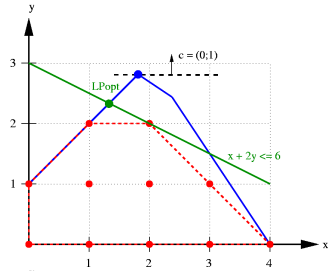
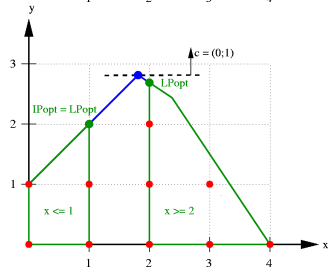
Kapitel 5

Branch-and-Cut





Inhalt

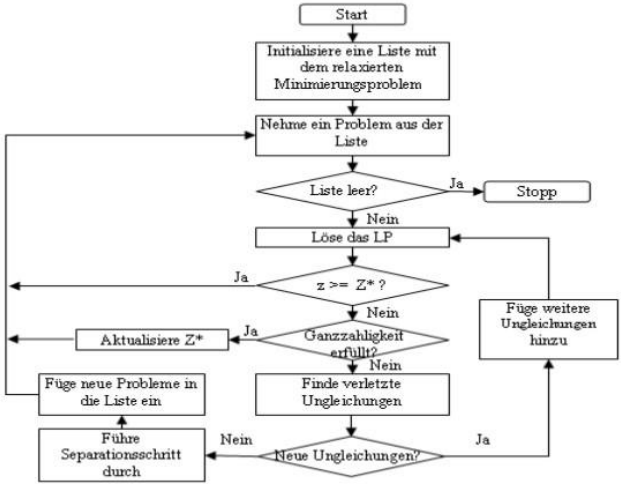
Branch-and-Cut

- Verwende Branch-and-Bound als übergeordneten Algorithmus.
- Nutze LP-Relaxationen zur Bestimmung oberer Schranken für die Teilprobleme P_i.
- Falls Lösung der LP-Relaxation nicht zu einer Auslotung führt:
  - ☞ Versuche durch Schnittebenen die obere Schranke zu verringern.
  - ☞ Im Idealfall führt dies zu einem ausgeloteten Teilproblem P_i.
- Branching erst dann, wenn weitere Schnittebenen "nichts mehr bringen".
- Als Schnittebenen nutzen wir i. d. R. problemspezifische gültige Ungleichungen.

Branch-and-Cut-Algorithmus

Algorithmus 5.1 (für Maximierungsprobleme)

1. $\mathcal{P} := \{\text{LP-Relaxation von } P\}$
   $z^* :=$ *Wert einer heuristischen Lösung (oder* $-\infty$*)*
2. $\mathcal{P} = \emptyset$*? Wenn ja, dann STOP!*
   Wähle $P \in \mathcal{P}$*.* $\mathcal{P} := \mathcal{P} \setminus \{P\}$
3. *Löse* P*. Sei* z *der zugehörige Zielfunktionswert.*
4. $z \leq z^*$*? Wenn ja, dann gehe zu 2.*
5. *Ist die Lösung von* P *ganzzahlig? Wenn, ja dann aktualisiere eventuell* z^* *und gehe zu 2.*
6. *Suche eine verletzte gültige Ungleichung.*
   Wenn Ungleichung gefunden, dann füge diese P *hinzu und gehe zu 3.*
7. *Wähle aus der Lösung von* P *eine gebrochene Variable* x_i *und erzeuge damit zwei neue Probleme* P_1 *und* P_2*.*
   Setze $\mathcal{P} := \mathcal{P} \cup \{P_1, P_2\}$ *und gehe zu 2.*


Start
Initialisiere eine Liste mit dem relaxierten Minimierungsproblem
Nehme ein Problem aus der Liste
Liste leer?
Ja
Stopp
Nein
Löse das LP
z >= Z* ?
Ja
Nein
Füge weitere Ungleichungen hinzu
Aktualisiere Z*
Ja
Ganzzahligkeit erfüllt?
Nein
Füge neue Probleme in die Liste ein
Finde verletzte Ungleichungen
Führe Separationsschritt durch
Nein
Neue Ungleichungen?
Ja

