令和元年度

筑波大学 応用理工学類

卒業研究論文

量子アニーリングマシンを用いた量子計算の評価

主専攻名：応用物理主専攻

学籍番号：201611030

著者名：土肥野 真也

指導教員：数理物質系 物理工学域 小林 伸彦

目次

要旨

D-Wave systems 社が提供する量子アニーリングマシン D-Wave2000Q は、組み合わせ最適化問題に特化した量子アニーリングという量子計算を行う。量子アニーリングでは、対象の組み合わせ最適化問題をイジングモデルに定式化して解探索を行う。この量子アニーリングによる量子計算を私たちが使う一般的なコンピュータによる計算と比較して、計算時間や扱うことのできる問題の大きさなどを評価した。まず、”数の分割問題”という組み合わせ最適化問題を解く量子計算の計算速度を、一般的なコンピュータによる bit 全探索のアルゴリズムによる計算と比較した。結果として、bit 全探索よりも圧倒的に早ことが分かった。次に、”部分和問題”という問題の計算速度を DP（動的計画法）のアルゴリズムの計算と比較した。量子計算は、条件によって DP よりも早く最適解を導いた。また、2 つの組み合わせ最適化問題両方において D-Wave2000Q が扱える問題のスケールの限界が一般的なコンピュータよりも小さいことが分かった。検証を通して、一般的なコンピュータにおいて効率の良いアルゴリズムが用意できない複雑性の高い問題に対して量子アニーリングはかなり効果的であるといえることが分かった。

第 1 章

序論

株価予測や電子状態計算のような物理シミュレーション、暗号分野など、量子コンピュータが活用される分野や領域は数多くあり、量子コンピュータの研究は盛んに行われている。量子コンピュータの中で組み合わせ最適化問題を解くことに特化した量子アニーリングマシンがあり、これも交通における渋滞予測や集積回路の設計最適化、結晶の構造決定などのような物理分野などで活用され始めている。現在開発されている量子アニーリングマシンに、D-Wave systems 社が開発した D-Wave2000Q があり、この量子アニーリングマシンをオンライン上にて 1 分間使用できるクラウドサービスが提供されている。このように現在量子アニーリングマシンは身近なものとなりつつあり、汎用的な計算のツールとして使うことができると思われる。本論では、D-Wave2000Q を利用するにあたっての計算速度や限界などの実用性の評価を行い、計算ツールとしての量子アニーリングマシンの利用について考察していく。

第 2 章 量子アニーリングマシンによる量子計算

2.1 量子コンピュータ

2.1.1 量子コンピュータとは

量子コンピュータは ”0” と ”1” を確率的に状態として表現する量子ビットを量子力学的な重ね合わせを用いて計算を行う。この性質上、量子コンピュータにおいては同じ入力値に対して結果は異なる可能性がある。量子コンピュータは、量子ビットに対して量子力学的な操作を行った後、量子ビットを測定することで解を得る。

2.1.2 量子ビット

量子ビットは、下記のように複素ベクトルで表せる。

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (2.1)$$

ここで左辺は量子ビットを、右辺の |0⟩,|1⟩ はそれぞれ通常のビットと等価の状態 0 と 1 を表します。左辺はベクトルで、α, β は確率振幅を表す複素数の値で、確率振幅は絶対値の二乗をとると通常の意味での確率になる。式(2.1)は一つの量子ビットを表したものであり、n量子ビットは重ね合わせにより

$$|\psi_{n-1}\rangle \cdots |\psi_1\rangle\ |\psi_n\rangle = a_1|11\cdots11\rangle + a_2|11\cdots10\rangle + \cdots a_{2^n}|00\cdots00\rangle \quad (2.2)$$

量子ビットに対する操作で2^n通りのビット状態が変化することが量子的な並列性を表しており、これが量子計算の有利な点である。

2.1.3 量子コンピュータの研究

現在では盛んに研究がされている量子コンピュータは、主に二つの方向性がある。我々が普段使うようなコンピュータのように論理回路としての量子ゲートを用いて計算を行う汎用量子コンピュータと、組み合わせ最適化問題を解くことに特化させた量子アニーリングマシンである。本論では後者である量子アニーリングに注目する。

2.2 組み合わせ最適化問題

組合せ最適化問題は、多数の選択肢がある中で、最も良い組合せを選ぶ問題である。例えば、乗換案内のような出発地と目的地とを結ぶ「最適」な経路探索が挙げられる。問題ごとに決めた制約下でもっとも良い組合せを決定することを組み合わせ最適化問題と呼ぶ。組合せ最適化問題は全ての組合せを調べれば良く、計算方法としては単純であるが、原理的には全ての可能性を列挙する必要があるということは、問題の規模や複雑性に対し爆発的に組合せの数が増えてしまい、現在のコンピュータをもってしても、計算に莫大な時間がかかってしまうこととなる。また、現在の金融システムやネットの RSA 暗号、ブロックチェーンの公開鍵と秘密鍵など、「素因数分解」を応用した暗号技術は、古典コンピュータでは解くのに膨大な時間を要することを前提に成り立っている。組み合わせ最適化問題を高速に解くということは、上記の暗号分野にとても大きな影響を与えることになる。

2.3 量子アニーリング方式

2.3.1 Simulated アニーリング

組み合わせ最適化問題の解探索法として、焼きなまし法（シミュレーテッドアニーリング）焼きなまし法がある。最適解の探索のために組合せを変化させていく過程で、必ずしもコスト関数が小さくならなくてもある確率で次の状態として選択する。この時、どの程度コスト(エネルギー)の低下を重要視するかを「温度」というパラメータで制御する。高い温度の場合には目まぐるしく組合せを変化させるのに対し、低い温度の場合にはエネルギーの低い方向にしか変化しなくなる。高温から低温に徐々に変化させることで局所解での凍結を回避し最適解の探索を行う。

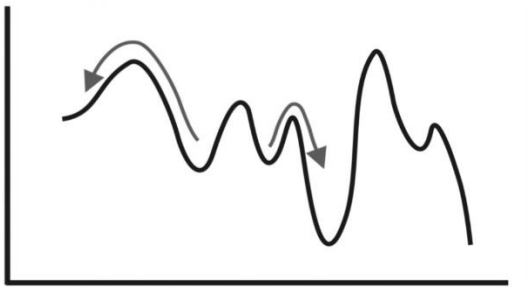

図 2.1: simulated アニーリングの概念図 [1]
横軸に状態、縦軸にコスト関数(エネルギー)

2.3.2 量子アニーリング

量子アニーリングでは温度の代わりに量子効果(トンネル効果)を変化させる。量子効果が非常に強い状況では、すべての組み合わせ状態の重ね合わせになっており、この時に測定を行うと全状態が等しい確率で出力される。つまり、量子効果が強いということ、これは各状態の間でトンネル効果による状態遷移が起きるということで、これに対して、量子効果を小さくすることで状態が動かなくなる。まさに温度アニーリング(シミュレーテッドアニーリング)の量子版といえるので量子アニーリングと呼ばれる。

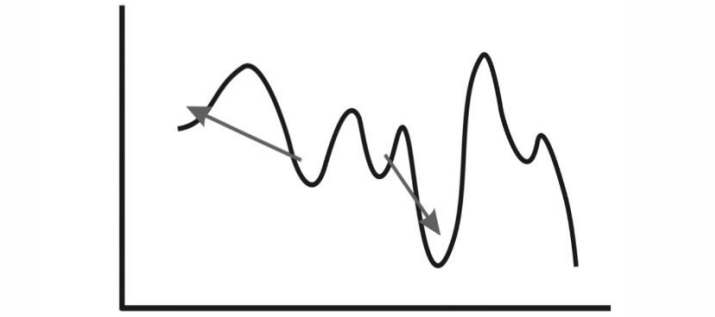

図 2. 2:量子アニーリングの概念図 [1]
横軸に状態、縦軸にコスト関数(エネルギー)

