Einführung in die Semidefinite Programmierung

Daniel Borchmann
Sommerakademie Görlitz 2007

24. September 2007

Die Lösung allgemeiner Optimierungsaufgaben stellt sich in der Praxis als häufig nicht durchführbar dar, so dass eine direkte Behandlung solcher Aufgaben nicht sinnvoll erscheint. Die geläufige Methode der Relaxation auf linerare Programme stellt dabei eine meist zu starke Abschwächung des Problems dar. Dieser Nachteil motivierte die Entwicklung der semidefiniten Programmierung, die eine meist vielversprechendere Relaxation erlaubt und erstmalig zu signifikanten Verbesserungen bei der Behandlung von MAX-CUT und MAX-2SAT seit über 20 Jahren geführt hatte.

1 Einführung

Wir wollen im folgenden Optimierungsaufgabe der Form

$$\begin{array}{rl} \text{maximiere} & f(x) \\ \text{so dass} & f_i(x) = b_i \text{ für } i \in \{1, \dots, m\} \\ & h_i(x) \geq 0 \text{ für } i \in \{1, \dots, l\} \end{array} \tag{1}$$

mit $f, f_i, h_i \colon \mathbb{R} \longrightarrow \mathbb{R}$ und $m, l \in \mathbb{N}$ betrachten, wobei f die *Zielfunktion* und f_i, h_i die *Nebenbedingungen* bezeichnen sollen. Die *Menge aller zulässigen Lösungen* G sei dabei definiert durch

$$G := \{x \mid x \text{ erfüllt alle Nebenbedingungen von (1)}\}.$$

Solche Optimierungsaufgaben sind meist in ihren durch die mathematische Modellierung vorgelegten Form numerisch nur schwer zu bearbeiten und ihre Lösung im Sinne des Findens einer optimalen Lösung $x^* \in G$, die f maximiert, ist effizient[1] nicht machbar.

[1] also hier in polynomieller Zeit

Daher sind vielfältige Strategien zur Behandlung solcher Aufgaben entworfen und untersucht worden. Eine Möglichkeit, diese Optimierungsaufgaben zu lösen, ist die Aufgabe (1) in eine einfachere Aufgabe zu überführen, deren Behandlung wohlverstanden ist und eine optimale Lösung einer solchen in festgelegter Weise zu einer optimalen Lösung von (1) führt. Diese Methodik ist als *Relaxation* des Problems (1) bekannt und wird meist dadurch realisiert, dass die Nebenbedingungen f_i und h_i derart vereinfacht werden, dass sich G so darstellen lässt, dass eine Maximierung von f über G im Vergleich zur Ausgangsaufgabe einfacher wird.

Wir wollen im folgenden den Ansatz der Relaxation von (1) zu einem linearen Programm diskutieren und werden dabei sehen, dass diese Form der Relaxation bei bestimmten Problemen nicht den erwünschten Erfolg verspricht und uns deswegen dem Konzept der *Semidefiniten Programmierung* zuwenden, was anschließend an Beispielen verdeutlicht werden soll.

1.1 Lineare Programmierung

Ein erster Versuch, (1) zu behandeln, ist eine Relaxation zu einer linearen Optimierungsaufgabe der Form

$$\begin{aligned} \text{maximiere} \quad & c \cdot x \\ \text{so dass} \quad & x \in \mathbb{R}^n, Ax = b, x \geq 0 \end{aligned}$$

wobei für ein festes $n \in \mathbb{N}$ die Vektoren $c, x, b \in \mathbb{R}^n$ und $A \in \mathbb{R}^{n \times n}$ gelten und $\cdot$ das im $\mathbb{R}^n$ kanonische Skalarprodukt bezeichnen soll. Solche Optimerungsaufgaben haben sich in der Praxis als sehr verlässlich erwiesen und auch die Theorie zur Lösung solcher Probleme (mit Hilfe von Simplexmethoden oder Innerer-Punkt Methoden) ist sehr ausgereift, so dass lineare Optimierungsaufgaben einfach zu handhaben sind.

