BwInf 2019 - Blumenbeet Dokumentation

## Team: Kings of Quantum, Team Nr: 00672

## Bearbeitet von Jakov D. Wallbrecher

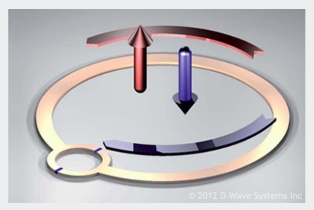
# Lösungsstrategie

Unsere Idee ist es, das Blumenbeet-Problem so darzustellen, dass es von einem adiabatischen Quantencomputer (auch Quantum-Annealer) gelöst werden kann. Voraussetzung dafür ist es, dass das Problem ein Optimierungsproblem ist, dass mit 0 und 1 dargestellt werden kann. Der Quantum-Annealer erwartet als Eingabe eine Hamilton-Matrix (nachfolgend erklärt), die das Problem repräsentiert. Um diese Matrix zu erhalten muss man das Problem zuerst mit 0 und 1 darstellen können, um dann die Matrix zu formulieren. Abschließend haben wir das Problem auf herkömmlichen PC’s mittels simulated Annealing, sowie auf dem Digital Annealer von Fujitsu, als auch auf einem echten Quantencomputer gelöst

# Funktionsweise eines adiabatischen Quantencomputers

## Quantenbit auf einem Quantencomputer

Quantencomputer rechnen - anders als herkömmliche Computer - nicht mit normalen Bits, sondern mit Quantenbits, kurz „Qubits“. Im Gegensatz zu klassischen Bits können sich Qubits in einer Überlagerung der Werte 0 und 1 (quantenmechanisch: Superposition) befinden. Erst beim Auslesen des Zustands nimmt es einen der beiden Zustände mit einer bestimmten Wahrscheinlichkeit an. Diese Wahrscheinlichkeit kann durch Coupler beeinflusst werden.

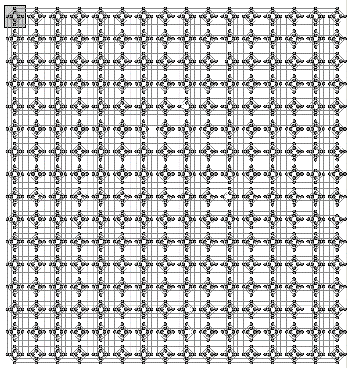


*Qubit auf einem D-Wave*

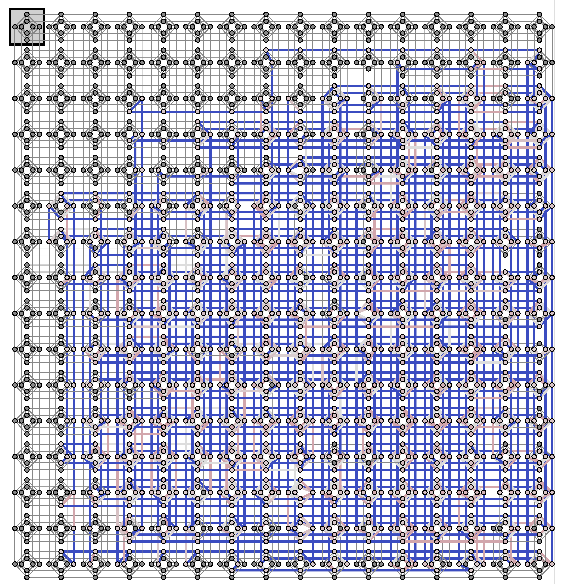
Qubits stellen einen Wert nicht wie Bits mit Transistoren durch Strom an oder aus dar, sondern durch den Spin (Drehrichtung) eines Elektrons, oder wie bei dem von uns verwendeten Rechner von D-Wave, durch die Flussrichtung des Stroms in einem supraleitenden Ring (siehe Abbildung rechts). Die Superposition entspricht also dem Fluss des Stroms in beide Richtungen.

## Chimera-Graph

Der sogenannte Chimera-Graph beschreibt die Verbindungen aller physikalischen Qubits auf einem D-Wave-Quanten-Annealer. Er zeigt alle aktuell aktiven Qubits (Ausfälle einzelner Qubits sind möglich) und deren Verbindungsmöglichkeiten zu anderen Qubits.