2.3.3 イジングモデル

アニーリングマシンは実装として、上記で述べたアニーリングを高速に実行し近似最適解を出力するのだが、このアルゴリズムの長所は汎用性にあり、一般的なアニーリングマシンでは汎用的に問題を受け付けられるイジングモデルと呼ばれる計算モデルを用いてアニーリングアルゴリズムへと適用している。[2]

イジングモデルは統計物理でよく使われる下記のエネルギー関数で表される。

$$E(\{s_i\}) = \sum_{i \in V} h_i s_i + \sum_{(i,j) \in E} J_{ij} s_i s_j \quad (2.3)$$

ここで s_iは入力変数を表し、統計物理での呼び名のままにスピンと呼ぶ。 $s_i \in \{-1,1\}$である。J_{ij}は(二体の)相互作用パラメータ、h_iは (一体の) パラメータとして磁場と呼ばれる。アニーリングマシンへの入力は h_i, J_{ij}を与えるのみでアニーリングを実行するように、エネルギーが最低となる $s_i \in \{-1,1\}$の組み合わせを出力するように実装されている。

2.3.4 量子アニーリングとイジングモデル

前で述べたように、量子アニーリングマシンにはイジングモデルが物理的に実装されている。この物理的なイジングモデルの実際のパラメータに、対象の問題をマッピングすることで、マシンが実現される。

2.3.5 イジングモデルによるアルゴリズム

量子アニーリングは外部からイジング模型を時間変化させることで行う。しかし、単純にイジング模型の相互作用や磁場のパラメータのみを時間変化させるのではなく、新たに量子ゆらぎと呼ばれる項を導入して、これを制御する。アニーリング開始時刻$(t = 0)$では量子ゆらぎの項のみ、アニーリング終時刻$(t = \tau)$では最適化したいイジング模型のみになるように状態の入れ替えを行う。式で表すと、

$$H(t) = A(t)H_A + B(t)H_B \quad (2.4)$$

となる。H_Aは量子効果を表す項を、H_Bは最適化したい関数の項をそれぞれ表す。また、$A(t)$、$B(t)$については

$$H(0) = A(0)H_A$$
$$H(\tau) = B(\tau)H_B$$

となる必要があり、

$$\frac{B(0)}{A(0)} = 0 \quad (2.5)$$

$$\frac{A(\tau)}{B(\tau)} = 0 \quad (2.6)$$

とできる。

量子効果を表すH_Aは、量子力学的な重ね合わせの度合いを表す。初期時刻では全ての取りうる状態が重ね合わせの状態にあり、これを徐々に弱くしていくと同時にエネルギー関数の項を強くすることで、終時刻ではエネルギーの低い最適解近傍のみ重ね合わせとして残るようにする。先に述べた量子アニーリングである。実際の実装ではエネルギー関数をイジング模型として、量子ゆらぎを「横磁場」と呼ばれる項で導入している。以下のようにして、イジング模型が量子力学的な重ね合わせの性質を持つようにする。

前に述べたように、量子ビットの状態は複素ベクトルで表すことができる。1 量子ビットの場合は(2.1)式より、

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

ここで、シュレーディンガー方程式

$$H|\psi\rangle = E|\psi\rangle \quad (2.7)$$

により、エネルギーから逆説的にハミルトニアンを求めることで、状態ベクトルとしての量子ビットを求めることができる。このように、イジングモデルのスピンを、量子力学的に、量子ビットとして扱っている。

これまでのハミルトニアンは最適化したい関数のみを考慮した場合、つまり$H = H_B$であった。次に、量子効果H_Aについて、H_Aは大きく言うとハミルトニアンの非対角成分のことで、横磁場という形で導入される。最適化の対象に対するスピンをs^z、横磁場に対応するスピンをs^xとあらわす。例えば、

$$H_A = -\Gamma s^x、H_B = -Js^z \quad (2.8)$$

$$H = H_A + H_B \quad (2.9)$$

とした場合のハミルトニアンを考える。s^z と s^x は同じスピンに対する別の見方を表しています。H_B の行列表現は、H_A で述べたことに対して対角部分にエネルギー値を全通り並べたものである。

$$H_B = \begin{pmatrix} -J & 0 \\ 0 & J \end{pmatrix} \quad (2.10)$$

$$H_B = -Js^z \quad (2.11)$$

より、

$$s^z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad (2.12)$$

であることがわかる。このときスピンが行列で表現されるようになっている。このs^zについて、s^xは次の行列で定義する。

$$s^x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad (2.13)$$

s^xは、演算子としてスピンの方向を変える。

$$s^x|1\rangle \ = |0\rangle \quad (2.14)$$

$$s^x|0\rangle \ = |1\rangle \quad (2.15)$$

1 スピンのハミルトニアンを行列式で書くと下記のようになる。

$$H \ = \ H_A + H_B = \begin{pmatrix} -J & -\Gamma \\ -\Gamma & J \end{pmatrix} \quad (2.16)$$

である。さらに、2 スピンの場合は、$H_A = -\Gamma(s_1^x + s_2^x)$とすると、1 スピンの場合と同様に、それぞれ自身のスピンを反転するため、

$$s_1^x|11\rangle \ = |10\rangle \ 、s_2^x|11\rangle \ = |01\rangle \quad (2.17)$$

という性質があり、これにより

$$H_A = \begin{pmatrix} 0 & -\Gamma & -\Gamma & 0 \\ -\Gamma & 0 & 0 & -\Gamma \\ -\Gamma & 0 & 0 & -\Gamma \\ 0 & -\Gamma & -\Gamma & 0 \end{pmatrix} \quad (2.18)$$

となる。これを言い換えると、非対角座標 $i,\ j$ のうち、行番号 iに対して 1 スピン(ビット)反転した列 j の要素に横磁場の値が入る。 例えば、H_A の 1 行目は $|11\rangle$ が対応しますが、スピン 1 が反転すると $|10\rangle$、スピン 2 が反転すると$|01\rangle$ なのでそれぞれ 1 行 2 列目、1 行 3 列目が埋まる。 これを全行で繰り返すと上のように非対角成分の一部が埋まる。

これはNスピンの場合にも同様で、横磁場による量子揺らぎを用いると、各行に対して 1 スピンだけ異なる列が横磁場の値で埋まる。そしてそれ以外の非対角要素は 0。H_B のように、対角項だけなら固有値・固有状態(固有ベクトル)は自明で互いに独立であるが、非対角項 H_A が導入されることで、ある状態と別の状態の間で遷移が発生する。これをトンネル効果と呼ぶ。

2.3.6 ハミルトニアンの時間変化

量子アニーリングは対角行列であるターゲット関数と非対角項の量子効果を時間変化させ、初期時刻ではH_Aのみ、終時刻ではH_Bのみが有効になるようにする。

まず、時刻$t = 0$で初期状態は、H_Aとして横磁場を用いる場合に

$$H_A = -\sum_i^N s_i^x \quad (2.19)$$

基底状態がすべてのスピン等確率で持つ状態の組み合わせ状態として設定する。

この初期状態 $H(0) = A(0)H_A$ から、終時刻 $H(\tau) = B(\tau)H_B$ までハミルトニアンを変化させる。行列式で表すと以下のように表せる。

$$H(t) = \begin{pmatrix} B(t)E_{2^N-1} & -A(t) & \cdots & 0 & 0 \\ -A(t) & B(t)E_{2^N-2} & & -A(t) & 0 \\ \vdots & & \ddots & \vdots & \\ 0 & -A(t) & & B(t)E_1 & -A(t) \\ 0 & 0 & \cdots & -A(t) & B(t)E_0 \end{pmatrix} \quad (2.20)$$

このハミルトニアンを用いて、シュレーディンガー方程式を解くことで状態ベクトル(量子ビット)が記述される。

$$i\hbar\frac{d}{dt}|\psi(t)\rangle = H(t)|\psi(t)\rangle \quad (2.21)$$

理想的なアニーリングマシンでは、量子ビットは上記の微分方程式に従う。

時間変化を極めてゆっくりと変化させるようなスケジューリングを選んだ場合、最適化問題の最低エネルギー状態へと必ず到達できることが分かっている。しかし、現実的には適当な時間で終了させる必要がある。また、適切な試行回数で最適解へと到達できれば良いから、必ずしも一回の試行で最適解にたどり着く必要がない。また、最適解近傍の解が得られれば十分という場合も問題によってありうる。これについて、後述する D-Wave へ渡すスクリプトを書く際に、用意されているパラメータで設定できる。