Diskussion Branch-and-Cut

- Der Unterschied zum reinen Branch-and-Bound besteht im Schritt 6.
- Hier werden üblicherweise Ungleichungen verwendet, die speziell für eine Problemklasse sind.
- Separationsproblem: Welche gültigen Ungleichungen werden von einer optimalen Lösung einer LP-Relaxation verletzt?
- Die Effizienz eines Branch-and-Cut-Algorithmus wird maßgeblich von der Güte der erkannten verletzten Ungleichungen und der Effizienz der Separation beeinflusst.

Wir untersuchen den Branch-and-Cut-Ansatz am Beispiel des Traveling-Salesman-Problems.

TSP als lineares Programm: Variablen und Zielfunktion

Vollständiger Graph (V, E) mit Knotenmenge $V = \{v_1, \ldots, v_n\}$.

c_{ij} bzw. c_e sei die Länge der Kante $e = \{v_i, v_j\}$.

Wegen Symmetrie gelte stets $i < j$.

$$x_{ij} = x_e = \begin{cases} 1 & \text{Kante } e = \{v_i, v_j\} \text{ ist in Tour enthalten} \\ 0 & \text{sonst} \end{cases}$$

Zielfunktion:

$$\min \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij} x_{ij} \quad \text{bzw.} \quad \min \sum_{e \in E} c_e x_e$$

Menge der Schnittkanten

Definition 5.2

Es sei $G = (V, E)$ ein Graph und $S \subseteq V$. Die Menge

$$\delta(S) = \{e = \{v, w\} \in E | v \in S, w \in V \setminus S\}.$$

heißt Menge der Schnittkanten von S.

Für einen einzelnen Knoten $v \in V$ schreiben wir kurz $\delta(v) := \delta(\{v\})$.

Bemerkung: $\delta(v)$ besteht aus allen Kanten, die inzident zu v sind.

Beispiel 5.3

Beispiele für die Menge der Schnittkanten. ✎

Bezeichnungen für TSP

Definition 5.4

Es sei $G = (V, E)$ ein vollständiger Graph.

Für eine Kantenteilmenge $F \subseteq E$ bezeichne

$$x(F) = \sum_{e \in F} x_e.$$

Für eine Knotenteilmenge $S \subseteq V$ bezeichne

$$x(S) = x(E(S)) \text{ mit } E(S) = \{e = \{v, w\} | v, w \in S\}.$$

Gültige Ungleichungen für das TSP (1)

In einer TSP-Tour ist jeder Knoten inzident mit genau zwei Kanten.

Für $i = 1, \ldots, n$:

$$\sum_{j=1}^{i-1} x_{ji} + \sum_{j=i+1}^{n} x_{ij} = 2,$$

oder kurz: für alle $v \in V$ gilt

$$x(\delta(v)) = 2.$$

Grad-Gleichungen

Gültige Ungleichungen für das TSP (2)

Jede Kante ist an einer TSP-Tour mit einem Gewicht von höchsten 1 beteiligt.

Für $1 \leq i < j \leq n$:

$$0 \leq x_{ij} \leq 1,$$

bzw. für alle $e \in E$ gilt:

$$0 \leq x_e \leq 1.$$

Variablenbeschränkungen (triviale Ungleichungen)

Gültige Ungleichungen für das TSP (3)

Für jede echte Teilmenge $S \subsetneqq V$ ist der induzierte Untergraph $T(S)$ einer TSP-Tour T kreisfrei, enthält also höchstens $|S| - 1$ Kanten.