Es gibt jedoch Probleme, die sich einer erfolgreichen Behandlung durch eine Relaxation hinzu einer linearen Optimierungsaufgabe widersetzen in dem Sinne, dass entweder die Relaxation keine neuen Ergebnisse liefern kann oder dass bessere Algorithmen bekannt sind, die ohne solch eine Relaxation auskommen. So führt zum Beispiel [GHL06] eine Relaxation von MAX-2SAT an, in der von $x_i \in \{0, 1\}$ zu $x_i \in [0, 1]$ relaxiert worden ist mit dem Hintergedanken, 0 mit falsch und 1 mit wahr zu indentifizieren und eine Lösung des entstandenen lineare Optimierungsproblems derart zu interpretieren, dass eine Präferenz einer Variablen x_i zu einer der Seiten 0 oder 1 hin eine Präferenz von x_i zu falsch oder wahr bedeutet. Allerdings gibt [GHL06] an, dass eine optimale Lösung der relaxierten Aufgabe dadurch zu erreichen ist, für alle x_i die Belegung $x_i = 0.5$ zu wählen, die in jedem Fall zu einer optimalen Lösung führt und damit die Relaxation im schlimmsten Fall nicht besser macht als der einfache Ansatz, jede Variable zufällig auf wahr oder falsch zu setzen.

Wir wollen im Folgenden daher eine recht neue Form der Relaxation bestimmter Probleme, meist sogenannter quadratischer Optimierungsprobleme, betrachten, die in vielen Fälle zu erstaunlichen Verbesserungen klassischer Approximationsaufgaben geführt hat.

1.2 Semidefinite Programmierung

Um im Folgenden bessere Relaxationen erreicht zu können, benötigen wir den folgenden Begriff:

Definition (positiv semidefinite Matrizen)**.** Sei $A \in \mathbb{R}^{n \times n}$. A heißt *positiv semidefinit*, wenn für alle $x \in \mathbb{R}^n$ stets $x^T Ax \geq 0$ gilt (in Zeichen $A \succeq 0$).

Die Eigenschaft der positiven Semidefinitheit ist ein Möglichkeit, die Eigenschaft einer reellen Zahl, positiv zu sein, auf Matrizen zu verallgemeinern.

In diesem Zusammenhang gilt dann folgender Satz

Theorem (Eigenschaften positiv semidefiniter Matrizen)**.** *Sei $A \in \mathbb{R}^{n \times n}$ symmetrisch. Dann sind folgenden Aussagen äquivalent:*

a) A ist positiv semidefinit

b) A hat keine negativen Eigenwerte

c) Für alle $x \in \mathbb{R}^n$ ist $x^T Ax \geq 0$

d) Es existiert eine Matrix $C \in \mathbb{R}^{n \times n}$ mit $A = CC^T$

Insbesondere die Eigenschaft (d) interessiert uns, da sie es uns erlaubt, Produkte der Form $x_i x_j$ durch Einträge einer positiv semidefiniten Matrix X zu ersetzen und damit eine Relaxation zu erreichen. Dies motiviert eine neue Klasse von Optimierungsaufgaben, den sogenannten *semidefiniten Programmen* (semidefiniten Optimierungsaufgaben, SDP), die sich allgemein durch Optimierungsaufgaben der Form

$$\begin{array}{rl} \text{maximiere} & \operatorname{tr}(WX) \\ \text{so dass} & \operatorname{tr}(A_j X) = b_j \text{ für } j \in \{1, \ldots, m\} \\ & X \succeq 0 \end{array}$$

mit $W, X, A_j \in \mathbb{R}^{n \times n}$ symmetrisch und $b_j \in \mathbb{R}$. Dabei ist

$$\operatorname{tr}(X) = \sum_{i=1}^{n} X_{ii}$$

die *Spur* der Matrix X.

Es lassen sich nun die aus der linearen Optimierung bekannten Verfahren wie das Simplexverfahren oder die Inneren-Punkt Methoden auf semidefinite Programme übertragen, allerdings nicht ohne Einbußen in der Laufzeit in Kauf nehmen zu müssen. Im Gegenzug erweisen sich Relaxationen zu semidefiniten Programmen als wesentlich schärfer und lassen die Berechnung verbesserter Approximationen zu.

Mit Hilfe Innerer-Punkt Methoden lassen sich semidefinite Programme in der Hinsicht einfach lösen, dass zu gegebenen $n \in \mathbb{N}$ und $\varepsilon > 0$ eine Lösung berechnet werden kann, die höchsten um ε von der Optimallösung abweicht. Diese Berechnung verläuft zeitlich in n und $\log \frac{1}{\varepsilon}$ polynomiell. Siehe hierzu insbesondere [Ali93].