３章 D-Wave system

3.1 D-Wave systems 社

D-Wave systems 社は量子コンピューティング・システム、およびそのソフトウェアを開発提供する企業で、2011 年 5 月 11 日、D-Wave Systems は「世界初の商用量子コンピュータ」を謳った D-Wave One を発表した。D-Wave One は、128 量子ビットが実装されている量子アニーリングマシンである。現在はさらに多くの量子ビットを実装した D-Wave 2000Q システム [3] を完成させた。D-Wave systems が提供するサービスに D-Wave 2000Q を用いて私的な計算を行えるというものがあり、これを利用して本論を進めていく。

3.2 D-Wave 2000Q

3.2.1 ハードウェア

D-Wave 2000Q は極低温下の環境が保たれており、装置の大きさのほとんどは冷却装置が占めている。また、量子計算を実行するために QPU には隔絶された環境が必要で、QPU は装置内部の外磁場や振動、RF 信号などからの影響のない液体ヘリウムを用いた絶対零度(15mK)の真空環境下にある。また、D-Wave 2000Q システムはデータセンターやクラウド、スーパーコンピューターとオンライン上でつながっている。隔絶されている QPU が外部のシステムと連携するために $1nT$以下の高透磁率の超伝導体を用いている。[3]

図 3.1: D-Wave 2000Q [3]

3.2.2 磁束量子ビット

D-Wave 2000Q システムには、量子ビットとして超伝導量子ビットの一つである磁束量子ビットが実装されている。

3.2.3 ジョセフソン結合

超伝導量子ビットの最小構成要素は、ジョセフソン接合と呼ばれる超伝導デバイスである。[4] ジョセフソン接合は、数ナノメートル程度の極めて薄い絶縁膜を二つの超伝導体で挟んだ構造を有している。(図) 超伝導は，超伝導を担う電子の対（クーパー対）が転移温度以下で凝縮して発現する巨視的量子現象である。その結果，超伝導体は一つの巨視的波動関数（複素関数） によって記述される。ジョセフソン接合においては、電圧を印加しなくて もクーパー対のトンネル効果によって電流が流れる。 これは直流ジョセフソン効果と呼ばれる。接合に流れる電流は、二つの超伝導体の巨視的波動関数の位相差 $\varphi = \varphi 1 - \varphi 2$ を用いて $I_J(\varphi) = I_C \, sin\varphi$ と表される。ここで、I_C はジョセフソン臨界電流と呼ばれる。このジョセフソン接合は，超伝導量子回路を構成する基本要素であり、位相差 φ に対する sin 依存性から非線形なインダクタとしても振る舞う。

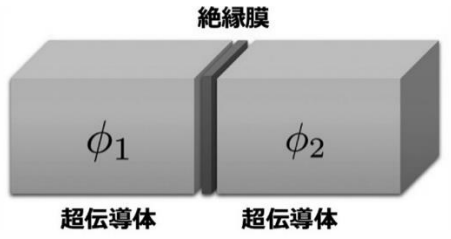

図 3.2:ジョセフソン接合 [4]

磁束量子ビットは，ジョセフソン接合を(図 3.2)のように配置した超伝導リングである。この量子ビットにおいては、外側の大きなリングを時計（反時計）回りに巨視的な超伝導電流が流れる状態をスピン変数 $+1(-1)$ に対応させる。それぞれの状態は、大きな外側リングを貫く磁束が $\pm\varphi_0/2$ の状態に対応する。ここで、$\varphi_0 = h/2e$ は磁束量子である（e は素電荷、h はプランク定数)。D-Wave 2000Q Systems この超伝導磁束量子ビットがスピンとして用いられている。前節で述べていることだが、磁束量子ビットを用いた超伝導量子アニーリングマシンにおいては、外側の大きなリングに印加する磁束 Φ_{out} を調整することで局所磁場 h_i を制御する。（図 3.3）また、量子アニーリングに必要な横磁場は、内側の小さなリングに外部から磁束を印加することで実現する。横磁場 Φ_{in}を印加することで、スピンが $+1$ と-1 の 2 状態の量子力学的重ね合わせが生成される。量子アニーリングの過程においては、この横磁場を徐々に弱めていく。

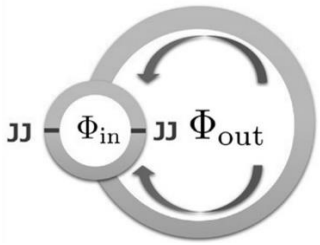

図 3.3: 磁束量子ビット [4]

磁束量子ビットを用いた超伝導量子アニーリングマシンにおいては，結合器を媒介した磁気的相互作用を用いてスピン間結合$J_{i,j}$ を間接的に実現している（図 3.4)。両端の超伝導リングは磁束量子ビットであり、真ん中の超伝導リングは結合器である。結合 器の中にある小さな超伝導リングに印加する磁束 Φ_{co} を制御することで$J_{i,j}$ を可変にできる。そのため、真ん中の超伝導リングはプログラマブル結合器とも呼ばれる。

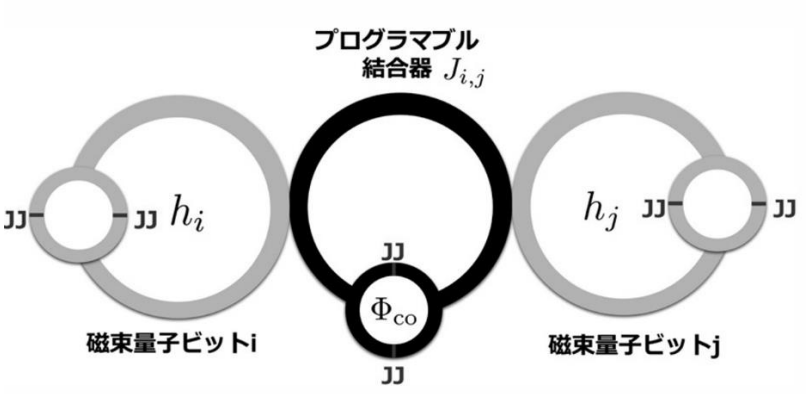

図 3.4: 量子ビット間の結合 [4]

3.2.4 キメラアーキテクチャ(グラフ)

量子アニーリングマシンのハードウェアにおいては、 最隣接スピンのみならず遠くのスピンとの結合も実装する必要がある。しかし、遠距離相互作用を実装することは現実的に極めて困難であるため、D-Wave 2000Q システムでは，キメラアーキテクチャ（グラフ）と呼ばれるグラフ理論に基づくハードウェアアーキテクチャを考案・実装している。[3] これによって，最隣接としか相互作用しないスピングラス模型（キメラグラフ）（図 3.4）の中に，遠くと相互作用するスピングラス模型を埋め込むことが可能となる。ただし、N スピンの全結合スピングラス模型を実装するためには， 約 N^2 個の最隣接相互作用するスピングラス模型が必要となる。つまり、相互作用のスピンとして運用しなければいけない量子ビットが全体の量子ビットの何割かを占めるのである。

