Wörter aufräumen

Bundeswettbewerb Informatik

Jakov D. Wallbrecher

2020

# Lösungsidee

Um zur Lösung zu kommen, sind wir in mehreren Schritten vorgegangen:

1. Für jede Lücke berechnen, welche Wörter „hineinpassen“
2. Alle Lücken mit nur einer Möglichkeit befüllen
3. Verteilung der Wörter auf die Lücken mit mehreren passenden Wörtern als „QUBO“ Optimierungsproblem formulieren und mit einem adiabatischen Quanten-Annealer lösen
4. Lücken ohne vorgegebenen Buchstaben mit übrigen Wörtern auffüllen

# Einführung „Quantum-Annealing“

Wie vorher beschrieben nahmen wir uns vor, ein Teil des Problems auf einem adiabatischen Quantencomputer zu lösen. Dazu hier einige vorangehende Erläuterungen.

Adiabatische Quantencomputer, auch Quantenannealer genannt, sind prädestiniert, um Optimierungsprobleme der QUBO-Form zu lösen. QUBO steht für „quadratic unconstrained binary optimization“. Das bedeutet also, dass die Optimierungsprobleme immer als Abhängigkeiten zwischen zwei Werten, ohne zusätzlich weitere Bedingungen, formuliert und in 1 und 0 dargestellt werden müssen.

Die Idee hinter der Funktionsweise eines Quantenannealers ist, dass die miteinander verschalteten Recheneinheiten (sog. Qubits) in die Superposition gebracht werden (also alle Qubits sowohl den Wert 0 als auch 1 haben) und im Laufe eines Durchlaufs die Verschaltungen „eingefadet“ werden. Am Ende des Durchlaufs werden alle Qubits gemessen, sodass sie sich zwischen 0 und 1 „entscheiden“ müssen. Mit welcher Wahrscheinlichkeit sie sich für welchen Wert entscheiden, kann durch die Verschaltungen („couplings“) und Wertungen („biases“) beeinflusst werden.

Ziel ist es also, Das Optimierungsproblem so mit Verschaltungen und Wertungen darzustellen, dass am Ende eines Durchlaufs des Quantencomputers eine richtige Lösung resultiert.

Als Eingabe erwartet der Quantencomputer eine Kostenmatrix (Hamiltonian). An x- und y-Achse sind jeweils alle Qubits angetragen. Wäre eine Lösung eindeutig falsch, wenn zwei Qubits (beispielsweise „a“ und „f“) gleichzeitig belegt sind, also den Wert 1 haben, wird in der Matrix an der Stelle (a|f) eine Bestrafung angetragen (z.B. „+2“). Im Quantencomputer bedeutet das eine „bestrafende“ (bzw. Energie-erhöhende) Verschaltung dieser beiden Qubits. Dadurch ist es sehr unwahrscheinlich, dass der Quantencomputer sich zur Auswahl dieser beiden Qubits entscheidet.

# Umsetzung in der Programmiersprache C#

## Klasse Lücke

Die Datenklasse Lücke beschreibt eine Lücke (z.B. „\_a\_“) mit den Eigenschaften char[] lücke, int passt, List<int> passtList und int index.

class Luecke {

public char[] luecke;

public int passt;

public List<int> passtList;

public int index;

public Luecke(char[] luecke, List<int> passtList, int index) {

this.luecke = luecke;

this.passtList = passtList;

this.index = index;

}

}

Char[] lücke: Char-Array mit allen Buchstaben einer Lücke (z.B. {‚\_‘, ‚a‘ , ‚\_‘})

List<int> passtList: Liste mit den indizes aller passenden Wörter

Int passt: index des endgültig passenden Wortes

Int index: Position der Lücke im Text

Die Funktion readFile() liest die Eingabedatei und gibt den ursprünglichen Text, eine Liste mit Lücken (gespeichert als Objekte der Klasse Lücke) und ein Array mit Wörtern zurück.

## findSolution()

Die Funktion findsolution() erwartet als Eingabe die Rückgabewerte von readFile, löst das Rätsel, und gibt die Lösung als string zurück. Wie schon beim Lösungsansatz beschrieben, gliedert sich der Lösungsweg in mehrere Teilschritte.

1. Für jede Lücke berechnen, welche Wörter „hineinpassen“. Die jeweiligen indizes der passenden Wörter werden in der jeweiligen passtList Eigenschaft der Lücke gespeichert.

