NTT DATA
Trusted Global Innovator

NTTデータ 量子コンピューティングガイドライン
量子アニーリング／イジングマシン編

2021年1月

NTT DATA
Trusted Global Innovator

免責事項

「NTTデータ 量子コンピューティングガイドライン 量子アニーリング/イジングマシン編」は、2021年1月時点における株式会社NTTデータ（以下：NTTデータ）の見解を表したものであり、特定のハードウェア、ベンダー、製品またはサービスを推奨するものではありません。今後の技術やサービスの進展、市場動向のために、本ガイドラインの記載内容と変化が生じる可能性がございます。記載している情報には細心の注意を払っていますが、一般の読者に対する分かりやすさをガイドラインの主たる目的としており、学術的な厳密性、記載するハードウェア・ソフトウェア・ソースコード例の動作・安全性などの一切を保証せず、法律上の瑕疵担保責任を持つものではありません。また、弊社の判断によって、本ガイドラインに記載している情報の変更や、掲載の継続と終了など管理の変更を行う可能性がございます。

商標について

発行元

株式会社NTTデータ 技術革新統括本部 技術開発本部
量子コンピュータ／次世代アーキテクチャ・ラボ

執筆者

香月 諒大

本ガイドラインに関するお問い合わせ先

矢実 貴志 / 田端 佑介 / 尾崎 史朗 / 香月 諒大 / 川又 裕也
E-mail: qcomputer@kits.nttdata.co.jp

目次

第 1 章　はじめに 1

1.1 量子コンピュータ／次世代アーキテクチャ・ラボ 1

1.2 本ガイドラインの使い方 1

第 2 章　量子ゲート方式／量子アニーリング方式／イジングマシン 3

2.1 量子コンピュータとは 3

2.2 量子ゲート方式 5

2.3 量子アニーリング方式 7

2.4 イジングマシン（QUBO ソルバー、アニーリングマシン） 9

第 3 章　組合せ最適化とは 10

3.1 組合せ最適化のイメージ 11

3.1.1 組合せ最適化 11

3.1.2 巡回セールスマン問題の例 11

3.1.3 巡回セールスマン問題の巡り方の候補の数 12

3.1.4 巡回セールスマン問題の組合せ爆発 13

3.2 決定変数・目的関数・制約条件と定式化 15

3.2.1 ナップサック問題 15

3.2.2 組合せ最適化のプロセス 17

3.2.3 ナップサック問題の定式化 18

3.2.3.1 決定変数 18

3.2.3.2 目的関数 18

3.2.3.3 制約条件 19

3.3 組合せ最適化が用いられるシーン 20

3.4 既存の組合せ最適化のアプローチ 22

3.4.1 最適化用のソフトウェアとアルゴリズム 22

3.4.1.1 最適化ソルバー 22

3.4.1.2 業務に特化したパッケージ . . . 23
3.4.1.3 アルゴリズムの開発 . . . 24
3.4.2 厳密解の保証とヒューリスティック . . . 24

第 4 章 量子アニーリング／イジングマシンの概要 26
4.1 ハードウェアの役割と特徴 . . . 26
4.1.1 量子アニーリング／イジングマシンのアプローチ . . . 27
4.1.2 マシンへの入出力データ概要 . . . 27
4.2 イジングモデル . . . 29
4.3 ナチュラルコンピューティング . . . 32
4.3.1 ナチュラルコンピューティングとは . . . 32
4.3.1.1 粘菌コンピュータ . . . 32
4.3.1.2 シャボン膜による計算 . . . 33
4.4 量子アニーリングのイメージ . . . 35

第 5 章 D-Wave Leap でのプログラミング例 37
5.1 動作環境について . . . 37
5.1.1 利用登録と接続情報の取得 . . . 37
5.1.2 環境構築 . . . 38
5.2 入出力処理の確認 . . . 39
5.2.1 ライブラリのインポート . . . 40
5.2.2 ユーザ情報の設定 . . . 40
5.2.3 sampler インスタンス . . . 40
5.2.4 Binary Quadratic Model . . . 41
5.2.5 sampler.sample メソッドと response の取得 . . . 41
5.3 分割問題 . . . 43
5.3.1 分割問題の定式化 . . . 43
5.3.2 数式処理の手計算と D-Wave システムによる求解 . . . 44
5.3.3 PyQUBO と D-Wave システムによる求解 . . . 46
5.4 巡回セールスマン問題 . . . 48
5.4.1 巡回セールスマン問題の定式化 . . . 48
5.4.1.1 目的関数 . . . 49
5.4.1.2 制約条件 . . . 49
5.4.2 PyQUBO と D-Wave システムによる求解 . . . 50

第 6 章 組合せ最適化の関連知識 56
6.1 QUBO とイジングモデル . . . 56
6.2 ペナルティ関数 . . . 59
6.2.1 制約条件付き組合せ最適化問題の探索方法 . . . 59
6.2.2 ペナルティ関数の考え方 . . . 60
6.2.3 ペナルティ関数の構成方法 . . . 61
6.2.4 巡回セールスマン問題におけるペナルティ関数 . . . 62

6.3 大域的最適解と局所的最適解 64
6.3.1 多峰性 64
6.4 シミュレーテッド・アニーリング 67
6.4.1 シミュレーテッド・アニーリング 67
6.5 組合せ最適化問題の前処理 70
6.5.1 最小化と最大化 70
6.5.2 定数倍 70
6.5.3 定数項 71
6.5.4 上三角化について 71

第 7 章 量子アニーリング／イジングマシン特有の処理 74
7.1 疎結合と全結合 74
7.1.1 疎結合／全結合なハードウェア 75
7.1.2 疎結合ハードウェアの全結合化 76
7.1.3 疎結合／全結合な（求解対象の）イジングモデル 77
7.1.4 チェーンの強度 78
7.2 階調 80
7.3 アニーリングタイム 80
7.4 解のサンプリング 81

1 はじめに

1.1 量子コンピュータ／次世代アーキテクチャ・ラボ

NTT データでは、お客様のビジネス課題の解決や、システム開発における新規技術の開拓を目的に、数理最適化などのデータサイエンス領域の技術から先進的な計算基盤技術の活用まで、幅広く調査や開発を実施しています。それらの取組の一環として、業務要件に基づいた検証・評価を支援する「量子コンピュータ／次世代アーキテクチャ・ラボ」のサービスを 2019 年 1 月 25 日より行っています[*1][*2] 。

本ガイドラインでは、量子コンピューティングの一領域として組合せ最適化への応用に注目を浴びている量子アニーリング、及び FPGA や GPU などのハードウェアを用いて組合せ最適化の処理基盤を提供するイジングマシンに関する弊社の調査を公開しています。

1.2 本ガイドラインの使い方

本ガイドラインでは、**量子アニーリング／イジングマシンや組合せ最適化に初めて携わるプログラマー／システムエンジニアや、技術的な背景を理解したい読者** に向けて、必要な知識の解説や、サンプルプログラムの提供を行っています。**実際にプログラミングを行う方以外にとっても、量子アニーリング／イジングマシンにまつわるニュースの前提を理解するために役立つような情報** を記載しております。

本ガイドラインは、順に読みすすめることによって、徐々に技術的に細かな内容を理解できるよう構成されています。最初に、量子コンピュータにおける、技術的な背景や用語・用途の把握を行いたい読者のための章を設けています。章が進むに従ってより具体的に、量子アニーリング／イジングマシンの主な利用用途である組合せ最適化等の技術の概要を説明します。なお、量子力学などの物理学、組合せ最適化に対する前提知識は、前提として習得していなくても良いように心がけ構成しています。

[*1] NTT データ - 量子コンピュータ／次世代アーキテクチャ・ラボのサービス開始 https://www.nttdata.com/jp/ja/news/release/2019/012501/

[*2] E-mail：qcomputer@kits.nttdata.co.jp

章	内容
2 章 **量子ゲート方式／量子アニーリング方式／イジングマシン**	量子コンピュータの概要を説明した後に、量子ゲート方式・量子アニーリング方式・イジングマシンの違いを解説しています。
3 章 **組合せ最適化とは**	量子アニーリング／イジングマシンの用途である「組合せ最適化」の概要と、ビジネスで現れるシーン・既存のアプローチを記載しています。
4 章 **量子アニーリング／イジングマシンの概要**	3 章 の内容を踏まえて、量子アニーリング／イジングマシンの計算の原理や対象とする問題を説明しています。
5 章 *D-Wave Leap* **でのプログラミング例**	具体的に量子アニーリングのイメージを持つために、D-Wave システムを利用したプログラミング例と解説を記載しています。
6 章 **組合せ最適化の関連知識**	より深い技術的な内容や本格的に利用を始める前に、組合せ最適化に関する知っておきたい技術的な特徴や専門用語を説明しています。
7 章 **量子アニーリング／イジングマシンの概要**	各社のマシン毎に多数のパラメータが存在しますが、それらを理解する前提となる情報や、特に重要なパラメータを説明しています。

組合せ最適化の理論やアルゴリズム、量子力学に関する物理学的な解説などの詳細情報は割愛しています。開発に必要な最低限の情報記載を行っているため、より専門的な学術的内容を習得するためには、適宜記載している文献・教科書等を参考にしてください。なお、後半の章では、機械学習（AI）やデータ分析などの領域において求められる線形代数・統計の知識や、Python によるプログラミングの経験を前提とする記述が含まれます。

本ガイドラインの対象読者

- 量子アニーリング／イジングマシンや組合せ最適化に初めて携わるプログラマー／システムエンジニア
- 量子コンピュータに関連するニュースの技術的背景を知りたい方向け

2 量子ゲート方式／量子アニーリング方式／イジングマシン

この章の流れ

初めて量子コンピュータの話題に触れる方に向けて、最初に知っておきたい背景や用語の説明を行います。**量子ゲート方式／量子アニーリング方式／イジングマシン** といった、複数の方式のハードウェアが存在すること、それらの用途や特徴の理解が重要です。技術的に細かな仕組みは一旦省き、**ニュースや記事が理解できるようになるために知っておきたい前提となる情報** を集約しています。

- 2.1 章 量子コンピュータとは
- 2.2 章 量子ゲート方式
- 2.3 章 量子アニーリング方式

2.1 量子コンピュータとは

近年、「**量子（Quantum）**」をキーワードとする技術が注目を浴びています。従来から存在するコンピュータでは、情報を表現するために、電圧の高低であるバイナリのビット（01）を用いてきました。一方「**量子コンピュータ（Quantum Computer）**」では、量子力学に基づく「とある現象」を用いて情報の表現を行っています。

情報の表現方法は、コンピュータの原理の中でも非常に低レイヤに位置しています。回路や計算の考え方を根本的に変えることによって、**従来の回路が苦手であった様々な用途が開拓されるようになり、計算原理の根幹を大きく変える**ことができるコンピュータとして期待されています。

量子コンピュータの「とある現象」は、ビットの「**重ね合わせ状態（Superposition）**」「**もつれ合い（Entanglement）**」と呼ばれるものであり、「**量子ビット（Qubit）**」という情報の単位が与えています。計算原理を学ぶ際には理解する必要がある概念ですが、難解であることに加えて、利用者の観点では必ずしも理解する必要がないため、一旦この章では用語の紹介だけに留めます。

なお、量子コンピュータと対比する意味で、バイナリのビット（01）で情報を表現する従来のコンピュータを

「**古典コンピュータ（Classical Computer）**」と呼ぶことがあります[*1]。

量子コンピュータは、現在研究開発が盛んになされているハードウェアであり、その過程で様々な方式が生まれています。

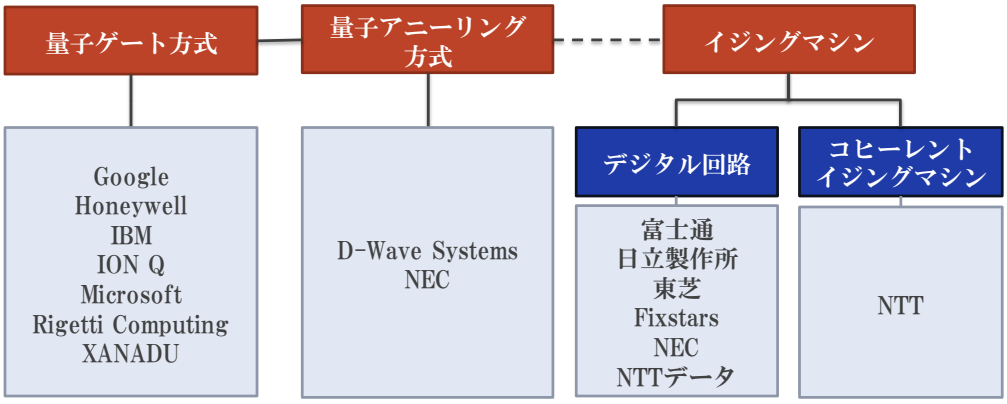


図 2.1　用途・実現方法の異なる様々な方式のハードウェアが存在する。

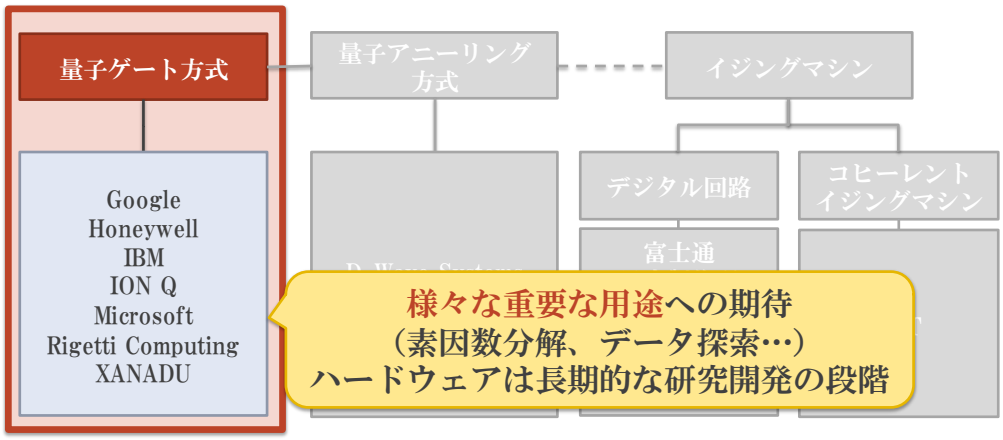
これらの中には、用途の類似性の観点から、量子コンピュータと似た文脈で記載されることが多いものの、実際の計算原理としては古典コンピュータに属するようなハードウェアなども多数開発されており、初めて聞く方にとって誤解を招きやすく、理解が難しい状況になっています。

ハードウェア	特徴	注意点
量子ゲート方式	量子回路にゲートを配置 素因数分解・データ探索 化学・金融など重要な**多数の用途**	ハードウェアの研究開発に **長期的な基礎研究**を要する
量子アニーリング方式	主に**組合せ最適化**に特化 量子ゲート方式と原理が異なり、 比較的規模が大きく安定	量子ゲート方式に比べ、 **想定用途は一部に特化**
イジングマシン （アニーリングマシン）	**FPGAやGPU等**を利用して、 量子アニーリング方式同様の **組合せ最適化**を高速に計算	理論的な高速性は示されておらず、 **将来的な性能向上に限界** が生じる可能性がある。

そこでこの章では、初めて量子コンピュータの話題に触れる方に向けて、現在発表されている方式の種類や特徴を説明します。特に、量子ゲート方式と量子アニーリング方式／イジングマシンの違いの理解は重要であり、用途や実現の方法にも大きな違いがあります。

[*1] 量子コンピュータは「**量子力学（Quantum Physics）**」、通常のコンピュータは「**古典力学（Classical Physics）**」という物理学に基づく考え方を利用しているために命名されています。

2.2 量子ゲート方式



「**量子ゲート方式（Gate-based Quantum Computer）**」は、最も古くから研究されている量子コンピュータの方式であり、1980 年代から学術領域で検討されてきました。なお、量子ゲート方式／量子アニーリング方式／イジングマシンなど採用している方式の言及がなく、「量子コンピュータ」とだけ書かれているときには、量子ゲート方式を直接指していることが多いです。

図 2.2　IBM による量子ゲート方式のマシン「IBM Q」[*3]

量子ゲート方式を動作させる際には、従来のコンピュータと同様に、計算したい用途に合わせて、プログラミングを行って回路を構成し、所望の入出力に合わせます。つまり、「**回路素子・ゲート（Gate）**」をプログラミングによって都度うまく配置して、所望の機能が得られるように設計します。

現状、量子ゲート方式のプログラミング言語は発展の途中の段階であるため、論理回路を直接編集するような形態を取っています。従来の古典コンピュータ上における論理回路の構成方法と考え方が類似している点も多

[*3] IBM - Image Gallery より画像引用 https://newsroom.ibm.com/image-gallery-research#gallery_closed

くありますが、制御の対象の情報が通常の 01 ビットではないために、量子力学固有の特性を考慮した「**量子回路（Quantum Circuit）**」に基づいている点に違いがあります。

量子ゲート方式の特徴的な点は、古典コンピュータよりも理論的な観点で「速い」アルゴリズムが存在する点です。「量子コンピュータにより、既存の暗号が破られるかもしれない」という話を聞いたことがあるかもしれません。もしも、量子ビットを大量に具えており、誤り訂正などの重要な機能が実装されている量子コンピュータが開発できた場合に、暗号を解読しうると考えられる「**ショアのアルゴリズム（Shor's Algorithm）**」が知られています。他にも、データ探索や化学計算など、重要なドメインでのアルゴリズムが複数存在します。

量子ゲート方式は、量子回路を構成する柔軟さがあるために、上記のように **重要で効率的なアルゴリズムが、机上で検討され発表されている** という特徴があります。一方で、**量子ゲート方式のハードウェアの製造は技術的に非常に難しく**、理論上の重要な計算をすぐに実行できる状態ではありません。特に、誤り訂正機能の実装や量子ビット数の大規模化、ハードウェアノイズの低減といった大きな技術的課題が多々あり、**長期的な研究が必要** であると考えられています。

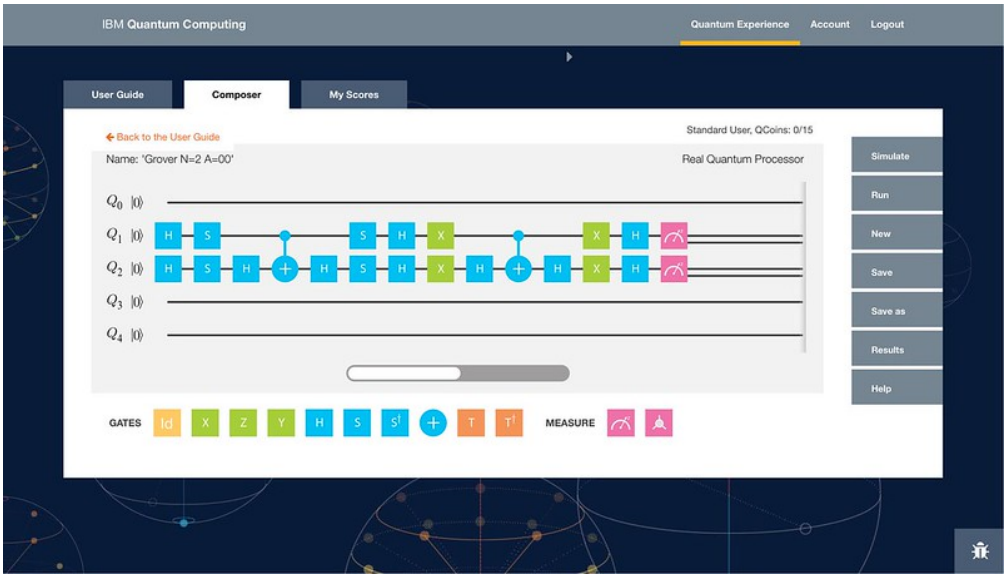
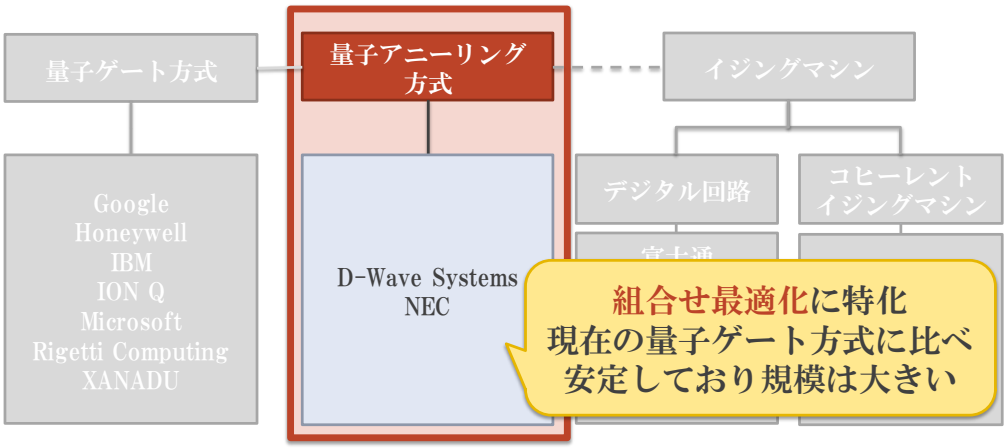


図 2.3　量子ゲート方式 IBM Q 用の量子回路の動作イメージ。ユーザがローカル PC 上から量子回路上にゲートを配置すると、クラウドを経由して実際に IBM Q で計算が行われる。[*2]

しかし、将来的に高性能化が実現できた際のインパクトが非常に大きいため、ハードウェアとアプリケーションの両面で、長期的な目線での研究が盛んになされています。ハードウェアの領域では、**IBM** による「**IBM Q**」、**Google** による「**Sycamore**」の他、**Microsoft**、**Intel**、**Rigetti Computing** などが基礎研究と開発を行っています。アプリケーションの領域では、クラウドサービスの提供、ミドルウェア・SDK やプログラミング言語の開発、化学や製造業、金融工学などでの基礎研究での活用領域開拓など、プレイヤーが国内外で多岐に広がっています。