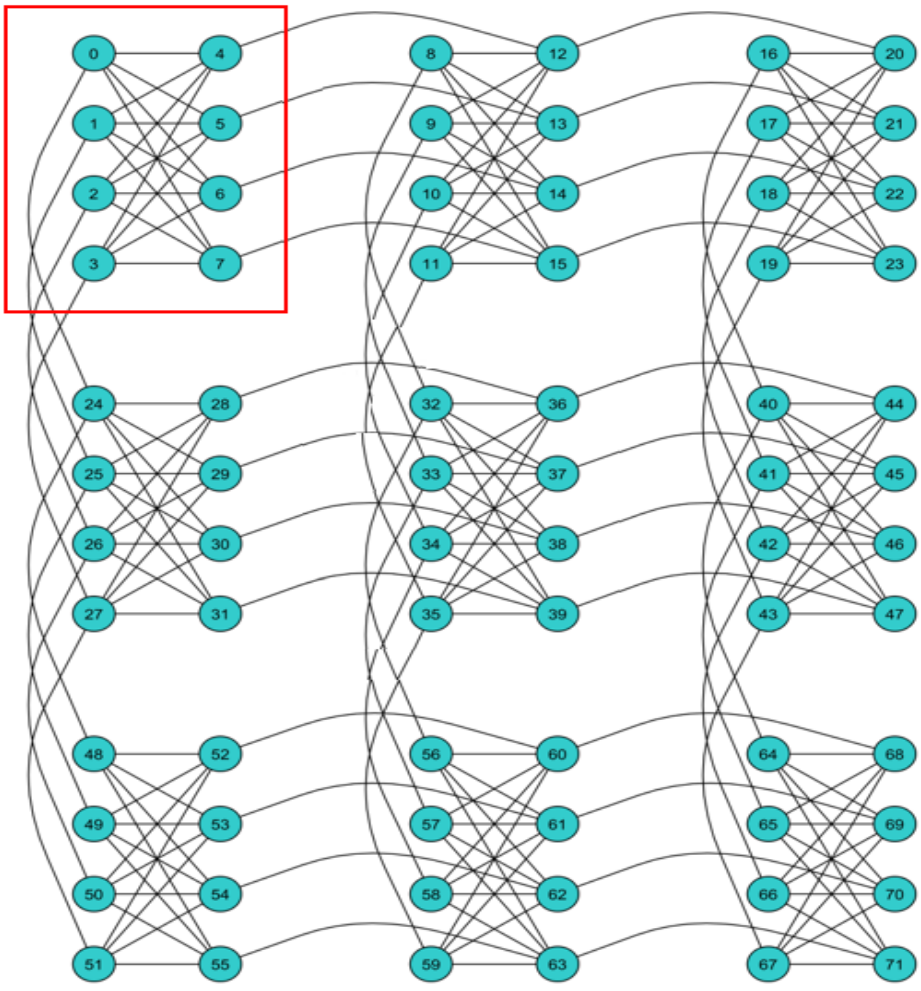

図 3.5: キメラアーキテクチャの概念図 9×9 の実装[3]

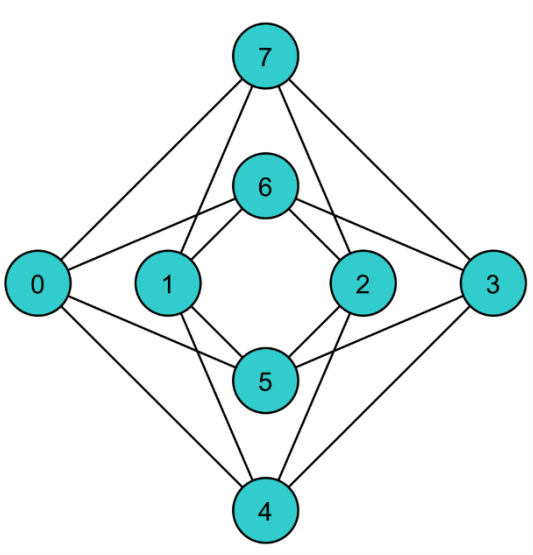

図 3.6: 1 ユニットを十字形で表したグラフ [3]

D-Wave 2000Q QPU のキメラグラフは 8 量子ビットを 1 ユニット（図 3.4 赤枠）として、16×16 の行列として並んだ 2048 量子ビットが実装されている。キメラグラフによって、量子ビットと結合器（couplers）に目的関数を表す過程は、Minor embedding と呼ばれている。

3.3 クラウドサービス

3.3.1 Leap

Leap は、D-Wave 2000Q にアクセスできるクラウドサービス。アカウントを登録することで API トークンのコードを取得して、無料で 1 か月に 1 分間だけ D-Wave 2000Q システムを利用して計算を行うことができる。具体的に Leap を利用する手順を以下に示す

(1) Leap に登録し API Token のコードを取得する。

(2)その後、Leap に認識させるために Python にて組み合わせ最適化問題を定義したコードを記述する。

(3)作成した Python のコードに API に渡すためのコードを追加する。

(4) D-Wave 2000Q が入力を受け取り、量子アニーリングによる計算を行って出力を返す。

また、Python で D-Wave 2000Q を利用するためには Python 環境にてコマンド入力

```
pip install dwave-system
```

によって必要なライブラリをインストールしておく。

3.3.2 QUBO(Quadratic Unconstrained Binary Optimization)モデル

Python にて組み合わせ最適化問題を定義、記述するにあたり、D-Wave 2000Q に計算をさせるためには対象の組み合わせ最適化問題をイジングモデルへと置き換えなければならないのだが、D-Wave ではこれについて下記の変数変換を行い、スピン変数以外にバイナリ変数として QUBO モデルへと入力することがある。[2]

$$q_i = \frac{1}{2}(s_1 + 1) \ , s_i \in \{-1,1\} \quad (3.1)$$

これにより、$q_i \in \{0,1\}$ となり、QUBO モデルを

$$E = q^T Q q \quad (3.2)$$

のような 2 次形式で表すことができる。Q はコストや制約を踏まえた行列で、辞書式(dict 型)に定義することで組み合わせ最適化問題を定義し、D-Wave に入力として渡すことができる。

x

3.3.3 PyQUBO

3.3.2 では、目的の組み合わせ最適化問題の定義は、二次形式の QUBO モデルで表し、Python において dict 型の辞書式の係数行列を構成する必要があると述べた。これにおいて、現在、（株）リクルートコミュニケーションズが 2018 年 9 月 25 日にオープンソースとして公開したドメイン固有言語 PyQUBO がある。[5] 定式化した組合せ最適化問題をアニーリングマシンで解くために PyQUBO を使用することで、QUBO への変換を自動的に行うことが可能となる。

具体的には、QUBO モデルにおいて複雑であった係数行列について、PyQUBO の利用によりコスト関数として直接的な数式で記述できるようになるため、簡潔かつ可読性の高いコードを書くことができるようになる。PyQUBO は Python での利用が可能になっているため、Python の環境にて以下のコマンドでライブラリをインストールしておく必要がある。

```
pip install pyqubo
```

また、以下に PyQUBO で用いるクラスまたはメソッドについて記す。[6]

クラスまたはメソッド	概要
Array	バイナリ変数またはスピン変数を取り扱うクラス
Constraint	コスト関数の制約項を記述するためのクラス
Express.compile()	記述したコスト関数をモデルにコンパイルするためのメソッド
Model.to_qubo()	モデルを QUBO に変換するためのメソッド。予め設定した Placeholder を dict 形式で渡す。
Model.decode_solution()	得られた解をデコードするためのメソッドデコードした結果に加えて制約を満たしているかの結果とエネルギーを返す。
Placeholder	コスト関数の各項のパラメータを記述するためのクラス。
solve_qubo()	QUBO を解くための関数。

Placeholder のパラメータの値を適宜決めることで、効率的な量子アニーリングを行うことができる。エネルギー関数の最小化で、QUBO の U が”制約なし”を意味するように、PyQUBO を用いない場合、制約条件はラグランジュの未定乗数項としてコスト関数に組み込んでいる。その際に各制約項の重みとなるパラメータを設定する。Placeholder は、このパラメータを一時的に保管するためのものであり、コスト関数の各項を計算した後に

