(i,j) \in \mathbf{T}$.

(b) $c_{ij}^{\pi} \geq 0$ for all $(i,j) \in \mathbf{L}$.

(c) $c_{ij}^{\pi} \leq 0$ for all $(i,j) \in \mathbf{U}$.

Beweis: Sei $x*$ die zu $(\mathbf{T}, \mathbf{L}, \mathbf{U})$ gehörige spannende Baumstruktur und es existiert ein Potential π das die Bedingungen erfüllt. Wir müssen zeigen, daß $x*$ eine optimale Lösung für das MCFP ist. Es gilt:

$$\text{Minimiere} \sum_{(i,j)\in A} c_{ij}x_{ij} \Leftrightarrow \text{ Minimiere} \sum_{(i,j)\in A} c_{ij}^{\pi}x_{ij} \Leftrightarrow \text{ Minimiere} \sum_{(i,j)\in \mathbf{L}} c_{ij}^{\pi}x_{ij} - \sum_{(i,j)\in \mathbf{U}} |c_{ij}^{\pi}|x_{ij}.$$

Die Definition von x^* impliziert, daß für jede beliebige Lösung x gilt: $x_{ij} \geq x_{ij}^*$ für alle $(i,j) \in \mathbf{L}$ und $x_{ij} \leq x_{ij}^*$ für alle $(i,j) \in \mathbf{U}$. Also ist die Zielfunktion x in jedem Fall größer oder gleich der von x^*. □

Algorithmusidee

- Beginne mit einer zulässigen spannenden Baustruktur und versuche durch jeweils Austausch einer Kante des Baums mit einer Kante außerhalb des Baums zu einer besseren spannenden Baustruktur zu kommen bis die optimale Baumstruktur erreicht wird, d.h. die MCF-Optimalitätsbedingungen erfüllt sind.

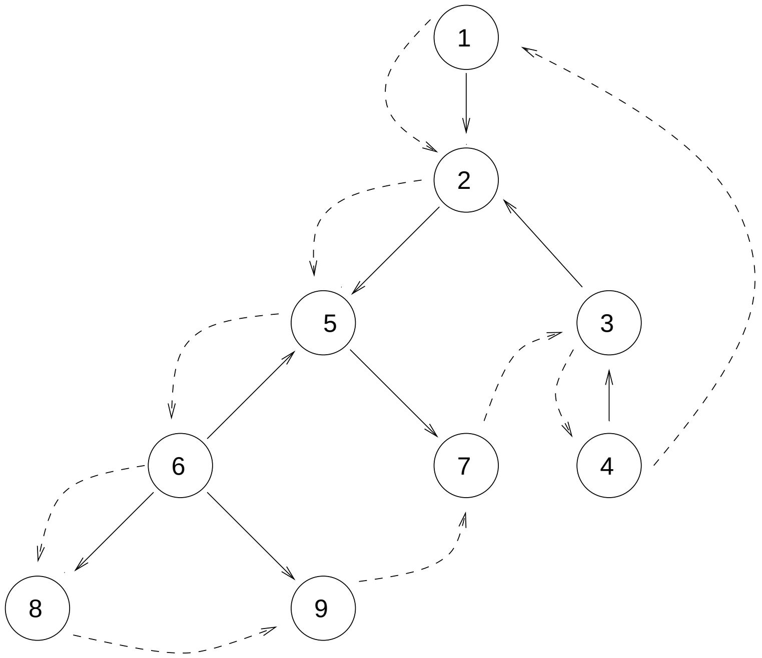
Verwalten einer spannenden Baumstruktur

Bei jedem Knoten werden drei Indizes gespeichert:

- **Vorgänger** *pred(i):* Jeder Knoten i besitzt einen eindeutigen Pfad zur Wurzel. *pred(i)* speichert den Vorgängerknoten von i.
- **Tiefe** *depth(i):* Anzahl der Kanten zwischen der Wurzel und Knoten i.
- **Gefädelter Index** *thread(i):* Index, der entsteht, wenn man bei der Wurzel startet und den Baum in depth-first-search-Ordnung durchläuft und jeweils den Nachfolgerknoten bei jedem Knoten bestimmt.

Anwendung: Angenommen man sucht alle Nachfolger von Knoten i, dann durchläuft man rekursiv den gefädelten Index ausgehend von *thread(i)* bis man zu einem Knoten gelangt, dessen Tiefe kleiner oder gleich der Tiefe von i ist.