Für $3 \leq |S| \leq n - 1$:

$$\sum_{v_i, v_j \in S, i<j} x_{ij} \leq |S| - 1,$$

oder kurz

$$x(S) \leq |S| - 1.$$

Subtour Elimination Constraints (SEC)

Diskussion Subtour Elimination Constraints

- Alternative Formulierung:
$$x(\delta(S)) \geq 2.$$
- Erste Formulierung besser für „kleine“ S, zweite für „große“ S.
- Es genügt, SECs für $3 \leq |S| \leq \frac{n}{2}$ zu formulieren.
- Problem: Die Anzahl der Subtour Eliminitation Constraints wächst exponentiell in n.
- Lösungsansatz:
  - Beginne nur mit den Grad-Gleichungen als LP-Relaxation.
  - Falls die Lösung der LP-Relaxation eine der gültigen Ungleichungen verletzt, nehme diese zur Relaxation hinzu.
- Mit diesem Ansatz lösen wir nun einige TSP-Probleme.

Beispiele für TSP mit Schnittebenen (1)

Beispiel 5.5

TSP von Folie 238.

LP: Grad-Ungleichungen für alle Knoten

$z = 189$

Verletzte Ungleichungen:

- Variablenbeschränkungen für: AaDü, BeHa, MüNü
- Subtour für: Ba – Fr – St

Fortsetzung Beispiel.

Nehme entsprechende Ungleichungen hinzu.

$z = 250$

- ☞ Optimale Lösung der LP-Relaxation ist (die bekannte) zulässige TSP-Tour!
- ☞ Branching war nicht erforderlich!

Beispiele für TSP mit Schnittebenen (2)

Beispiel 5.6

Beispiel mit $n = 17$ aus TSPLIB (gr17). ✎ Tafel.

LP-Relaxationen siehe Homepage.

Wiederum ist kein Branching notwendig.

Beispiel 5.7

Beispiel mit $n = 13$ aus TSPLIB (gr13). ✎ Tafel.

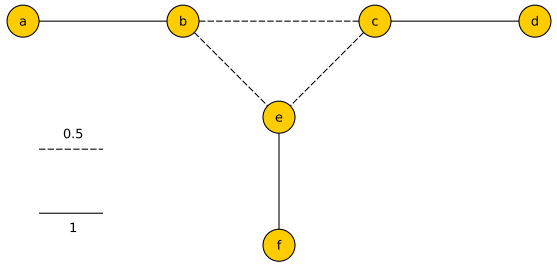
Im Gegensatz zu den beiden vorangegangenen Beispielen treten hier Brüche bei den optimalen Lösungen der Relaxation auf.

Trotzdem genügen die bekannten Ungleichungen, um das das Problem zu lösen.

LP-Relaxationen siehe Homepage.

2-Matching-Ungleichungen (1)

Bei der Lösung einer TSP-Relaxation kann die folgende Situation (als Teil der gesamten Lösung) auftreten.



Hier ist keine der uns bekannten gültigen Ungleichungen verletzt. Was tun?

2-Matching-Ungleichungen (2)

Von den sechs Kanten können höchstens vier an einer TSP-Tour beteiligt sein.

- Es können maximal zwei der Kanten $\{b, c\}, \{b, e\}, \{c, e\}$ enthalten sein, sonst wäre die SEC für $S = \{b, c, e\}$ verletzt.
- Wenn zwei Kanten von $\{b, c\}, \{b, e\}, \{c, e\}$ enthalten sind, z. B. $\{b, c\}$ und $\{c, e\}$, dann ist eine äußere Kante nicht enthalten, hier $\{c, d\}$.
- Damit folgt:

$$x_{a,b} + x_{b,c} + x_{b,e} + x_{c,e} + x_{c,d} + x_{e,f} \leq 4$$

ist eine gültige Ungleichung, die die dargestelle Situation separiert.