*Chimera Graph und Embedding*



Qubits können entweder mit dem Gleichheits- oder dem Ungleichheitscoupler gekoppelt werden. Wenn zwei Qubits mit einem Gleichheitscoupler gekoppelt sind, nehmen sie bei einer Messung zu einer hohen Wahrscheinlichkeit den gleichen Wert an. Beim Ungleichheitscoupler nehmen sie entgegengesetzte Werte an (also 1 und 0 oder 0 und 1). Die Kombination mehrerer gekoppelter Qubits nennt man Embedding. Das Embedding basiert auf der sog. Hamilton-Matrix.

## Hamilton-Matrix und Energieberechnung

Der Hamiltonian bildet die Basis für die Koppelung der Qubits auf dem Chimera-Graphen. Er bildet - in Form einer Matrix - ab, welche Kombination der Zustände zweier Qubits bestraft oder belohnt werden soll. Wenn man also nicht möchte, dass zwei Felder besetzt sind (z.B. die Felder a und b) trägt man an der Position (a|b) der Matrix den Wert 1 an. Wenn dann zwei Damen auf diese beiden Felder gesetzt werden, bestraft der Hamiltonian dies mit 1. Alle diese „Strafen“ oder auch „Fehler“ entsprechen zusammen der Energie, die bei einer optimalen Lösung minimal sein muss.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | b | c | d |
| a | 0 | 1 | 0 | 0 |
| b | 0 | 0 | 0 | 0 |
| c | 0 | 0 | 0 | 0 |
| d | 0 | 0 | 0 | 0 |

|  |  |
| --- | --- |
| a | b |
| c | d |

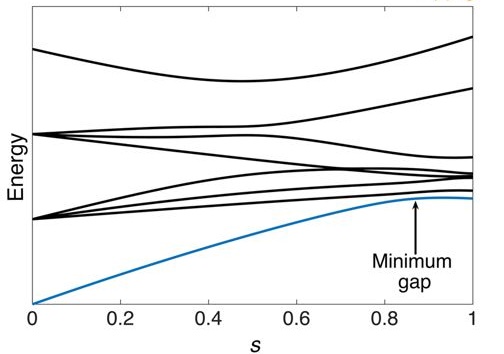
*Beispiel für eine Hamiltonian-Matrix*

Diese Energie wird auf Basis der Hamiltonian-Matrix H und eines Vektors berechnet, in dem die Werte der Qubits angetragen sind.

Die Schwierigkeit ist es also, für das Problem eine Hamiltonmatrix zu finden, die für die optimale Lösung die niedrigste Energie liefert.

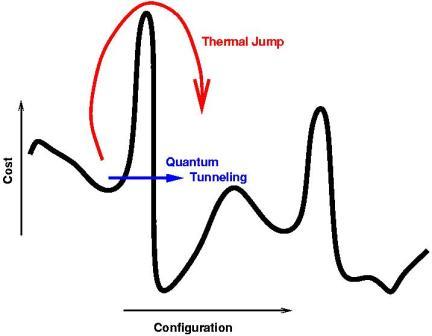
## Quantum Annealing

Ein Durchlauf eines Quantum-Annealing-Programms beginnt damit, dass mittels des Initial-Hamiltonian alle Qubits in Superposition „gesetzt“ werden. Dann wird der eigentliche problemspezifische Hamiltonian langsam „eingeblendet



*Minimum Gap in Zeit-Energie-Diagramm*

In der Grafik rechts sieht man (blau) den besten Energiezustand zu jedem Zeitpunkt des Annealing-Vorgangs. An der Stelle der Minimum Gap ist die Wahrscheinlichkeit, in einen schlechteren Zustand zu rutschen am größten. Je größer die Minimum Gap, desto leichter lässt sich das Problem mit einem Quantencomputer lösen.