Dazu iteriert das Programm über alle Lücken. Es prüft in jeder Lücke zuerst, ob eines der Wörter die entsprechende Länge hat. Wenn ja, wird durch ablaufen der Buchstaben überprüft, ob die vorgegebenen Buchstaben mit denen des Wortes zusammenfallen. Durch geschicktes setzen der Variable equal in der Schleife können wir ablesen, ob die Lücke a) keine Vorgaben hat und das Wort reinpasst (equal=0), b) das Wort die richtige Länge hat und die Vorgaben übereinstimmen (equal=1) oder c) es zwar die richtige Länge hat, nicht aber mit den vorgegebenen Buchstaben zusammenpasst. Der Index des Wortes wird im Fall a) in die Liste LueckenLeer und im Fall b) in die Liste luecken hinzugefügt.

for (int i = 0; i < luecken.Count; i++) {

for (int j = 0; j < woerter.Length; j++) {

if (luecken[i].luecke.Length == woerter[j].Length) {

int equal = 0;

for (int k = 0; k < woerter[j].Length; k++) {

if (luecken[i].luecke[k] != '\_') {

if (luecken[i].luecke[k] == woerter[j][k]) {

equal = 1;

}

else {

equal = 2;

break;

}

}

}

if (equal == 1) {

luecken[i].passtList.Add(j);

}

else if (equal == 0) {

try {

lueckenLeer[lueckenLeer.FindIndex(ind => ind.index.Equals(lueckenPaare[i].index))].passtList.Add(j);

}

catch {

lueckenLeer.Add(new Luecke(luecken[i].luecke, new List<int> { j }, luecken[i].index));

}

}

}

}

}

1. Alle Lücken mit nur einer Möglichkeit befüllen

Um sicherzustellen, dass Wörter nicht mehrmals eingesetzt werden, benutzen wir als Wertkopie des arrays wörter das Array unbenutzeWörter, in dem der Eintrag aller verwendeten Wörter auf „“ gesetzt wird.

Zuerst wird festgestellt, in welche Lücke die meisten Wörter reinpassen, bzw. wie viele es sind (erste Schleife; a).

In der zweiten Schleife (i) geht das Programm alle Lücken ab, in denen nur ein Wortindex gespeichert ist (also nur ein Wort hineinpasst), und setzt den endgültigen passt Wert der Lücke auf den einzig möglichen Wortindex. Außerdem wird das entsprechende Wort wie beschrieben aus dem unbenutzteWörter Array gelöscht. Nach dieser zweiten Schleife werden alle nun verwendeten Wörter aus der passtList anderer Wörter gelöscht. So kann die zweite Schleife wiederholt werden um sicherzustellen, dass auch Lücken, die durch Benutzen von einem der passenden Wörter eindeutig geworden sind, gleich eingefüllt werden (zB ist für die Lücke „d\_\_“ „der“ und „die“ möglich und in der Lücke „\_e\_“ nur „der“. Wenn die zweite Lücke gesetzt wurde, ist auch die erste Lücke eindeutig lösbar). Dieser Vorgang wird so oft wiederholt, wie die längste passtList einer Lücke lang ist.

//klar feststehende (nur eine möglichkeit) einfüllen

string[] unbenutzteWoerter = new string[woerter.Length];

woerter.CopyTo(unbenutzteWoerter, 0);

//längste passtList ermitteln

int meistePasst = 0;

for (int a = 0; a < luecken.Count; a++) {

if (luecken[a].passtList.Count > meistePasst) {

meistePasst = luecken[a].passtList.Count;

}

}

for (int j = 0; j < meistePasst; j++) {

//eindeutige Wörter setzen

for (int i = 0; i < luecken.Count; i++) {

if (luecken[i].passtList.Count == 1 && unbenutzteWoerter[luecken[i].passtList[0]] != "") {

luecken[i].passt = luecken[i].passtList[0];

unbenutzteWoerter[luecken[i].passt] = "";

}

}

//jetzt gesetzte Wörter aus allen passtList entfernen

for (int l = 0; l < luecken.Count; l++) {

for (int w = 0; w < luecken[l].passtList.Count; w++) {

if (unbenutzteWoerter[luecken[l].passtList[w]] == "") {

luecken[l].passtList.RemoveAt(w);

break;

}

}

}

}

1. Verteilung der Wörter auf die Lücken mit mehreren passenden Wörtern als „QUBO“ Optimierungsproblem formulieren und mit einem adiabatischen Quanten-Annealer lösen

Dieser Schritt ist nicht bei allen Rätseln nötig, da oft auch die erste Methode reicht, um das Problem zu lösen.