2 Beispiele für Semidefinite Programme

Im Folgenden soll es uns darum gehen, semidefinite Programmierung anhand von Beispielen zu verdeutlichen und die erstaunlichen Resultate insbesondere bei der Behandlung von MAX-CUT, die durch diese neue Form der Relaxation erreicht worden sind, genauer zu betrachten.

2.1 MAX-CUT

Es sei im Folgenden $G = (V, E)$ ein Graph mit Knotenmenge $V \neq \emptyset$ und Kantenmenge $E \subseteq V \times V$, $|E| > 0$, dessen Kanten $(i, j) \in E$ mit nichtnegativen Gewichten w_{ij} versehen sind. Ohne Einschränkungen sei G vollständig, da für das Problem MAX-CUT nicht vorhandene Kanten und Kanten mit Gewicht 0 gleichwertig sind.

Definition (Gewicht eines Schnittes)**.** Sei $S \subseteq V$. Dann ist das *Gewicht des Schnittes* $(S, V \setminus S)$ gegeben durch

$$w(S, V \setminus S) := \sum_{i \in S, j \notin S} w_{ij}.$$

Wir können dann MAX-CUT definieren als die Optimierungsaufgabe

$$\begin{aligned} \text{maximiere} \quad & w(S, V \setminus S) \\ \text{so dass} \quad & S \subseteq V \end{aligned}$$

Es ist MAX-CUT $\in \mathcal{NPC}$, wie schon Karp zeigen konnte und ist damit eines der ältesten bekannten $\mathcal{NPC}$ Probleme überhaupt. Damit ist ein direkter Lösungsversuch mit dem heutigen Wissen nicht praktikabel und zur Lösung von MAX-CUT werden im allgemeinen Approximationsalgorithmen verwendet.

Eine Möglichkeit, MAX-CUT approximativ zu lösen, ist, für jeden Knoten mit Wahrscheinlichkeit 0.5 zu entscheiden, ob er zur Menge S hinzuzunehmen ist oder nicht. Dies ist ein 2–Algorithmus und war lange Zeit (über 20 Jahre) der best-bekannte Algorithmus zur Behandlung von MAX-CUT. Erst mit der Entwicklung von SDP gelang eine Neuentwicklung eines $(\alpha + \varepsilon)$–Algorithmus' mit $\alpha > 1.1382$ und $\varepsilon > 0$.

Zuerst modellieren wir MAX-CUT als quadratische Optimierungsaufgabe. Dazu assoziieren wir mit jedem Knoten $j \in V$ eine Variable $y_j \in \{-1, 1\}$ mit der Intention, dass $y_j = 1$ genau dann, wenn $j \in S$ gelten soll. Dann kann $w(S, V \setminus S)$ ausgedrückt werden durch

$$w(S, V \setminus S) = \frac{1}{2} \sum_{i<j} (1 - y_i y_j)$$

und wir können MAX-CUT durch die folgende Optimierungsaufgabe beschreiben:

$$\begin{aligned} \text{maximiere} \quad & \frac{1}{2} \sum_{i<j} w_{ij}(1 - y_i y_j) \\ \text{so dass} \quad & y_i \in \{-1, 1\} \text{ für alle } j \in V \end{aligned} \tag{MAX-CUT}$$

Diese Optimierungsaufgabe ist *quadratisch* in dem Sinne, dass alle Variablen y_i in einem Produkt $y_i y_j$ vorkommen. Setzen wir nun

$$Y := (y_i y_j)_{i,j=1}^n$$

so ist nach dem oben angegebenen Satz $Y \succeq 0$, symmetrisch und es ist $Y_{ii} = 1$. Damit ist die folgenden Relaxation zu einem semidefiniten Programm naheliegend:

$$\begin{aligned} \text{maximiere} \quad & \frac{1}{2} \sum_{i<j} w_{ij}(1 - Y_{ij}) \\ \text{so dass} \quad & Y_{ii} = 1 \text{ für alle } i \in V \\ & Y \succeq 0 \end{aligned}$$