*Tunneling in einer Energielandschaft*

Durch quantenmechanische Gesetzmäßigkeiten ist es außerdem möglich, beim unendlich langsamen Übergang vom einen in den anderen Hamiltonian alle Erhebungen in der Energielandschaft zu tunneln (siehe Grafik rechts). In der Theorie kann man durch unendlich langsamen Übergang zwischen beiden Hamiltonians immer das beste Ergebnis erzielen. In der Praxis reichen allerdings oft zweistellige Mikrosekundenbeträge, um zu einem Ergebnis zu kommen. Da Quantum-Annealer aber oft keine perfekte Lösung (globales Minimum), sondern nur eine gute Lösung (lokales Minimum) finden, lässt man das gleiche Problem üblicherweise ca. 10.000-mal nacheinander mit den gleichen Parametern berechnen. Abschließend soll erwähnt sein, dass die Anzahl richtiger Ergebnisse pro 10.000 Durchläufe sehr stark schwankt, da der Quantencomputer nicht immer im besten Energiezustand endet. Quantencomputer können allerdings bisher nur sehr kompakt formulierbare Probleme lösen.

## Von der Kostenfunktion bis zum Output

Nachdem ein Problem in eine Kostenfunktion verpackt wurde, wird diese an die Oceans-SDK von D-Wave weitergegeben, die das Embedding setzt. Das Embedding wird dann gemeinsam mit Parametern wie Durchlaufdauer und Anzahl der Durchläufe an den Server in Vancouver gesendet, der den Annealvorgang steuert. Nach Ende jeden Rechenvorgangs werden die Werte aller Qubits gemessen. Die Messwerte werden am Ende aller Anneals gesammelt zurückgeschickt und von unserem Programm problemspezifisch dekodiert.

# Formulieren der Hamilton-Matrix

## Darstellen des Problems

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Topf 0 | Topf 1 | Topf 2 | Topf 3 | Topf 4 | Topf 5 | Topf 6 | Topf 7 | Topf 8 |
| Blau | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Gelb | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| Grün | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| orange | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| rosa | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| rot | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| türkis | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 0 |  |  |
|  | 1 |  | 2 |  |
| 3 |  | 4 |  | 5 |
|  | 6 |  | 7 |  |
|  |  | 8 |  |  |

tabellarische Darstellung des Blumenbeets

Nummerierung des Blumenbeets

Ziel ist es nun, das Blumenbeet mit den möglichen Farbvarianten jedes Topfes mit den Ziffern 1 und 0 darzustellen. Dazu stellen wir uns eine Tabelle vor, auf deren x-Achse die einzelnen Blumentöpfe, und auf deren y-Achse die möglichen Farben für jeden einzelnen Blumentopf angetragen sind. In dem Beispiel rechts ist im Topf e eine gelbe Blume und im Topf 7 eine rote Blume platziert.

## Regeln und Umsetzung in die Matrix

Anhand dieses Modells können wir nun einige Regeln ableiten, die gelten müssen.

1. Jedem Topf darf nur eine Farbe zugeordnet werden -> pro Spalte höchstens eine 1
2. Jede Farbe soll mind. Einmal vorkommen -> möglichst wenig gleiche Farben -> möglichst wenig 1er pro Reihe
3. Insgesamt aber möglichst viele 1er, solange sie den vorherigen Regeln nicht widersprechen
4. Möglichst viele Kombinationswünsche erfüllen -> Die im Blumenbeet nebeneinanderliegenden Töpfe an den entsprechenden Farben „belohnen“

Um nun die Hamilton-Matrix zu berechnen, müssen wir die zweidimensionale Darstellung in eine eindimensionale umwandeln: Topf0blau, 0gelb, 0grün, …, 0türkis, Topf1blau, 1gelb, ……, 8rot, 8türkis.