2-Matching-Ungleichungen (3)

Zur Definition einer 2-Matching-Ungleichung benötigen wir:

1. eine Knotenmenge (handle) $H \subseteq V$ mit $3 \leq |H| \leq |V| - 1$,
2. eine Kantenmenge (teeth) $T \subseteq \delta(H)$ mit
   - einer ungeraden Anzahl an Kanten,
   - die keinen Knoten gemeinsam haben, also $e \cap e' = \emptyset$ für alle $e, e' \in T$ mit $e \neq e'$.

Die 2-Matching-Ungleichung lautet dann:

$$x(H) + x(T) \leq |H| + \left\lfloor \frac{|T|}{2} \right\rfloor .$$

Herleitung der 2-Matching-Ungleichungen

- Vorzeichenbedingungen für alle $e \in \delta(H) \setminus T$:

$$-x_e \leq 0.$$

- Variablenbeschränkungen für alle $e \in T$:

$$x_e \leq 1.$$

- Gradgleichungen für alle $v \in H$ als $\leq$-Ungleichung:

$$x(\delta(v)) \leq 2.$$

Wir multiplizieren alle Gleichungen mit $\frac{1}{2}$ und summieren auf.

Damit erhalten wir:

$$\begin{array}{rcrcrcr} & & -\frac{1}{2}x(\delta(H) \setminus T) & & & \leq & 0 \\ & & & & \frac{1}{2}x(T) & \leq & \frac{1}{2}|T| \\ x(H) & + & \frac{1}{2}x(\delta(H) \setminus T) & + & \frac{1}{2}x(T) & \leq & |H| \end{array}$$

Die Summation dieser drei Ungleichungen liefert:

$$x(H) + x(T) \leq |H| + \frac{|T|}{2}.$$

Jetzt können wir die rechte Seite abrunden.

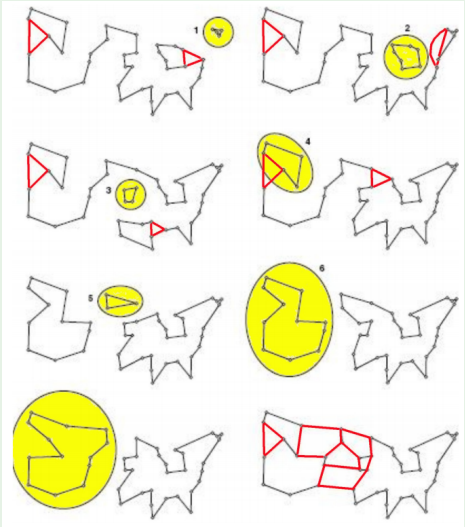
Beispiel 5.8 (Dantzig, Fulkerson, Johnson (1954))

Problem mit 42 US-Städten aus dem Jahr 1954.

schwarz $= 1$, rot $= \frac{1}{2}$

Sieben SECs liefern die Tour unten rechts.

Hier ist keine SEC mehr verletzt, aber eine 2-Matching-Ungleichung.

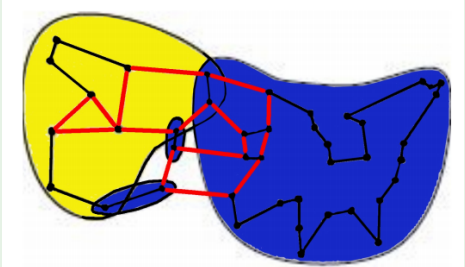


Fortsetzung Beispiel.

Die Grafik oben zeigt die verletzte 2-Matching Ungleichung.

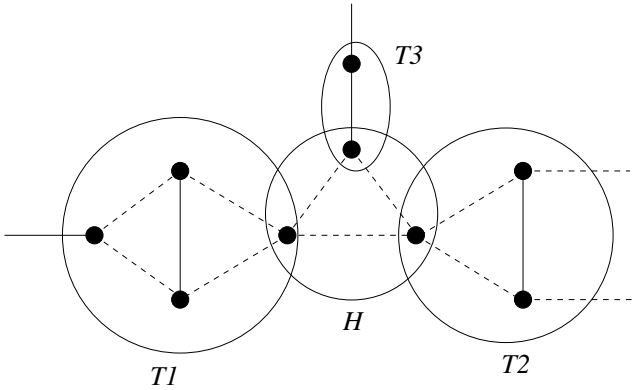
Nach Separation erhalten wir die Lösung unten.