Um einen besseren Approximationsalgorithmus für MAX-CUT zu erreichen, ist allerdings ein anderer Weg erfolgsversprechender: Wir fassen die Variablen y_i als Einheitsvektoren im $\mathbb{R}^1$ auf und schreiben (MAX-CUT) als

$$\begin{aligned} \text{maximiere} \quad & \frac{1}{2} \sum_{i<j} w_{ij}(1 - y_i y_j) \\ \text{so dass} \quad & y_i \in S_0 \text{ für alle } j \in V \end{aligned}$$

wobei S_n die Einheitskugel im $\mathbb{R}^{n+1}$ bezeichne, also

$$S_n = \{x \in \mathbb{R}^{n+1} \mid \|x\|_2 = 1\}.$$

Die Relaxation besteht nun darin, statt S_0 jede Kugel S_n zuzulassen:

$$\begin{aligned} \text{maximiere} \quad & \frac{1}{2} \sum_{i<j} w_{ij}(1 - v_i^T v_j) \\ \text{so dass} \quad & v_i \in S_{n-1} \text{ für alle } j \in V \end{aligned}$$

Diese Relaxation lässt sich als semidefinites Programm schreiben, indem wir

$$Y := \text{Gram}(\{v_1, \ldots, v_{|V|}\}) := (v_i^T v_j)_{i,j=1}^n$$

setzen und damit

$$\begin{aligned} \text{maximiere} \quad & \frac{1}{2} \sum_{i<j} w_{ij}(1 - Y_{ij}) \\ \text{so dass} \quad & Y_{ii} = 1 \text{ für alle } j \in V \\ & Y \succeq 0 \end{aligned} \qquad \text{(MAX-CUT 2)}$$

Um aus einer Lösung von (MAX-CUT 2) eine Lösung der ursprünglichen Relaxation zu machen, müssen wir nur die berechnete Matrix Y zerlegen in der Form, dass $Y = CC^T$ gilt. Diese Zerlegung kann in einer Laufzeit von $\mathcal{O}(n^3)$ durchgeführt werden und es sind die Zeilen von C die gesuchten Vektoren v_i.

Anschließend müssen wir aus der Lösung der Relaxation zu einer Lösung von (MAX-CUT) kommen. Dazu wählen wir gleichverteilt $r \in S_{n-1}$ und setzen

$$S := \{v_i \mid v_i^T r \geq 0\}.$$

Anschaulich gesprochen legen wir eine Hyperebene im $\mathbb{R}^n$ durch den Koordinatenursprung und wählen alle die Vektoren der gelösten Relaxation aus, die sich im positiven Halbraum dieser Hyperebene befinden. Als Algorithmus formuliert ergäbe sich dann

Algorithmus (Approximation von MAX-CUT nach [GW95]). *Gegeben sei ein Graph $G = (V, E)$, $|E| > 0$, $\varepsilon > 0$.*

1) Berechne eine optimale Lösung von (MAX-CUT 2) bis auf ε genau.

2) Wähle zufällig $r \in S_{n-1}$.

3) Setze $S := \{v_i \mid v_i^T r \geq 0\}$.

Da der erste Schritt in polynomieller Zeit durchgeführt werden kann, hat der gesamte Algorithmus eine polynomielle Laufzeit.

Es ist dann S ein Schnitt im Graphen G, dessen erwartete Güte $E_w \geq 0.87856 \cdot (\mathrm{OPT} - \varepsilon)$ ist:

Theorem (erwartete Güte der Approximation). *Die erwartete Güte E_w des durch den oben angegebenen Algorithmus berechneten Schnittes ist*

$$E_w \geq \alpha \cdot (OPT - \varepsilon)$$

wobei

$$\alpha = \min_{0 \leq \theta \leq \pi} \frac{2}{\pi} \frac{\theta}{1 - \cos\theta}$$

gilt.

Beweis. Wir werden zum Nachweis der Aussage die folgenden Teilschritte durchführen:

1) Es ist $E_w \geq \frac{1}{\pi} \sum_{i<j} w_{ij} \arccos(v_i^T v_j)$.

2) Es ist für $-1 \leq y \leq 1$ die Ungleichung $\frac{1}{\pi} \arccos(y) \geq \alpha \cdot \frac{1}{2}(1 - y)$ korrekt.