Placeholder の値を変えることで、パラメータの調整が簡便になる。PyQUBO を用いない場合は、何度も各項の計算を行う必要があった。

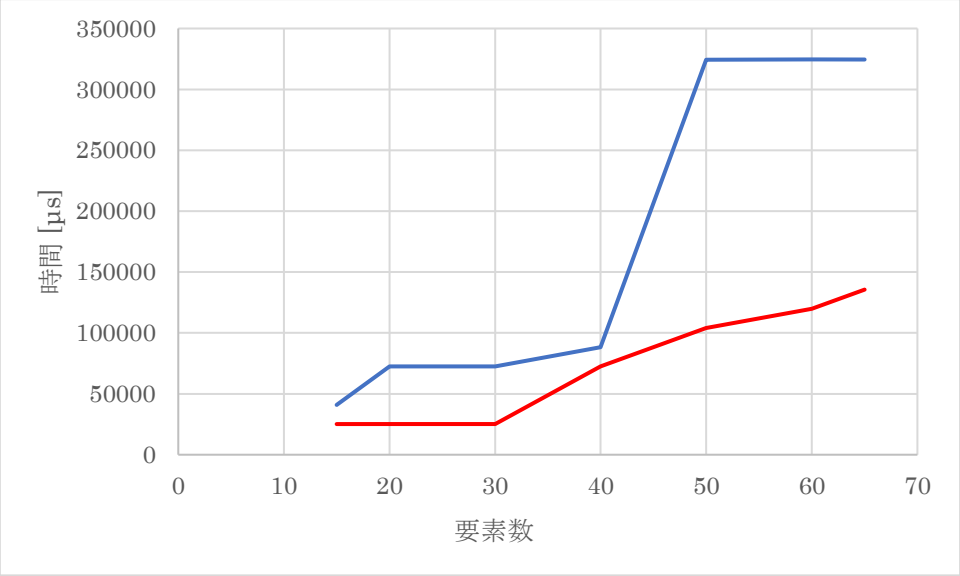

図 3.7: 部分和問題（後述）において、イジングモデルのパラメータを常に一定とした場合（青）と、適宜値を設定した場合（赤）の最適解を得るまでの計算時間の比較

また、Placeholder の値を問題と問題のスケールに合わせて適切に選ぶ必要がある。図 3.7 のグラフのようになるのは、要素数の増加とともに量子アニーリングにおける解候補が増大し、量子アニーリングの試行回数を大きくしないと最適解を得られなくなってしまう。これに対し、要素数の増加に従って、Placeholder の値を大きくすることでエネルギー関数の最小化のペナルティを強く課すようにして、少ない量子アニーリングの試行回数で最適解を得ることができる。

第 4 章 D-Wave 2000Q と古典コンピュータにおける計算時間の比較

検証で用いた一般的なコンピュータとは、Intel Core i5-8400 2.8GHz をプロセッサーとして搭載した ThinkPad X1 である。

4.1 数の分割問題

量子アニーリングマシンである D-wave と古典コンピュータとの計算比較を行うための組み合わせ最適化問題に数の分割問題を選ぶ。

4.1.1 概要

"N個の正の数の集合$S = \{n_1, n_{2,} \ . \ . \ . n_N\}$が与えられる。この集合を二つの非交な部分集合$R$と $S - R$ に分けて、R 内の数の和と $S - R$ 内の数の和を等しくすることができるかどうか判定する ” という問題である。

$$H = \{n_1, n_{2,} \ . \ . \ . n_N\}$$

4.1.2 問題の具体例

$S = \{5,2,1,9,11\}$ という正の整数からなる集合があった場合、この集合は$R = \{2,1,11\}$ と $S - R = \{5,9\}$ の和の大きさが等しい二つの集合へと分割することができる。

また、$S = \{13,2,5,8,11\}$ の場合、集合 S の要素の和は奇数であるから、和の大きさが等しい二つの集合へとは分割できない。

4.1.3 イジングモデルへの変換

数の分割問題を解くイジングモデルのエネルギー関数は以下のように表現できることが、A.Lucas らによって明らかにされている。[7]

集合 S の各要素にイジングスピン変数$s_i = \ \pm 1$ を、さらに任意の正の定数 A を用いて

$$H = A\left(\sum_{i=1}^{N} n_i s_i)\right)^2 \quad (4.1)$$

と表現できる。これは、エネルギー関数が最小化されて $H = 0$ が達成されたとする。このとき、$s_i = +1$ となっている n_i を集めて集合$R = \{n_i | s_i = +1\}$ をつくり、残りでできあがる集合 $S - R = \{n_i | s_i = -1\}$ をつくったとき、H は次のように表すことができるから、確かに数の分割問題が解けていることになる。

$$H = A\left(\sum_{s_i=+1} n_i - \sum_{s_i=-1} n_i\right)^2 = A\left(\sum_{n_i \in R} n_i - \sum_{n_i \in S-R} n_i\right)^2 = 0 \quad (4.2)$$

4.2 Python によるコーディング

数の分割問題は判定問題であるが、以下に示すプログラムでは分割できない集合にたいしては、近似解(分割される二つの集合の要素和の差はなるべく小さい)を最適解として求めている。判定の対象となる集合は、乱数によって決めた。

4.2.1 D-wave 2000Q システムへのコード

式での定数 A が制約項におけるペナルティウェイトにあたる。ここでは、$A = 1.0$とした。また、ライブラリ PyQUBO の利用によって、本来は辞書式に行列を構築しなければいけなかったところを、式で表したイジングモデルをほとんどそのままコード上に記述することができている。Python の random.randint()で、1~N の乱数を生成する。

```
import numpy as np
from pyqubo import Array, Constraint, Placeholder, solve_qubo
from dwave.system.samplers import DWaveSampler
from dwave.system.composites import EmbeddingComposite
import pandas as pd
import matplotlib.pyplot as plt
import itertools
import time
```

```
start = time.time()
goal = 0
np.random.seed(10)
NUM = #要素数
x = Array.create('x', shape = NUM, vartype='SPIN')

member = np.random.randint(1, NUM, NUM)
print("member")
print(member)
print("goal")
goal = sum(member) / 2
print(goal)
detail = False
#関数項
def func(N,p,x):
    sum = 0
    for i in range(N):
        sum += p[i]*x[i]

    return sum

# プレースホルダー
param_cover1 = Placeholder("cover1")
start = time.time()
#数の分割問題
H = param_cover1*(func(NUM_VER,member,x))**2

# モデルをコンパイル
model = H.compile()
# 制約項のペナルティウェイト
param_cover1 = #適宜決める
# プレースホルダーと合わせて QUBO を作成
feed_dict = {'cover1': param_cover1}
qubo, offset = model.to_qubo(feed_dict=feed_dict)

#dwave 計算
```

```
sampler = EmbeddingComposite(DWaveSampler())
print("Start anealing", time.time() - start)
response = sampler.sample_qubo(qubo,num_reads = #実行回数)
print("End anealing", time.time() - start)
if detail == True:
#結果の出力
    df_result = pd.DataFrame()
    k = 0
    for sample, energy, num_occurrences, chain_break_fraction in
list(response.data()):
        print(sample, "Energy: ", energy, "Occurrences: ", num_occurrences)
        df_tmp = pd.DataFrame(dict(sample), index=[k])
        df_tmp['Energy'] = -energy
        df_tmp['Occurrences'] = num_occurrences
        df_result = df_result.append(df_tmp)
        k+=1

    result =
df_result.pivot_table(index=df_result.columns[:n].tolist()+['Energy'],
values=['Occurrences'], aggfunc='sum').sort_values('Energy', ascending=False)

    print(result)

else:
    print(list(response.data())[0])

print("Total_real_time ", response.info["timing"]["total_real_time"], "us")
print(time.time() - start)
print(" ")
print(next(response.samples()))
p = dict(next(response.samples()))
ans = 0
for i in range(NUM):
      ans += p["x[{}]".format(i)] * member[i]

print('answer')
```

```
print(ans)
```