Sowohl an der x- als auch an der y-Achse wird diese eindimensionale Reihe angetragen. Jetzt geht es darum, die „Belohnungen“ (negative Wert) und „Bestrafungen“ (positiver Wert) in der Matrix anzutragen. Daraus ergibt sich eine spiegelsymmetrische Matrix. Diese Belohnungen und Bestrafungen treten dann in Kraft, wenn beide Felder 1 sind (Also wenn z.b. Topf3grün und Topf7blau 1 sind, tritt die Bestrafung/Belohnung an der Stelle (3grün|7blau) in Kraft).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0blau | 0gelb | 0grün | 0orange | 0rosa | 0rot | 0tuerkis | 1blau | … |
| 0blau |  |  |  |  |  |  |  |  |  |
| 0gelb |  |  |  |  |  |  |  |  |  |
| 0grün |  |  |  |  |  |  |  |  |  |
| 0orange |  |  |  |  |  |  |  |  |  |
| 0rosa |  |  |  |  |  |  |  |  |  |
| 0rot |  |  |  |  |  |  |  |  |  |
| 0tuerkis |  |  |  |  |  |  |  |  |  |
| 1blau |  |  |  |  |  |  |  |  |  |
| … |  |  |  |  |  |  |  |  |  |

Zur Umsetzung der Regel A) werden u.a. an den Koordinaten (0blau|0gelb), (0blau|0grün), (0blau|0orange), …, (0gelb|0grün), (0gelb|0orange), … Bestrafungen angetragen. Dieses Prozedere wird bei jedem Topf widerholt (jedes Feld in einer Spalte des Modells wird mit jedem anderen Feld in derselben Spalte bestraft).

Für die Regel B) wird jedes Feld in einer Reihe des Modells mit jedem anderen Feld in derselben Reihe bestraft (z.B. (0grün|5grün) oder auch (2rot|8rot)).

Um, wie die Regel C) erfordert, möglichst viele 1er insgesamt zu setzen, muss man jedem Feld eine gewisse Grundbelohnung zuweisen, die immer gilt sobald das Feld besetzt ist. Dazu wird auf der Matrix-Diagonale eine Belohnung angetragen, da auf der Diagonale immer die Felder selbst doppelt liegen (z.B. (3orange|3orange) oder (6rosa|6rosa)).

Die Regel D) kann umgesetzt werden, indem, z.B. bei einem Kombinationswunsch „blau rosa 3“ bei allen zwei nebeneinanderliegenden Töpfen an den Farben blau und rosa Belohnungen angetragen werden. Beispiel: Topf 4 und Topf 7 liegen nebeneinander -> in der Matrix wird an der Stelle (Topf4blau|Topf7rosa) und (Topf4rosa|Topf7blau) eine Belohnung von -3 angetragen.

## Programmierte Matrixgenerierung

Nun wolle wir die Umsetzung der eben beschriebenen Matrixgenerierung in der Programmiersprache Processing (https://processing.org/) erklären.

Zur Matrixgenerierung geht das Programm mit einer verschachtelten for-Schleife durch die gesamte Matrix. Auf der x-Achse stehen die Variablen l für die l-te Farbe und x für den x-ten Topf und auf der y-Achse die Variablen m für die m-te Farbe und a für den a-ten Topf.

Für die Regel A) muss geprüft werden, ob die beiden Variablenpaare zwei Felder in der Matrix darstellen, die zum gleichen Topf gehören (x==a) und aber unterschiedliche Farben darstellen (l!=m). Bestrafung von z.B. 12

Zur Umsetzung der Regel B) wird geprüft, ob man die gleiche Spalte (l==m) aber nicht die gleiche Farbe betrachtet. Bestrafung von z.B. 6

Um auf der Diagonale der Matrix Belohnungen anzutragen (Regel C) ) überprüft das Programm, ob x==a und l==m ist. Belohnung von z.B. -4

Für die Belohnung einer Wunschkombination (Regel D) ) wird zuerst geprüft, ob man zwei nebeneinanderliegende Töpfe betrachtet. Dazu werden alle möglichen Kombinationen von a und x durchgegangen, an denen die beiden Töpfe nebeneinanderliegen. Danach wird die Liste aller Wunschkombinationen durchgegangen und überprüft, ob eine dieser Kombinationen grade von l und m dargestellt werden. Bestrafung von z.B. -1,5\*PunktefürWunsch

Die Belohnungs- und Bestrafungswerte müssen für jedes Beispiel neu angepasst werden, da z.B. bei umso weniger Farben immer öfter gegen Regel B) verstoßen werden muss.