Sei $\{v_1, \ldots, v_{|V|}\}$ eine Lösung von (MAX-CUT 2) bis auf ε genau. Dann ergibt sich die Aussage wegen

$$\begin{aligned} E_w &= \frac{1}{\pi} \sum_{i<j} w_{ij} \arccos(v_i^T v_j) \\ &\geq \alpha \cdot \frac{1}{2} \sum_{i<j} w_{ij} (1 - v_i^T v_j) \\ &\geq \alpha \cdot (\mathrm{OPT} - \varepsilon) \end{aligned}$$

womit die Aussage gezeigt wäre.

zu (1) Es sei X_{ij} eine Zufallsvariable mit

$$X_{ij} = \begin{cases} 0 & \text{falls } v_i \text{ und } v_j \text{ nicht durch } r \text{ getrennt werden} \\ 1 & \text{falls } v_i \text{ und } v_j \text{ durch } r \text{ getrennt werden.} \end{cases}$$

Dann ist

$$E[X_{ij}] = \Pr[v_i \text{ und } v_j \text{ werden durch } r \text{ getrennt}]$$

und wegen der Linearität des Erwartungswertes ist

$$\begin{aligned} E_w &= E[\sum_{i<j} w_{ij} X_{ij}] \\ &= \sum_{i<j} w_{ij} E[X_{ij}] \\ &= \sum_{i<j} w_{ij} \Pr[v_i \text{ und } v_j \text{ werden durch } r \text{ getrennt}] \\ &= \sum_{i<j} w_{ij} \Pr[\operatorname{sgn}(v_i^T r) \neq \operatorname{sgn}(v_j^T r)] \end{aligned}$$

Betrachten wir nun den durch v_i und v_j aufgespannten, höchstens zweidimensionalen Untervektorraum von $\mathbb{R}^n$, so trennt r genau dann die beiden Vektoren, wenn die Hyperebene zwischen den Vektoren verläuft. Die Wahrscheinlichkeit, dass die Hyperebene zwischen den beiden Vektoren verläuft, ist aber $\frac{2\theta}{2\pi} = \frac{\theta}{\pi}$ wegen der vorausgesetzten Gleichverteilung und mit

$$\theta = \arccos(v_i^T v_j)$$

ist dann

$$\begin{aligned} E_w &= \sum_{i<j} w_{ij} \Pr[\operatorname{sgn}(v_i^T r) \neq \operatorname{sgn}(v_j^T r)] \\ &= \frac{1}{\pi} \sum_{i<j} w_{ij} \arccos(v_i^T v_j). \end{aligned}$$

zu (2) Es ergibt sich sofort

$$\begin{aligned} \frac{\frac{1}{\pi}\arccos(y)}{\frac{1}{2}(1-y)} &\geq \min_{-1 \leq y < 1} \frac{\frac{1}{\pi}\arccos(y)}{\frac{1}{2}(1-y)} \\ &\overset{y=\cos\theta}{=} \min_{0<\theta\leq\pi} \frac{2}{\pi} \frac{\theta}{1-\cos\theta} \\ &= \alpha \end{aligned}$$

und für $y = 1$ gilt die Abschätzung offensichtlich.

□

Der angegeben Algorithmus ist nach Angabe von [Wa06] der zur Zeit beste bekannte Algorithmus zur Behandlung von MAX-CUT. Weiterhin wurde bereits in [GW95] gezeigt, dass die obige Abschätzung sehr gut ist, indem ein Beispiel eines Graphen angegeben worden ist, dessen Approximationsgüte nahe an α lag.

Es ist noch zu bemerken, dass, da semidefinite Programme im allgemeinen nicht exakt lösbar sind, die Lösung von (MAX-CUT 2) nur mit einem Fehler $\varepsilon > 0$ berechnet werden kann, also

$$\mathrm{OPT}_{\mathrm{SDP}} \geq \mathrm{OPT} - \varepsilon.$$

Dies beeinträchtigt jedoch nicht die Richtigkeit der obigen Untersuchung, da dieser Fehler in den Approximationsfaktor des Algorithmuses einbezogen werden kann, denn es ist α nur eine untere Schranke der möglichen Approximationsgüte. Ist genauer $\varepsilon > 0$ die gewünschte Genauigkeit der Approximation und $c := \min\{w_{ij} \mid i, j \in V, w_{ij} \neq 0\}$, so ist die erwartete relative Güte

$$\alpha \cdot (1 - \frac{\varepsilon}{c}) \cdot \text{Optimalwert} \quad \text{bzw.} \quad \frac{1}{\alpha \cdot (1 - \frac{\varepsilon}{c})}.$$