Die hier verletzte Ungleichung ist eine Verallgemeinerung einer 2-Matching-Ungleichung.



Comb-Ungleichungen (1)

2-Matching-Ungleichungen sind ein Spezialfall von Comb-Ungleichungen.



Comb-Ungleichungen (2)

Für eine Comb-Ungleichung benötigen wir:

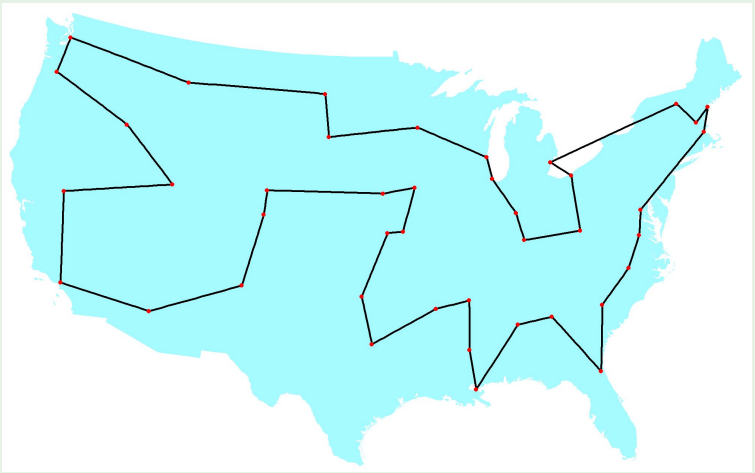
1. eine Knotenmenge (handle) $H \subseteq V$
2. eine ungerade Anzahl an Knotenmengen (teeth) $T_1, \ldots, T_k \subseteq V$ für die gilt:
   - $T_i \cap T_j = \emptyset$ für $1 \leq i < j \leq k$,
   - $H \cap T_i \neq \emptyset$ für $1 \leq i \leq k$,
   - $T_i \setminus H \neq \emptyset$ $1 \leq i \leq k$.

Die Comb-Ungleichung lautet dann:

$$x(H) + \sum_{i=1}^{k} x(T_i) \leq |H| + \sum_{i=1}^{k} |T_i| - \left\lceil \frac{3k}{2} \right\rceil .$$

Beispiel 5.9

Separation der Comb-Ungleichung von Beispiel 5.8 führt zur optimalen Lösung.



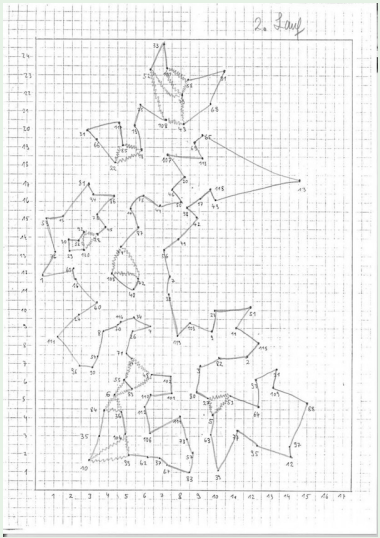
Beispiel 5.10 (Grötschel (1977))

Grötschel 1977, 120 Städte in Deutschland, damals Weltrekord

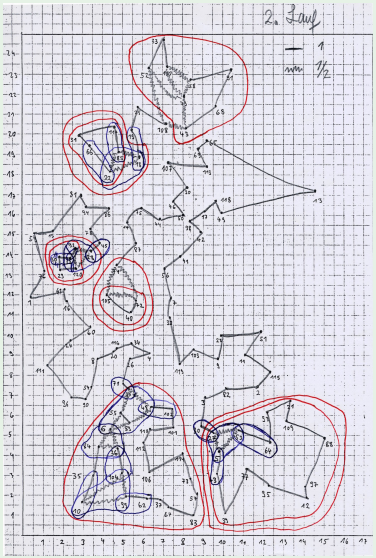
LP-Solver löst die Relaxationen, die dann gezeichnet werden.