void calculateHamiltonMatrix() {

  for (int l=0; l<colorNum; l++) {

    for (int x=0; x<flowerNum; x++) {

      for (int m=0; m<colorNum; m++) {

        for (int a=0; a<flowerNum; a++) {

          if (x==a && l!=m) { //nur eine Farbe pro Kästchen

            hamiltonMatrix[x\*colorNum+l][a\*colorNum+m] += 12;

          }

          if (x!=a && l==m) { //möglichst wenig gleiche Farben

            hamiltonMatrix[x\*colorNum+l][a\*colorNum+m] +=6;

          }

          if (x==a&&l==m) { //Grundbelohnung

            hamiltonMatrix[x\*colorNum+l][a\*colorNum+m] -= 4;

          }

//Wunschkombinationen belohnen

          if ((x==0&&a==1||x==1&&a==0)||(x==0&&a==2||x==2&&a==0)||

            (x==1&&a==2||x==2&&a==1)||(x==1&&a==3||x==3&&a==1)||(x==1&&a==4||x==4&&a==1)||

            (x==2&&a==4||x==4&&a==2)||(x==2&&a==5||x==5&&a==2)||

            (x==3&&a==4||x==4&&a==3)||(x==3&&a==6||x==6&&a==3)||

            (x==4&&a==5||x==5&&a==4)||(x==4&&a==6||x==6&&a==4)||(x==4&&a==7||x==7&&a==4)||

            (x==5&&a==7||x==7&&a==5)||

            (x==6&&a==7||x==7&&a==6)||(x==6&&a==8||x==8&&a==6)||

            (x==7&&a==8||x==8&&a==7)) {

            for (int i=0; i<wishes.length; i++) {

              if (wishes[i][0]==m&&wishes[i][1]==l || wishes[i][0]==l&&wishes[i][1]==m) {

                hamiltonMatrix[x\*colorNum+l][a\*colorNum+m] -= 1.5\*wishes[i][2];

              }

            }

          }

        }

      }

    }

  }

}

## Programmablauf

Zuerst wird die Einagbedatei (input.txt) vom Programm gelesen, in der sich die Anweisungen einer Aufgabe von der BwInf-Website finden. Im nächsten Schritt wird jeder Farbe eine ID zugewiesen. Bei weniger als sieben zu verwendenden Farben werden zuerst den in der Wunschkombinations-liste vorkommenden Farben eine ID zugewiesen und danach so viele weitere ID’s vergeben bis die Grenze an erwünschten Farben erreicht ist. Danach wird die Hamilton-Matrix, wie oben beschrieben, berechnet und als Textdatei exportiert. Um unsere Matrix zu testen haben wir außerdem ein einfaches simulated-Annealing implementiert. Dieses optimiert so lange, bis es die möglichst beste Lösung gefunden hat und gibt diese Lösung in der Konsole aus.

## Lösen auf dem Quantencomputer und dem DAU

Quantencomputer stehen noch ganz am Anfang ihrer Entwicklung. Besonders größere Probleme können sie nur schwer lösen. Sie sind dafür ausgelegt, viele sehr gute, nicht aber unbedingt die Beste Lösung zu finden. Die Rechenzeit auf dem Quantum-Annealer von D-Wave konnten wir über das Forschungszentrum, mit dem wir auch schon bei unserem letztjährigen Projekt zum Thema Quantum computing zusammengearbeitet haben, bekommen.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 0 |  |  |
|  | 1 |  | 2 |  |
| 3 |  | 4 |  | 5 |
|  | 6 |  | 7 |  |
|  |  | 8 |  |  |

Nummerierung des Blumenbeets

Der DAU (Digital Annealing Unit) von Fujitsu ist ein herkömmlicher Hochleistungsrechner, der ebenfalls für Optimierungsprobleme ausgelegt ist. Auf ihn konnten wir Dank Herrn Fritz Schinkel von Fujitsu München zugreifen.