[*2] flickr - IBM Q より画像引用 https://www.flickr.com/photos/ibm_research_zurich/sets/72157663611181258/

2.3 量子アニーリング方式



量子ゲート方式は、プログラマーが柔軟に回路を作成できる一方で、ハードウェアの製造が非常に難しいという課題がありました。回路の柔軟性は失われるものの、もしも特定の用途に限定した場合には、回路がシンプルになるために、比較的安定して大規模化が実現できます。カナダ「**D-Wave Systems**」による「**量子アニーリング方式（Quantum Annealing）**」のマシンは、特定の領域の計算に特化して、量子力学に基づく現象を利用するハードウェアです[*4]。

図 2.4　D-Wave Systems による量子アニーリング方式のマシン「D-Wave Advantage」[*5]

量子アニーリング方式が対象としている計算は、「**組合せ最適化（Combinatorial Optimization）**」と呼ばれます。組合せ最適化は、機械学習（AI）やデータ分析の他、様々なシステムのアルゴリズムを構成する技術であり、量子アニーリングよりも古くから、数学・情報工学に関連する学術分野として確立されています。

[*4] D-Wave Systems https://www.dwavesys.com/
[*5] D-Wave - Media Resources https://www.dwavesys.com/resources/media-resources

組合せ最適化では、計算を行うために非常に長い時間がかかる場合があるという特徴があります[*6] 。計算時間上の扱いにくさを解決するために、効率的なアルゴリズムと、計算基盤の構成が必須とされています。量子アニーリングは、その高速化を実現するためのハードウェアの一つとして着目されています。

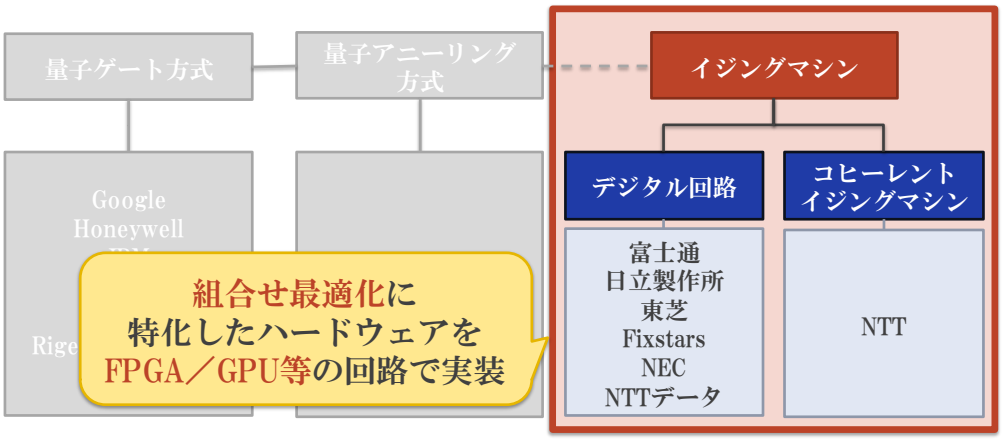
量子アニーリング方式の理論は、1998 年に東京工業大学の西森秀稔氏により提案された後に[*7] 、D-Wave Systems により 2011 年にハードウェアとして開発されており、量子ゲート方式に比べると、より近年に着目されているといえます。

本ガイドラインでは、この量子アニーリング方式に関する解説を主としています。

[*6] 組合せ最適化の対象と採用されるアプローチによっては、効率的な計算手法や既存のソフトウェアも多数存在します。本ガイドラインでは、既存の組合せ最適化技術もあわせた全体概要が理解できるよう解説していきます。

[*7] Quantum annealing in the transverse Ising model https://arxiv.org/abs/cond-mat/9804280

2.4 イジングマシン（QUBO ソルバー、アニーリングマシン）



量子アニーリング方式は、組合せ最適化に用途を特化することによって、量子ビット数の増加やノイズの低減などの高性能化を実現しました。しかし、組合せ最適化の実用のためには、非常に多くの量子ビットが必要になる一方で、製造上の難しさから求解が不安定な側面があったり、量子ビットが不足する場合があり、取り扱いが難しいことがあります。

そのため、直近での実用や、量子アニーリング方式の将来的な発展を見据えた事前検討を見据えた際には、古典コンピュータに属する既に確立された計算基盤を作り込んだほうが、ハードウェアとして安定して柔軟に大規模化しやすいという考え方も生まれてきており、D-Wave システムと同じ組合せ最適化問題に対して、FPGA や GPU で高速化を行うハードウェアが開発されています[*8] 。

組合せ最適化には、多数の問題が含まれていますが、その中でも量子アニーリング方式は「**イジングモデル（Ising Model）の基底状態探索問題**」や「**Quadratic Unconstrained Binary Optimization（QUBO）**」という問題を対象としています[*9] 。こういった、イジングモデルの基底状態探索問題や QUBO を求解対象に設定したハードウェアを「**イジングマシン（Ising Machine）**」「**QUBO ソルバー（QUBO Solver）**」「**アニーリングマシン（Annealing Machine）**」と呼称することがあります[*10][*11] 。

まとめ

- 量子コンピュータは、非常に低レイヤな **情報の表現方法** において量子力学に基づく現象を利用し、従来の回路であれば苦手である計算の高速化を図る。
- 「**量子ゲート方式**」「**量子アニーリング方式**」「**イジングマシン**」の違いの整理が重要である。

[*8] なお、「**コヒーレントイジングマシン（Coherent Ising Machine）**」は「**縮退光パラメトリック発振器（Degenerated Optical Parametric Oscillator, DOPO）**」と呼ばれる装置を主として計算に利用しており、他の GPU と FPGA のみを主とする装置と比べ、前提となる計算原理が大きく異なります。

[*9] イジングモデルの定義は 4.2 章 に、QUBO の定義は 6.1 章 にて後述します。

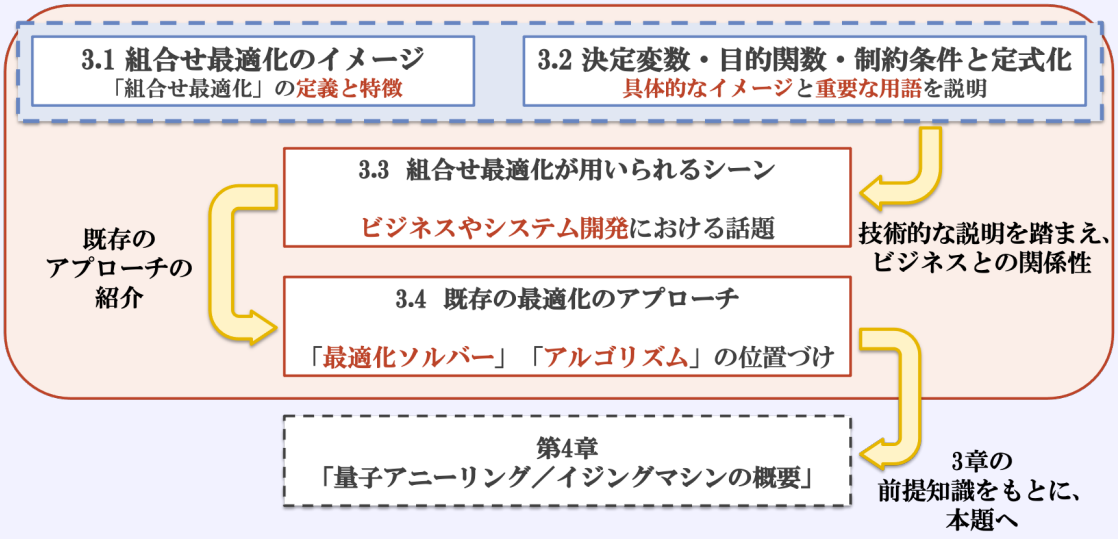
[*10] 量子コンピュータ・量子ゲート方式・量子アニーリング方式・イジングマシンが指す範囲は、メディアによって定義が異なることが多く、ひとまとめに量子コンピュータと呼ばれている場合もあります。

[*11] 本ガイドラインでは、簡単のために「イジングマシン」「QUBO ソルバー」「アニーリングマシン」をまとめて「イジングマシン」とし、その中でも D-Wave Systems による量子アニーリング方式のハードウェアを「量子アニーリング」として区別しています。

3 組合せ最適化とは

この章の流れ

本ガイドラインで解説する量子アニーリング／イジングマシンは、組合せ最適化の用途に特化して利用されています。本章では、**組合せ最適化の概要と特徴** を、具体例と図を交えて説明します。



- 3.1 章 組合せ最適化のイメージ
- 3.2 章 決定変数・目的関数・制約条件と定式化
- 3.3 章 組合せ最適化が用いられるシーン
- 3.4 章 既存の組合せ最適化のアプローチ

前段に、やや細かな技術的なトピックを記載しています。これらは、ビジネスとの関係性や既存のソフトウェア・アルゴリズムなどを説明するうえで、前提となるために意図的に配置しています。専門知識が無くとも理解できるように構成しており、この章の後段は実用に近いトピックを記載しているため、順に読み進めていくことを推奨します。

3.1 組合せ最適化のイメージ

コンテンツ

この節では、初めて組合せ最適化に触れる方に向けて、最も有名な問題の一つである **巡回セールスマン問題** を例に、用語や概念を説明します。

3.1.1 組合せ最適化

「**組合せ最適化（Combinatorial Optimization）**」は **多数の選択肢の中から、定量的に定めた基準に沿って、最良のものを選ぶ** ことをいいます[*2]。この説明だけでは抽象的であり捉えにくいため、これから具体例を挙げながら複数の節をまたいで説明していきます。

組合せ最適化には様々な問題が含まれていますが、まずは非常に有名な **巡回セールスマン問題** を例に説明します。

3.1.2 巡回セールスマン問題の例

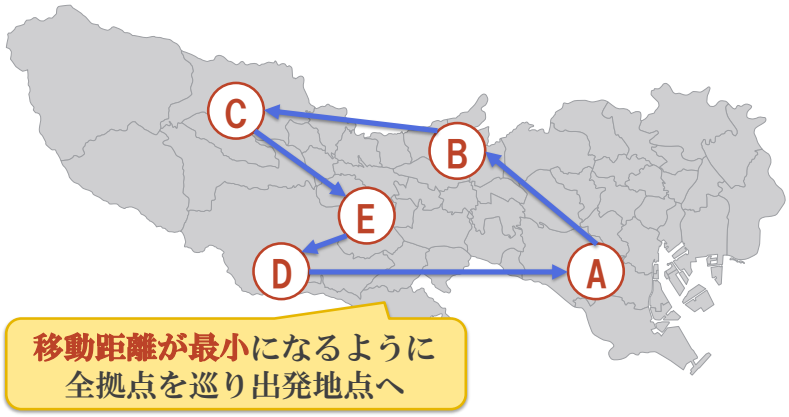


図 3.1　巡回セールスマン問題のイメージ。全拠点を巡り、出発地点へ戻る最短のルートを探す。

A 本社、B 支店〜E 支店は会社の拠点を表しています。サラリーマンの S さんは、普段 A 本社で働いていますが、とある用事があり、B 支店〜E 支店を巡った後に、最後に本社へ戻る必要があります。このとき、できるだけ無駄な移動を省き、距離が短くなるような一筆書きルートを計算する問題が「**巡回セールスマン問題（Traveling Salesman Problem, TSP）**」です。

[*2] 「**連続最適化（Continuous Optimization）**」という分野とも関連しており、包括して「**数理最適化（Mathematical Optimization）**」「**数理計画（Mathematical Programming）**」や単に「**最適化（Optimization）**」とも呼ばれることがありますが、ここでは一旦違いを無視して進めて良いです。

3.1.3 巡回セールスマン問題の巡り方の候補の数

前の節に記載した 図 3.1 には、巡り方の候補の一例を示しています。実際には、その他にも多数の巡り方の候補があります。例えば、次のように多数の巡り方を列挙することができます。

- A→B→C→E→D→A
- A→B→E→C→D→A
- A→B→C→D→E→A
- ...

では、巡り方の候補全体を考えると、合計でどれだけのパターンあるでしょうか。具体的には、次の 表 3.1 のように巡り方を「**全探索・総当り（Brute-force Search）**」することができます。

表 3.1　A 本社を開始と終了の地点として、B~E 支店の巡り方。

パターン	1 番目（出発）	2 番目	3 番目	4 番目	5 番目	6 番目（到着）	移動距離
1	A	B	C	D	E	A	120km
2	A	B	C	E	D	A	100km
3	A	B	D	C	E	A	180km
4	A	B	D	E	C	A	110km
5	A	B	E	C	D	A	130km
6	A	B	E	D	C	A	120km
7	A	C	B	D	E	A	190km
...	...	...	...	...	...	...	
24	A	E	D	C	B	A	160km

2 番目に巡る候補は B〜E の 4 通り、3 番目に巡る候補は B〜E 支店のうち 2 番目に巡ったものを除き 3 通り、という手順を繰り返すことにより、合計で、$4 \times 3 \times 2 \times 1 = 24$ 通りのパターンが存在します。[*3]

B~E 支店の巡り方

$$4 \times 3 \times 2 \times 1 = 24 \text{ 通り}$$

S さんは 24 通りのパターンの中から、距離が最短になるルートを探し出せばよいです。この例の場合には、総当りを行った結果の表 表 3.1 の各行を見れば、移動距離が最短のルートを選び出すことができそうです。

[*3] 文献によっては、若干異なる定義が記載されていることがあります。「経路として A→B→C→E→D→A とその逆周りである A→D→E→C→B→A を同一視する」「本店 A を出発地点と限らない」「距離ではなく移動費用とする」など、前提が異なる場合があります。ここでは、出発地点は本店 A に限定して、逆回りを区別し距離を最小化します。

3.1.4 巡回セールスマン問題の組合せ爆発

では、 考慮する支店の数が増えた場合を考えます。例えば、図 3.2 のように、既存の B〜E 支店に加えて F〜Z 支店も新設されたとしましょう。

図 3.2　多数の拠点を一筆書きに巡り、最短距離で A 本社に戻りたい。

表 3.1 と同じように、何番目にどの支店を巡るか総当りを行うと、実は

B~Z 支店の巡り方

$$26 \times 25 \times \cdots \times 1 = 403291461126605635584000000 \text{ 通り}$$

ものパターンが現れます。これほどの膨大なパターンの探索は、例えコンピュータを使っても数億年以上かかるような規模になってしまいます。

このように、想定する拠点が増えるにつれて、組み合わせが爆発的に増えていき、問題の複雑さが巨大化していく性質は、しばしば「**組合せ爆発（Combinatorial Explosion）**」と呼ばれます。そのため、組合せ最適化を扱う際には、全探索による組合せ爆発を避けながら、最適な選択肢を効率的に選ぶアルゴリズムを設計することが重要です。

巡回セールスマン問題は有名な問題であるため、様々な研究結果により、ノートパソコンであっても数千〜数万拠点の巡回セールスマン問題を解くことが可能になっています[*4][*5][*6] 。しかし、ビジネス上で現れる、複雑で大規模な組合せ最適化の場合、一般の開発者が効率的なアルゴリズムを逐次開発することは容易ではないため、頻繁に組合せ爆発による計算時間の問題に直面することがあります。

[*4] The Traveling Salesman Problem http://www.math.uwaterloo.ca/tsp/index.html
[*5] Concorde TSP Solver http://www.math.uwaterloo.ca/tsp/concorde.html
[*6] LKH-3 http://akira.ruc.dk/~keld/research/LKH-3/

まとめ

- 組合せ最適化は、**多数の選択肢の中から、定量的に定めた基準に沿って、最良のものを選ぶ** 問題。
- 問題の質と採用するアルゴリズムによっては、莫大な計算資源が必要になる「**組合せ爆発**」という扱いにくさが存在する。

3.2 決定変数・目的関数・制約条件と定式化

コンテンツ

組合せ最適化に関連する開発では、「**目的関数**」「**決定変数**」「**制約条件**」といった用語が現れます。ここでは、有名な組合せ最適化問題の一つである「**ナップサック問題**」を例に、「**定式化**」と呼ばれるプロセスと用語の説明を行います。
この節の後に、より具体的にビジネスと組合せ最適化の関係性を説明しますが、具体的にイメージしやすくなるように多少細かな説明も含んでいます。数式に対して抵抗感のある方は、数式の記号や意味は深追いせず、組合せ最適化のプロセスにだけ着目して進めることをお勧めします。

3.2.1 ナップサック問題

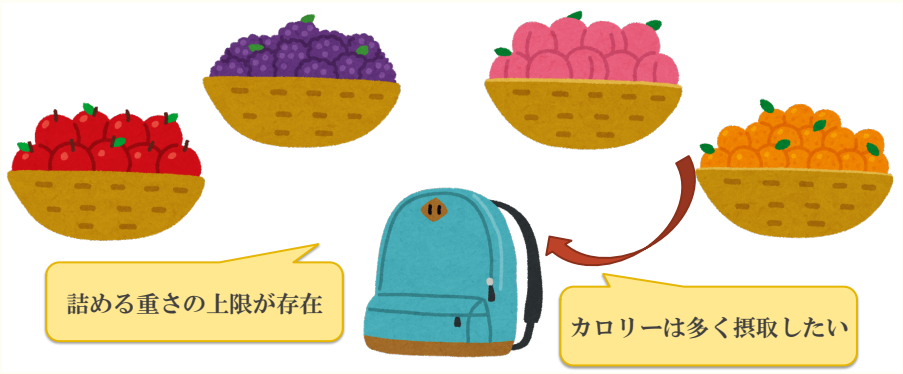
組合せ最適化のイメージ では、巡回セールスマン問題の例を説明しました。組合せ最適化の中には、巡回セールスマン問題の他にも多数の問題が含まれていますが、「**ナップサック問題（Knapsack Problem）**」もその一つです。これから、ナップサック問題を例に、組合せ最適化の分野でよく使われる重要な用語の説明を行います。

ナップサック問題

A さんはピクニックを行う計画を立てている。様々な食料の候補がある中で、できるだけ多くカロリーの摂取できる食料を持っていきたい。そこで、3000 g まで食料を詰めることができるナップサックの中に、いくつかの食料を選んでカロリーが最大になるように計画を立てている。なお、1 種類の食料を複数個詰めることも許す。

表 3.2　ナップサックに入れる食料のデータ

食料名	重さ	カロリー
りんご	250 g	138 kcal
ぶどう	200 g	120 kcal
もも	300 g	106 kcal
みかん	80 g	35 kcal



文章と図のイメージから、数学の文章題を思い浮かべる方も多いと思います。実際、ビジネス上の業務要件を組合せ最適化で表現するという取り組みは、

1. 文章題の文章を読み、
2. 方程式を記述し、
3. 方程式に対する算法を適用する

という数学の文章題の一連のプロセスと類似している側面があります。

3.2.2 組合せ最適化のプロセス

数学における文章題を解く際、求めたい未知の値を x と表現し、文章に合わせた方程式を立式しました。一次方程式／二次方程式／連立方程式で表現すれば、教科書で学んだ解き方を適用することで、方程式の解を求めることができます。

組合せ最適化においても同様に、求めたい未知の値を「**決定変数（Decision Variable）**」という変数で表現します。

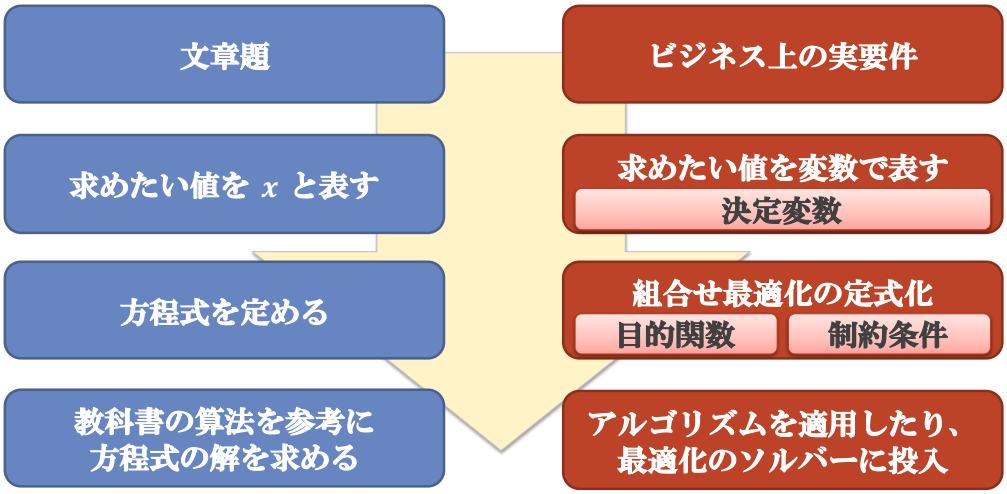


図 3.3　「学校数学の方程式の文章題」と「組合せ最適化」のよくあるプロセス

学校数学の方程式の文章題の場合には、文章題に書いてある諸条件を満足するような（＝方程式を満足するような）未知の値 x を求めることが目的でした。一方、組合せ最適化の場合には、巡回セールスマン問題の場合には距離が最短の巡り方、ナップサック問題の場合にはカロリーが最大の詰め方を探していることからわかるように、特定の定量化した値を「**最小化（Minimize）**」や「**最大化（Maximize）**」することが目的です。

最小化／最大化を行いたい量に対して、決定変数を用いて表現した関数を「**目的関数（Objective Function）**」と呼び、守らなければならない要件を決定変数で表現した関係式を「**制約条件（Constraints）**」と呼びます。そして、対象を決定変数などを用いて数式で表現することを「**定式化（Formulation）**」や「**モデル化（Modeling）**」と呼ぶことがあります[*1]。

中学・高校数学で扱う方程式と違って、定式化による表現を行っただけで定石の解法がすぐに与えられるとは限らないですが、後続の分析を施したり、最適化問題用のソフトウェアに投入するために重要なプロセスです。この詳細は、後に 3.4 章 **既存の組合せ最適化のアプローチ** でも説明します。