2.2 MAX-BISECTION

Eine Variation von MAX-CUT ist die Fragestellung nach einem möglichst großen Schnitt S im Graphen G, so dass $|S| = |V \setminus S|$ gilt. Sinnvollerweise fordern wir $|E| \in 2\mathbb{N}$. Dann kann MAX-BISECTION durch die Optimierungsaufgabe

$$\begin{aligned} \text{maximiere} \quad & \frac{1}{2} \sum_{i<j} w_{ij}(1 - y_i y_j) \\ \text{so dass} \quad & y_i \in \{-1, 1\} \text{ für alle } i \in V \\ & \sum_{i=1}^{|V|} y_i = 0 \end{aligned} \tag{MAX-BISEC}$$

formuliert werden mit den gleichen Bezeichnung wie bei MAX-CUT. Schreiben wir nun

$$\sum_{i=1}^{|V|} y_i = 0 \text{ als } e^T y = 0$$

mit $e = (1)_{i=1}^n$ und $y = (y_i)_{i=1}^n$, so gilt

$$e^T y = 0 \implies \mathrm{tr}(ee^T Y) = 0$$

mit $Y = (\hat{y}_i^T \hat{y}_j)_{i,j=1}^n$ und $\hat{y}_i = (y_i, 0 \dots 0)^T$. Lassen wir nun allgemein $\hat{y}_i \in S_{n-1}$ zu, dann ist die Optimierungsaufgabe

$$\begin{array}{rl} \text{maximiere} & \frac{1}{2} \sum_{i<j} w_{ij}(1 - Y_{ij}) \\ \text{so dass} & \operatorname{tr}(ee^T Y) = 0 \\ & Y_{ii} = 1 \text{ für alle } i \in V \\ & Y \succeq 0 \end{array} \tag{2}$$

eine Relaxation von (MAX-BISEC). Damit lässt sich nun der folgende Algorithmus zur Berechnung einer Näherungslösung von MAX-BISECTION angeben:

Algorithmus (Frieze/Jerrum, [Ye99]). *Gegeben sei ein Graph $G = (V, E)$, $|V| \in 2\mathbb{N}$.*

1) *Berechne einen Schnitt S' wie in Algorithmus (2.1) mit dem semidefiniten Programm (2) statt (MAX-CUT 2)*
2) *Ist $|S'| \neq |V \setminus S'|$, so sei ohne Einschränkung $|S'| > |V \setminus S'|$. Entferne dann so lange die leichtesten Knoten aus S', bis $|S'| = |V \setminus S'|$ gilt. Dabei heißt ein Knoten aus S'* am leichtesten*, wenn er die Verringerung des Wertes des Schnittes S' bei Entfernung minimiert.*

Dieser Algorithmus hat eine erwartete relative Güte von $0.651 \cdot \mathrm{OPT}$ bzw. 1.536. In [Ye99] wurde ein Verfahren vorgestellt, welches eine Güte von $0.699 \cdot \mathrm{OPT}$ bzw. 1.431 aufweist.

2.3 MAX-2SAT

Auch die Behandlung von MAX-SAT ist mit Hilfe semidefiniter Programmierung möglich und wir wollen in diesem Abschnitt speziell MAX-2SAT untersuchen, da hier ein sehr einfachen Zugang möglich ist.

Gegeben sei eine Menge

$$C = \{C_j \mid C_j = x_{j_1}^{\alpha_{j_1}} \vee x_{j_2}^{\alpha_{j_2}}, j \in \{1, \dots, n\}, \alpha_{j_1}, \alpha_{j_2} \in \{-1, 1\}\}$$

von logischen Klauseln mit genau zwei Literalen. Dabei bedeutet $x^1 = x$ und $x^{-1} = \bar{x}$. Die Optimierungsaufgabe MAX-2SAT ist dann: Finde eine Belegung der Variablen derart, dass die Anzahl der erfüllten Klauseln maximal ist.

Um diese Optimierungsaufgabe mit Hilfe semidefiniter Programmierung zu behandeln, stellen wird für jede Variable x_i in den logischen Klauseln eine Variable $y_i \in \{-1, 1\}$ bereit. Des weiteren benötigen wir noch $y_0 \in \{-1, 1\}$, welche die Rolle von `true` in dieser Probleminstanz übernehmen soll. Wir setzen dann $x_i = \texttt{true} \iff y_i = y_0$.