Anhand der Zeichnung werden neue verletzte Ungleichungen identifiziert.

Welche SECs und 2-Matching-Ungleichung können Sie erkennen?



Fortsetzung Beispiel.



Separationsverfahren

- Für die effiziente Lösung eines TSP müssen wir eine verletzte gültige Ungleichung algorithmisch effizient erkennen können.
- Hierzu können Heuristiken und exakte Verfahren eingesetzt werden.
- Grundlage für diese Verfahren ist der stützende Graph für die Lösung einer Relaxation.
- Die Separation von verletzten Variablenbeschränkungen ist offensichtlich trivial.
- Im Folgenden betrachten wir die Separation von SECs.

Support Graph

Definition 5.11

Es sei der vollständige Graph $G = (V, E)$ für ein TSP und $\mathbf{x}$ die optimale Lösung einer LP-Relaxation.

Dann heißt der Graph $G(\mathbf{x}) = (V, E')$ mit

$$E' = \{e \in E | x_e > 0\}$$

stützender Graph (support Graph) von $\mathbf{x}$.

Wir ordnen dabei der Kante $e \in E'$ das Gewicht x_e zu.

- Der Stützgraph enthält also genau die Kanten, die zu Entscheidungsvariablen mit einem positiven Wert gehören.
- Wir haben in unseren Beispielen schon implizit mit stützenden Graphen gearbeitet.

Separation von SECs: Zusammenhang

Wir beginnen mit einer sehr einfachen Heuristik.

Lemma 5.12

Es sei $G(\mathbf{x})$ ein stützender Graph.

Dann gilt: Wenn $G(\mathbf{x})$ nicht zusammenhängend ist, dann bildet jede ZHK eine verletzte SEC.

Folgerung:

- Mittels Tiefen- oder Breitensuche können wir effizient verletzte SECs erkennen.
- In Beispiel 5.8 trifft dies auf 6 von 7 SECs zu.

Separation von SECs: Schnitt

Grundlage für alle exakten Separationsverfahren für SECs ist das folgende Lemma.

Lemma 5.13

Es sei $G(\mathbf{x})$ ein stützender Graph.

Die SEC für eine Knotenmenge S ist genau dann verletzt, wenn

$$x(\delta(S)) < 2$$

gilt, d. h. die Schnittkanten zu S haben in $G(\mathbf{x})$ eine Kapazität (Gewicht) < 2.

Folgerung:

- Wir berechnen einen minimalen Schnitt für $G(\mathbf{x})$.
- Wenn dieser Schnitt eine Kapazität < 2 hat, haben wir eine verletzte SEC gefunden.

Schnitte berechnen

- gerichteter Graph mit ausgezeichneter Quelle s und Senke t: Ford-Fulkerson- bzw. Edmonds-Karp-Algorithmus zur Berechnung eines Maximalflusses bzw. eines minimalen Schnitts.
- prinzipiell möglich: Wir ersetzen im Stützgraph $G(\mathbf{x})$ jede ungerichtete Kante durch zwei gerichtete Kanten und
- lösen das Maximalflussproblem für jede mögliche s, t-Menge.
- also: $\frac{n(n-1)}{2}$ Fluss-/Schnittprobleme
- ist polynomiell!
- Verbesserung mit dem Gomory-Hu-Algorithmus auf $n-1$ Fluss-/Schnittprobleme.

Reduktionsverfahren für SECs (1)

Insbesondere für eine exakte Separation von SECs bietet es sich an, den Stützgraph vor Berechnung der Schnitte zu „schrumpfen“.