[*1] 対象を記述する定式化の表現は一つであるとは限らず、決定変数の定め方などによって複数の表現の方法があります。

3.2.3 ナップサック問題の定式化

目的関数・決定変数・制約条件をひとまとめに記述することで、ナップサック問題を定式化することができます。ナップサック問題の場合は、以下のような数式で定式化することができます。これから、この数式の具体的な意味の読み解き方を簡易に説明します。

ナップサック問題の定式化

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{N}^4}{\text{maximize}} \quad & 138x_{\text{りんご}} + 120x_{\text{ぶどう}} + 106x_{\text{もも}} + 35x_{\text{みかん}} \\ \text{subject to} \quad & 250x_{\text{りんご}} + 200x_{\text{ぶどう}} + 300x_{\text{もも}} + 80x_{\text{みかん}} \leq 3000 \end{aligned}$$

3.2.3.1 決定変数

求めたい未知の値、ここでは各食料を持っていく個数を、それぞれ $x_{\text{りんご}}, x_{\text{ぶどう}}, x_{\text{もも}}, x_{\text{みかん}}$ 個として表現します。ここでは、各値は負ではない整数値であり、便宜上 4 つをまとめて $\mathbf{x}$ と表します。$\mathbf{x}$ のように、求めたい対象の値を表す未知変数が決定変数です。

3.2.3.2 目的関数

続いて、ナップサック問題の目的関数を決定変数を用いて表現していきます。まず、総カロリーに対して、

総カロリーと各食料のカロリー・個数の関係性

$$\begin{aligned} [\text{総カロリー}] = {} & [\text{りんごのカロリー}] \times [\text{りんごの個数}] + [\text{ぶどうのカロリー}] \times [\text{ぶどうの個数}] \\ & + [\text{ものカロリー}] \times [\text{もの個数}] + [\text{みかんのカロリー}] \times [\text{みかんの個数}] \end{aligned}$$

という関係式が成りたちます。これを、決定変数を用いて書き直すと、次のような式になります。

総カロリーと各食料の個数、カロリーの関係性の数式による表現

$$[\text{総カロリー}] = 138x_{\text{りんご}} + 120x_{\text{ぶどう}} + 106x_{\text{もも}} + 35x_{\text{みかん}}$$

最後に、最大化したいのか最小化したいのかといった情報や、探索する対象の決定変数を明らかにするために、慣例的に使われている表現方法に揃えます。

ナップサック問題 の目的関数

$$\underset{\mathbf{x}\in\mathbb{N}^4}{\text{maximize}} \; 138x_{\text{りんご}} + 120x_{\text{ぶどう}} + 106x_{\text{もも}} + 35x_{\text{みかん}}$$

このように、最小化／最大化を行いたい対象を、決定変数を用いて数式表現したものを目的関数といいます。

3.2.3.3 制約条件

文章中で、ナップサックに重さの容量の制約が存在すると述べられています。

ナップサックの食料の総重量

$$[\text{ナップサックの食料の総重量}] = 250x_{\text{りんご}} + 200x_{\text{ぶどう}} + 300x_{\text{もも}} + 80x_{\text{みかん}}$$

であり、これがナップサックの重さの限界である 3000 g 以下でなければならないので、

ナップサック問題 の制約条件

$$250x_{\text{りんご}} + 200x_{\text{ぶどう}} + 300x_{\text{もも}} + 80x_{\text{みかん}} \leq 3000$$

このように、業務上の要件などを、決定変数を用いて表現したものを制約条件といいます。制約条件は、「**等式制約（Equality Constraint）**」や「**不等式制約（Inequality Constraint）**」など、様々な表現が存在します。問題によっては、複数の等式制約・不等式制約が同時に複数発生することがあります。

なお、決定変数に対して割り当てる結果となる数値である「**解（Solution）**」は、この制約条件を満たすような範囲の値である必要があります[*2] 。

まとめ

- 組合せ最適化に取り組む際には、数式を使って「**定式化**」を行うことが多い。
- 定式化の重要な構成要素である「**決定変数**」「**目的関数**」「**制約条件**」。

[*2] 制約条件が定める決定変数の取りうる領域を、「**実行可能領域（Feasible Region）**」といいます。また、制約条件を満足しているときに、その決定変数の値を 「**実行可能解（Feasible Solution）**」 といいます。

3.3 組合せ最適化が用いられるシーン

コンテンツ

組合せ最適化を実際のビジネスで利用する場合には、様々な業務上の要件を考慮する必要があります。それらの中には、3.2 章の **決定変数・目的関数・制約条件と定式化** で説明した制約条件のような数式で書き表すことができるものから、業務上の都合から求められる計算速度や信頼性まで多岐に渡ります。この節では、組合せ最適化のよくある用途と、考慮すべき観点を大まかに説明します。

3.1 章 **組合せ最適化のイメージ** と 3.2 章 **決定変数・目的関数・制約条件と定式化** では、「巡回セールスマン問題」「ナップサック問題」を例として、組合せ最適化の大まかな概要と定義を述べてきました。

よりビジネスに近い関連分野では、金融工学の領域でのポートフォリオ最適化などでの応用、交通における混雑の解消や混雑を避けるような配送計画の策定、製造業におけるサプライチェーン・マネジメントなど、多数の領域で最適化が用いられています。また、機械学習（AI）の領域や様々なシステム・ソフトウェアの内部で、性能を最大化したり、計算を効率化するためのアルゴリズムを提供しています。中には、「AI」として一括にされつつも、中身は組合せ最適化で動いているようなシステムも多数あります。

図 3.4　組合せ最適化の関連する領域。エンドユーザに近い領域から、基盤をなす領域まで幅広く活用されている。

一方で、**巡回セールスマン問題の例** で記載したように、組合せ最適化を効率的に計算しないと、計算時間が爆発的に多くかかり、取り扱いにくい場合があります。また、**定式化** のフェーズでは、そもそも文章題（＝顧客の業務要件）と合わない定式化を行っていないか、都度検証する必要があります。ユーザに提供する結果にも影響を与えるため、システム全体の中でも特に綿密なヒアリングと設計を意識しなければならない領域にあたります。

例えば、要件定義における大きな観点で見たときには、特に次のような点が重要になることが多いです。

- 設定した定式化が、**真に顧客・システムのやりたいことと一致しているか** 十分に検証したか[*1] 。
- 各制約条件は、**必ず守らなければならない** ものなのか、**できれば守って欲しい** 程度のものなのか。
- 計算する解は、**必ず最適解である必要がある** のか、**ある程度違っても許容** されるのか。

計算時間に絞ると、次のような観点も頻繁に現れます。

- **夜間などに事前にバッチ計算** し、日中の業務時間内は計算結果が参照さえできればよい。
- 定式化が妥当であるか数値実験するために、**数十分程度** で検証サイクルを回したい。
- 災害避難経路の案内など、**オンライン処理** を行わなければならない。

上記の観点とあわせて、以下のように非常に様々な事情を総合的に考慮して、組合せ最適化の導入を行わなければなりません。

- 組合せ最適化の定式化と業務要件との一致
- 求められる解の信頼性
- 必要な計算の即時性
- 実装できる機能の柔軟性
- 開発に要する工数や専門性
- 必要となる計算基盤のスペック

これらの事情に応じて、組合せ最適化問題の解を求めるためのアプローチは、異なるものが採用されます。量子アニーリング／イジングマシンを利用する際にも、これら業務要件とマシンの特性を照らし合わせることは非常に重要です。後続の 3.4 章 **既存の組合せ最適化のアプローチ** の節では、従来から存在する、組合せ最適化に対して頻繁に用いられるアプローチを紹介しています。

まとめ

- 金融・交通・物流・製造業・機械学習（AI）など、応用先は多岐に渡る。
- 目的関数の設定・求められる解の信頼性・計算時間などの観点と、業務要件とを照らし合わせてアプローチを検討することが重要。

[*1] 例えば、カーナビが「距離最小化」を目的関数として細い山道を案内しても、運転手の真にやりたいことにとっては良くない選択かもしれません。これは目的関数の設定を誤っている典型例であると考えられます。

3.4 既存の組合せ最適化のアプローチ

コンテンツ

量子アニーリング／イジングマシンを利用する前に、**従来の組合せ最適化に対する取り組みのフロー** を紹介します。アプローチによって様々なメリット・デメリットがあるため、求められる業務要件と照らし合わせて変える必要があります。

3.4.1 最適化用のソフトウェアとアルゴリズム

ここまでの説明では、組合せ最適化の概念と、定式化を行うというプロセスについて説明しました。また、定式化した解を具体的に得たいときには、3.3 章 **組合せ最適化が用いられるシーン** のように、考慮しなければならない観点は多数あるため、業務要件に応じたアプローチを採用することがあることを説明しました。

この節では、最小化／最大化したい値や、制約条件が明らかになった後に、解を実際に計算する方法について紹介します。量子アニーリング／イジングマシンも、解の計算のフェーズで用いられるハードウェアですが、既存の最適化問題へのアプローチ方法・ソフトウェアの背景を理解することで、役割がより明確になると考えられます。

アプローチ	特徴	メリット	デメリット
最適化ソルバーの適用	定式化を行い、数式を記述すると、解を自動で計算	最適化の観点で汎用的**専門性・工数を要さず**、**手軽に検証**できる	**計算が遅いこともある**ソルバーの対象や特性を考慮した**独特な定式化**
業務に特化したパッケージの適用	業務によっては、既存のパッケージ商品やソフトウェアが存在する	**専門性・工数を要さず**、計算速度・機能などが**作り込まれ**、**費用対効果**に優れることが多い	業務によっては**世に無い**ことがある自力でロジックの変更ができず、**柔軟性に乏しい**
問題に応じた**アルゴリズム**の開発	最適化・アルゴリズムの専門家が独自にプログラミングする	**業務要件に合致**したシステムを開発可能	既存の解法があるケースを除き、**求められる専門性・工数が大きい**

3.4.1.1 最適化ソルバー

例えば、勤務シフトを組むようなシーンにおいて、シフト管理担当者が時間をかけて計画を立ててよい場合には、「**最適化ソルバー（Optimization Solver, Optimizer）**」を利用する方法があります[*1]。最も身近な例でいう

[*1] 最適化問題を解く際に汎用的に利用できることから「**汎用ソルバー**」と呼んだり、より解く対象の問題に関し言及して「**混合整数計画問題ソルバー（Mixed Integer Programming Solver, MIP Solver）**」と呼ぶことがあります。

と、実は Excel にも最適化ソルバーが実装されており[*2][*3]、セルに記述されているデータに基づいて計算をすることができます。

より数理最適化の用途に特化したソルバーを利用して問題を解きたい場合には、**商用・OSS の最適化ソルバー**も便利です。多くのソルバーは、ユーザが定式化を行った組合せ最適化の数式を、ほぼそのまま書き下すことによって解を計算するモデリング連携機能も提供しています。この場合、**定式化さえ行えば、解の算出をある程度自動的に行なってくれる** ために、組合せ最適化のアルゴリズム部分を専門家が深く理解した上で開発する必要が無く、比較的少ない手間で結果を得ることができます。

最適化ソルバーを利用する場合も、必ずしも毎回人手を介する必要は無く、既存のシステムの API に沿う形で入れ込んで自動化することも可能です。これは、一度だけ人が定式化を行えば、与える係数部分はコンピュータにより自動で置き換えをすればよく、人が都度定式化を変更する必要は無いからです。例えば、ナップサック問題において、ナップサックの容量サイズが変わったり、アイテムの重さが変わったとしても、カロリーを最大化したい・容量の制限が存在するといった要件自体が変わらない限り、与えられた数式の係数を変える処理だけで十分です。

最適化ソルバーは、このように汎用性が高く、簡易に組合せ最適化の結果を検証できるメリットがある一方で、**特定の業務に対して作り込まれているわけではない** ために、本格的に業務に特化して作り込まれたソフトウェアやアルゴリズムに比べると、計算速度が遅くなる傾向にあります[*4]。また、何らかの対象を定式化した数式は一意ではなく複数ありえますが、その違いによってソルバーの計算速度が大きく左右されたり、ソルバーの入力形式に沿うよう人手での数式変形がある程度必要な場合があるなど、独特なテクニックを若干必要とします。

3.4.1.2 業務に特化したパッケージ

定式化のプロセスも避けたい場合は、**業務に特化したパッケージの導入** なども考えられます。最も身近なシステムで例えると、カーナビ・在庫管理・シフト管理のように、車や店によって共通する業務に特化して機能を提供するソフトウェアが開発・販売されています。他にも、巡回セールスマン問題に対する様々な研究結果が反映され高速化された特化型ソルバー[*5] なども存在します。解決したい問題に合致するパッケージが存在した場合には、開発のための特殊な専門性・工数も少なく、最適化ソルバーや後述の自作のアルゴリズムよりも作り込まれており、高速・多機能である事が多いです。一方で、**要件を満たさないような場合に、自分で追加機能を実装することが困難であるため、採用できない** 場合があるデメリットがあります。また、新規事業であったり、あまり一般的でない業務である場合には、**そもそもパッケージが存在しないために採用できない場合も多々あります**。

[*2] Excel で始める数理最適化 http://www.orsj.or.jp/archive2/or57-04/or57_4_175.pdf

[*3] Microsoft - Using Solver to determine the optimal product mix https://support.microsoft.com/en-us/office/using-solver-to-determine-the-optimal-product-mix-c057e214-962f-4339-8207-e593e340491f

[*4] 例えば、ソート（数の並び替え）の問題をわざと組合せ最適化として定式化することは可能ですが、ソート用の効率的なアルゴリズムは既に知られているため、最適化ソルバーを利用すると手軽さ・計算速度の両側面で非効率的と考えられます。

[*5] Concorde TSP Solver http://www.math.uwaterloo.ca/tsp/concorde.html

3.4.1.3 アルゴリズムの開発

既存の最適化ソルバーでは性能が不足していたり、カスタマイズができないパッケージでは要件を解決できないような場合には、**アルゴリズムに詳しい開発者が、組合せ最適化の数式や業務要件を個別に見て、効率的に計算できる方法を独自に開発** する場合もあります。例えば、3.2 章 **決定変数・目的関数・制約条件と定式化** で説明したナップサック問題であれば、動的計画法と呼ばれるアルゴリズムを工夫することによって、ある程度大規模な問題であっても正確に最適解を計算することができることが知られています[*6] 。ただし、実際のビジネスにおける複雑な組合せ最適化を解決するためには、そもそも既存の有名な解法が存在しない場合があります。組合せ最適化やアルゴリズムに強みのある専門家・プログラマーによる研究開発や、専用の計算基盤の検討など **専門性や工数が多く必要** になる可能性があります。

3.4.2 厳密解の保証とヒューリスティック

前の 3.2 章 **決定変数・目的関数・制約条件と定式化** の節では、組合せ最適化の解を得るためのアプローチの種類を説明しました。同じ「解」であっても、組合せ最適化に対して採用したアプローチに応じて、得られる解の意味合いが変わることがあります。組合せ最適化の本来の目標のように、計算された結果の解が最適解であることが保証されているとき、「**厳密解が保証されている**」といいます。更に、その計算が効率的である場合には、「**多項式時間（Polynomial Time）**」のアルゴリズムであると呼ばれます[*7] 。

一方で、組合せ最適化問題の性質によっては、多項式時間の効率的なアルゴリズムが考案されていない問題も多数存在します[*8] 。実用上は、これらの難しい問題に対しても何らかの解は得たいため、最適であることは妥協した上で、できるだけ良質な解を得ることを目標にする考え方も存在します。このような背景で、最適性が保証されていないアルゴリズムを「**ヒューリスティック（Heuristic）**」と呼び、得られた最適とは限らない解を「**近似解（Approximate Solution）**」といいます[*9][*10] 。

各アプローチ（最適化ソルバー・パッケージ・アルゴリズム）が厳密解を与えるかヒューリスティックであるかは、それらの実装と対象の組合せ最適化問題との相性や使い方によって都度変わります。そのため、パッケージの場合には実装されている仕様の確認、独自開発のアルゴリズムの場合には専門家による最適解への収束の証明などが別途必要です[*11] 。

[*6] （書籍）プログラミングコンテストチャレンジブック [第 2 版]　～問題解決のアルゴリズム活用力とコーディングテクニックを鍛える～「2-3 値を覚えて再利用”動的計画法”」より https://book.mynavi.jp/ec/products/detail/id=22672

[*7] 簡単のために省略していますが、正確には「効率的」といった直感的な定義ではなく、アルゴリズムの計算量を評価したときに多項式で表現されることをいいます。

[*8] 正確な定義は割愛しますが、「**NP 困難（NP-Hard）**」「**NP 完全（NP-Complete）**」と呼ばれるクラスの組合せ最適化が該当します。

[*9] 単に解ということなどもあり、文脈により表現が変わることがあります。また、**大域的最適解と局所的最適解** でも説明を行っています。

[*10] 「**メタヒューリスティック（Metaheuristic）**」の場合、特定の問題を想定せずに、広く様々な問題に対して対応できるアルゴリズムの構成の「枠組み」を指すニュアンスがあります。初学者でイメージが付きにくい場合は、一旦、同じものとして理解することを勧めます。

[*11] 最適化ソルバーの多くの場合、計算を行っている最中はヒューリスティックなアルゴリズムも動作させながら、時間をかけて近似解の改善を行っていきます。ただし、逐次生成されている近似解が、最適解からどれくらいギャップがあるかを見積もる機構が備わっています。このギャップがゼロになった場合には、生成された解が厳密解であることが保証されます。

まとめ

- 「**最適化ソルバー**」「**パッケージ**」「**独自開発**」といった選択肢を、計算速度・開発工数・業務要件などに応じて選ぶ。
- 「**厳密解保証**」が計算時間的に難しい場合には「**ヒューリスティック**」が用いられる。

4 量子アニーリング／イジングマシンの概要

この章の流れ

量子アニーリング／イジングマシンは、組合せ最適化の求解が目的のハードウェアです。本章では、これらのハードウェアが **システム全体でどのようなシーンで利用されるのか**、特に入出力データのフローなどを中心に解説します。また、**量子アニーリングの計算の仕組み** について、開発を行う上で最低限必要な内容の紹介を記載しています。

- 4.1 章 **ハードウェアの役割と特徴**
- 4.2 章 **イジングモデル**
- 4.3 章 **ナチュラルコンピューティング**
- 4.4 章 **量子アニーリングのイメージ**

節によっては、技術的に細かく入り込むことを防ぐために、少し抽象的な用語の説明で終えている場合があります。次の 5 章 *D-Wave Leap* **でのプログラミング例** でより具体的に説明を行うため、不明瞭に感じた部分は深堀りしすぎず、一旦読み進めることを推奨します。

4.1 ハードウェアの役割と特徴

コンテンツ

本節では、3.4 章 **既存の組合せ最適化のアプローチ** で説明した内容を踏まえて、組合せ最適化問題を解く際に、量子アニーリング／イジングマシンいた場合の特徴を説明します。計算を行う原理については一旦後に回し、マシンへ入出力されるデータに注目してください。

4.1.1 量子アニーリング／イジングマシンのアプローチ

量子アニーリング／イジングマシンは、3.4 章 **既存の組合せ最適化のアプローチ** で説明した中の枠組みでいうと、**最適化ソルバーを利用するアプローチに近いフロー** を取っています。ユーザは、まず最初に対象とする業務の定式化を行い、マシンに定式化の結果を投入します。その後、求解プロセスは基本的に量子アニーリング／イジングマシンの内部でのみ実行されて、ユーザはアルゴリズムの構築を意識することなく結果を得ることができます。

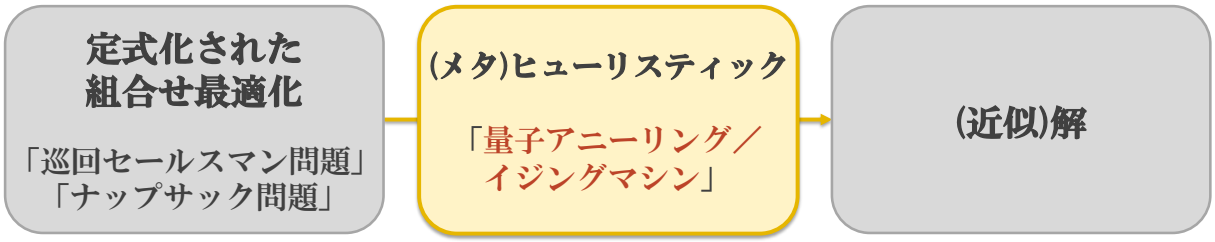


図 4.1　量子アニーリング／イジングマシンによる求解のフロー

ただし、量子アニーリング／イジングマシンから得られる解は最適解である保証は無いため、ヒューリスティックなアルゴリズムの一種であるといえます[*1][*2] 。

4.1.2 マシンへの入出力データ概要

直前の節では、量子アニーリング／イジングマシンが定式化した数式を、そのまま入力として受け取るかのような説明を行いました。正確には、量子アニーリングやイジングマシンは「**イジングモデル（Ising Model）** の基底状態探索問題」と呼ばれる、特定の組合せ問題の求解だけに特化しています[*3][*4] 。

定式化の結果は業務要件によって様々であるため、いきなりイジングモデルと呼ばれる形式に定式化されることはありません。例えば、**ナップサック問題を定式化した数式** はイジングモデルの形式とは異なっているため、直接量子アニーリング／イジングマシンに投入することは不可能です。

ただし、組合せ最適化の定式化結果を、イジングモデルに合致するように強制的に変形する前処理を施すことにより、様々な問題を取り扱うことができます[*5][*6][*7] 。次の 図 4.2 はそのフローを示しています。

[*1] 理論上、十分に長い計算時間の場合に、最適解へ収束することが証明されていますが、ノイズが存在する現実のハードウェア上で、持続可能な時間で打ち切った場合を想定しています。

[*2] 7.4 章 **解のサンプリング** で後述しますが、大量の近似解が短時間で得られるなど、通常のアルゴリズムとは異なる特色も存在します。

[*3] マシンによっては、後に紹介する **QUBO** というインタフェースを採用しています。イジングモデルと QUBO には小さな違いはあるものの、一旦両者とも似たものであると考えて良いです。