Beispiel:



i	1	2	3	4	5	6	7	8	9
pred(i)	0	1	2	3	2	5	5	6	6
depth(i)	0	1	2	3	2	3	3	4	4
thread(i	2	5	4	1	6	8	3	9	7

Berechnen der Knotenpotentiale zu gegebener Baumstruktur

- Geforderte Eigenschaft: $c_{ij}^{\pi} = 0$ für alle $(i,j) \in T$.
- Addiert man eine Konstante auf alle Potentiale, so ändern sich die reduzierten Kosten nicht. Daher kann man $\pi(1) := 0$ setzen.
- Aus der Beziehung $c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j) = 0$ lassen sich alle Potential sukzessive im Baum berechnen.

```
procedure compute-potentials;
begin
    π(1) := 0;
    j :=thread(1);
    while j ≠ 1 do
    begin
        i := pred(j);
        if (i, j) ∈ A then π(j) := π(i) − c_ij;
        if (j, i) ∈ A then π(j) := π(i) + c_ji;
        j :=thread(j);
    end;
end;
```

- Laufzeit: $\mathcal{O}(n)$

Berechnen des Flusses zu gegebener Baumstruktur

- Betrachte unkapazitierten Fall: Löscht man eine Kante (i, j) aus dem spannenden Baum, so entstehen zwei Teilbäume. Sei T_1 der Baum der i enthält und T_2 der Baum der j enthält. Dann gilt: $\sum_{k \in T_1} b(k) = -\sum_{k \in T_2} b(k)$. Da im spannenden Baum nur Kante (i, j) die beiden Teilbäume verbindet, müssen darauf $\sum_{k \in T_1} b(k)$ Einheiten transportiert werden.

- Geschicktes Berechnen der Flüsse, indem man eine Kante wählt, so daß ein Teilbaum immer ein einzelner Knoten ist.

```
procedure compute-flows;
begin
    b'(i) := b(i), for all i ∈ N;
    for each (i, j) ∈ U do
        set x_ij := u_ij subtract u_ij from b'(i) and add u_ij to b'(j);
    for each (i, j) ∈ L do set x_ij = 0;
    T' := T;
   while  T' ≠ {1} do
   begin
        select a leaf node j (other than node 1) in the subtree T';
        i :=pred(j);
        if (i, j) ∈ T' then x_ij := −b'(j)
        else x_ij := b'(j);
        add b'(j) to b'(i);
        delete node j and the arc incident to it from T';
   end;
end;
```

- Ein Blatt läßt sich in $\mathcal{O}(1)$ bestimmen, wenn wir den thread-Index in umkehrter Richtung durchlaufen. (Zu Beginn thread-Index durchlaufen und Knoten auf Stack speichern.)
- Laufzeit: $\mathcal{O}(m)$

Der Algorithmus

algorithm network simplex;
begin
  determine an initial feasible tree structure $(\mathbf{T}, \mathbf{L}, \mathbf{U})$;
  let x be the flow and π be the node potentials associated with this tree structure;
  while some nontree arc violates the optimality conditions **do**
  begin
    select an entering arc (k, l) violating its optimality condition;
    add arc (k, l) to the tree and determine the leaving arc (p, q);
    perform a tree update and update the solutions x and π;
  end;
end;

Bemerkungen

- Initiale Lösung läßt sich z.B. durch Berechung eines Maximalen Flusses mit anschließender Konvertierung zu einer spannenden Baum-Lösung.
- Optimale Lösung gefunden, falls $c_{ij}^{\pi} \geq 0$ für jede Kante $(i, j) \in L$ und $c_{ij}^{\pi} \leq 0$ für jede Kante $(i, j) \in U$.
- Eine die Optimalität verletzende Kante aus U oder L wird als die einzufügende gewählt.
- Welche Kante (die eine MCF-Optimalitätsbedingung verletzt) soll gewählt werden? Verschiedene Regeln:
  - Dantzig's Pivot-Regel: Wähle Kante, bei der die „Violation“ $|c_{ij}^{\pi}|$ maximal ist. Vorteil: Relativ große Abnahme des Zielfunktionwerts pro Iteration. Nachteil: Zu aufwendige Suche.
  - Nehme die erste brauchbare Kante. Zyklische Abarbeitung der Kantenliste. (Leider zu viele Iterationen mit einer zu geringen Reduzierung der Zielfunktion!)
  - Eine Mischung aus den oberen beiden Regeln. Zwei Phasen: In Hauptiterationen werden potentielle Kandidaten in einer Liste gesammelt. In der folgenden Subiterationen wird aus der Liste immer der Kandidat mit der größten Violation gesucht. Subiterationen enden, wenn Liste leer wird.