4.2.2 bit 全探索アルゴリズム

以降に示していくのは、一般的なコンピュータにおける数の分割問題の解探索アルゴリズムである。bit 全探索は状態が 2^N 通りある問題に対して 2 進数をフラグに見立てることで全探索を行う方法。各要素に対してどちらかの集合に属すか、属さないかの 2 通りあるので、時間計算量としては $O(2^N)$ になるはずである。

```
import numpy as np
import time
np.random.seed(10)
N = #要素数
array = np.random.randint(1, N, N)
team_A = np.zeros((2**N,N))
total_A = np.zeros(2**N)
print(array)
start = time.time()
# 目標値
ans = sum(array)/2
print("理想値:",ans)
#bit 全探索
best = 0
for bits in range(2**N):
    for i in range(N):
        if ((bits >> i) & 1):
            # 順に右にシフトさせ最下位 bit のチェックを行う
            # フラグが立っていたなら以下の処理
            team_A[bits][i] = array[i]
            total_A[bits] += array[i]

    if((best <= total_A[bits] <= ans)):
        best = total_A[bits]
```

```
        A = bits

for j in range(N):
    if(team_A[A][j]== 0):
         comtinue
    else:
         print(team_A[A][j])

print("answer")
print(best)
elapsed_time = time.time() - start
print ("elapsed_time:{0}".format(elapsed_time) + "[sec]")
```

4.3 出力結果

D-Wave に渡した上記のコードに注釈しているように、初めに集合 S の内容をナンバリングしながら列挙した後、集合 R として選んだものに 1 を格納、選ばなかったものに 0 を格納し、それを表示する。最後に、計算の時間とともに集合 R として選んだものの総和を求めることで最適解が導かれているかどうかが分かる。

bit 全探索についても、同様に集合 S の要素と選んだ集合 R 、さらに最適解を出力するようにコードを実装している。これらは以下に示す例を一つに省略する。

4.3.1 D-Wave2000Q からの出力

（ｉ）最適化を得ている結果

要素数 $N = 15$、量子アニーリング試行回数を十分にとった(num_read = 3000)場合

```
member
[10 14  5  1  2 12 13 10 14  1 14  2 11  9 10]
goal
64.0
Start anealing 2.5755269527435303
End anealing 7.717666387557983
Sample(sample={'x[0]': 1, 'x[10]': 0, 'x[11]': 0, 'x[12]': 1, 'x[13]': 1,
'x[14]': 0, 'x[1]': 0, 'x[2]': 1, 'x[3]': 1, 'x[4]': 1, 'x[5]': 1, 'x[6]': 1,
'x[7]': 0, 'x[8]': 0, 'x[9]': 1}, energy=-16384.0, num_occurrences=1,
chain_break_fraction=1.0)
Total_real_time  954278 us
7.94819188117981
answer
64
```

要素数の和を 2 で割った値から、サンプルによっては数の分割問題の判定としては”不可能”と返す場合があるが、D-Wave2000Q の量子計算が最もよい（もとの集合の要素和をほとんど二分している）近似解を出力したとき、量子アニーリングはうまく行えているとする。これは、イジングモデルで表したエネルギー関数の低い状態を出力していることが示唆できるからである。

(ⅱ) 最適化を行えていない

要素数 $N = 15$、量子アニーリング試行回数が不十分である(num_read = 10)場合。アニーリングの試行回数が不十分であるため、最適解近傍を出力している。

```
member
[10 14  5  1  2 12 13 10 14  1 14  2 11  9 10]
goal
64.0
Start anealing 2.191833257675171
End anealing 12.245985269546509
Sample(sample={'x[0]': 1, 'x[10]': 1, 'x[11]': 0, 'x[12]': 0, 'x[13]': 1,
'x[14]': 0, 'x[1]': 0, 'x[2]': 1, 'x[3]': 0, 'x[4]': 0, 'x[5]': 1, 'x[6]': 0,
'x[7]': 1, 'x[8]': 0, 'x[9]': 1}, energy=-16348.0, num_occurrences=1,
chain_break_fraction=1.0)
Total_real_time  12578 us
12.259592294692993

answer
61
```

4.3.2 bit 全探索による出力

縦列に並んだ数字は集合 R の要素である。

```
[10 14  5  1  2 12 13 10 14  1 14  2 11  9 10]
理想値: 64.0
1.0
2.0
14.0
1.0
14.0
2.0
11.0
9.0
10.0
answer
64.0
elapsed_time:0.7052679061889648[sec]
```

4.4 評価の方法

横軸を集合 S の要素数、縦軸を、最適解を得るまでにかかる時間(解を出力するまでの時間)として、上述した２つの解探索法をプロットして比較を行った。

4.5 評価

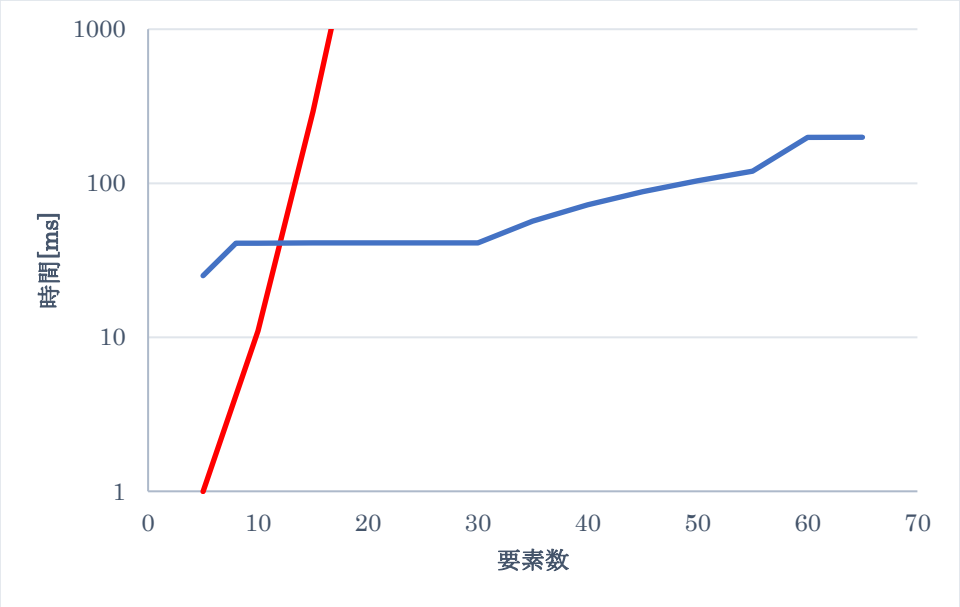

図 4.1: 3 つの解探索法の計算時間の推移（赤：bit 全探索 青：D-Wave）

bit 全探索による解探索法は、計算量が $O(2^N)$ となるだけあって、計算時間は要素数にたいして指数関数的に増加しており、すぐに莫大な計算量となってしまっている。D-Wave2000Q による量子アニーリングについて、要素数の増加とともに計算時間が増えているのは、問題のスケールが大きくなればなるほど、とるべき状態の数が増えて、量子アニーリングの試行回数を増やさないと最適解へとたどり着くことができないからである。また、$N = 65$ 以上の N では量子アニーリングは実行されず、以下に示すエラーとなった。

```
if bqm and not embedding:
raise ValueError("no embedding found")
bqm_embedded = embed_bqm(bqm, embedding, target_adjacency,


ValueError: no embedding found
```

上記のエラーメッセージによれば、パラメータ bqm が構築できていない、つまり要素数が 65 を超えると QUBO が構築できなくなることが知らされている。これにより、D-Wave2000Q が行う量子アニーリングにおける、問題のサイズの限界を知ることができた。

4.6 部分和問題

ここで、数の分割問題とは少し変えた他の組み合わせ最適化問題について量子アニーリングによる解探索を行っていきたい。部分和問題について着目する。

4.6.1 概要

“ 与えられた正の整数の集合 $T = \{a_1, a_2 \ldots a_N\}$から部分集合をうまく選んで、選んだ集合の要素の和が与えられた数 A に等しくなるようにできるかどうかを判定する ”という問題である。