[*4] 観点に応じて「最適化ソルバー」ではなく「イジングモデル基底状態探索問題に対する業務に特化したパッケージ」と見なすこともできますが、ここでは一般的な組合せ最適化を解く場合としての役割を想定しています。

[*5] イジングモデルに数式を変形する前処理用のツール「**PyQUBO**」が提供されており、5 章 *D-Wave Leap* **でのプログラミング例** で説明します。

[*6] この入力形式の変換は「**多項式時間帰着（Polynomial-time Reduction）**」と呼ばれます。

[*7] 既存の最適化ソルバーも、実は「**混合整数計画問題（Mixed Integer Programming, MIP）**」や半正定値計画問題といった、特定の問題を対象としたアルゴリズムが実装されています。それらへの強制的な変形を、ユーザが意識しないで済むようなモデリング連携機能ライブラリが開発なされています。

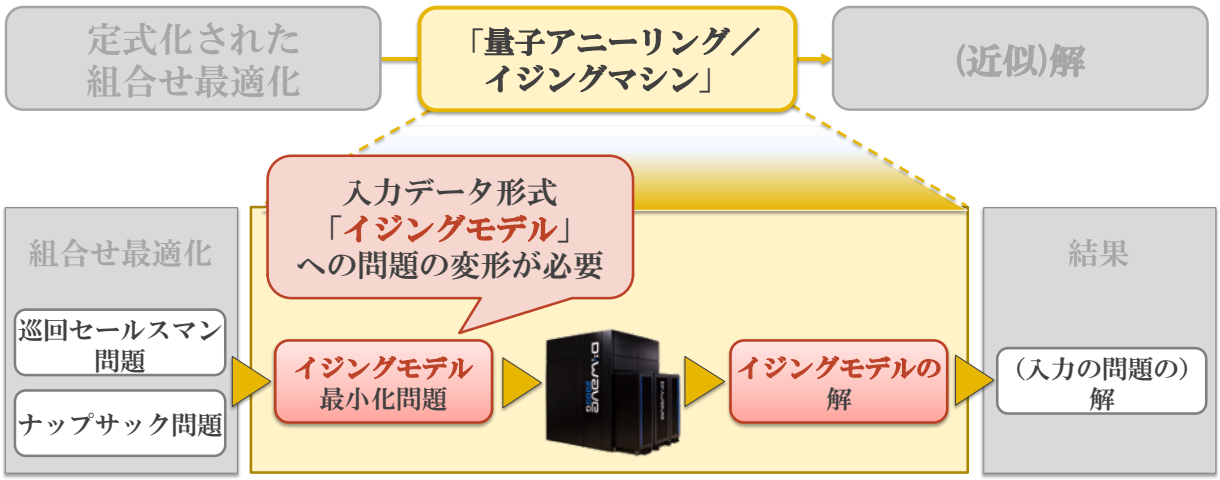


図 4.2　量子アニーリング／イジングマシンはイジングモデルに特化し求解、広く組合せ最適化に用いるためには前後に追加処理が必要。

この変形の方法には、量子アニーリング／イジングマシン特有の様々な種類の処理が必要であり、その施し方によって得られる解の質や安定性が変わることがあります[*8]。本ガイドラインの後の章でも、変形のテクニックについて記載しています。

まとめ

- 量子アニーリング／イジングマシンは、**ヒューリスティックなアルゴリズムが実装された最適化ソルバー** という立場に近い。
- 入力データの形式は「**イジングモデル**」である。

[*8] 組合せ最適化の定式化と変形の処理は、従来の最適化ソルバーでも必要であるため、既存の教科書なども参考になります。

4.2 イジングモデル

コンテンツ

前の 4.1 章 **ハードウェアの役割と特徴** では、量子アニーリング／イジングマシンにおける入力のデータがイジングモデルであることを説明しました。この節では、具体的に **イジングモデルとは何か** について説明します。

3.1 章 **巡回セールスマン問題** や 3.2 章 **ナップサック問題** において、定式化した数式の前提として現実の要件が存在していたように、イジングモデルにおいても、**定式化した数式** と、**その数式が前提とする事象** が存在します。イジングモデルが意味する事象は、とある物理学的なモデルです。実は、量子アニーリングのハードウェア内部の物理的な構造と深く関連しています。

物理学的な意味が分かりにくいため、躓きやすいポイントです。しかし、この節の一番下で解説しているような、イジングモデルの定式化した数式の要点と、各単語の意味を覚えることができれば、初めて触るユーザとしての理解は十分であると考えられます。書いてある解説の意図が分からない場合、まずは斜め読みをし、次節以降にて解説を行う **ナチュラルコンピューティング** や **シミュレーテッド・アニーリング** を読んだ後に再読することを推奨します。

「**イジングモデル（Ising Model）**」は、次のような意味を持つモデルです[*1] 。

[*1] 正確には、ハードウェアが固有に採用しているイジングモデル上での計算を考える必要があります。例えば、(4.1) で示されている入力データはそのまま解けず、後に説明する 7.1 章 **疎結合と全結合** の手法が必要になります。

イジングモデル

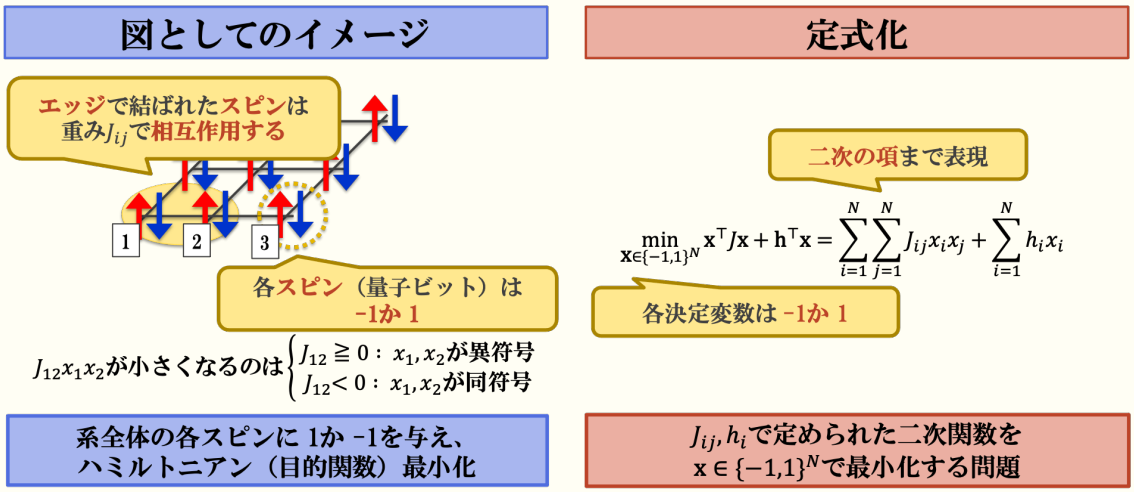


図 4.3　イジングモデルの基底状態探索問題の物理的な意味と、その組合せ最適化としての定式化。

−1 か 1 のいずれかの値を取る「**スピン（Spin）**」という素子が複数個存在する。各スピンは、−1 か 1 の取りやすさの傾向の偏りを意味する「**局所磁場項**」を持っている。更に、エッジで結線されている周辺のスピンの値を見ながら、エッジの重み「**相互作用項**」が負であれば互いに同じ値を、正であれば互いに違う値を取ろうとする傾向が生じる。局所磁場項と相互作用項の値が大きく設定されているほど、優先的に傾向が守られるように設定される。

このようなスピンとエッジ全体がなす系を「**イジングモデル（Ising Model）**」と呼ぶ。与えられたイジングモデルに対して、最も局所磁場項と相互作用項の要請を満足する（＝「**ハミルトニアン (Hamiltonian)**」を安定させる）ために、各スピンが −1 か 1 のどちらを取ればよいか探索することを、「**イジングモデルの基底状態探索問題**」という[2][3]。

[2] イジングモデルの基底状態探索問題では長いため、イジングモデルと呼ぶだけで問題を意味したり、イジング最適化と呼ばれることもありますが、どれも同一のことを指しており、本書でも省略して呼ぶことがあります。

[3] 「スピンとハミルトニアン」は文脈に応じて適切な呼び方が変わり、組合せ最適化では「決定変数と目的関数」、ハードウェア諸元では「(量子) ビットとエネルギー」となる場合があります。また、相互作用項を 2 次項、局所磁場項を線形項や 1 次項と呼ぶ場合もあります。

このイジングモデルの基底状態探索問題を、組合せ最適化として定式化すると、次の (4.1) のような目的関数が得られます。

イジングモデルの基底状態探索問題

x_i $(i = 1, \cdots, N)$ は、スピンが -1 か 1 である値を表す決定変数であり、J_{ij} $(i = 1, \cdots, N; j = 1, \cdots, N)$ はスピン i, j 間の相互作用項を表し、h_i $(i = 1, \cdots, N)$ はスピン i の局所磁場項を表す定数である[4]。

$$\underset{(x_1, x_2, \cdots, x_N) \in \{-1,1\}^N}{\text{minimize}} \quad \sum_{i=1}^{N} \sum_{j=1}^{N} J_{i,j} x_i x_j + \sum_{i=1}^{N} h_i x_i + \text{const.} = \mathbf{x}^\top J \mathbf{x} + \mathbf{h}^\top \mathbf{x} + \text{const.} \tag{4.1}$$

[4] なお、目的関数の符号の正負が違ったり、Σ の添字に $i < j$ という条件がついていることもありますが、実質的に同一であり、詳細は 6.5 章 **組合せ最適化問題の前処理** で説明します。

この定式化には、以下のような特徴があります。

- 決定変数が -1 か 1 である。
- 目的関数の次数が 2 次までの多項式である。
- 制約条件は明示的に存在しない。

ユーザから見ると、量子アニーリング／イジングマシンに $J, \mathbf{h}$ を入力データとして与えるだけで、その結果の解が -1 か 1 で構成されて返却されます。一方、量子アニーリング／イジングマシンから見ると、$J, \mathbf{h}$ によってイジングモデルが一意に特定できるため、それらを 2 次までの多項式からなる目的関数としてセットし、ハードウェアを動作させて計算を実施します。

実は、量子アニーリングでは、ユーザの入力に基づいて、イジングモデルの物理的な現象を、実際にハードウェアの内部に再現しています。この考え方は、次の 4.3 章 の **ナチュラルコンピューティング** にて説明します。

まとめ

- 「**イジングモデルの基底状態探索問題**」は物理学・数学的なモデルを定式化したもの。
- 利用する観点では、イジングモデル (4.1) の以下の 3 つの特徴を抑えたい。
- 決定変数が -1 か 1 である。
- 目的関数の次数が 2 次までの多項式である。
- 制約条件は明示的に存在しない。

4.3 ナチュラルコンピューティング

コンテンツ

量子アニーリングのハードウェアである **D-Wave システム** は計算の原理が通常のコンピュータの考え方と大きく異なり、**ナチュラルコンピューティング** と呼ばれる種類に属します。
D-Wave システムの物理学・量子力学に関連する挙動をいきなり学ぶことは難しいです。そのためこの節では、先にナチュラルコンピューティングの考え方について、分かりやすい既存の例で説明します。
その後、D-Wave システムがナチュラルコンピューティングに属するイメージを説明します。

4.3.1 ナチュラルコンピューティングとは

量子アニーリングの一種である D-Wave システムは、「**ナチュラルコンピューティング (Natural Computing)**」という計算方法の一種であると考えられています。

ナチュラルコンピューティングでは、通常のコンピュータと計算の原理が大きく異なり、CPU やメモリの概念もありません。自然現象の性質をコンピューティングに用いること、とまとめることができますが、抽象的な説明であるため、「粘菌コンピュータ」「シャボン膜による計算」という有名な 2 つの具体例で説明します。

4.3.1.1 粘菌コンピュータ

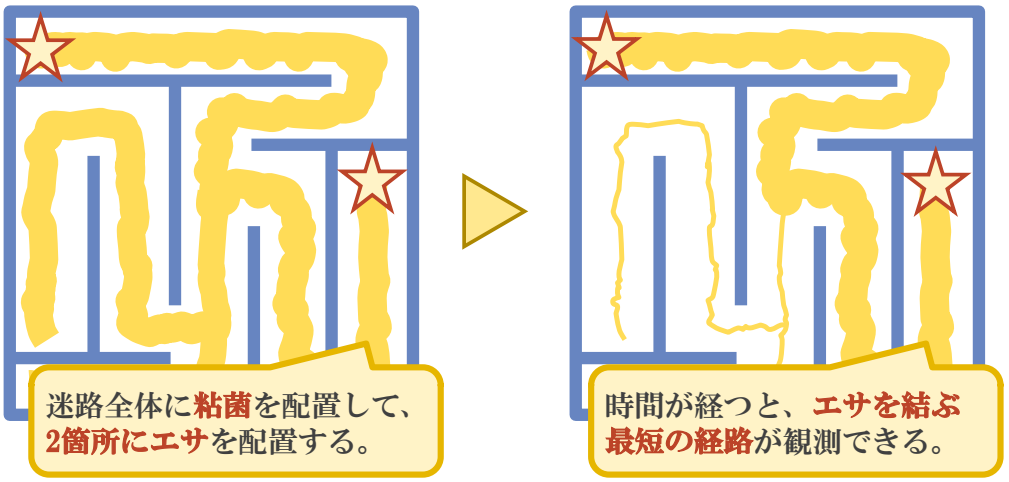


図 4.4　粘菌コンピュータによる迷路上のエサの探索[*1]

粘菌コンピュータは、ナチュラルコンピューティングの代表的な例の一つとして知られています[*2] 。粘菌はア

[*1] Path finding by tube morphogenesis in an amoeboid organism https://pubmed.ncbi.nlm.nih.gov/11527578/
[*2] TED - What humans can learn from semi-intelligent slime https://www.ted.com/talks/heather_barnett_what_humans_can_learn_from_semi_intelligent_slime

メーバの一種であり、這い回って形を自在に変形させながら、栄養になるエサを探す性質があります[*3] 。

迷路上のとある地点から、別の地点に向かって最短の経路を探したいとします。通常のコンピュータで計算する場合には、その地点から経路をルールを定めて順に探索し、出口を見つけるまで計算します。

一方、粘菌コンピュータでは、探索を行うためのアルゴリズムを構築しません。迷路全体に万遍なく粘菌を配置し、更に迷路上の 2 点にエサを配置します。実は、粘菌は広がった細胞全体が連絡を取り合い、エサを探索するという性質を持っているため、迷路上を粘菌で連結する形を時間をかけて観測することにより、最短の経路を計算することができます。

したがって、迷路を解きたい場合には、その迷路の形に合わせて同じような壁を実際に準備して粘菌を配置し、物理実験の結果を観測して解を得ることができ、探索問題のアルゴリズムの代替とすることができます。

4.3.1.2 シャボン膜による計算

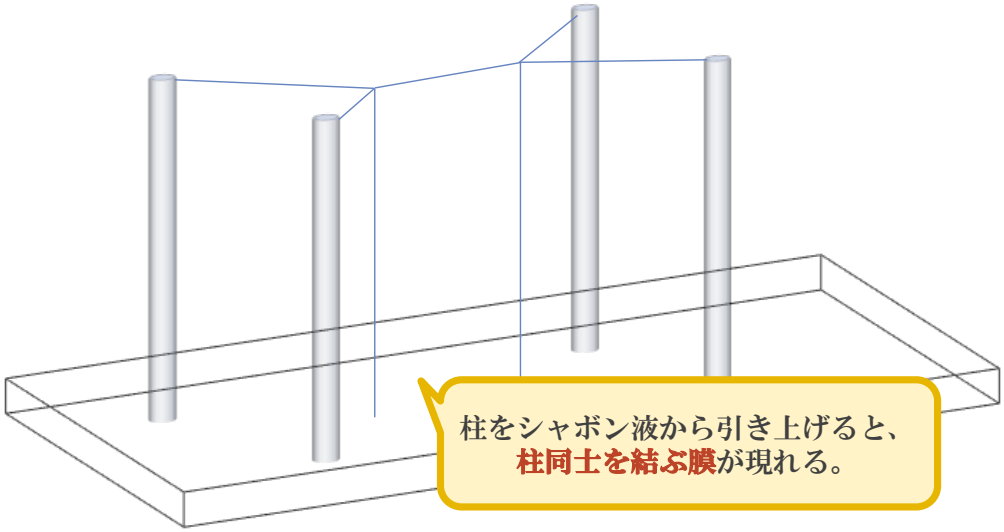


図 4.5　シャボン膜がなす壁が、柱同士を繋ぐ最小の線分をなす

平面上に点がいくつか存在するときに、それら全体を結ぶ最小の長さの線分を計算する問題を、最小シュタイナー木問題といいます[*4] 。柱を数本準備して適当な位置に立て、シャボン液から引き上げることで、シュタイナー木問題の近似解の一つが得られることが知られています[*5] 。この場合も、粘菌コンピュータと同様に、解きたい最小シュタイナー木問題の形に合わせて柱をシャボン液上に配置して、物理実験の結果を観測して解を得ることができます。

[*3] 澤井 哲, 脳を持たない粘菌が集団行動する秘密 https://www.works-i.com/works/series/macro/detail005.html
[*4] 最小シュタイナー木問題は、一般的には平面上の点の距離がコストであるとは限りませんが、ここでは分かりやすさのために限定した定義をしています。
[*5] 近道とシャボン膜の数学 https://mathsoc.jp/publication/tushin/2201/2201koiso.pdf

まとめ

- ナチュラルコンピューティングの例として「**粘菌コンピュータ**」「**シャボン膜による計算**」。
- 計算したい対象と同じ振る舞いをする物理実験に計算を委ね、結果の観測によって解を得る。

4.4 量子アニーリングのイメージ

コンテンツ

量子アニーリング（D-Wave システム）に入力したイジングモデルに対して、最小解の探索を行うイメージを説明します。

D-Wave システムで実装されている量子アニーリングも、**ナチュラルコンピューティング** の一種に属します。解きたい「イジングモデルの基底状態探索問題」の解を、量子力学に基づく自然現象を用いて計算し、結果を観測します。

本節では、量子アニーリングが実際にイジングモデルを計算しているときの仕組みの概要を説明します。なお、本節に記載している内容は、ユーザとして必ずしも理解しなくても良い内容です。ハードウェアの様々なパラメータを調整して、性能を引き出す際には計算原理を理解していたほうが望ましいですが、ここまでの内容で、説明していない事項も存在します。初学者の場合には斜め読みをし、**シミュレーテッド・アニーリング** などの関連項目を学んでから再読することを推奨します。逆に、実際の D-Wave システムに実装されている物理素子を簡易化して説明しているため、ハードウェアに関する正確な実装の説明は教科書等を参照ください[*1][*2] 。

量子アニーリングの計算対象は 4.2 章 で説明した **イジングモデル** であり、その数式は (4.1) で表されます。実は、D-Wave システムでは、その内部の物理的な状態を、解きたい対象を表すイジングモデルと全く同じような物理実験系を持つように設定します。具体的には、スピンに該当する素子を準備し、それらに $1, -1$ の取りやすさを制御する局所磁場項に相当する作用を与えます。スピン同士のエッジに該当する結線を行い、互いの相互作用項に対応する重みを設定します。

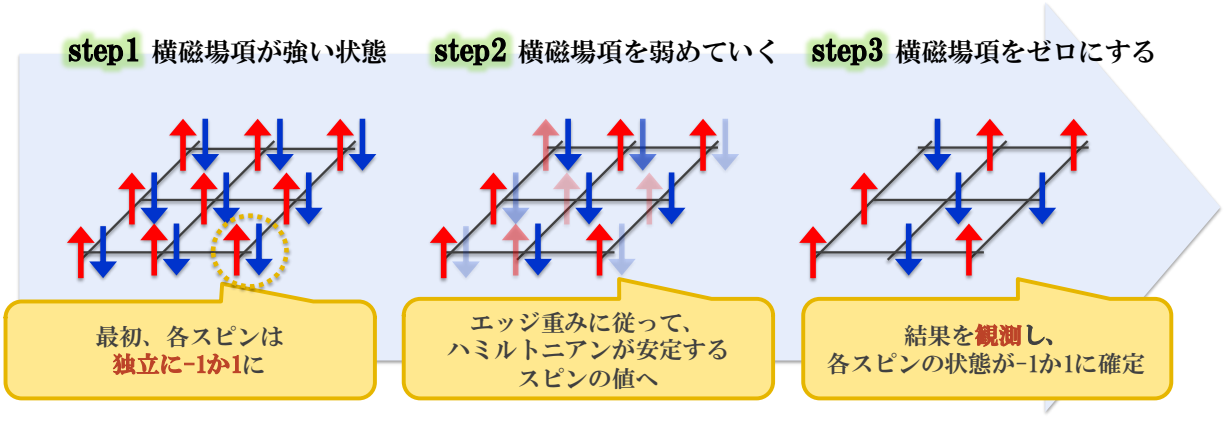


図 4.6　量子アニーリングの動作のイメージ。解きたい問題と同じ物理実験系を準備して、物理実験の結果の観測により解を得る。

「**横磁場項（Transverse Magnetic Field）**」は、各スピンが $-1, 1$ のどちらを取るかを同時探索させる（重ね合わせ状態とする）ような力を加えています。この系の、横磁場項をゆっくりと下げていくと、ハミルトニアン

[*1] 量子アニーリングのためのハードウェア技術-超電導エレクトロニクスと超伝導量子回路- https://www.orsj.or.jp/archive2/or63-6/or63_6_335.pdf

[*2] （書籍）量子アニーリングの基礎 https://www.kyoritsu-pub.co.jp/bookdetail/9784320035386

（目的関数）が小さくなるようにスピンが 1 か −1 に収束するという性質があります。結果、ハミルトニアンを小さくするような解を観測できます。

局所磁場項や相互作用項に対応する結線の設定値を、解きたい組合せ最適化問題の目的関数の構造と一致するよう、人為的に設定することによって、イジングモデルのハミルトニアンの最小化を通じた、目的関数の最小解をの取得ができるようになります。このように、自然現象に計算を委ねて、観測によって結果を得ることから、D-Wave システムによる量子アニーリングはナチュラルコンピューティングの一種であると呼ばれます。

