Dokumentation zum Projekt:
Branch-and-Bound-Verfahren an einem geeignetem Beispiel

Anna Daschkovska

Generelle Erläuterung des Algorithmus

Branch and Bound ist eine Methode, die in der diskreten Optimierung verwendet wird, um eine Lösung für ein Optimierungsproblem zu finden. Es ist insbesondere nützlich für Probleme, bei denen die Suche nach der optimalen Lösung mit Brute-Force-Methoden zu zeitaufwändig wäre. Die Probleme sind dabei ganzzahlige Probleme, welche als Gleichungssysteme dargestellt werden.

Er besteht aus zwei Teilen: dem Branch (Verzweigung) und dem Bound (Abgrenzung). Der *Branch*-Schritt dient dazu, das vorliegende ganzzahlige Problem in zwei oder mehr Teilprobleme (lineare Probleme) aufzuteilen, die eine Vereinfachung des ursprünglichen Problems darstellen. Durch rekursives Ausführen des Branching-Schritts für die erhaltenen Teilprobleme entsteht eine Baum-Struktur, wobei der Wurzelknoten das ursprüngliche Problem darstellt und die nachfolgenden Knoten die Teilprobleme darstellen.
Dabei kann man auf verschiedene Weisen, je nach Bedarf, die nächsten zu bearbeiteten Knoten (Teilprobleme) wählen. Oft verwendet man Tiefensuche, Breitensuche und Bestensuche. Ich habe die Tiefensuche verwendet.

Jedes Mal, wenn neue Teilprobleme erzeugt wurden, werden diese zum Beispiel mit dem Simplex-Verfahren gelöst und anschließend erhält man exakte Lösung für ein Maximum oder Minimum. Nun wird der Bound-Schritt relevant: Man wählt nicht ganzzahlige Variablen aus und schränkt diese nach oben („upper bound“) und nach unten („lower bound“) ein. Durch solche Schranken ergeben sich neue Teilprobleme und wir nähern uns ganzzahligen Ergebnissen an. Zweige werden dabei abgeschnitten, wenn eine ganzahlige Lösung für ein Teilproblem gefunden wurde und diese somit im Ausgangsproblem (der Wurzel) zulässig ist. Solche Lösungen werden sich gemerkt und durch weiteres Suchen immer verglichen, um die optimalste ganzzahlige Lösung zu finden.

Dieses Verfahren kann man in einem Binärbaum, wenn man immer eine Variable wählt und einschränkt darstellen. Da es bei der Tiefensuche zum Beispiel passieren kann, dass im Worst Case ewig tief gesucht wird, kann man eine maximale Tiefe festlegen, um ein möglichst breite Suche zu ermöglichen.

Erläuterung an einem Beispiel:

Ganzzahliges Problem (GP):

$x_1 + x_2 \leq 6$ dabei soll für GP gelten: $x_1, x_2 \geq 0$

$x_1, x_2 \in \mathbb{Z}$

$5x_1 + 9x_2 \leq 45$

$5x_1 + 8x_2 \rightarrow max$

Wir entfernen die Ganzzahligkeitsbedingung ($x_1, x_2 \in \mathbb{R}$) und lösen dieses Gleichungssystem als ein Problem (LP1) mit dem Simplex-Verfahren.

Es ergibt sich:
$x_1 = 2{,}25$
$x_2 = 3{,}75$
$z = 41{,}25$ (z ist hierbei dar optimalste Wert, den $5x_1 + 8x_2$ annehmen kann)

Dieses Ergebnis ist in GP natürlich nicht zulässig, sodass wir uns eine Verzweigungsvariable wählen, die nicht ganzzahlig ist und zwei neue Teilprobleme erzeugen.

Wir wählen x_2 und erstellen die Bedingungen $x_2 \leq 3$ (für LP2) und $x_2 \geq 4$ (für LP3), da wir wissen, dass alle Werte dazwischen in GP sowieso nicht zulässig sind. (Würde man zb. keinen Binärbaum wählen, würde man für mehrere oder alle nicht ganzzahligen Variablen jeweils obere und untere Schranken setzen und somit mehr Probleme erzeugen).

Wir fügen die neuen Bedingungen hinzu und erhalten diese zwei neuen Gleichungssysteme:

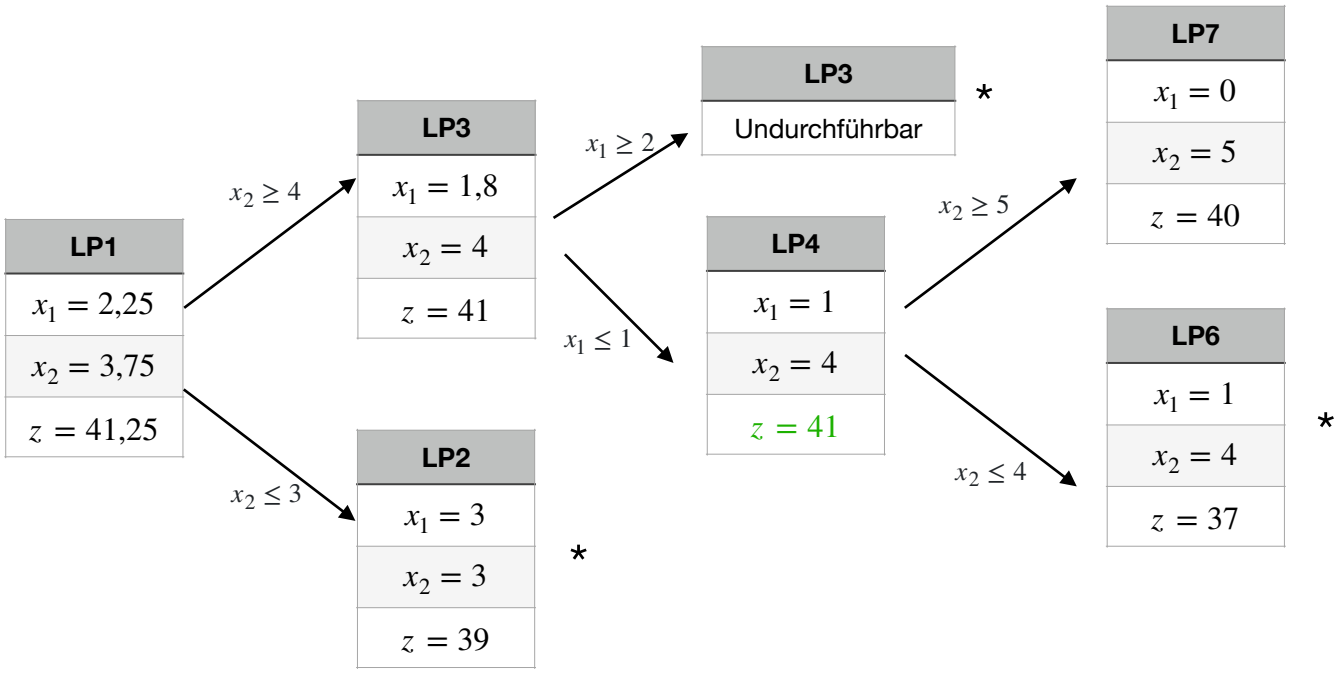
LP2:

$x_1 + x_2 \leq 6$
$5x_1 + 9x_2 \leq 45$
$5x_1 + 8x_2 \rightarrow max$
$x_2 \leq 3$
$x_1, x_2 \geq 0$

LP3:

$x_1 + x_2 \leq 6$
$5x_1 + 9x_2 \leq 45$
$5x_1 + 8x_2 \rightarrow max$
$x_2 \geq 4$
$x_1, x_2 \geq 0$

Beide Probleme lösen wir mit dem Simplex und so weiter. Es entsteht ein solcher Baum:



Die Ergebnisse von LP2, LP4, LP7 und LP6 waren alles mögliche Lösungen, aber der höchste Wert entstand natürlich bei LP4 mit 41. Die Zweige der anderen wurden abgeschnitten, da danach keine bessere Lösung als diese mehr möglich wäre. Je nach dem wie genau man den optimalsten Wert haben möchte sucht man weiter. Hier wurde in diesem Baum die optimalste Lösung z=41 gefunden. Je nach dem wie und mit welchen Methoden man die Verzweigungsvariblen und deren Anzahl wählt erhält man Abweichungen. Mein Programm erhält zum Beispiel für dieses Beispiel den optimalsten Wert 40, da es die Verzweigungsvariable x_1 am Anfang wählt. Wie gesagt, dieses Verfahren ist ein Näherungsverfahren.

Implementiert habe ich dieses Verfahren in C++ mithilfe eines Stapels. Beim Implementieren des Simplex-Algorithmus beziehe ich mich auf die Quelle: https://github.com/PetarV-/Algorithms/blob/master/Mathematical%20Algorithms/Simplex%20Algorithm.cpp

Hinweis: Beim Input handelt es sich um gehardcodete Variablen, die man verändern muss. Die auszuführende Datei heißt main.cpp und benötigt alle anderen Dateien zum funktionieren.

Ausgabe von Beispielen:

1. GP:	Input:	Output:
$x_1 + x_2 \leq 20$	matrix vorfaktoren {{1,1},{2,9}}	x-Werte: [1x1, 0x2]
$2x_1 + 9x_2 \leq 45$	rechte_seiten {20,3}	Optimalster Wert: 5
$5x_1 + 4x_2 \rightarrow max$	vorfaktoren_max_gleichung {5,4};	
$x_1, x_2 \geq 0$		

2. GP:	Input:	Output:
$x_1 + x_2 \leq 88$	matrix vorfaktoren {{1,1},{2,9}}	x-Werte: [2x1, 0x2]
$2x_1 + 9x_2 \leq 4$	rechte_seiten {88,4}	Optimalster Wert: 108
$54x_1 + 4x_2 \rightarrow max$	vorfaktoren_max_gleichung {54,4};	
$x_1, x_2 \geq 0$		

3. GP:	Input:	Output:
$x_1 + x_2 + 5x_3 \leq 88$	matrix vorfaktoren {{1,1,5},{2,9,1}}	x-Werte: [0x1, 0x2, 4x3]
$2x_1 + 9x_2 + x_3 \leq 4$	rechte_seiten {88,4}	Optimalster Wert: 120
$54x_1 + 4x_2 + 30x_3 \rightarrow max$	vorfaktoren_max_gleichung {54,4,30};	
$x_1, x_2, x_3 \geq 0$		

4. GP:	Input:	Output:
$2x_1 + 2x_2 \leq -20$	matrix vorfaktoren {{-2,-2},{4,1}}	x-Werte: [0x1, 8x2]
$4x_1 + x_2 \leq 8$	rechte_seiten {20,8}	Optimalster Wert: 48
$18x_1 + 6x_2 \rightarrow max$	vorfaktoren_max_gleichung {18,6};	
$x_1, x_2 \geq 0$		

Hier finden Sie meinen Programmcode:

https://github.com/andasc/Branch-and-Bound-Algorithm