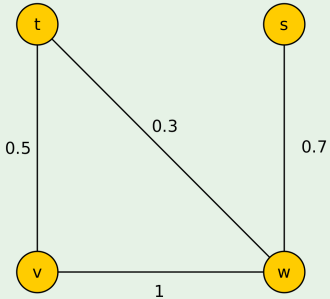
Algorithmus 5.14 (Padberg/Crowder)

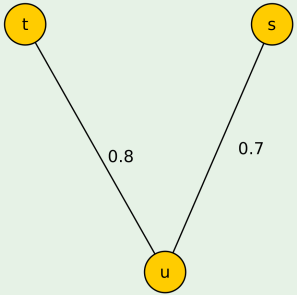
Solange es Kanten $e = \{v, w\}$ mit Gewicht $x_e = 1$ gibt, wiederhole:

1. *Wähle eine Kante $e = \{v, w\}$ mit Gewicht 1.*
2. *Ersetze die Kante e durch einen neuen Knoten u.*
3. *Wenn es einen oder mehrere Knoten t gibt, die zu beiden Knoten v und w adjazent sind, dann ersetze die Kanten $\{v, t\}$ und $\{w, t\}$ durch eine Kante $\{u, t\}$ mit Gewicht $x_{\{u,t\}} = x_{\{v,t\}} + x_{\{w,t\}}$.*
4. *Wenn es einen oder mehrer Knoten t gibt, die entweder nur zu v oder nur zu w adjazent sind, so ersetze die jeweilige Kante durch $\{u, t\}$ mit dem gleichen Gewicht wie zuvor.*

Reduktionsverfahren für SECs (2)

Beispiel 5.15





Vor-und Nachteile von Branch-and-Cut

Vorteile:

- Für viele Probleme führt der Branch-and-Cut-Ansatz zu vergleichsweise effizienten Lösungsalgorithmen.
- Für die Lösung der LP-Relaxationen stehen leistungsfähige LP-Solver zur Verfügung.

Nachteile:

- hoch spezialisierter Ansatz: Schnittebenen und Separation sind problemspezifisch.
- theoretische Hürde: Es müssen leistungsfähige Schnittebenen bekannt sein, in Verbidung mit effizienten Separationsalgorithmen.
- hoher Implementierungsaufwand: Die Separationsalgorithmen selbst können sehr komplex sein.

Branch-and-Cut mit rein ganzzahligen Lösungen (1)

Problem: Wir wären gerne in der Lage, schnell einen Prototyp zu implementieren, um die Leistungsfähigkeit eines Branch-and-Cut-Ansatzes für ein Optimierungsproblem einschätzen zu können.

Idee:

- Einsatz moderner leistungsfähiger Solver für die ganzzahlige Programmierung.
- Wir nutzen die eingebauten Techniken der Solver, um die LP-Relaxationen optimal ganzzahlig zu lösen.
- Separation nur auf den ganzzahligen Lösungen

Branch-and-Cut mit rein ganzzahligen Lösungen (2)

Das beschriebene Verfahren eignet sich für kombinatorische Optimierungsprobleme mit einer großen Anzahl an Nebenbedingungen in der LP-Formulierung (Beispiel: TSP).

Algorithmus 5.16

Es sei I eine Probleminstanz eines kombinatorischen Optimierungsproblems.

1. *Lege eine LP-Relaxation P für I fest.*
2. *Bestimme eine ganzzahlige Lösung $\mathbf{x}$ für P.*
3. *Ist $\mathbf{x}$ zulässig für I? Wenn ja, dann STOP!*
4. *Bestimme für $\mathbf{x}$ eine verletzte gültige Ungleichung und füge diese P hinzu.*

   Gehe zu 2.

Beispiel TSP

Wenn wir nur ganzzahlige Lösungen der LP-Relaxationen betrachten,

- dann ist durch Gradgleichungen und SECs das Problem vollständig beschrieben.
- Die Variablenbeschränkungen entsprechen SECs auf zwei Knoten.
- Die Separation für SECs ist sehr einfach zu implementieren.