まとめ

- D-Wave システムによる量子アニーリングはナチュラルコンピューティングの一種。
- 解きたい対象のイジングモデルと同じ物理系を設けて、ハミルトニアン（目的関数）が小さくなるような実験を実施。

5 D-Wave Leap でのプログラミング例

この章の流れ

ここまでの章では、**組合せ最適化** や、**量子アニーリング／イジングマシン** の概要について説明しました。この章では、細かな説明に入る前に、具体的なイメージが持てるよう、**D-Wave システムによるプログラミングの例** を記載しています。

- 5.1 章 **動作環境について**
- 5.2 章 **入出力処理の確認**
- 5.3 章 **分割問題**
- 5.4 章 **巡回セールスマン問題**

また、実際に組合せ最適化の定式化を行っていき、**イジングモデル** へ変形する手順を追うことができます。D-Wave システムを利用するための環境設定を行った後に、簡単な入出力の確認を行い、具体的な問題を例に解き方を確認します。

5.1 動作環境について

5.1.1 利用登録と接続情報の取得

この節では、D-Wave システムを利用するための準備を説明します[*1] 。

通常、D-Wave システムを利用する際には、クラウドサービス **D-Wave Leap** 経由で登録を行い、接続情報を得ます[*2] 。D-Wave Leap には、有償の月額制サービスと、無償体験版があり、無償体験版は 1 分/月の利用ができます[*3] 。クラウドでの利用契約を行うと、ログイン先のサイトを経由して、接続のための **endpoint**, **token**,

[*1] AWS Braket のような、他のクラウドサービスを経由する方法もありますが、ここでは D-Wave Leap での利用を想定しています。
[*2] D-Wave Leap https://www.dwavesys.com/take-leap
[*3] 実稼働時間で換算されます。実験条件次第ですが、1 回の求解あたりミリ秒以下であることが大半であるため、トライアルのためには十分な時間が提供されています。

solver の情報が発行されます。

本ガイドラインでのコーディング例を実行するためには、D-Wave Leap のトップページに記載されている、以下の文字列情報を手元に取得してください。

- Solver API Endpoint
- Supported Solvers
- API Token

5.1.2 環境構築

D-Wave システムを利用する場合、OSS として開発されている D-Wave Ocean SDK を利用します[*4][*5] 。D-Wave Ocean SDK は Python によって動作し、クラウド上の D-Wave システムに REST API で通信し求解を要求する機能や、ユーザのローカル環境で数値実験等を行うための、データ分析や前・後処理用の機能を提供しています。

Python 3.5+ 系が動作する環境であれば、通常のノートパソコン上などでも十分に利用が可能です[*6] 。以下のようにライブラリのインストールを行ってください。

```
pip install dwave-ocean-sdk
```

[*4] Ocean Documentation https://docs.ocean.dwavesys.com/en/stable/
[*5] GitHub - dwavesystems/dwave-ocean-sdk https://github.com/dwavesystems/dwave-ocean-sdk
[*6] Ocean Documentation - Installing Ocean Tools https://docs.ocean.dwavesys.com/en/stable/overview/install.html

5.2 入出力処理の確認

コンテンツ

具体的な問題に取り組む前に、D-Wave システムへのデータの入出力の方法について確認します。大まかなイメージを掴むために、細かなパラメータに関する理解は先送りして進めてよいですが、**手元にプログラミングの環境を準備して、実際に動かすことを推奨** します。

本来、D-Wave システムを用いて組合せ最適化を解くためには、**組合せ最適化問題の定式化** を行った後に、**イジングモデル** への変形が必要でした。しかし、変形方法は若干専門的なテクニックも含まれるため、まずは **イジングモデルへの変形が既に済んでいて、D-Wave システムへの入力を行う直前の段階** を前提として考えます。

(5.1) の数式のようなイジングモデルを解きたいとします。

$$\underset{(x_0,x_1)\in\{-1,1\}^2}{\text{minimize}} \quad -x_0x_0 + 2x_0x_1 - x_1x_1 - x_0 \tag{5.1}$$

動作確認も兼ねて、以下のようなコードを実行し、`response` が取得できることを確認してください。

```
import dwave.system
import dimod

ENDPOINT = 'https://cloud.dwavesys.com/sapi/' # 5.1.1 で取得した ENDPOINT を記載する
SOLVER = '***' # 5.1.1 で取得した SOLVER を記載する
TOKEN = '***' # 5.1.1 で取得した TOKEN を記載する

sampler = dwave.system.composites.EmbeddingComposite(
    dwave.system.samplers.DWaveSampler(
        endpoint=ENDPOINT,
        token=TOKEN,
        solver=SOLVER
    )
)

h = {0: -1}
J = {(0, 0): -1, (0, 1): 2, (1, 1): -1}
bqm = dimod.BinaryQuadraticModel.from_ising(h, J)

response = sampler.sample(bqm, chain_strength = 3)
response
```

実は、上記のような短いコードだけで、実際に D-Wave システムによる計算の結果が得られています。以下では、このコードの各行の意味を説明します。

5.2.1 ライブラリのインポート

```
import dwave.system
import dimod
```

D-Wave Ocean SDK では、機能毎にライブラリが分割されています[*1][*2] 。

5.2.2 ユーザ情報の設定

```
ENDPOINT = 'https://cloud.dwavesys.com/sapi/' # 読者自身のものに変更する
SOLVER = '***' # 読者自身のものに変更する
TOKEN = '***' # 読者自身のものに変更する
```

D-Wave システムとユーザ間は、「**Solver API（SAPI）**」と呼ばれる REST API によって通信を行います[*3] 。これらの情報の取得方法は、5.1 章 **動作環境について** を参照してください。

D-Wave Systems が提供しているハードウェアのバージョンは複数あるため、`SOLVER` で特定しています。利用する時点において、ユーザがアクセス可能であるマシンは、D-Wave Leap のトップページから確認することができます。

5.2.3 sampler インスタンス

```
sampler = dwave.system.composites.EmbeddingComposite(
    dwave.system.samplers.DWaveSampler(
        endpoint=ENDPOINT,
        token=TOKEN,
        solver=SOLVER
    )
)
```

D-Wave システムへ SAPI 経由で通信を行うために、D-Wave Ocean SDK が設けているクラスに対して、接続情報を設定し、`sampler` インスタンスを作成しています。D-Wave システムへの疎通を行う `sampler` に対して、後段では、解きたい問題の情報を入力するという使い方をします[*4] 。

なお、通信状況や利用契約の関係上、D-Wave システムの利用ができない場合には、以下のような `sampler` を作成することで、ユーザのローカルコンピュータ上において、代替となる計算を実行することもできます[*5] 。

[*1] GitHub - dwavesystems/dwave-ocean-sdk https://github.com/dwavesystems/dwave-ocean-sdk

[*2] D-Wave システムへの通信処理を行う機能だけではなく、イジングモデルに関連する前・後処理や、データ分析に役立つ機能も提供しています。

[*3] D-Wave System Documentation - Solver API REST Web Services Guide https://docs.dwavesys.com/docs/latest/doc_rest_api.html

[*4] クラス名の意味はここでは割愛していますが、より発展的なイジングモデルの投入方法を行う際には、別のクラスを利用する必要があります。

[*5] この例では、後述する 6.4 章 の **シミュレーテッド・アニーリング** のアルゴリズムを実行します。

```
import neal

sampler = neal.SimulatedAnnealingSampler()
```

D-Wave システムに接続する sampler と、入出力の形式が近くなるように設計されています。そのため、sampler を切り替えるだけで、前後の処理はそのままで、D-Wave システム／ローカルコンピュータを変更しています。

5.2.4 Binary Quadratic Model

```
h = {0: -1}
J = {(0, 0): -1, (0, 1): 2, (1, 1): -1}
bqm = dimod.BinaryQuadraticModel.from_ising(h, J)
```

この 2 行は、D-Wave システムに与えるイジングモデルの情報を記述しています。

具体的には、解きたいイジングモデル (5.1) において、「x_0 と x_0 間の係数の値が -1 である」ことを、相互作用項を表す辞書型 J の要素として、(0, 0):-1 と表現しています。また、x_0 に対する局所磁場項 h は -1 であるために、0: -1 と表現しています。このルールに従って、数式全体の情報を h と J に記述します。なお、辞書型 J のキー (1, 0) や、h のキー 1 に対応する値はゼロであり、値を設定せずにそのまま引数へ渡しています。

dimod.BinaryQuadraticModel で作成されている bqm は、イジングモデルの情報を扱いやすくするためのインスタンスです。

5.2.5 sampler.sample メソッドと response の取得

```
response = sampler.sample(bqm, chain_strength = 3)
```

ここまでに準備してきた、sampler に対して bqm を投入します。これにより、クラウド上の D-Wave システムへ、解きたいイジングモデル (5.1) を実際に送信しています。D-Wave システムには、様々な前処理・動作パラメータがあり、chain_strength もその一つです。技術的な意味は、7.1 章 の **疎結合と全結合** で後述します。

その後、D-Wave システムからの応答を待ち、response を取得するという一連の操作が自動的に行われます。response は、D-Wave システムが出力した解の情報を保持しており、以下のような値が格納されています。

```
SampleSet(
    rec.array(
        [([ 1, -1], -5., 1, 0.)],
        dtype=[('sample', 'i1', (2,)), ('energy', '<f8'), ('num_occurrences', '<i8'),
 ('chain_break_fraction', '<f8')]
    ),
```

(次のページに続く)

(前のページからの続き)

```
    [0, 1],
    {
        'timing': {
            'qpu_sampling_time': 239,
            'qpu_anneal_time_per_sample': 20,
            'qpu_readout_time_per_sample': 198,
            'qpu_access_time': 10937,
            'qpu_access_overhead_time': 1952,
            'qpu_programming_time': 10698,
            'qpu_delay_time_per_sample': 21,
            'total_post_processing_time': 449,
            'post_processing_overhead_time': 449,
            'total_real_time': 10937,
            'run_time_chip': 239,
            'anneal_time_per_run': 20,
            'readout_time_per_run': 198
        },
    'problem_id': '***'
    },
    'SPIN'
)
```

通信にかかった時間や、その他周辺事項に関する記載がありますが、ここでは割愛します[*6][*7][*8] 。上記の `response` の 3 行目に着目してください。

```
[([ 1, -1], -5., 1, 0.)],
```

`[1, -1]` が、D-Wave システムから得られた求めたいイジングモデル (5.1) に対する解 x_0, x_1 に相当します。

$$x_0 = 1, x_1 = -1$$

ここまでの流れを通じて、ユーザは (5.1) の情報さえ送れば、実際に計算を行うアルゴリズムやコードを書かずに解を得ることができました。

まとめ

- **D-Wave Ocean SDK** を利用して、解きたいイジングモデルを簡単に D-Wave システムへ送信できる。

[*6] dimod - Samples https://docs.ocean.dwavesys.com/projects/dimod/en/latest/reference/sampleset.html

[*7] D-Wave System Documentation - QPU Timing Information from Ocean Software https://docs.dwavesys.com/docs/latest/c_timing_5.html#timing-cloud-client

[*8] D-Wave System Documentation - Breakdown of Service Time https://docs.dwavesys.com/docs/latest/c_timing_4.html

5.3 分割問題

コンテンツ

D-Wave システムを用いて、**分割問題** というシンプルな組合せ最適化問題に取り組みます。この問題を定式化したときには、イジングモデルの形式にはなっていません。D-Wave システムが扱うことができるイジングモデルへ変形するために、必要な **数式の手計算** の前処理を実際に体験した後に、それらの手計算を自動化する **PyQUBO** というライブラリを紹介します。

5.3.1 分割問題の定式化

本節では、「**分割問題（Number Partitioning）**」という組合せ最適化を扱います[*1][*2]。この問題を実際に定式化を行い、イジングモデルへ変形した後に、D-Wave システムで解いてみます。非常にシンプルな例ではありますが、ここまでの章で説明した量子アニーリング／イジングマシンの利用方法が、一通り詰まっています。

分割問題

N 個の整数 $a_1, \cdots, a_N$ を 2 つの集合に分けたい。このとき各々の集合の要素の和が互いに等しくなることができるか判定し、その答えを出力する。
例えば、$\{1, 5, 6\}$ という整数の集合に対して、$A = \{1, 5\}$ とし、$B = \{6\}$ という 2 集合に分けると、集合 A の要素の和が 6、集合 B の要素の和が 6 であり、互いに一致している。

分割問題に対する、組合せ最適化による定式化は次のように与えられます。

分割問題の組合せ最適化問題としての定式化

整数を分割する 2 集合の名前を A, B とするときに、

$$x_i = \begin{cases} -1 & (i \text{ 番目の整数が集合 } A \text{ に属する}) \\ 1 & (i \text{ 番目の整数が集合 } B \text{ に属する}) \end{cases}$$

という決定変数を $i = 1, 2, \cdots, N$ に対して定める。次のような定式化によって、分割問題の解が得られる（制約条件は存在しない）。

$$\underset{\mathbf{x} \in \{-1,1\}^N}{\text{minimize}} \ (a_1 x_1 + a_2 x_2 + \cdots + a_N x_N)^2 \tag{5.2}$$

[*1] GitHub - pyqubo / notebooks / integer_partition.ipynb https://github.com/recruit-communications/pyqubo/blob/master/notebooks/integer_partition.ipynb

[*2] Ising formulations of many NP problems https://arxiv.org/pdf/1302.5843.pdf

まずは、目的関数の意味を確認するために、$\{1,5,6\}$ という集合に対して、総当りを考えます。

$$\underset{\mathbf{x}\in\{-1,1\}^N}{\text{minimize}} \ (1x_1+5x_2+6x_3)^2$$

分割問題を総当りする場合、各要素が集合 A に属するか、集合 B に属するかを探索します。決定変数の $\{-1,1\}$ は、それぞれ集合 A,B に属することを表現しています。

表 5.1　$\{1,5,6\}$ に対する総当り

$a_1=1$	$a_2=5$	$a_3=6$	目的関数値	対応する分割	分割できているか
-1	-1	-1	144	$A=\{1,5,6\}$	NG
-1	-1	1	**0**	$A=\{1,5\},B=\{6\}$	**OK**
-1	1	-1	4	$A=\{1,6\},B=\{5\}$	NG
-1	1	1	100	$A=\{1\},B=\{5,6\}$	NG
1	-1	-1	100	$A=\{5,6\},B=\{1\}$	NG
1	-1	1	4	$A=\{5\},B=\{1,6\}$	NG
1	1	-1	**0**	$A=\{6\},B=\{1,5\}$	**OK**
1	1	1	144	$A=\{1,5,6\}$	NG

「分割できているか」の列に注目すると、目的関数値がゼロになっている行は **OK** であり、非ゼロになっている行は **NG** となっています。これは、集合 A に属する決定変数は -1 を取るため和がマイナスの項が作成され、集合 B に属する数字は数字が 1 であるためプラスの項を作成し、これらの和が釣り合うときにだけゼロになるためです。

したがって、**目的関数値がゼロとなる行を探索することができれば、所望の分割が得られる** という考え方で定式化されています。

5.3.2 数式処理の手計算と D-Wave システムによる求解

通常、組合せ最適化を定式化したときに、そのままではイジングモデルではありません。実際、**イジングモデル** と (5.2) を比較したときに、数式の形が異なることが分かります。そのため、定式化した問題をイジングモデルに合うような変形を行う必要があります。具体的には、次のような数式変形です[*3] 。

$$\begin{aligned}\underset{\mathbf{x}\in\{-1,1\}^N}{\text{minimize}} \ (1x_1+5x_2+6x_3)^2 &= 1x_1^2+25x_2^2+36x_3^2+10x_1x_2+60x_2x_3+12x_3x_1 \\ &= 10x_1x_2+60x_2x_3+12x_3x_1+62 \\ &= \mathbf{x}^\top\begin{pmatrix}0&10&12\\0&0&60\\0&0&0\end{pmatrix}\mathbf{x}+62\end{aligned} \tag{5.3}$$

イジングモデルとして記述を行うことができたため、D-Wave システムへ投入可能になりました。前の **入出力処理の確認** の例で紹介した流れに沿って、$J,\mathbf{h}$ の値を書き換えます。

[*3] 決定変数 $x_i=1,-1$ に対して、$x_i^2=1$ が常に成り立つため、定数と見なすことができます。このテクニックは 6.5 章で紹介します。

なお、値がゼロであるときには、辞書型の値を与えなくてよいです。**h** はゼロベクトルであるため、空の辞書型を設定しています。

```
import dwave.system
import dimod
import neal

ENDPOINT = '***' # 5.1.1 で取得した ENDPOINT を記載する
SOLVER = '***' # 5.1.1 で取得した SOLVER を記載する
TOKEN = '***' # 5.1.1 で取得した TOKEN を記載する

sampler = dwave.system.composites.EmbeddingComposite(
    dwave.system.samplers.DWaveSampler(
        endpoint=ENDPOINT,
        token=TOKEN,
        solver=SOLVER
    )
)

J = {(1, 2): 10, (2, 3): 60, (1, 3): 12}
h = {}
offset = 62

bqm = dimod.BinaryQuadraticModel.from_ising(h, J, offset)

response = sampler.sample(bqm, chain_strength=70)
response
```

これを実行した結果、次のような出力が得られます。

```
SampleSet(
    rec.array([([1, 1,  -1], -6., 1, 0.)],
    dtype=[('sample', 'i1', (3,)), ('energy', '<f8'), ('num_occurrences', '<i8'), (
↪'chain_break_fraction', '<f8')]), [1, 2, 3],
        {
... 略
```

`[1, 1, -1]` は答えを表していて、

$$x_1 = 1, x_2 = 1, x_3 = -1$$

であるために、決定変数の定義から、

$$A = \{6\}, B = \{1, 5\}$$

とすることが最適であるという結果が得られています。なお、このコードは一つ前に紹介した 5.2 章 **入出力処理の確認** の例と、ほぼ全く同じ構造です。

5.3.3 PyQUBO と D-Wave システムによる求解

数式処理の手計算と *D-Wave* システムによる求解 で紹介した数式の手計算での変形は、線形代数の知識がある設計者や開発者にとっては可能であるものの、そうではない方にとっては難しい場合があります。また、分割問題は極めてシンプルな例ですが、現実の複雑な問題になった場合には、専門知識がある設計者であったとしても、手計算の手間が非常に大きくなり、計算ミスの可能性も大きくなります。

そういった数式処理の計算をサポートするためのライブラリが存在します。Mathematica[*4], Sympy[*5] などの既存のソフトウェアも利用可能ですが、量子アニーリング／イジングマシンでの利用に特化したライブラリとして、**PyQUBO** というライブラリが開発されています[*6] 。

実際に、PyQUBO を利用して、分割問題を解いてみます。

```
import pyqubo

x = pyqubo.Array.create('x', shape=(3), vartype='SPIN') # 'SPIN'は決定変数{-1, 1}を表す
objective_function = (1 * x[0] + 5 * x[1] + 6 * x[2]) ** 2

model = objective_function.compile()
bqm = model.to_bqm()

print(bqm.to_ising())
```

このコードを動かしたとき、次のような出力を得ることができます。

```
({'x[0]': 0.0, 'x[1]': 0.0, 'x[2]': 0.0}, {('x[0]', 'x[1]'): 10.0, ('x[0]', 'x[2]'):␣
↪12.0, ('x[1]', 'x[2]'): 60.0}, 62.0)
```

このタプルの要素は、それぞれ (5.3) で手計算を行ったときに得られた係数 $\mathbf{h}, J$ と定数に一致しています[*7] 。PyQUBO 利用時は、`objective_function` の行に記しているように、数式の手計算を行わず、目的関数を数式の形でそのまま記述しています。

その後、`model.to_bqm()` のように、イジングモデルを表現するインスタンスである `bqm` を直接生成しています。**入出力処理の確認** で説明した手順と同様にして、`bqm` は対象とする最適化問題を記述するクラスインスタンスであるため、D-Wave システムを表す `sampler` に投入することによって解を得ることが出来ます。

その他にも、制約条件を記述する機能や、連続変数を扱う機能などが実装されており、様々な数値実験を効率的に行うことができます。引き続き、**巡回セールスマン問題** にて、PyQUBO の機能を利用していきます。

[*4] Wolfram https://www.wolfram.com/index.ja.html
[*5] Sympy https://www.sympy.org/en/index.html
[*6] PyQUBO https://pyqubo.readthedocs.io/en/latest/
[*7] `{'x[0]': 0.0, 'x[1]': 0.0, 'x[2]': 0.0}` は (5.3) の対角成分に相当します。

まとめ

- 分割問題の目的関数を、イジングモデルへ手計算で数式変形して、D-Wave システムへ投入することができる。
- **PyQUBO** と連携することによって、手計算の手間を削減することができる。

5.4 巡回セールスマン問題

コンテンツ

通常、定式化した組合せ最適化には、制約条件が含まれていたり、決定変数が $\{-1, 1\}$ ではないなど、**イジングモデルの構造と異なる要素が多数含まれています**。本節では、PyQUBO と D-Wave Ocean SDK を使って、巡回セールスマン問題を解いてみます。後の章にて説明する予定のテクニックも先行して含まれていますが、ソフトウェアが自動的に処理してくれる部分も多いため、ユーザはそれらのテクニックをある程度意識せずに、D-Wave システムを利用することができます。

5.4.1 巡回セールスマン問題の定式化

この節では、組合せ最適化のイメージ で説明した巡回セールスマン問題に対して、D-Wave システムを利用する方法を記載しています。ただし、ここまでの説明では、巡回セールスマン問題の定義しか説明していなかったため、まずは以下のように定式化を行います。

巡回セールスマン問題の定式化

N 個の支店があるときに、支店 i と支店 j の間の距離を、$d_{i,j}$ $(i = 1, \cdots, N)$ とする。決定変数を次のように定める。

$$x_{i,t} = \begin{cases} 1 & (\text{支店 } i \text{ へ } t \text{ 番目に訪れる}) \\ 0 & (\text{支店 } i \text{ に } t \text{ 番目に訪れない}) \end{cases}$$