Wie schon in der Erklärung zu Quantencomputern beschrieben, ist das binäre Darstellen eines Problems die erste Herausforderung. Bei uns steht jede Kombination aus Lücke mit passendem Wort für ein Qubit. Also zum Beispiel wäre bei der Lücke „d\_\_“ mit den passenden Wörtern „die“ und „der“ ein Qubit a *Lücke „d\_\_“ in Kombination mit Wort „der“* und ein anderes Qubit b *Lücke „d\_\_“ in Kombination mit Wort „die“*. Eine Regel wäre bei diesem Beispiel, dass nicht sowohl das eine als auch das andere Qubit 1 sein dürfen, da sonst in der Lücke „d\_\_“ beide Wörter stehen würden. Also bestrafen wir in der Matrix an der Stelle (Qubit a|Qubit b).

Um alle diese Bestrafungen in die Matrix einzufüllen, durchlaufen wir alle Lücken in Kombination mit allen Wörtern, die in diese Lücke passen. Um aber wie beschrieben zwei solcher Lücke-Wort-Kombinationen zu vergleichen, iterieren wir innerhalb dieser Schleifen nochmals über Lücken und Wörter.

*\*Der Übersichtlichkeit halber lasse ich die Schleifen hier weg und erkläre nur die Bedingungen innerhalb der verschachtelten Schleifen.\**

Wenn die gleiche Lücke und das gleiche Wort verglichen wird, wird in die Matrix eine Grundbelohnung eingetragen, um zu verhindern, dass der Quantencomputer gar keinen Qubit auf 1 setzt (wenn er nur bestraft wird, ist eine Lösung ohne gesetzte Qubits auch optimal). Wenn die betrachtete Lücke gleich, aber das Wort verschieden ist, wird eine Bestrafung eingetragen, damit der Quantencomputer pro Lücke nur EIN Wort vorsieht (wie beim Beispiel vorhin beschrieben). Zuletzt muss sichergestellt werden, dass jedes Wort nur einmal eingefüllt wird, indem bestraft wird, wenn das Wort gleich, aber die Lücke verschieden ist.

int wort1 = luecken[lücke1].passtList[wort1ind];

int wort2 = luecken[lücke2].passtList[wort2ind];

if (lücke1 == lücke 2) {

if (wort1 == wort2) { matrix[x, y] = -2; } //grundbelohnung

else { matrix[x, y] = 2; } //bestrafung: nur ein Wort pro Lücke

}

if (wort1 == wort2 && lücke1 != lücke2) {

matrix[x, y] = 2; //bestrafung: jedes Wort nur einmal

}

Die jetzt generierte Matrix wird über eine selbstgeschriebene, auch schon für andere Projekte eingesetzte, Library zusammen mit einigen Rahmenparametern an den Quantencomputer übertragen. Die wichtigsten Parameter sind „annealing\_time“ (gibt in Mikrosekunden an, wie lang ein Durchlauf dauern soll) und „num\_reads“ (bestimmt wie viele Durchläufe gemacht werden sollen; oft findet der Quantenannealer nicht sofort die optimale Lösung). Als Rückgabe erhalten wir vom Quantencomputer alle n Ergebnisse (in 0 und 1) mit deren jeweiliger Energie („Güte des Ergebnis“). Zugriff auf den Quanteannealer der kanadischen Firma D-Wave erhielten wir über das Forschungszentrum in Jülich.

qaConstellation constellation;

try {

Dictionary<string, string> qaArguments = new Dictionary<string, string>() {

{"annealing\_time","30"},

{"num\_reads","1000"},

{"chain\_strength","1.7" }

};

Dictionary<string, string> pyParams = new Dictionary<string, string>() {

{"problem\_type","qubo"},

{"dwave\_solver", "DW\_2000Q\_6"},

{"dwave\_inspector","false" }

};

Task<qaConstellation> constellationTask = QA\_Classification.Program.qaCommunication(matrix, qaArguments, pyParams);

constellation = constellationTask.Result;

constellation.printConstellation();