Fazit: Geringer Aufwand zur Implementierung eines TSP-Solvers.

Mit diesem einfachen Ansatz kann man schon erstaunliches erreichen.

Anwendung TSP

Beispiel 5.17

Lösungszeiten (real time) für TSP-Probleme auf i5-8250U CPU (4 Kerne/8 Threads) Notebook mit Gurobi 9 als ILP-Solver:

Problem	#Städte	Zeit
gr21	21	0,016s
fri26	26	0,032s
gr48	48	0,312s
gr96	96	1,636s
rd100	100	0,558s
gr120	120	2,079s

Problem	#Städte	Zeit
bier127	127	0,679s
gr202	202	5,367s
gr229	229	19,521s
rd400	400	1m30,070s
gr431	431	7m37,847s
gr666	666	3m45,436s

Alle Probleme stammen aus der TSPLIB.

Linear Ordering Problem

Gegeben seien n Objekte $o_1, \ldots, o_n$, die in eine Reihenfolge gebracht werden sollen.

Gesucht ist also eine Permutation der n Objekte.

Wenn in der Permutation Objekt o_i (irgendwo) vor Objekt o_j steht, dann fällt Nutzen/Kosten c_{ij} an.

Welche Reihenfolge ist optimal?

Linear Ordering Problem als LP

Variablen

$$x_{ij} = \begin{cases} 1 & o_i \text{ ist vor } o_j \\ 0 & \text{sonst} \end{cases}$$

Maximiere

$$\min \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} c_{ij} x_{ij}$$

unter den Nebenbedingungen

$$x_{ij} + x_{ji} = 1 \text{ für alle } i < j$$
$$x_{ij} + x_{jk} + x_{ki} \leq 2 \text{ für alle } i < j, i < k, j \neq k$$
$$x_{ij} \in \{0, 1\}.$$

Die Ungleichungen heißen Dicycle-Ungleichungen.

Diskussion Linear Ordering Problem

- LOP ist (als Entscheidungsproblem) NP-vollständig.
- $O(n^3)$ viele Dicycle-Ungleichungen, problematisch für größere n
- einfacher Branch-and-Cut-Ansatz:
  - beginne nur mit den Gleichungen $x_{ij} + x_{ji} = 1$,
  - prüfe in einer ganzzahligen Lösung, ab alle Dicycle-Ungleichungen erfüllt sind und
  - füge verletzte Dicycle-Ungleichungen der Relaxation hinzu.

LP-Solver-Frameworks

- Die modernen LP-Solver bieten ein Framework für die Entwicklung von Algorithmen an.
- Man kann nicht nur einen LP-Solver nutzen, sondern über das Framework selbst in die Lösung eingreifen.
- Hierzu registriert man beim Solver eine Callback-Funktion, die bei bestimmten Ereignissen aufgerufen wird.
- Die Callback-Funktion kann dazu genutzt werden
  - Schnittebenen zu erzeugen und
  - heuristische Lösungen zu generieren.

Beispiel Gurobi

Reference Manual

- Monitoring Progress - Logging and Callbacks
  - `GRBsetcallbackfunc`
  - `GRBcbget`
- Modifying Solver Behavior - Callbacks
  - `GRBcbcut`
  - `GRBcblazy`
  - `GRBcbsolution`
- Callback Codes
  - `MIPNODE`
  - `MIPSOL`

Examples: Gurobi TSP Beispiel

Zusammenfassung

- Branch-and-Cut: Branch-and-Bound in Verbindung mit problemspezifischen Schnittebenen und effizienter Separation
- sehr erfolgreich bei der Suche nach optimalen Lösungen
- Schnittebenen für TSP: SECs, 2-Matching- und Comb-Ungleichungen
- hoher Implementierungsaufwand
- praktischer Ansatz:
  - Separation rein ganzzahliger Lösungen
  - Nutzung leistungsfähiger LP-Solver als Branch-and-Cut-Framework