このとき、組合せ最適化のイメージ で説明した巡回セールスマン問題は、次のような組合せ最適化として定式化できる[1] 。

$$\begin{aligned} \underset{\mathbf{x}\in\{0,1\}^{N\times N}}{\text{minimize}} \quad & \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{t=1}^{N} d_{i,j}x_{i,t}x_{j,(t+1)\%N} \\ \text{subject to} \quad & \sum_{i=1}^{N} x_{i,t} = 1,\ t = 1, \ldots, N, \\ & \sum_{t=1}^{N} x_{i,t} = 1,\ i = 1, \ldots, N. \end{aligned}$$

[1] $(t+1)\%N$ は、$t+1$ を N で割ったときの剰余を表す。

ナップサック問題 や 分割問題 と違って、定式化された目的関数と制約条件の意味が難しいため、順に説明します。定式化に不慣れな方にとっては複雑な内容であるため、目的関数と制約条件の意味を理解をせずに、一旦斜め読みをしてソースコード例に飛んでも、後続の説明上問題ありません。

この巡回セールスマン問題の定式化では、

- 決定変数が 01 である。
- 制約条件が存在する。

といった点から、イジングモデルと異なる点が多く、変換するためには数学的な前処理・変形が必要になります。

5.4.1.1 目的関数

$$\underset{\mathbf{x}\in\{0,1\}^{N\times N}}{\text{minimize}} \quad \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{t=1}^{N} d_{i,j}x_{i,t}x_{j,(t+1)\%N}$$

目的関数の $\sum$ の中の、$x_{i,t}x_{j,(t+1)\%N}$ の部分が重要です[*2]。「支店 i に t 番目に訪れる かつ 支店 j に $t+1$ 番目に訪れる」場合に、$x_{i,t}x_{j,(t+1)\%N}$ が 1 となります。それ以外の場合には 0 となります。総和記号 $\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{t=1}^{N}$ は、「支店 i に t 番目に訪れる かつ 支店 j に $t+1$ 番目に訪れる」場合に、支店 i と j の間の距離 $d_{i,j}$ を足し合わせる、という操作を、セールスマンが巡回しうる支店・順番全体に対して足し合わせていくことを意味します。回りくどい言い方となりましたが、これはセールスマンが通る経路の総距離と対応づいています。

5.4.1.2 制約条件

$$\begin{aligned}\text{subject to} \quad & \sum_{i=1}^{N} x_{i,t} = 1,\ t = 1,\dots,N,\\ & \sum_{t=1}^{N} x_{i,t} = 1,\ i = 1,\dots,N.\end{aligned}$$

制約条件は、巡回セールスマン問題において、「一筆書き」でなければならないことを表現しています。$t=1$ 番目に訪れる支店が、$i=1,\cdots,N$ のうちのどこか一つに存在しなければなりません。これは、$\sum_{i=1}^{N} x_{i,t}=1, t=1$ の制約に対応づいています。$x_{i,t}$ の定義を考えると、$t=1$ 番目において、どこか一箇所だけ支店 i に対応するビットを 1 とし、その他を 0 とします[*3]。同様にして、支店 $i=1$ に、$t=1,\cdots,N$ のうちなの何番目かには訪問しなければなりません。これは $\sum_{t=1}^{N} x_{i,t}=1, i=1$ で表現されています。全部の順序・全部の支店に対して、これらの制約条件を同時に課すことによって、「一筆書き」であることが担保されます。

[*2] $(t+1)\%N$ は、$N+1$ 番目都市を 1 番目と見なすための境界を処理している剰余であり、本質的な意味は無いため、式を理解するためには一旦無視してよいです。

[*3] このように、どれか一つを選択する操作を、多次元の決定変数の中から 1 ビットを選択する操作に置き換えた制約を「**One-Hot 制約（One-Hot Constraint）**」と呼ぶことがあります。

5.4.2 PyQUBO と D-Wave システムによる求解

巡回セールスマン問題をイジングモデルへ変形する処理の中には、まだ説明していないテクニックなども含まれています。しかし、**分割問題** と同じように、PyQUBO がそれらをある程度自動的に処理してくれるため、ユーザは高度なテクニック・数式変形を意識せずに D-Wave システムを利用できます[*4] 。

そのため、ここまでに説明していない内容を含む、巡回セールスマン問題に敢えて挑戦します。説明されていない事項で気になった場合は、後続の章で説明するため、読み飛ばしながら進めることを推奨します。なお、この例は、PyQUBO で管理されている利用例から引用しています[*5] 。

```
import dwave.system
from pyqubo import Array, Placeholder, Constraint
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np

def plot_city(cities, sol=None):
    n_city = len(cities)
    cities_dict = dict(cities)
    G = nx.Graph()
    for city in cities_dict:
        G.add_node(city)

    # draw path
    if sol:
        city_order = []
        for i in range(n_city):
            for j in range(n_city):
                if sol.array('c', (i, j)) == 1:
                    city_order.append(j)
        for i in range(n_city):
            city_index1 = city_order[i]
            city_index2 = city_order[(i+1) % n_city]
            G.add_edge(cities[city_index1][0], cities[city_index2][0])

    plt.figure(figsize=(3,3))
    pos = nx.spring_layout(G)
    nx.draw_networkx(G, cities_dict)
    plt.axis("off")
    plt.show()

def dist(i, j, cities):
    pos_i = cities[i][1]
    pos_j = cities[j][1]
    return np.sqrt((pos_i[0] - pos_j[0])**2 + (pos_i[1] - pos_j[1])**2)
```

(次のページに続く)

[*4] GitHub - recruit-communications/pyqubo - https://github.com/recruit-communications/pyqubo/blob/master/notebooks/TSP.ipynb
[*5] GitHub - pyqubo / notebooks https://github.com/recruit-communications/pyqubo/tree/master/notebooks

(前のページからの続き)

```
cities = [
    ("a", (0, 0)),
    ("b", (1, 3)),
    ("c", (3, 2)),
    ("d", (2, 1)),
    ("e", (0, 1))
]

n_city = len(cities)

# 決定変数
x = Array.create('c', (n_city, n_city), 'BINARY')

# 制約条件
time_const = 0.0
for i in range(n_city):
    time_const += Constraint((sum(x[i, j] for j in range(n_city)) - 1)**2, label=
↪"time{}".format(i))

city_const = 0.0
for j in range(n_city):
    city_const += Constraint((sum(x[i, j] for i in range(n_city)) - 1)**2, label=
↪"city{}".format(j))

# 目的関数
distance = 0.0
for i in range(n_city):
    for j in range(n_city):
        for k in range(n_city):
            d_ij = dist(i, j, cities)
            distance += d_ij * x[k, i] * x[(k+1)%n_city, j]

# ハミルトニアン（イジングモデル）の記述
A = Placeholder("A")
H = distance + A * (time_const + city_const)

# コンパイル
model = H.compile()

# QUBO（イジングモデル）の作成
feed_dict = {'A': 4.0}
bqm = model.to_bqm(feed_dict=feed_dict)
bqm.normalize()

# イジングモデルの計算
ENDPOINT = 'https://cloud.dwavesys.com/sapi/' # 5.1.1 で取得した ENDPOINT を記載する
SOLVER = '***' # 5.1.1 で取得した SOLVER を記載する
```

(次のページに続く)

(前のページからの続き)

```
TOKEN = '***' # 5.1.1 で取得した TOKEN を記載する

sampler = dwave.system.composites.EmbeddingComposite(
    dwave.system.samplers.DWaveSampler(
        endpoint=ENDPOINT,
        token=TOKEN,
        solver=SOLVER
    )
)
sampleset = sampler.sample(bqm, num_reads=100, chain_strength=1.2)

decoded_samples = model.decode_sampleset(sampleset, feed_dict=feed_dict)
best_sample = min(decoded_samples, key=lambda x: x.energy)
num_broken = len(best_sample.constraints(only_broken=True))
print("number of broken constarint = {}".format(num_broken))

if num_broken == 0:
    plot_city(cities, best_sample)
```

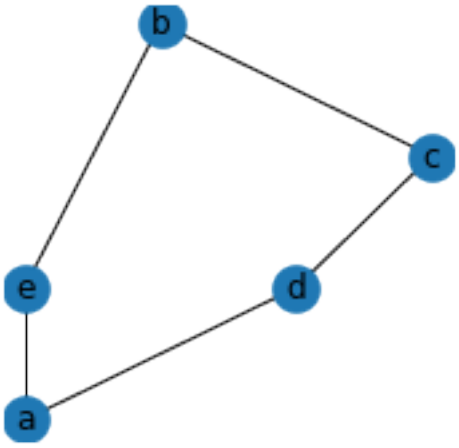


図 5.1　実行した結果得られる巡回セールスマン問題の解

最初に、巡回セールスマン問題に必要な、支店の位置情報の定義や距離の計算を行っています。通常のデータサイエンスで用いるようなライブラリも、よく活用されます。

```
import dwave.system
from pyqubo import Array, Placeholder, Constraint
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np

def plot_city(cities, sol=None):
    n_city = len(cities)
    cities_dict = dict(cities)
```

(次のページに続く)

(前のページからの続き)

```
    G = nx.Graph()
    for city in cities_dict:
        G.add_node(city)

    # draw path
    if sol:
        city_order = []
        for i in range(n_city):
            for j in range(n_city):
                if sol.array('c', (i, j)) == 1:
                    city_order.append(j)
        for i in range(n_city):
            city_index1 = city_order[i]
            city_index2 = city_order[(i+1) % n_city]
            G.add_edge(cities[city_index1][0], cities[city_index2][0])

    plt.figure(figsize=(3,3))
    pos = nx.spring_layout(G)
    nx.draw_networkx(G, cities_dict)
    plt.axis("off")
    plt.show()

def dist(i, j, cities):
    pos_i = cities[i][1]
    pos_j = cities[j][1]
    return np.sqrt((pos_i[0] - pos_j[0])**2 + (pos_i[1] - pos_j[1])**2)

cities = [
    ("a", (0, 0)),
    ("b", (1, 3)),
    ("c", (3, 2)),
    ("d", (2, 1)),
    ("e", (0, 1))
]

n_city = len(cities)
```

続いて、決定変数を定義しています。$x_{i,j}$ は 2 次元配列的な構造を持つ変数ですが、PyQUBO の文法に従って n_city × n_city サイズで作成することができます。このとき注意したい点は、BINARY として定義していることです。PyQUBO では、$\{-1, 1\}$ が SPIN、$\{0, 1\}$ が BINARY と定義されています。決定変数が取りうる値が違うと、定式化が正しく PyQUBO に渡されていないため、異なる組合せ最適化を解くことになってしまいます。

```
x = Array.create('c', (n_city, n_city), 'BINARY')
```

ここから先は、決定変数 x を用いて、制約条件と目的関数を表現していきます。制約条件の表現方法は、後に紹介する **ペナルティ関数** のテクニックとも関係します。とはいえ、sum と Σ、for i in range(n_city)

と　$i = 1, \cdots, N$ を対応付ければ、ほぼ全く同じように式をそのまま記述していることに注目してください。

```
# 制約条件
time_const = 0.0
for i in range(n_city):
    time_const += Constraint((sum(x[i, j] for j in range(n_city)) - 1)**2, label=
↪"time{}".format(i))

city_const = 0.0
for j in range(n_city):
    city_const += Constraint((sum(x[i, j] for i in range(n_city)) - 1)**2, label=
↪"city{}".format(j))
```

次のコードは、目的関数の記述を行っています。

```
# 目的関数
distance = 0.0
for i in range(n_city):
    for j in range(n_city):
        for k in range(n_city):
            d_ij = dist(i, j, cities)
            distance += d_ij * x[k, i] * x[(k+1)%n_city, j]
```

最後に、H の行のように、目的関数と制約条件をセットにして、組合せ最適化の定式化に必要な情報をひとまとめにしています。Placeholder と feed_dict は後の章で説明するため、ここでは読み飛ばして良いです。

```
# ハミルトニアン（イジングモデル）の記述
 A = Placeholder("A")
 H = distance + A * (time_const + city_const)

 # コンパイル
 model = H.compile()

 # QUBO（イジングモデル）の作成
 feed_dict = {'A': 4.0}
 bqm = model.to_bqm(feed_dict=feed_dict)
```

以上のように、定式化した数式を数式として、PyQUBO に与えることで、イジングモデル表記の bqm を得ることができました。**入出力処理の確認** の説明にしたがって、D-Wave システムにより、イジングモデルとしての結果を得ることができます。最後に、決定変数 $x_{i,j}$ の定義に従って得られた 01 値を、セールスマンの訪問順序として対応付けた後に、別途準備していた可視化用の関数 plot_city に渡し、図を出力しています。

```
decoded_samples = model.decode_sampleset(sampleset, feed_dict=feed_dict)
best_sample = min(decoded_samples, key=lambda x: x.energy)
num_broken = len(best_sample.constraints(only_broken=True))
print("number of broken constarint = {}".format(num_broken))
```

(次のページに続く)

(前のページからの続き)

```
if num_broken == 0:
    plot_city(cities, best_sample)
```

ここから後の章では、より発展的な使い方と理解のために、PyQUBO や D-Wave Ocean SDK の内部で行われている処理を説明します。

入出力処理の確認、**分割問題**、**巡回セールスマン問題** で説明したように、ユーザの観点では数式処理やテクニックを深く把握しなくても扱える問題は多くあります。細かな技術の把握が難しい場合には、対応する自動化機能が存在する場合があることだけでも念頭に置くとよいと思われます。

まとめ

- 巡回セールスマン問題のように、イジングモデルと構造が大きく異なる問題が存在する。
- PyQUBO と D-Wave Ocean SDK を利用すれば、数式処理をある程度自動化できる。

6 組合せ最適化の関連知識

この章の流れ

量子アニーリング／イジングマシンを用いて開発を行う場合に、組合せ最適化の構造にあわせて制約条件を扱ったり、パラメータをチューニングしたりする必要があります。より本格的に開発を始めていく上で、必要となる組合せ最適化の知識を記載しています。

- 6.1 章 *QUBO* とイジングモデル
- 6.2 章 ペナルティ関数
- 6.3 章 大域的最適解と局所的最適解
- 6.4 章 シミュレーテッド・アニーリング
- 6.5 章 組合せ最適化問題の前処理

ここまでの章と異なり、技術的に重要な概念が、やや独立しながら別のトピックとして説明しています。そのため、読む順序に関しては、読者の興味に応じてある程度変更しても大丈夫なように構成されていますが、いずれも頻出するトピックであるため、実際にシステム開発を行う設計者・開発者にとってはどれも知っておきたい内容です。

6.1 QUBO とイジングモデル

コンテンツ

量子アニーリング／イジングマシンを利用する上で、QUBO という組合せ最適化問題が頻繁に現れます。問題の定義を確認して、イジングモデルと似たような立ち位置であることを説明します。

量子アニーリング／イジングマシンに関する説明の中に、しばしば「**Quadratic Unconstrained Binary Optimization (QUBO)**」という単語が記載されています。QUBO は組合せ最適化問題の一つであり、イジングモデルと非常に類似した構造を持っています。

組合せ最適化を QUBO に変換することができれば、イジングモデルと QUBO の間で、簡単に相互変換することができます。そのため、多くの量子アニーリング／イジングマシンでは、イジングモデルだけではなく QUBO での入力も受理するインタフェースが備わっています[*1] 。

数学的には異なる問題ではありますが、ハードウェアのユーザとしての観点から言うと、イジングモデルと QUBO は、ほぼ同じものであると理解してよいと考えられます。

Quadratic Unconstrained Binary Optimization (QUBO) の定義

次の目的関数で書き表されている組合せ最適化問題を、「**Quadratic Unconstrained Binary Optimization（QUBO）**」と呼ぶ。

$$\underset{(x_1,x_2,\cdots,x_N)\in\{0,1\}^N}{\text{minimize}} \quad \sum_i \sum_j J_{i,j} x_i x_j + \sum_i h_i x_i + \text{const.} = \mathbf{x}^\top J \mathbf{x} + \mathbf{h}^\top \mathbf{x} + \text{const.} \tag{6.1}$$

イジングモデル と同じように目的関数の特徴を記載すると、以下の 3 点が挙げられます。

- 決定変数が 0 か 1 である。
- 目的関数の次数が 2 次までの多項式である。
- 制約条件は明示的に存在しない。

イジングモデルでは決定変数が $\{-1, 1\}$ でしたが、QUBO では $\{0, 1\}$ になっています。その他に変更点はありません。

QUBO は、次のような変数変換を行うことによって、イジングモデルに帰着することができます。D-Wave Ocean SDK などのソフトウェア上にも、相互変換を行うための機能が備わっているため、ユーザが直接実装する必要はありませんが、参考までに記載しています。

[*1] ハードウェアの内部のアルゴリズムが、イジングモデルではなく QUBO を解いていることがあります。この場合は、イジングモデルを QUBO へ変換した後に求解します。このように QUBO を主とするハードウェアは、「**QUBO ソルバ（QUBO Solver）**」と呼ぶことがあります。

イジングモデルと QUBO の相互変換

$\mathbf{y} \in \{-1, 1\}^N$ で定義されているイジングモデルの目的関数 $\mathbf{y}^\top J\mathbf{y} + \mathbf{h}^\top \mathbf{y} + \text{const.}$ に対して、$\mathbf{x} = \frac{\mathbf{y}+\mathbf{1}}{2}$ という変数変換を考える。ただし、ここで $\mathbf{0}$ は、N 次元であり、それぞれの要素が全て 0 だけからなるベクトルを表している（$\mathbf{1}$ も同様）。

$\mathbf{y}$	$-\mathbf{1} \to \mathbf{1}$
$\mathbf{x}$	$\mathbf{0} \to \mathbf{1}$

$\mathbf{x} = \frac{\mathbf{y}+\mathbf{1}}{2} \iff \mathbf{y} = 2\mathbf{x} - \mathbf{1}$ であり、イジングモデルの目的関数に対して代入を行うと、

$$\begin{aligned}
\mathbf{y}^\top J\mathbf{y} + \mathbf{h}^\top \mathbf{y} + \text{const.} &= (2\mathbf{x} - \mathbf{1})^\top J(2\mathbf{x} - \mathbf{1}) + \mathbf{h}^\top(2\mathbf{x} - \mathbf{1}) + \text{const.} \\
&= 4\mathbf{x}^\top J\mathbf{x} - 4\mathbf{1}^\top(J + J^\top)\mathbf{x} + \mathbf{1}^\top\mathbf{1} + 2\mathbf{h}^\top\mathbf{x} - \mathbf{h}^\top\mathbf{1} + \text{const.} \\
&= 4\mathbf{x}^\top J\mathbf{x} - (4\mathbf{1}^\top J + 4\mathbf{1}^\top J^\top - 2\mathbf{h}^\top)\mathbf{x} + \text{const.}
\end{aligned}$$

となる。このとき、 $\mathbf{x} \in \{0, 1\}^N$ であるため、この問題は QUBO である。
同様に、 $\mathbf{y} = 2\mathbf{x} - \mathbf{1}$ という変換を行えば、QUBO をイジングモデルへ変形することができる。

まとめ

- QUBO はイジングモデルと容易に相互変換でき、互いにほぼ同じような以下の 3 つの特徴を持っている。
- 決定変数が 0 か 1 である。
- 目的関数の次数が 2 次までの多項式である。
- 制約条件は明示的に存在しない。

6.2 ペナルティ関数

コンテンツ

量子アニーリング／イジングマシンは、無制約な組合せ最適化であるイジングモデルのみを対象としているため、制約条件を明示的に入力として受け付けることはできないですが、ペナルティ関数という処理を利用することで制約条件がまるで無くなったかのように見なすことができます。
この節では、ペナルティ関数の考え方の説明の後に、巡回セールスマン問題 における制約条件の扱い方を、改めて説明します。

6.2.1 制約条件付き組合せ最適化問題の探索方法

次のような、シンプルな制約条件付き組合せ最適化問題を考えましょう。

$$\begin{aligned} \underset{\mathbf{x}\in\{-1,1\}^3}{\text{minimize}} \quad & x_1 \\ \text{subject to} \quad & x_1 + x_2 + x_3 = 1 \end{aligned} \tag{6.2}$$

この組合せ最適化問題の答えを全探索した場合には、以下の表のような結果が得られます。

表 6.1　目的関数値の総当り

x_1	x_2	x_3	制約条件	目的関数値	最適性
-1	-1	-1	NG	**-1**	制約条件違反
-1	-1	1	NG	**-1**	制約条件違反
-1	1	-1	NG	**-1**	制約条件違反
-1	1	1	**OK**	**-1**	**最適解**
1	-1	-1	NG	1	制約条件違反
1	-1	1	**OK**	1	最適ではない
1	1	-1	**OK**	1	最適ではない
1	1	1	NG	1	制約条件違反

制約条件付きの組合せ最適化問題の場合には、制約条件を守る（制約条件列が OK と書かれている）範囲内で、目的関数値の最小値を探す必要があるため、$x_1 = -1, x_2 = 1, x_3 = 1$ が最適解であることが分かります。この場合、制約条件と目的関数値の、両方の列を考慮しながら探索する必要があります。しかし、量子アニーリング／イジングマシンの場合、制約条件を考慮しながら探索する機構が無いため、そのままでは制約条件を満たすことはできません。

実は、本来は制約条件と目的関数の両方を考慮しながら行わなければならない探索を、目的関数だけに着目すれば十分であるかのように変形する手法があります。このときに用いられる表現方法を「**ペナルティ関数（Penalty Function）**」といいます。

ペナルティ関数のアイデアは、「探索対象の解が制約条件を破った場合に、目的関数値を非常に悪化させる」というものです。制約条件を破っている解は、目的関数のスコアが悪いために、自然と解としてされることが少なくなります。