Um nun eine Lösung der Verteilung der Wörter auf die Lücken zu bekommen, wählen wir das beste Ergebnis vom Quantencomputer aus und dekodieren es. Dazu iteriert das Programm wieder über alle Lücken und die jeweils passenden Wörter und setzt, wenn der entsprechende Wert im Ergebnis 1 ist, den dazugehörigen Wortindex in die endgültige „passt“ Variable der Lücke.

int[] solution = constellation.results[constellation.getLowest(1)].Item4;

int count = 0;

for (int wort1 = 0; wort1 < luecken.Count; wort1++) {

if (luecken[wort1].passtList.Count > 1) {

for (int passt1ind = 0; passt1ind < luecken[wort1].passtList.Count; passt1ind++) {

if (unbenutzteWoerter[luecken[wort1].passtList[passt1ind]] != "") {

if (solution[count] == 1) {

luecken[wort1].passt = luecken[wort1].passtList[passt1ind];

}

count++;

} } } }

Zum Schluss werden die errechneten Wörter wieder in den Originaltext eingefüllt. Um Satzzeichen o.ä. zu erhalten, haben wir versucht, den Text der Eingabe Schritt für Schritt durch die entsprechenden Wörter zu ersetzen. Dazu laden wir die Lücke, die sich an der aktuellen Stelle befindet (die Lücken sind nicht zwingend sortiert, daher der Klassenparameter „index“), und ersetzen jeweils von einem Leerzeichen ausgehend die folgenden Zeichen durch das jeweilige Wort.

string[] originalText = input.Item3.Split(' ');

for (int i = 0; i < luecken.Count; i++) {

Luecke aktuelleLuecke = luecken[luecken.FindIndex(ind => ind.index.Equals(i))];

string oldstring = originalText[i].Substring(0, woerter[aktuelleLuecke.passt].Length);

originalText[i] = originalText[i].Replace(oldstring, woerter[aktuelleLuecke.passt]);

}

return string.Join(" ", originalText);

# Ergebnisse

## Raetsel0

oh je, was für eine arbeit!

## Raetsel1

Am Anfang wurde das Universum erschaffen. Das machte viele Leute sehr wütend und wurde allenthalben als Schritt in die falsche Richtung angesehen.

## Raetsel2

Als Gregor Samsa eines Morgens aus unruhigen Träumen erwachte, fand er sich in seinem Bett zu einem ungeheueren Ungeziefer verwandelt.

## Raetsel3

Informatik ist die Wissenschaft von der systematischen Darstellung, Speicherung, Verarbeitung und Übertragung von Informationen, besonders der automatischen Verarbeitung mit Digitalrechnern.

## Raetsel4

Opa Jürgen blättert in einer Zeitschrift aus der Apotheke und findet ein Rätsel. Es ist eine Liste von Wörtern gegeben, die in die richtige Reihenfolge gebracht werden sollen, so dass sie eine lustige Geschichte ergeben. Leerzeichen und Satzzeichen sowie einige Buchstaben sind schon vorgegeben.

# Programmausgabe (bei raetsel4):

Wörter:

Es, in, in, so, aus, der, die, die, ein, ist, Opa, sie, und, und, von, dass, eine, eine, sind, einer, Liste, schon, sowie, einige, findet, Jürgen, Rätsel, sollen, werden, ergeben, gegeben, lustige, Wörtern, Apotheke, blättert, gebracht, richtige, Buchstaben, Geschichte, vorgegeben, Leerzeichen, Reihenfolge, Satzzeichen, Zeitschrift

Aufgabe:

\_\_a \_\_\_\_\_n \_\_\_\_\_\_\_t \_n \_i\_\_\_ \_e\_\_\_\_\_\_\_\_\_ \_\_s d\_\_ \_\_\_\_\_e\_\_ \_\_d \_i\_\_\_\_ e\_\_ \_\_t\_\_\_. \_s \_s\_ \_\_n\_ \_\_\_\_e \_\_n \_\_\_\_\_\_n \_\_\_e\_\_\_, \_i\_ i\_ \_i\_ \_\_\_h\_\_\_\_ \_\_\_\_\_\_\_\_\_\_e \_\_b\_\_\_\_\_ \_\_\_d\_\_ s\_\_\_\_\_, \_o \_\_s\_ s\_\_ \_i\_\_ \_\_\_t\_\_\_ \_e\_\_\_\_\_\_\_\_ \_r\_\_\_\_\_. \_\_\_\_\_\_\_\_\_\_n \_n\_ \_a\_\_\_\_\_\_\_\_\_ s\_\_\_\_ e\_\_\_\_\_ \_\_\_\_\_\_\_b\_\_ \_\_\_d \_\_\_\_n \_\_\_\_\_\_e\_\_\_.