Um die Anzahl der erfüllten Klauseln zu maximieren, benötigen wir eine Zielfunktion v, welche für uns die Anzahl der erfüllten Klauseln in einer Klauselmenge C erfüllt. Diese Funktion lässt sich auf den einzelnen Literalen sehr einfach angeben:

$$v(x_i) = \begin{cases} 0 & \text{falls } x_i = \texttt{false} \\ 1 & \text{falls } x_i = \texttt{true} \end{cases}$$

oder mit Hilfe der Variablen y_i und y_0:

$$v(x_i) = \frac{1 + y_i y_0}{2}$$

und

$$v(\overline{x_i}) = \frac{1 - y_i y_0}{2}.$$

Auf den Klausen ergibt sich dann v sehr kanonisch mit

$$\begin{aligned} v(x_i \vee x_j) &= 1 - v(x_i \wedge x_j) \\ &= 1 - \frac{1 - y_0 y_i}{2} \frac{1 - y_0 y_j}{2} \\ &= \frac{1 + y_0 y_i}{4} + \frac{1 + y_0 y_j}{4} + \frac{1 - y_i y_j}{4} \end{aligned}$$

Mit Hilfe dieser von v können wir nun MAX-2SAT schreiben als

$$\begin{aligned} \text{maximiere} \quad & \sum_{c \in C} v(c) \\ \text{so dass} \quad & y_i \in \{-1, 1\} \text{ für alle } x_i \end{aligned} \tag{MAX-2SAT}$$

Es ist die Summ $\sum_{c \in C} v(c)$ von der Form

$$\sum_{c \in C} v(c) = \sum_{i<j} a_{ij}(1 + y_i y_j) + b_{ij}(1 - y_i y_j).$$

Lassen wir dann wieder statt $y_i \in S_0$ auch $y_i \in S_{n-1}$ zu, so relaxiert sich das Problem zu

$$\begin{aligned} \text{maximiere} \quad & \sum_{c \in C} a_{ij}(1 + v_i^T v_j) + b_{ij}(1 - v_i^T v_j) \\ \text{so dass} \quad & v_i \in S_{n-1} \text{ für alle } i \\ & Y = (v_i^T v_j)_{i,j=0}^n \succeq 0 \end{aligned} \tag{3}$$

Lösen wir wieder diese Relaxation mit Lösung $\{v_0, \ldots, v_{|V|}\}$, so wählen wir wieder $r \in S_{n-1}$ zufällig und setzen alle Variablen x_i auf `true`, die

$$\operatorname{sgn}(v_i^T r) = \operatorname{sgn}(v_0^T r)$$

erfüllen, also mit v_0 „im selben Schnitt liegen“, die restlichen auf `false`.

Es gilt dann

Theorem (Approximationsgüte für MAX-2SAT)**.** *Es gilt für den oben angegebenen Algorithmus*

$$E_w \geq \alpha \cdot \sum_{i<j} a_{ij}(1 + v_i^T v_j) + b_{ij}(1 - v_i^T v_j)$$

mit demselben α wie im MAX-CUT Algorithmus.

Literatur

[Ali93] Farid Alizadeh; Interior Point Methods in Semidefinite Programming with Applications to Combinatorial Optimization;

[BV04] Stephen Boyd und Lieven Vandenberghe; Convex Optimization; Cambridge University Press, 2004

[FJ95] Alan Frieze und Mark Jerrum; Improved Approximation Algorithms for MAX k-CUT and MAX-BISECTION;

[GHL06] Carla P. Gomes et. al; The Power of Semidefinite Programming Relaxations for MAXSAT; In Proc. Conf. Intregration of AI/OR (CPAIOR06), 2006

[GW95] Michel X. Goemans und David P. Williamson; Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming; Proceedings of the 26th Annual ACM Symposium on Theory of Computing,

[Has97] Johan Håstad; Some optimal inapproximability results;

[helm] Christop Helmberg; Semidefinite Programming; zu finden unter `http://www-user.tu-chemnitz.de/~helmberg/semidef.html`

[Wa06] Rolf Wanka; Approximationsalgorithmen; Teubner Verlag, Wiesbaden, 2006

[Ye99] Yinyu Ye; A .699-Approimation Algorithm for Max-Bisection;