6.2.2 ペナルティ関数の考え方

本節では、制約条件付き組合せ最適化問題 (6.2) を例に、ペナルティ関数の考え方について記載します。

制約条件付き組合せ最適化問題 (6.2) に対して、以下の (6.3) に示す新しい目的関数を設けます。目的関数の意味と導出方法は後述しますが、元々の目的関数 x_1 に対して、制約条件の情報を反映した項 $\lambda\left(x_1+x_2+x_3-1\right)^2$ が付与されています。

$$\underset{\mathbf{x}\in\{-1,1\}^3}{\text{minimize}} \quad x_1+\lambda\left(x_1+x_2+x_3-1\right)^2 \tag{6.3}$$

ここで、 λ は十分に大きな正の数を設定します。例えば、 $\lambda=100$ としましょう。

表 6.2　(6.3) の総当り

x_1	x_2	x_3	目的関数値	最適性	（(6.2) の制約条件）
-1	-1	-1	1599	最適ではない	NG
-1	-1	1	399	最適ではない	NG
-1	1	-1	399	最適ではない	NG
-1	1	1	-1	**最適解**	**OK**
1	-1	-1	401	最適ではない	NG
1	-1	1	1	最適ではない	OK
1	1	-1	1	最適ではない	OK
1	1	1	401	最適ではない	NG

(6.3) は制約条件無し組合せ最適化問題であるため、目的関数値の列にだけ基づいて、最適解の探索を行います。実は、この制約条件無し最適化問題 (6.3) で得られた最適解は、元々の制約条件付き最適化問題 (6.2) に対して得られた最適解 $x_1=-1, x_2=1, x_3=1$ と一致します。

(6.3) の目的関数に付与した $\left(x_1+x_2+x_3-1\right)^2$ がペナルティ関数です。ペナルティ関数は、「目的関数値」列に制約条件の情報をひとまとめにすることにより、「制約条件」列を考慮しなくて済むような処理を行っています。

ペナルティ関数はその名前のように、制約条件を破った場合に目的関数にペナルティ（悪化させる力）を課すことで、できるだけ制約条件は守って最適化するように働きかける役割を持ちます。その力の大きさを表すパラメータのことを「**ペナルティパラメータ（Penalty Parameter）**」といいます。本節の例でいえば、$\lambda=100$ がペナルティパラメータです。

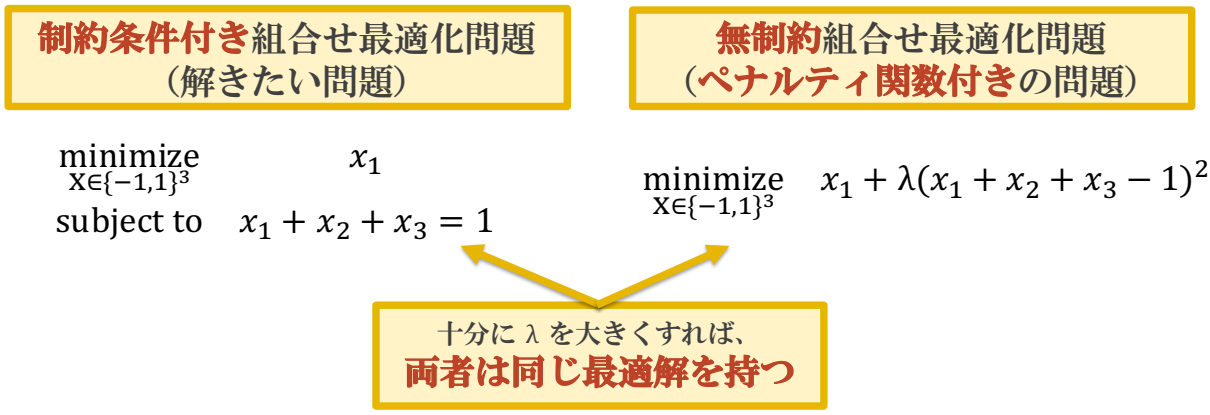


図 6.1　ペナルティ関数を利用して、制約条件付き組合せ最適化問題を無制約に変換できる。

6.2.3 ペナルティ関数の構成方法

上の節では、ペナルティ関数の考え方は紹介したものの、具体的な構成方法について説明を飛ばしていました。以下では等式制約に対する、「**二乗ペナルティ関数**」という最もシンプルな構成方法を紹介します。

等式制約に対するペナルティ関数の一例

$f(\mathbf{x})$ の最小化問題に対して、$g_i(\mathbf{x}) = \alpha_i,\ (i = 1, 2, \cdots, P)$ という制約条件が付与されているとき、ペナルティ関数付きの無制約最適化問題として、次の目的関数を利用することができる。ペナルティパラメータ $\lambda > 0$ は十分に大きい正の数である。

$$f(\mathbf{x}) + \lambda \sum_{i=1}^{P} (g_i(\mathbf{x}) - \alpha_i)^2$$

$(g_i(\mathbf{x}) - \alpha_i)^2$ は、$g_i(\mathbf{x}) = \alpha_i$ であるときにのみゼロとなり、$g_i(\mathbf{x}) \neq \alpha_i$ であるときには正の値を取ります。ここで、目的関数は最小化を目指しているため、$(g_i(\mathbf{x}) - \alpha_i)^2$ はゼロになろうとします。つまり、$g_i(\mathbf{x}) = \alpha_i$ になるような力を、解 $\mathbf{x}$ に対して加えていると解釈することができます。

前の節で、制約条件付き組合せ最適化問題 (6.2) に対して、無制約（ペナルティ関数付き）組合せ最適化問題 (6.3) を導出した際にも、上記の二乗ペナルティ関数を利用して導出しています。ただし、不等式制約など、他の形式の制約条件の場合には、そのまま利用することができないことがあります。また、$g_i(\mathbf{x})$ が 1 次以上の次数を持っていると、二乗ペナルティ関数は 2 次以上の次数となり、イジングモデルが 2 次の関数である条件を超えてしまいます。その他の発展的なペナルティ関数の構成方法や工夫も存在するため、必ずしもそのまま導入しても解決できるとは限らず、**問題に応じて様々なテクニックが必要であることに注意** してください。

また、数式上では、$\lambda > 0$ はある一定の大きさを超えるような値であれば良いですが、実際にハードウェアに投入する場合には、闇雲に大きくすると制約条件を満たさないような解が出力されることがあります。そのため、ペナルティ関数の作り方とパラメータのチューニングは、量子アニーリング／イジングマシンの性能にとって、非常に重要なものです。

6.2.4 巡回セールスマン問題におけるペナルティ関数

ここでは、巡回セールスマン問題を例に、本節で説明したペナルティ関数を導入する方法を説明します。5.4 章で示したように、**巡回セールスマン問題** は次の数式で表現することができます。

巡回セールスマン問題の定式化（再掲）

N 個の支店があるときに、支店 i と支店 j の間の距離を、$d_{i,j}$ $(i = 1, \cdots, N)$ とする。決定変数を次のように定める。

$$x_{i,t} = \begin{cases} 1 & (\text{支店 } i \text{ へ } t \text{ 番目に訪れる}) \\ 0 & (\text{支店 } i \text{ に } t \text{ 番目に訪れない}) \end{cases}$$

このとき、**組合せ最適化のイメージ** で説明した巡回セールスマン問題は、次のような組合せ最適化として定式化できる[1] 。

$$\begin{aligned} \underset{\mathbf{x}\in\{0,1\}^{N\times N}}{\text{minimize}} \quad & \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{t=1}^{N} d_{i,j}x_{i,t}x_{j,(t+1)\%N} \\ \text{subject to} \quad & \sum_{i=1}^{N} x_{i,t} = 1,\ t = 1, \ldots, N, \\ & \sum_{t=1}^{N} x_{i,t} = 1,\ i = 1, \ldots, N. \end{aligned}$$

[1] $(t+1)\%N$ は、$t+1$ を N で割ったときの剰余を表す。

まずは、イジングモデルに変形するため、制約条件をペナルティ関数として目的関数に入れ込む操作を行います。1 次までの項からなる等式制約であるために、**二乗ペナルティ関数** をそのまま利用することができます。

巡回セールスマン問題の二乗ペナルティ関数

$$\underset{\mathbf{x}\in\{0,1\}^{N\times N}}{\text{minimize}} \quad \sum_{i}\sum_{j}\sum_{t} d_{i,j}x_{i,t}x_{j,t+1} + \lambda\left\{\sum_{t}\left(\sum_{i}x_{it}-1\right)^2 + \sum_{i}\left(\sum_{t}x_{it}-1\right)^2\right\} \tag{6.4}$$

ここで、 $\lambda > 0$ は十分に大きい正のペナルティパラメータである。

この問題は、*QUBO とイジングモデル* に記載している以下の条件を満たすため、QUBO であると確認できます。

- 決定変数が 0 か 1 である。
- 目的関数の次数が 2 次までの多項式である。
- 制約条件は明示的に存在しない。

なお、二乗ペナルティ関数付きの目的関数 (6.4) から、QUBO の $\mathbf{x}^\top J\mathbf{x} + \mathbf{h}^\top \mathbf{x}$ に該当する $J, \mathbf{h}$ を算出する数式変形を手計算で行うと手間が多くかかりますが、**巡回セールスマン問題** で既に説明したように、PyQUBO を利用することで自動化することができます。

まとめ

- 制約条件をペナルティ関数として目的関数に入れ込めば、無制約な組合せ最適化と見なすことができる。
- ペナルティ関数のシンプルな構成方法の一つとして二乗ペナルティ関数が存在する。ただし、採用するペナルティ関数が性能を左右することは多いため、他の構成方法も検討する。

6.3 大域的最適解と局所的最適解

コンテンツ

組合せ最適化のアルゴリズムを構成する上で、気をつけなければならない **多峰性** や **局所的最適解** という性質を説明します。これらの性質は、後の節で説明するシミュレーテッド・アニーリングというアルゴリズムや、量子アニーリング／イジングマシンの計算原理を考えていく上で、念頭に置いておくべき性質です。

ここまでの説明では、組合せ最適化の探索の際に、全探索・総当りを行う前提での説明をしてきました。しかし、実際に組合せ最適化を解く際には、全探索・総当りといった方法を取ることはまれです。3.4 章 の **既存の組合せ最適化のアプローチ** で述べたように、求められる要件に応じて、最適化ソルバーやパッケージを用いたり、問題固有の効率的な計算方法を開発するといった例が存在します。

それらの内部で動作している具体的なアルゴリズムは膨大な数存在するため、数学・アルゴリズム・データ構造等に関する様々な参考書を参照する必要があります。ここでは、量子アニーリング／イジングマシンの対象とする、**離散的で計算が難しい問題のクラス** に対して、**ヒューリスティック** なアルゴリズムを構成する際に重要となる概念だけに、ポイントを絞って説明しています[*1] 。

後続の節で、 **シミュレーテッド・アニーリング** というアルゴリズムを紹介しますが、具体的なアルゴリズムを考える前に、組合せ最適化問題自体の重要な特徴について把握する必要があります。この節では、組合せ最適化問題の **多峰性** や **局所的最適解** といった概念を説明します。

6.3.1 多峰性

組合せ最適化問題は、多数の解の候補の中から、目的関数が最大もしくは最小なものを選択する問題でした。多数ある解全体をグラフで表すことは本来できないですが、無理やりイメージしてみると、次のような 図 6.2 になります[*2] 。

理想的な最小／最大の解は、解の候補が無数にある場合には容易に見つけることができません。また、仮にとりあえず任意に解 x を決め打ちしただけ（「**初期解（Initial Values）**」 という）だと、最適解はほぼ得られないと考えられます。このままでは手がかりがないので、初期解を改善していくという方針を採ります。

最も単純な方法は、解の一部を少し変更して、隣接する解に移動させ、改善する方向へと導いていくアプローチです。目的関数がなす山や谷を登り降りするために、「**山登り法（Hill Climbing）**」と呼ばれます。

[*1] 最適化問題の対象や用途によっては、後述の多峰性が見られなかったり、離散的な解の探索ではなく微積分学による数学的解析を行う方針を採るなど、前提が大きく異なるため注意が必要です。

[*2] 本来、グラフを書くためには、解に対応する目的関数値を評価する必要があります。組合せ爆発するようなシチュエーションでは、グラフを書いて最適解を見るという操作はできません。

[*3] 通常は、ある解 x に対する目的関数値だけが点として取得可能であり、図のように解全体に渡ってグラフを見渡せる状態でないことに注意してください。

図 6.2　暫定的に選んだ初期解・求めたい未知の解と目的関数値の全体イメージ[*3]

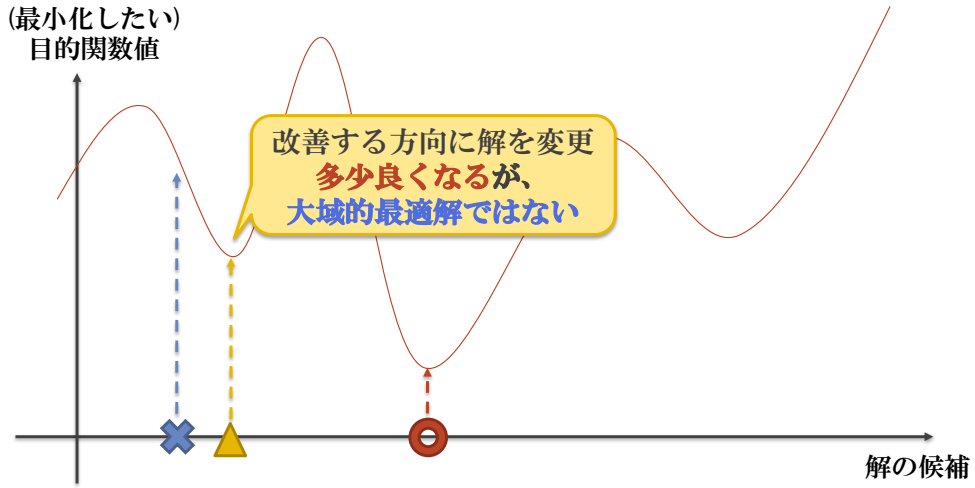


図 6.3　山登り法を利用した解の改善

初期解を改善してくれるメリットがあるものの、単純に山を降っただけでは、小さな谷の底で計算が止まり、目的関数全体の最適解に行き着くことができません[*4]。組合せ最適化問題において、複数の山や谷があることを「**多峰性**」といい、一つ一つの谷を「**局所的最適解（Local Optimal Solution）**」、最も深い谷を「**大域的最適解（Global Optimal Solution）**」といいます。（メタ）ヒューリステックを考える上で、**局所的最適解を抜け出し、大域的最適解に辿り着く** ことを考慮することが重要です。

初期解を定めて、近傍に移りながら最適解を探索する手法全般のことを「**局所探索法（Local Search）**」といいます。山登り法や、次の節で説明する シミュレーテッド・アニーリング は、局所探索法の一つです。局所探索法を設計する際には、局所的最適解をうまく抜け出して、大域的最適解に収束できるように設計する考え方が重要です。

[*4] 山登り法で最適解が得られる問題も存在しますが、ここではメタヒューリスティックが対象とする計算の難しい組合せ最適化問題に対してのみ言及しています。

まとめ

- 組合せ最適化問題の目的関数に「**多峰性**」がある場合には、多数の「**局所的最適解**」が発生する。
- 「**山登り法**」のようなシンプルな方法の場合、「**大域的最適解**」を見つけられない場合がある。
- 「**局所探索法**」を設計する際には局所的最適解を回避できるよう考慮する必要がある。

6.4 シミュレーテッド・アニーリング

コンテンツ

頻繁に量子アニーリングと比較される、古典コンピュータ上でのアルゴリズムとして、**シミュレーテッド・アニーリング** が有名です。
量子アニーリングは、シミュレーテッド・アニーリングのアイデアと深く関連するアルゴリズムであるため、量子アニーリングの原理を理解する上でも重要なトピックを含んでいます。

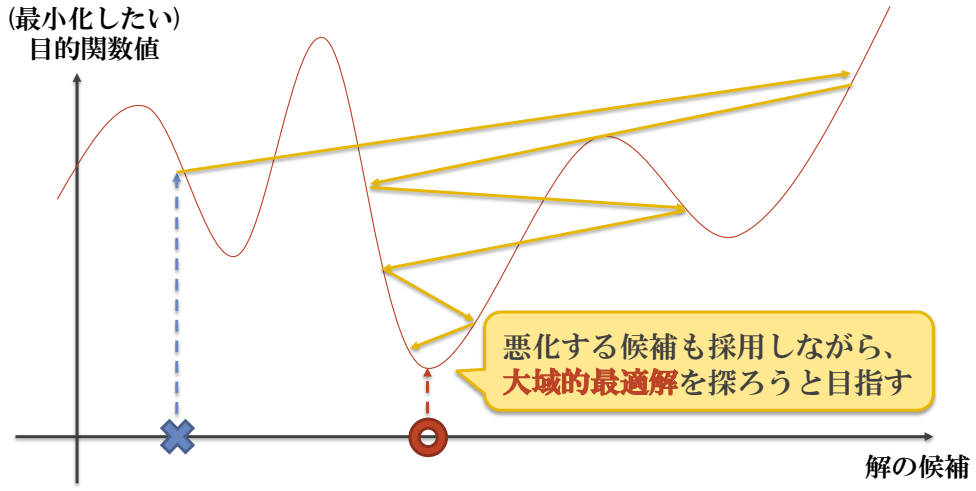
4.1 章の **ハードウェアの役割と特徴** にて、量子アニーリング／イジングマシンが、メタヒューリスティックと呼ばれるアルゴリズムに位置づけられることを説明しました。ヒューリスティックなアルゴリズムは、何らかの理由で求解が難しいクラスに属する組合せ最適化に対して、最適解を保証することは難しくても、できるだけ良い解を見つけるアルゴリズムを指していました。「**シミュレーテッド・アニーリング（焼き鈍し法, Simulated Annealing, SA）**」も、古くからあるメタヒューリスティックの一つです[*1]。

シミュレーテッド・アニーリングは、量子アニーリングのもととなったアルゴリズムであるため、考え方に類似点も多く、一緒に知っておくとよいアルゴリズムです。量子アニーリングと比較する対象の既存手法として頻繁に言及されています[*2]。また、イジングマシンの GPU や FPGA 上には、シミュレーテッド・アニーリングをベースとして、並列計算向けに工夫したアルゴリズムが実装されています。

6.4.1 シミュレーテッド・アニーリング

6.3 章で説明した **山登り法** では、探索が局所的最適解に詰まってしまい、大域的最適解が得られないことが問題となりました。シミュレーテッド・アニーリングは、局所的最適解に詰まってしまわないように、山登り法を工夫した局所探索法であるとイメージすれば良いです。

[*1] 他にも、タブー探索法や遺伝的アルゴリズム、粒子群最適化などの多数の手法が存在しますが、ここでは割愛しています。
[*2] なお、量子アニーリングと比較される場合には、イジングモデルのスピンに対し定義されたシミュレーテッド・アニーリングが言及されることが多いです。実際には、後述する「近傍」などの設計方法には多数の選択肢があります。



シミュレーテッド・アニーリング

1. 初期解をランダムに与える。現在解を初期解とする。
2. （問題の構造に基づいて近傍の生成ルールがあり、）現在解に対する近傍解を生成する。
3. 近傍解の目的関数値を評価して、
    3a. 現在解よりも改善するならば、現在解を近傍解に変更。
    3b. 現在解よりも悪化するならば、「悪化していても受理」するか判定し、
        3b-1. 受理ならば現在解を近傍解に変更。
        3b-2. 拒否ならば解の変更を行わない。
4. 「悪化していても受理」する判定条件を厳しくする
5. 終了条件の結果次第で、2. に戻るか終了する。

「**アニーリング（Annealing）**」は、日本語で「**焼き鈍し**」といいます。鉄を精製するときに、熱を加えた状態から、ゆっくりと冷やしていくことによって、鉄の内部のエネルギーが安定し、質の良いものが得られることを指しています。シミュレーテッド・アニーリングは焼き鈍しとイメージが似ているために名付けられています。最初は悪化する解であっても受理して様々な近傍の解に移動しますが（高熱）、反復が進むにつれてゆっくりと近傍への移動をゆるやかにし（冷やす）、目的関数の大域的最適解を探ります（エネルギーが安定）。

シミュレーテッド・アニーリングのアルゴリズムは、近傍の定義の仕方や、「悪化していても受理」する条件の設計によって、複数の実装方法がありえます。イジングモデルの場合には、$\mathbf{x} \in \{-1, 1\}^N$ で解が表現されているため、それらのハミング距離の観点での近傍を解として探索していることが多いです。例えば、N ビットあるうちの数個を、-1 であれば 1 にし、1 であれば -1 に変更した値を近傍とします。

ただし、**業務要件に特有の近傍的構造がある場合には、単にビットを反転させるよりも効率的** であることが多いです。例えば、巡回セールスマン問題における、都市のまわり方の順序を一部入れ替える近傍として、2opt 近傍などが有名です[*3]。このような場合には、シミュレーテッド・アニーリングの近傍は、6.2.4 章 の **巡回セール**

[*3] 巡回セールスマン問題への招待 III http://www.orsj.or.jp/~archive/pdf/bul/Vol.39_03_156.pdf

スマン問題におけるペナルティ関数 に記載した QUBO のビット反転ではなく、「巡り方」の性質を踏まえた近傍のほうが効率的であると考えられます。また、ペナルティ関数付きの QUBO に対してシミュレーテッド・アニーリングを適用しても、ヒューリスティックなアルゴリズムであるため、制約違反をしている解が得られる可能性がありますが、2opt 近傍などの場合には、制約条件を満たす範囲内での探索を行うために、必ず制約条件を守るというメリットもあります。

まとめ

- 「**シミュレーテッド・アニーリング**」は、大域的最適解を目指して、広めに近傍を探索しながら、凹凸の登り降りを許すように構成されている。

6.5 組合せ最適化問題の前処理

コンテンツ

量子アニーリング／イジングマシンを用いて組合せ最適化問題を解く際に、定式化した組合せ最適化の数式を扱うために、最低限知っておきたい関係式を記載しています。ハードウェアの特性に合わせて処理を施しておくことで、得られる目的関数の質が向上したり、パラメータのチューニングが効率的になることがあります。
ただし、ここに記載されている関係式は、頻繁に利用されて簡単なごく一部のものをピックアップしていますが、実際には様々なテクニックにより性能が左右されます。より詳細な処理方法は、最適化ソルバーや組合せ最適化の教科書を参照してください。