Bestimmung der zu verlassenden Kante

- Hinzunahme einer Kante erzeugt genau einen Zykel mit spannenden Baum.

```
procedure identify-cycle;
begin
   i := k and j := l;
   while i ≠ j do
   begin
      if depth(i) > depth(j) then i := pred(i)
      else if depth(j) > depth(i) then j := pred(j)
         else i := pred(i) and j := pred(j);
   end;
end;
```

- War hinzugenommene Kante aus L, so senden wir Einheiten in Richtung des neuen Pfeils und reduzieren den Zielfunktionswert pro Einheit um $|c_{ij}^{\pi}|$. War die Kante aus U, so augmentieren wir in der Gegenrichtung.

- Der Fluß wird solange erhöht, bis eine oder mehrere Kante beschränkt wird. Eine dieser Kanten wird je nach Beschränkung zu U oder L zugeordnet.

- Laufzeit des Flußupdates: $\mathcal{O}(n)$.

Potentialupdate

- Das Herauslösen der Kante (p, q) zerteilt den ursprünglichen Baum T in zwei Teile T_1 und T_2, wobei T_1 die Wurzel von T enthalte.

- Die Potentiale in T_1 bleiben erhalten. Dagegen ändern sich alle Potentiale in T_2 um einen konstanten Wert.

```
procedure update-potentials;
begin
   if q ∈ T2 then y := q else  y := p;
   if k ∈ T1 then change := −c_kl π else  change := c_kl π;
   π(y) := π(y)+ change;
   z :=thread(y);
   while depth(z) > depth(y) do
   begin
      π(z) := π(z)+ change;
      z := thread(z);
   end;
end;
```

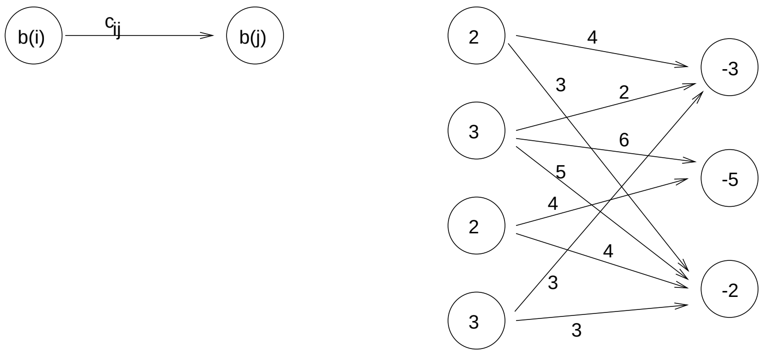
- Der Update der Indizes ist in $\mathcal{O}(|W| + \min\{|T_1|, |T_2|\})$ möglich.

Terminierung

- Falls jede Pivot-Operation nichtdegeneriert ist, das heißt, es läßt sich mindestens eine Einheit auf einem Pivotzykel augmentieren, dann terminiert das Verfahren sicher.
- Anderfalls besteht die Gefahr des Kreisens (cycling): Unendliche Folge von degenerierten Pivot-Operationen.
- Spezielle Implementierung verhindert das:
  strongly feasible spanning tree implementation.
  (Ein spannender Baum heißt verschärft zulässig, falls jede Baumkante mit Fluß null Richtung Wurzel zeigt und jede Kante mit saturiertem Fluß von der Wurzel wegzeigt.)
  Das Verfahren benutzt nur verschärft zulässige spannende Bäume.
  Neue „Leaving Arc Rule“: Wähle als verlassende Kante die letzte auftretende blockierende Kante die gefunden wird, wenn man den Pivotzykel in seiner Orientierung durchläuft und beim Apex w startet. (Der Apex ist der erste gemeinsame Vorfahre von Anfangs- und Endpunkt der eingefügten Kante im Baum.)

3.8 Das Transportproblem(TP)

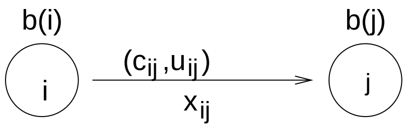
- MCFP-Netzwerk $G = (N_1 \cup N_2, A)$ ist ein gerichteter, bipartiter Graph.
- N_1: Angebotsknoten; N_2: Nachfrageknoten; keine Umladeknoten
- Es gibt keine Kapazitätsschranken $(u_{ij} = \infty)$.