4.6.2 イジングモデルへの変換

部分和問題を解くイジングモデルのエネルギー関数は、以下のように定義した。ここで、前節の数の分割問題とは異なり、集合 T の各要素にイジングバイナリ変数$q_i = \{0,1\}$ を用いた。定数 A はターゲット定数で、任意の正の定数 B を用いて

$$H = B\left(A - \sum_{i=1}^{N} a_i q_i\right)^2 \quad (4.3)$$

と表現できる。エネルギー関数が最小化されて $H = 0$ が達成されたとする。このとき、$q_i = 1$ となっている整数 a_i を集めてできる集合 $R = \{a_i | q_i = +1\}$ の要素の和はターゲット定数 A と等しい。

4.7 Python によるコーディング

4.7.1 D-Wave 2000Q システムへのコード

```
import numpy as np
from pyqubo import Array, Constraint, Placeholder, solve_qubo
from dwave.system.samplers import DWaveSampler
from dwave.system.composites import EmbeddingComposite
import pandas as pd
import matplotlib.pyplot as plt
import itertools
import time
start = time.time()
np.random.seed(10)
NUM = #要素数
target = #ターゲット定数
x = Array.create('x', shape = NUM, vartype='BINARY')

member = np.random.randint(1, NUM, NUM)
print("member")
print(member)
print("target")
print(target)
detail = False
#関数項
def func(N,p,x):
    sum = 0
```

```
    for i in range(N):
        sum += p[i]*x[i]

    return sum

# プレースホルダー
param_cover1 = Placeholder("cover1")
start = time.time()
#部分和問題
H = param_cover1*(target - func(NUM_VER,member,x))**2

# モデルをコンパイル
model = H.compile()
# 制約項のペナルティウェイト
param_cover1 = #適宜決める

# プレースホルダーと合わせて QUBO を作成
feed_dict = {'cover1': param_cover1}
qubo, offset = model.to_qubo(feed_dict=feed_dict)
#dwave 計算
sampler = EmbeddingComposite(DWaveSampler())
print("Start anealing", time.time() - start)
response = sampler.sample_qubo(qubo,num_reads = #実行回数)
#解の出力
print("End anealing", time.time() - start)
if detail == True:

    df_result = pd.DataFrame()
    k = 0
      for sample, energy, num_occurrences, chain_break_fraction in
list(response.data()):

        df_tmp = pd.DataFrame(dict(sample), index=[k])
        df_tmp['Energy'] = -energy
        df_tmp['Occurrences'] = num_occurrences
        df_result = df_result.append(df_tmp)
```

```
        k+=1

   result=
df_result.pivot_table(index=df_result.columns[:n].tolist()+['Energy'],
values=['Occurrences'], aggfunc='sum').sort_values('Energy', ascending=False)

    print(result)

else:
    print(list(response.data())[0])

print("Total_real_time ",response.info["timing"]["total_real_time"], "us")
print(time.time() - start)
print(" ")
print(next(response.samples()))
p = dict(next(response.samples()))
ans = 0
for i in range(NUM):
      ans += p["x[{}]".format(i)] * member[i]

print('answer')
```

4.7.2　DP(動的計画法)を用いたアルゴリズム

DP（動的計画法）とは、対象となる問題を帰納的に解く場合にくり返し出現する小さな問題例について、解(bool 型)を表に記録し表を埋めていく、メモ化という形で計算をすすめていくことで冗長な計算を省くアルゴリズムのことである。部分和問題においては、漸化式を以下に表すことができる。[7]

$dp[i+1][j]$ ： “i 番目までの整数の中からいくつか選んで総和を j とすることが可能かどうか ”として、$dp[i][j]\ (j = 0,1,\dots,A)(j = 0,1,\dots,A)$ を使って $dp[i+1][j](j = 0,1,\dots,A)$ の値を更新することを考えます。$dp[i+1][j]$ の値を求めるには、以下のようになります。

・整数 $a[i]$ を選ぶ場合　（$j \geq a[i]$ の場合のみ）

$$dp[i][j-a[i]] \text{ が True なら、} dp[i+1][j] \text{ も True}$$

・整数 $a[i]$ を選ばない場合

$$dp[i][j] \text{ が True なら、} dp[i+1][j] \text{ も True}$$

$$dp[i+1][j] = \begin{cases} dp[i][j-a[i]] \mid dp[i][j] & (j \geqq a[i]) \\ dp[i][j] & (j < a[i]) \end{cases} \quad (4.4)$$

ただし、初期条件を $dp[0][j] = \begin{cases} True\ (j = 0) \\ false\ (j \neq 0) \end{cases}$ (4.5) とする。

メモ化を用いることで、要素数 N とターゲット定数 A それぞれについて一回の組み合わせについて調べるだけで済むため、計算量としては $O(NA)$ になる。

```
import numpy as np
import pandas as pd
import itertools
import time
num = #要素数
np.random.seed(10)
member = np.random.randint(1, num, num)
print(member)
A = ターゲット定数
print(A)
```

```
start = time.time()
def part_sum(a,A):
  #初期化
  N = len(a)
  dp = [[0 for i in range(A+1)] for j in range(N+1)]
  dp[0][0] = 1
  #DP
  for i in range(N):
    for j in range(A+1):

      if (a[i] <= j): #集合に i+1 番目の要素 a[i]を追加できる可能性がある
        dp[i+1][j] = dp[i][j-a[i]] or dp[i][j]

      else: #集合に要素 a[i]を追加できる可能性はない
        dp[i+1][j] = dp[i][j]

  return dp[N][A]
if(part_sum(member,A)):
    print("Yes")
else:
    print("No")

totaling_time = time.time() - start
print ("elapsed_time:{0}".format(totaling_time) + "[sec]")
```

4.8 出力結果

D-Wave に渡した上記のコードは、前節で述べたことと同様に、初めに集合 S の内容をナンバリングしながら列挙した後、集合 R として選んだものに 1 を格納、選ばなかったものに 0 を格納し、それを表示する。最後に、計算の時間とともに集合 R として選んだものの総和を求めることで最適解（ターゲット定数と等しいかどうか）が導かれているかどうかが分かる。bit 全探索についても、同様に集合 S の要素と選んだ集合 R 、さらに最適解を出力するようにコードを実装している。これらは以下に示す例を一つに省略する。

4.8.1 D-Wave2000Q からの出力

サンプルによって、集合 S の全体の総和がターゲット定数に満たない場合がある。この場合での D-Wave の量子計算は行っていない。D-Wave の量子計算において、本論で注目しているのは最適解にたどり着くまでにかかる計算時間だからである。

（ｉ）最適化を得ている結果

要素数 $N = 15$、ターゲット定数$A = 100$、量子アニーリング試行回数を十分にとった（num_read = 500）場合

```
member
[10 14  5  1  2 12 13 10 14  1 14  2 11  9 10]
target
100
Start anealing 1.1862208843231201
End anealing 5.1847758293151855
Sample(sample={'x[0]': 0, 'x[10]': 1, 'x[11]': 1, 'x[12]': 1, 'x[13]': 1,
'x[14]': 1, 'x[1]': 1, 'x[2]': 0, 'x[3]': 0, 'x[4]': 0, 'x[5]': 1, 'x[6]': 1,
'x[7]': 0, 'x[8]': 1, 'x[9]': 1}, energy=-10000.0, num_occurrences=1,
chain_break_fraction=0.9333333333333333)
Total_real_time  166897 us
5.2374022006988525

answer
100
```

（ⅱ）最適化を行えていない

要素数 $N = 15$、ターゲット定数$A = 100$、量子アニーリング試行回数が不十分な（num_read = 10）場合

```
member
[10 14  5  1  2 12 13 10 14  1 14  2 11  9 10]
target
100
Start anealing 1.02010178565979
End anealing 5.626614093780518
Sample(sample={'x[0]': 1, 'x[10]': 1, 'x[11]': 0, 'x[12]': 1, 'x[13]': 1,
'x[14]': 1, 'x[1]': 1, 'x[2]': 1, 'x[3]': 0, 'x[4]': 1, 'x[5]': 1, 'x[6]': 1,
'x[7]': 1, 'x[8]': 0, 'x[9]': 0}, energy=-9900.0, num_occurrences=1,
chain_break_fraction=0.8666666666666667)
Total_real_time  12574 us
5.628643751144409



answer
110
```