PRINT MATRIX

-2 2 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0

2-2 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0

0 0-2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 2-2 2 0 0 0 0 2 0 0 0 2 0 0 0 0 0

0 0 2 2-2 0 0 0 0 0 2 0 0 0 2 0 0 0 0

0 0 0 0 0-2 2 0 0 0 0 0 0 0 0 0 0 2 0

0 0 0 0 0 2-2 0 0 0 0 0 0 0 0 0 0 0 2

0 0 0 0 0 0 0-2 2 0 0 0 0 0 0 2 0 0 0

0 0 0 0 0 0 0 2-2 0 0 0 0 0 0 0 2 0 0

0 0 0 2 0 0 0 0 0-2 2 0 0 2 0 0 0 0 0

0 0 0 0 2 0 0 0 0 2-2 0 0 0 2 0 0 0 0

2 0 0 0 0 0 0 0 0 0 0-2 2 0 0 0 0 0 0

0 2 0 0 0 0 0 0 0 0 0 2-2 0 0 0 0 0 0

0 0 0 2 0 0 0 0 0 2 0 0 0-2 2 0 0 0 0

0 0 0 0 2 0 0 0 0 0 2 0 0 2-2 0 0 0 0

0 0 0 0 0 0 0 2 0 0 0 0 0 0 0-2 2 0 0

0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 2-2 0 0

0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0-2 2

0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 2-2

RUN MATRIX ON DWAVE

Preprocess matrix, qaArguments and pyParams for transfer to python

Start python program & send data . . .

got response

Convert response to data

PRINT RESPONSE

QA\_ARGUMENTS:

annealing\_time = 30

num\_reads = 1000

chain\_strength = 1.7

PY\_PARAMS:

problem\_type = qubo

dwave\_solver = DW\_2000Q\_6

dwave\_inspector = false

RESULTS:

energy num cbf results

-18 63 0 [0 1 1 0 0 0 1 0 1 0 1 1 0 1 0 1 0 1 0]

-18 66 0 [0 1 1 0 0 0 1 0 1 1 0 1 0 0 1 1 0 1 0]

-18 93 0 [0 1 1 0 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0]

-18 80 0 [0 1 1 0 0 0 1 1 0 1 0 1 0 0 1 0 1 1 0]

-18 76 0 [0 1 1 0 0 1 0 0 1 0 1 1 0 1 0 1 0 0 1]

-18 46 0 [0 1 1 0 0 1 0 0 1 1 0 1 0 0 1 1 0 0 1]

-18 97 0 [0 1 1 0 0 1 0 1 0 0 1 1 0 1 0 0 1 0 1]

-18 63 0 [0 1 1 0 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1]

-18 66 0 [1 0 1 0 0 0 1 0 1 0 1 0 1 1 0 1 0 1 0]

-18 26 0 [1 0 1 0 0 0 1 0 1 1 0 0 1 0 1 1 0 1 0]

-18 61 0 [1 0 1 0 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0]

-18 48 0 [1 0 1 0 0 0 1 1 0 1 0 0 1 0 1 0 1 1 0]

-18 59 0 [1 0 1 0 0 1 0 0 1 0 1 0 1 1 0 1 0 0 1]

-18 53 0 [1 0 1 0 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1]

-18 56 0 [1 0 1 0 0 1 0 1 0 0 1 0 1 1 0 0 1 0 1]

-18 40 0 [1 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1]

-16 1 0 [0 1 0 0 0 1 0 0 1 1 0 1 0 0 1 1 0 0 1]

Opa Jürgen blättert in einer Zeitschrift aus der Apotheke und findet ein Rätsel. Es ist eine Liste von Wörtern gegeben, die in die richtige Reihenfolge gebracht werden sollen, so dass sie eine lustige Geschichte ergeben. Leerzeichen und Satzzeichen sowie einige Buchstaben sind schon vorgegeben.