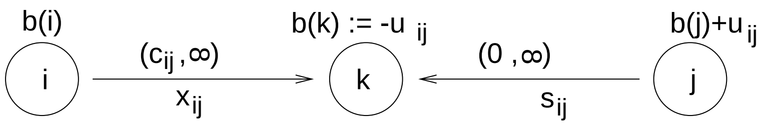


- Spezialfall: Zuordnungs- bzw. Assignmentproblem (Bipartites gewichtetes Matchingproblem):
  $|N_1| = |N_2|$, $b(i) = 1$ für alle $i \in N_1$ und $b(i) = -1$ für alle $i \in N_2$
  Effiziente Lösungsmethode: Ungarisches Verfahren (Primal-duales Verfahren)
  Laufzeit: $\mathcal{O}(|N_1| S(n, m, C))$.

Satz: Jedes TP läßt sich in ein MCFP transferieren und umgekehrt.

Beweis: „$\Longrightarrow$“ offensichtlich.
„$\Longleftarrow$“ Gegeben MCFP-Netzwerk G=(N,A). Transformation jeder Kante:

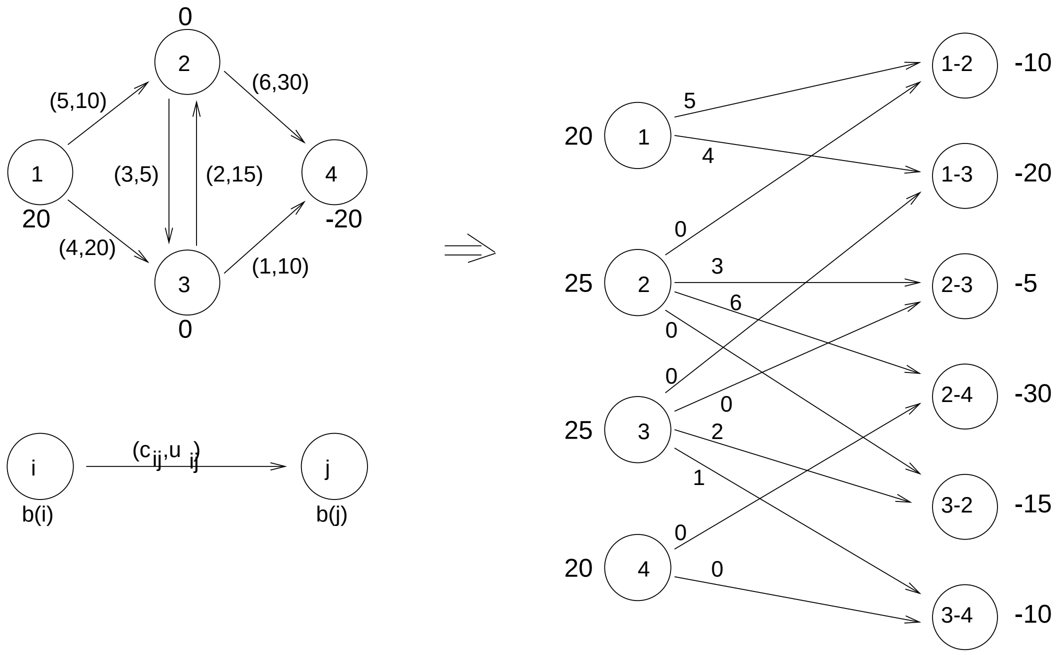




Entspricht Einführung einer Slackvariablen s_{ij} für alle $(i, j) \in A$. Kapazitätsbedingung: $x_{ij} \leq u_{ij} \Rightarrow x_{ij} + s_{ij} = u_{ij} \Rightarrow -x_{ij} - s_{ij} = -u_{ij}$ Diese letzte Gleichung kann man als lokale Gleichgewichtsgleichung eines neuen Knoten k interpretieren. Transformation des MCFP-Netzwerks G mit Kapazitäten zu einem bipartiten unkapazitierten Grpahen G' mit

$(n+m)$ Knoten und $2m$ Kanten.
Fluß im alten und neuen Netzwerk:
x_{ij} auf Kante (i,j) in G
x_{ij} auf Kante (i,k) und $u_{ij} - x_{ij}$ auf Kante (j,k) in G'.
$\Rightarrow$ Gleiche Kosten in G und G'. $\square$

Beispiel



- Zur Lösung von Transportproblemen existieren spezielle Adaptionen der herkömmlichen MCFP-Verfahren (z.B. Netzwerksiplex).
- Ist $|N_1| << |N_2|$ oder umgekehrt, so lassen sich die allgemeinen Verfahren beschleunigen.
- Sind Anbieter und Nachfrager in der Ebene eingebettet, so läßt sich die geometrische Struktur zur Gewinnung effizienter Verfahren ausnutzen.