4.8.2 DP を用いたアルゴリズムによる出力

部分和問題は判定問題であるから、集合T の中からターゲット定数と等しい和を作ることができるような組み合わせがあれば ”Yes” を、なければ ”No” を出力する。

$N = 15$、ターゲット定数 $A = 100$

```
[10 14  5  1  2 12 13 10 14  1 14  2 11  9 10]
100
Yes
elapsed_time:0.0047571659088134766[sec]
```

$N = 15$、ターゲット定数 $A = 300$

```
[10 14  5  1  2 12 13 10 14  1 14  2 11  9 10]
300
No
elapsed_time:0.010830163955688477[sec]
```

4.9 評価の方法

横軸を集合 S の要素数、縦軸を、最適解を得るまでにかかる時間(解を出力するまでの時間)として、上述した 3 つの解探索法をプロットして比較を行った。

4.10 評価

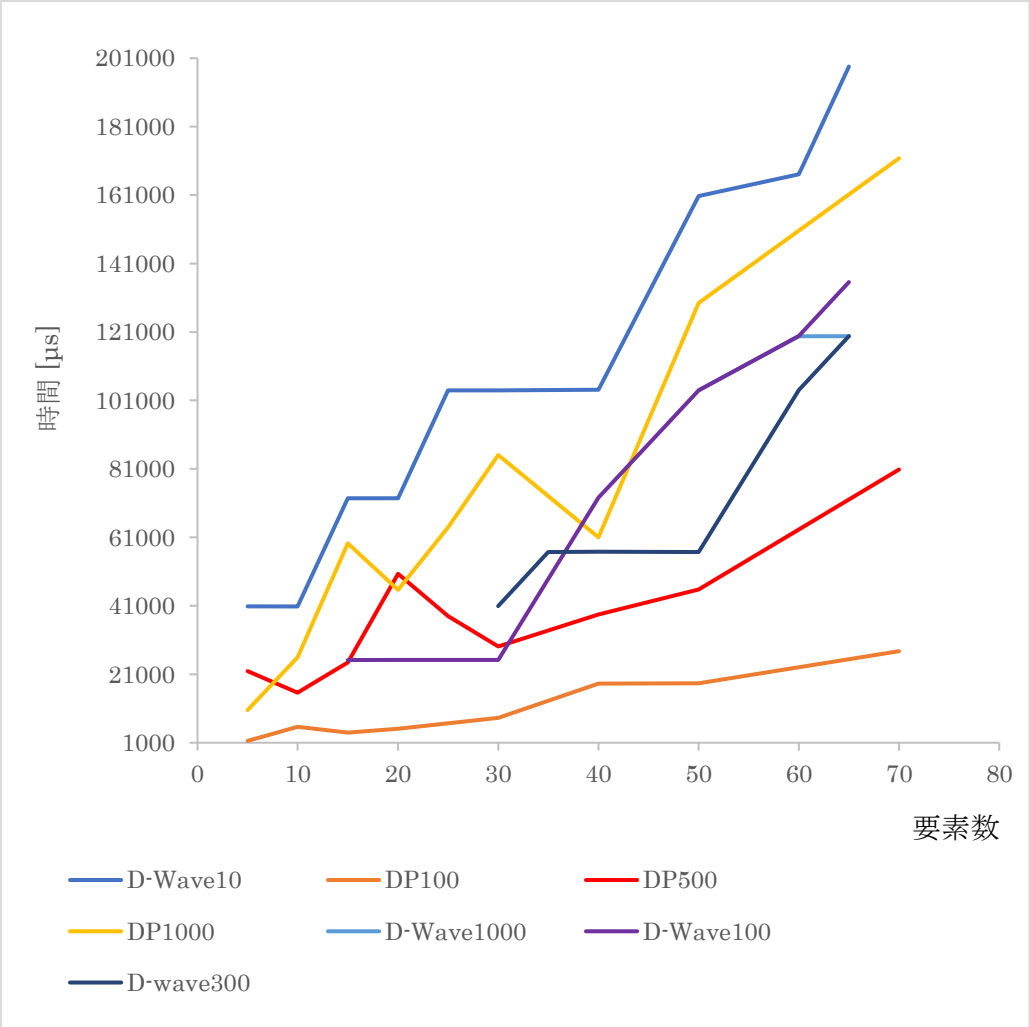

図 4.2： 部分和問題の計算時間の比較

計算量が$O(NA)$の DP の方が全体的に D-Wave より計算が速い。グラフから、D-Wave の計算が$N = 1000$ になって、DP の計算時間を下回っていることが分かる。量子ビットの実装の限界が障壁となっているが、これを考慮しなければ要素数 $A = 1000$ 程度から、これ以上の A では DP の計算量を D-Wave の量子計算が下回り続けることが予想できる。また、$N = 10$ において、D-Wave の計算時間が大きくなってしまうのは、ターゲット定数が小さい（乱数生成される各要素の値に近い）と近似解の候補が多く、精度の高い量子計算を必要とするため、量子アニーリングの試行回数を大きくとる必要があるからである。

5 章 結論

5.1 まとめ

D-Wave の量子アニーリングで行う組み合わせ最適化問題は、DP のような効率的なアルゴリズムと比べたときは、計算時間は大きいか同等くらいであったが、計算時間としては現実的であることが分かった。短所として、一般的なコンピュータがもつリソースと比べて D-Wave2000Q は量子ビットの実装に限界があり解くことのできる組み合わせ最適化問題のサイズに制限があることが挙げられる。また、一般的なコンピュータが、ある組み合わせ最適化問題に適したアルゴリズムを考える必要があることに対して、D-Wave で行う量子アニーリングはあくまでも全探索であるから、イジングモデルによる定式化さえできてしまえば、実用的な処理時間で計算を行えるということが言える。序論でも述べたように、イジングモデルへの定式化さえ達成できれば良いので、分野を選ばない活用ができる。

また、本論で取り上げたものより計算複雑性の増した問題や、計算量が$O(2^N)$となる全探索による解探索法しか知られていない問題に対して、適切な制約項やパラメータを設定し、うまくエネルギー関数の最小化へと組み合わせ最適化問題を帰着させることが、計算ツールとしてアニーリングマシン D-Wave2000Q を利用する際に求められる。

5.2 課題と展望

今回は“数の分割問題”と“部分和問題”の二つの組み合わせ最適化問題を解いた。この他に、取り上げた 2 つの問題よりも計算複雑性の高い“ハミルトン閉路問題”や“ナップザック問題”に取り組んでいたが、未だに D-Wave2000Q から最適解を得ることができなか

った。これらの問題について、D-Wave2000Q が適切な計算が行えるようにうまくイジングモデルへの変換を計画したい。また、結晶の構造決定のような物性分野における問題を、組み合わせ最適化問題へと帰着させ、量子アニーリングを用いて解決させることを試みてみたいと思った。

謝辞

本論文を作成するにあたり、ご指導を頂いた指導教員の小林伸彦教授に心より感謝致します。また、生まれてから大学卒業まで自分を支え、育ててくれた両親に深く感謝したいと思います

参考

[1]西森 秀稔 . 量子アニーリングの数理 物性研究 3.033203（2014）
[2]株式会社 fixters 技術リソース . https://quantum.fixstars.com/
[3]D-Wave systems 社 . https://www.dwavesys.com/
[4]川畑史郎 . 量子アニーリングのためのハードウェア技術―超伝導エレクトロニクスと超伝導量子回路―（2018）
[5]PyQUBO 株式会社リクルートコミュニケーションズ . Github コード https://github.com/recruit-communications/pyqubo
[6]東北大学量子アニーリング研究開発室 . https://qard.is.tohoku.ac.jp/T-Wave/
[7]A.Lucas,Front.Phys.2.5 （2014）
[8]春日伸弥、紀平拓男 . プログラミングの宝箱アルゴリズムとデータ構造 . 宝島社（2011）