Hier die Lösungen vom DAU:

Aufgabe:             0

Punkte: 14

Lösung:           [0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0]

ros

bla tue

rot rot gru

bla ora

gel

0:rosa 1:blau 2:tuerkis 3:rot 4:rot 5:gruen 6:blau 7:orange 8:gelb

Mehr Wunschfarben als max. Farbanzahl! Wunschkombination „gruen rot 1“ ausgelassen

Aufgabe:             1

Punkte: 36

Lösung:           [0 1 1 0 1 0 0 1 0 1 0 1 1 0 1 0 0 1]

tue

rot rot

tue tue tue

rot rot

tue

0:tuerkis 1:rot 2:rot 3:tuerkis 4:tuerkis 5:tuerkis 6:rot 7:rot 8:tuerkis

Aufgabe:             2

Punkte: 32

Lösung:           [0 0 1 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 0]

gru

rot rot

tue tue tue

rot rot

tue

0:gruen 1:rot 2:rot 3:tuerkis 4:tuerkis 5:tuerkis 6:rot 7:rot 8:tuerkis

Aufgabe:             3

Punkte: 13

Lösung:           [0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0]

ora

tue rot

gel rot tue

bla gru

ros

0:orange 1:tuerkis 2:rot 3:gelb 4:rot 5:tuerkis 6:blau 7:gruen 8:rosa

Aufgabe:             4

Lösung:           [1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0]

Lösung falsch!

2 Farben Topf 4 zugeordnet (Regel A) )

Farbe gelb nicht benutzt (Regel B) )

Aufgabe:             5

Punkte: 22

Lösung:           [0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0]

ros

bla rot

gel tue tue

ora rot

gru

0:rosa 1:blau 2:rot 3:gelb 4:tuerkis 5:tuerkis 6:orange 7:rot 8:gruen

Lösungen des Quantum-Annealers:

Aufgabe:             0

Punkte: 12

Lösung:           [0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0]

gel

ros ora

rot bla tue

tue rot

gru

0:gelb 1:rosa 2:orange 3:rot 4:blau 5:tuerkis 6:tuerkis 7:rot 8:gruen

Aufgabe:             1

Punkte: 36

Lösung:           [1 0 0 1 0 1 1 0 1 0 1 0 0 1 0 1 1 0]

rot

tue tue

rot rot rot

tue tue

rot

0:rot 1:tuerkis 2:tuerkis 3:rot 4:rot 5:rot 6:tuerkis 7:tuerkis 8:rot

Aufgabe:             2

Punkte: 32

Lösung:           [1 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0]

tue

rot rot

tue tue gru

rot rot

tue

0:tuerkis 1:rot 2:rot 3:tuerkis 4:tuerkis 5:gruen 6:rot 7:rot 8:tuerkis

Aufgabe:             3

Punkte: 13

Lösung:           [0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0]

rot

tue tue

ros rot gru

bla ora

gel

0:rot 1:tuerkis 2:tuerkis 3:rosa 4:rot 5:gruen 6:blau 7:orange 8:gelb

Aufgabe:             4

Punkte: 19

Lösung:           [0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0]

rot

gru ros

tue rot bla

ora ros

gel

0:rot 1:gruen 2:rosa 3:tuerkis 4:rot 5:blau 6:orange 7:rosa 8:gelb

Aufgabe:             5

Punkte: 21

Lösung:           [0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0]

gel

rot tue

ros tue rot

bla gru

ora

0:gelb 1:rot 2:tuerkis 3:rosa 4:tuerkis 5:rot 6:blau 7:gruen 8:orange

Wie man sieht, konnte der DAU meist deutlich bessere Ergebnisse erzielen als der Quantencomputer. Mit unserem simulated Annealing konnten wir allerdings für die Aufgabe 4 eine noch bessere Lösung bekommen als mit dem DAU oder dem Quantum-Annealer:

Aufgabe:             4

Punkte: 22

Lösung:           [0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0]

gel

tue ora

rot rot gru

tue ros

bla

0:gelb 1:tuerkis 2:orange 3:rot 4:rot 5:gruen 6:tuerkis 7:rosa 8:blau

Die von dem DAU und dem Quantencomputer zurückgegebenen Lösungen in 0 und 1 werden von unserem decodeSolution Programm ausgewertet. Es liest die Liste mit Wunschkombinationen, die Matrix und die Lösung ein und berechnet daraus die Punkteanzahl, zeigt ob gegen Regeln verstoßen wurde und gibt die Verteilung der Farben auf die Töpfe in der Konsole aus.