6.5.1 最小化と最大化

慣習上、特に断りがない場合には、最適化問題の目的関数は **最小化** とすることが多いです。そのため、量子アニーリング／イジングマシンだけでなく、既存の様々なアルゴリズムにおいても、対象とする問題の入力は最小化を前提として設定されていることが多く、単に「最適解」と書かれているときに「最小化」を指す場合があります。

ただし、実応用上では、最大化を行いたい場面もあるため、定式化した段階では最小化問題になっていないことがあります。その場合、次のような簡易な変形を施すことにより、最大化問題を最小化問題として見なすことができます。

符号反転による最小化と最大化の変換

$f(\mathbf{x})$ に最適解が存在するとき、次の 2 つの最適化問題の最適解は同じである。

$$\underset{\mathbf{x}}{\text{minimize}} \quad f(\mathbf{x})$$

$$\underset{\mathbf{x}}{\text{maximize}} \quad -f(\mathbf{x})$$

6.5.2 定数倍

巡回セールスマン問題において、移動する距離の単位系を km で記述するか、m で記述するかによって、目的関数や制約条件の係数の値の桁が見た目上異なります。しかし、異なる単位系で記述したとしても、本質的な距離は変わらないため、通るべき都市の順序が変わることにはなりません。

より広く言うと、最適化問題の目的関数や制約条件は、定数倍を行ったとしても、得られる解が変わりません。また、等式制約・不等式制約に対して、方程式に対する操作と同様の考え方で変形を行うことができます。ただ

し、負の数をかける場合には、**最小化と最大化** や不等号の反転なども関係するといった点などには注意してください。

目的関数と制約条件に対する定数倍

$f(\mathbf{x})$ に最適解が存在するとき、定数 $\alpha > 0, \beta_i > 0, \gamma_j \neq 0$ を変化したとしても、次の 2 つの最適化問題の最適解は同じである。

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f_0(\mathbf{x}) \\ \text{subject to} \quad & f_i(\mathbf{x}) \leq b_i,\ i = 1, \ldots, n, \\ & g_j(\mathbf{x}) = c_j,\ j = 1, \ldots, m. \end{aligned}$$

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \alpha f_0(\mathbf{x}) \\ \text{subject to} \quad & \beta_i f_i(\mathbf{x}) \leq \beta_i b_i,\ i = 1, \ldots, n, \\ & \gamma_j g_j(\mathbf{x}) = \gamma_j c_j,\ j = 1, \ldots, m. \end{aligned}$$

6.5.3 定数項

目的関数の最適解を計算するときに、下記の α のような定数項が存在することがあります。定数項は、目的関数値を変更するものの、得られる最適解自体には影響を与えません。つまり、 $f(\mathbf{x})$ の最適解が $\mathbf{x}^*$ であるとすると、$f(\mathbf{x}) + \alpha$ の最適解も $\mathbf{x}^*$ です。したがって、定式化を行ったり、上三角化を行った際に生じた定数項は無視しても、得られる解は変わらないです[*1] 。

定数項

$f(x)$ に最適解が存在するとき、定数 α に対して、次の 2 つの最適化問題の最適解は同じである。

$$\underset{\mathbf{x}}{\text{minimize}} \quad f(\mathbf{x})$$

$$\underset{\mathbf{x}}{\text{minimize}} \quad f(\mathbf{x}) + \alpha$$

6.5.4 上三角化について

量子アニーリング／イジングマシンのハードウェアによっては、イジングモデルや QUBO の係数行列が上三角行列として設定されています[*2] 。実は、D-Wave システムも上三角行列であるイジングモデルを入力の形式として採用しています[*3] 。ハードウェアとインタフェースによって、広義／狭義の上三角行列であったり、上三角行列ではなくても入力を受け付けたり、イジングモデルの符号が異なったり等、微細な違いが存在します。

[*1] 定数分の差異は解に影響を及ぼさないことから、具体的な数値ではなく const. という記号でまとめてしまうこともあります。

[*2] 他にも、マシンによっては様々なパターンがあります。ほとんどのパターンに対して、本節に記載している内容を多少改変し応用すれば（特に対角成分に注目する）、式の変形方法を導出することができます。

[*3] D-Wave System Documentation - Solving Problems with D-Wave Solvers https://docs.dwavesys.com/docs/latest/c_gs_3.html

微細な違いの対処は個々のユーザが考慮する必要があるため、対象のハードウェアで定義されている入力形式を確認した上で、本節の数式変形などを参考に、複数組み合わせて適切な入力になるように前処理を行ってください。

イジングモデル（狭義上三角行列）

$$\underset{(x_1,x_2,\cdots,x_N)\in\{-1,1\}^N}{\text{minimize}} \quad \sum_{i<j}^{N} J_{i,j}x_ix_j + \sum_{i=1}^{N} h_ix_i + \text{const.} \tag{6.5}$$

Σ の下添字が $i<j$ とされていますが、「ある行 i に対して、それよりも大きな添字の列 j に対して和を取る」というルールを意味します。これは、二次の係数行列 J における、上三角形の部分に該当します。$j=i+1$ といった添字の場合も同様です。

組合せ最適化を定式化したときに、一般的には J は上三角行列になるとは限りません。そのため、一見すると (6.5) で記述される対象は限定的である可能に見受けられます。しかし、同じ問題の最適解が得られるという条件下で、 J を上三角行列に変形するための手法が存在します。$\mathbf{x}\in\{-1,1\}^N$ であるとき、 $x_i^2=1$ であることに注意した上で、以下のような変形を行うことができます。

$$\begin{aligned}
\underset{(x_1,x_2,\cdots,x_N)\in\{-1,1\}^N}{\text{minimize}} \quad & \sum_{i=1}^{N}\sum_{j=1}^{N} J_{i,j}x_ix_j + \sum_{i=1}^{N} h_ix_i + \text{const.} \\
&= \sum_{i<j}^{N} J_{i,j}x_ix_j + \sum_{i=j}^{N} J_{i,i}x_i^2 + \sum_{i>j}^{N} J_{i,j}x_ix_j + \sum_{i=1}^{N} h_ix_i + \text{const.} \\
&= \sum_{i<j}^{N} J_{i,j}x_ix_j + \sum_{i<j}^{N} J_{j,i}x_jx_i + \sum_{i=1}^{N} h_ix_i + \text{const.} + J_{1,1} + \cdots + J_{N,N} \\
&= \sum_{i<j}^{N} \left(J_{i,j}+J_{j,i}\right)x_ix_j + \sum_{i=1}^{N} h_ix_i + \text{const.} + J_{1,1} + \cdots + J_{N,N}
\end{aligned}$$

この数式は、

1. 非上三角行列である J に対して、新しく $J'=(J+J^\top)$ を計算する。
2. $i<j$ となる要素だけを J' から抽出し（狭義上三角行列）、新しい 2 次項と見なす。
3. 線形項 $\mathbf{h}$ は変更しない。定数項は、J の対角成分の総和分だけ加算する。

というプロセスを踏めば、最適解 $\mathbf{x}$ が変わらないまま、イジングモデルの 2 次項の係数を上三角行列に変形できることを意味します。

QUBO（広義上三角行列、線形項無し）

$$\underset{(x_1,x_2,\cdots,x_N)\in\{0,1\}^N}{\text{minimize}} \quad \sum_{i\le j}^{N} J_{i,j}x_ix_j + \text{const.} \tag{6.6}$$

次に、QUBO を入力として求められていて、2 次項が広義上三角行列であり、線形項 $\mathbf{h}$ であるとき、$x_i = 0$ ならば $x_i^2 = x_i = 0$ であり、 $x_i = 1$ ならば $x_i^2 = x_i = 1$ であること、つまり $x_i^2 = x_i$ であることに注意した上で、以下のような変形を行うことができます。

$$
\begin{aligned}
\underset{(x_1,x_2,\cdots,x_N)\in\{0,1\}^N}{\text{minimize}} \quad & \sum_{i=1}^{N}\sum_{j=1}^{N} J_{i,j}x_i x_j + \sum_{i=1}^{N} h_i x_i + \text{const.} \\
&= \sum_{i<j}^{N} J_{i,j}x_i x_j + \sum_{i=j}^{N} J_{i,i}x_i^2 + \sum_{i>j}^{N} J_{i,j}x_i x_j + \sum_{i=1}^{N} h_i x_i^2 + \text{const.} \\
&= \sum_{i<j}^{N} J_{i,j}x_i x_j + \sum_{i<j}^{N} J_{j,i}x_j x_i + \sum_{i=1}^{N} (J_{i,i} + h_i)x_i^2 + \text{const.} \\
&= \sum_{i<j}^{N} (J_{i,j} + J_{j,i})\, x_i x_j + \sum_{i=1}^{N} (J_{i,i} + h_i)x_i^2 + \text{const.}
\end{aligned}
$$

この数式は、

1. 非上三角行列である J に対して、新しく $J' = (J + J^\top)$ を計算する。
2. $i < j$ となる要素だけを J' から抽出する（狭義上三角行列）。
3. 狭義上三角行列 J' の対角成分に、元々の J の対角成分と、線形項 $\mathbf{h}$ の和を順に代入し、新しい 2 次項と見なす。
4. 線形項 $\mathbf{h}$ はゼロベクトルにする。定数項 const. は変更しない。

というプロセスを踏めば、最適解 $\mathbf{x}$ が変わらないまま、QUBO の 2 次項の係数を広義上三角行列に変形し、線形項が除外できていることを意味しています。

まとめ

- 「**定数項**」「**最小化と最大化**」「**定数倍**」などの操作を行って係数を変えても、組合せ最適化の解は変わらない。
- ハードウェアによって、入力するイジングモデルの形式に条件があり、上三角化などの処理が必要になる場合がある。

7 量子アニーリング／イジングマシン特有の処理

この章の流れ

量子アニーリング／イジングマシンを利用する場合には、組合せ最適化に関連するパラメータだけではなく、利用するハードウェア固有の構成やパラメータに応じた処理が必要になる場合があります。
非常に数多くのパラメータが各ハードウェアのマニュアル毎に存在しますが、そのようなパラメータが派生する元となっている、特に重要な概念がいくつか存在します。
この章では、以下の 4 つのトピック

- 7.1 章 **疎結合と全結合**
- 7.2 章 **階調**
- 7.3 章 **アニーリングタイム**
- 7.4 章 **解のサンプリング**

に関して説明しています。

7.1 疎結合と全結合

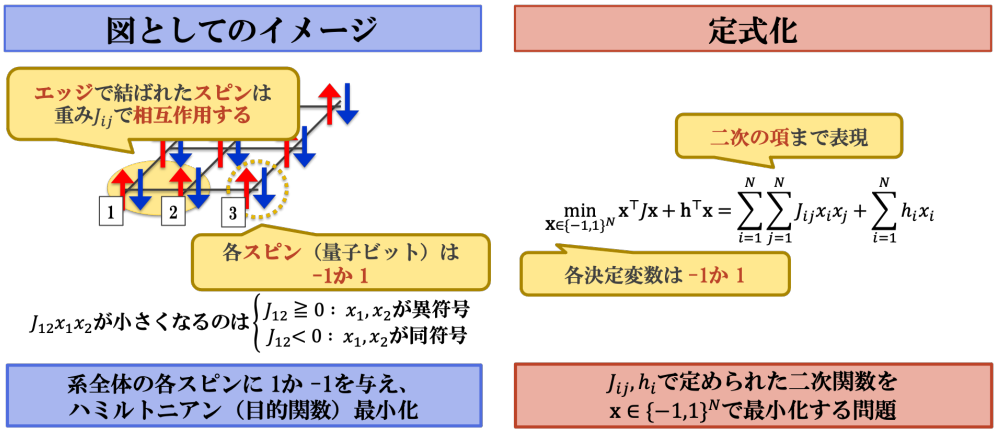
コンテンツ

この節までの説明では、特別な断りなく **イジングモデル** の目的関数を前提に説明しました。実際のハードウェアを利用するときは、どんなイジングモデルでも解けるわけではなく、より限定的な対象しか入力できないことがあります。この制限は、量子アニーリング／イジングマシンのハードウェア、更に解きたい対象のイジングモデルの **疎結合** と **全結合** という概念に関連します。
まず、これらの結合の意味を説明した後に、制限を解決するための **マイナーエンベディング** という方法や、ビット数・パラメータに与える影響を説明します。

7.1.1 疎結合／全結合なハードウェア

量子アニーリング／イジングマシンは、**イジングモデル** を入力とすると説明しました。実は、ハードウェアによっては、イジングモデルの中でも、更に限定的な形式だけを入力として受けつけます[*2] 。

4.2 章 に表示していた図を再掲します。量子アニーリングでは、4.4 章 で説明したように、「定式化」に記載されているような求解対象のイジングモデルの J, h を、「図としてのイメージ」に該当するような物理系のエッジ・ノード部分へ直接マッピングを行っていました。



左図に示している各スピンは、エッジで結線されています。もしも、各々のスピン間が相互に結線されている場合には「**全結合（Fully-Connected）**」であるといいます。全結合なハードウェアの場合、「マッピング先のエッジが存在しない」という問題が生じないため、解きたいイジングモデルをそのまま入力することができます。

一方で、スピン間の全結合を実現するためには、ハードウェア上のスピン間をすべて相互に結線しなければならず、あまりに回路構成上複雑になるという課題があります。そのため、互いに近傍にあって、結線しやすいスピン間だけにエッジを張るハードウェアを「**疎結合（Sparsely-Connected）**」であるといいます。疎結合である場合、解きたい対象のイジングモデルをそのままマッピングしようとしても、対応するエッジが存在しない可能性があり、入力することができなくなります[*3] 。

この節までに説明した量子アニーリング／イジングマシンでは、「イジングモデルであれば解ける」という説明だったため、全結合である前提を置いていました。

疎結合の場合でも、全結合なハードウェアと大半の利用方法やパラメータは類似しています。ただし、入力できないようなイジングモデルに対して前処理を施し、後述の **全結合化** や **マイナー・エンベディング** と呼ばれるフェーズと関連するパラメータを追加する必要があります。

[*2] 例えば、D-Wave 2000Q は Chimera グラフ、D-Wave Advantage は Pegasus グラフであり、日立製作所によるイジングマシンの CMOS アニーリングは King's グラフを採用しています。

[*3] 求解対象のイジングモデルが、ハードウェアグラフの「**誘導部分グラフ（Induced Subgraph）**」であれば、入力することが可能です。

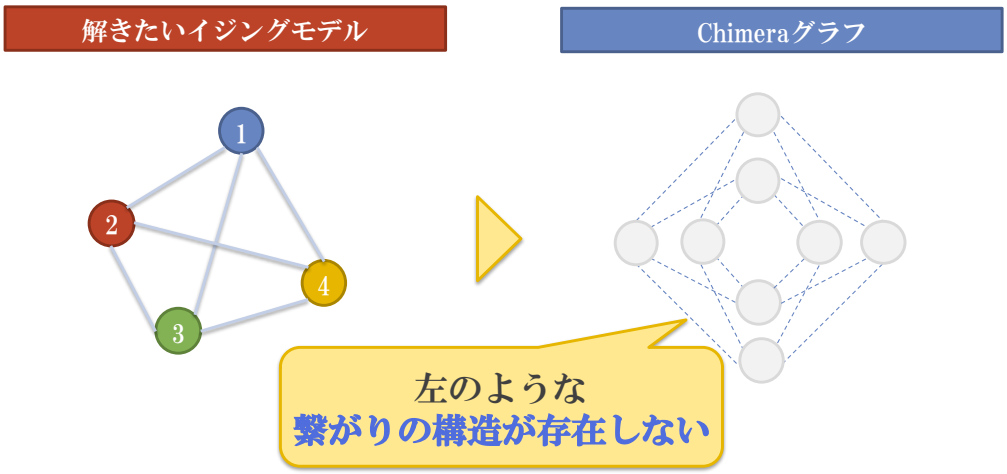


図 7.1　Chimera グラフへの 4 ビット（全結合）イジングモデルの埋め込みはできない

7.1.2 疎結合ハードウェアの全結合化

ハードウェアが疎結合であるときに、マッピングできないイジングモデルを入力するために、ハードウェアが擬似的に全結合かのように振る舞うような前処理を施します。この前処理は、しばしば「**全結合化**」と呼ばれます。この処理は、スピンを解きたいイジングモデルよりも多く消費することで、汎用性を向上させるという特徴があります。

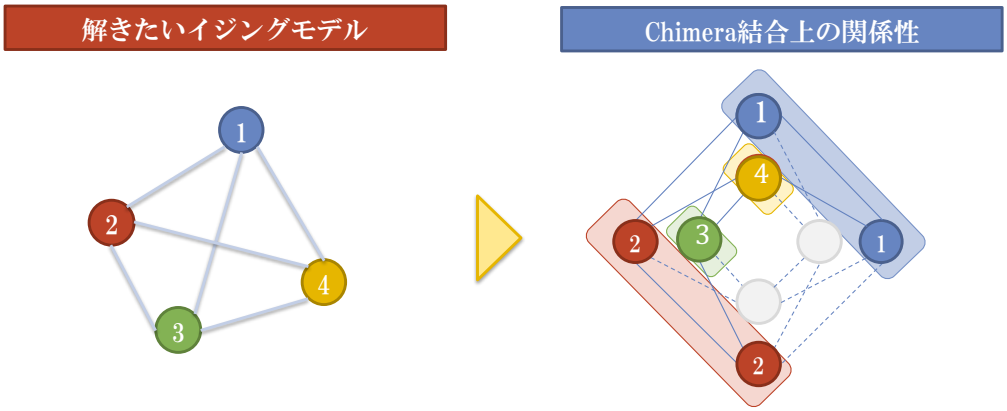


図 7.2　解きたいイジングモデルを、Chimera グラフへマッピングする処理のイメージ

具体的には 図 7.2 に記しているような処理を実施しています。左図に記載している求解対象のイジングモデルの 1 つのビットを、右図に記載しているハードウェア上の複数のスピンに対応付けます。例えば、左側の "1 ノード" は、右側では "1 ノード× 2 個" に対応づいています。右側 "1 ノード× 2" のノード間を結ぶ青色ブロックのエッジには、互いのスピンが同じ値を取るように力を加える係数を設定しています。

右側の 1, 2, 3, 4 を囲うブロックを、一つの新たなノードとみなせば、ノード間は互いにエッジで結ばれていることが確認できます。これらのエッジに、左図の解きたいイジングモデルのエッジの重みをマッピングします。このように、同じ値を取るように力を加えあったブロックを、スピンの「**チェーン（Chain）**」といいます。

今回は 4 ビットの小規模な例を記載しましたが、大規模な問題を解く場合にも、同様の操作に基づいて全結合化がなされます。疎結合なハードウェアの場合、全結合化の前処理によって、搭載されているビットよりも少ない数しか利用できなくなることがあります。そのため、性能表において、疎結合であるか全結合であるかによって、1 ビットが持つ意味合いが異なる場合があります。

7.1.3 疎結合／全結合な（求解対象の）イジングモデル

前の節では、ハードウェアが疎結合／全結合であることを説明しました。実は、疎結合／全結合の概念は、ハードウェアに搭載されているスピンだけではなく、**解きたい対象側のイジングモデルに対しても存在** します。

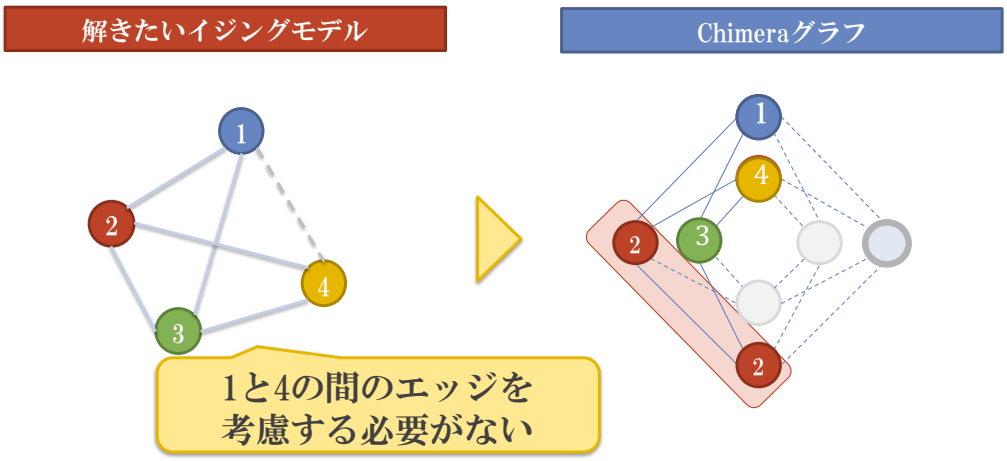


図 7.3　求解対象のイジングモデルが疎結合な場合の Chimera グラフへのマッピング

図 7.3 の左側に示している図が、解きたい対象側のイジングモデル疎結合である場合です。図 7.2 と違い、ノード 1 と 4 の間のエッジが存在しないです。そもそも解きたい対象に、ノード 1 と 4 の間の直接的な関係性は存在しないということであるため、D-Wave システムの Chimera グラフのエッジにも、重みを設定する必要がありません。このような場合には、Chimera グラフ上で作成しなければならないチェーンの長さが小さくなり、結果ビット数の消費を抑えることができます。

したがって、疎結合なハードウェアを利用する場合には、解きたい対象側のイジングモデルの結合の度合いに応じて、利用できるビット数が変動します。両者のチェーンへのマッピングを行うことを、「**マイナーエンベディング（マイナー埋め込み, Minor Embedding）**」といいます。全結合化は、解きたい対象のイジングモデルが全結合であった場合のマイナーエンベディングであるといえます。

ノードとエッジがなすグラフにおいて、その存在と結びつき方の構造を「**トポロジー（Topology）**」といいます。ハードウェアの疎結合のグラフのノードとエッジのトポロジーと、解きたい対象側のイジングモデルのトポロジーが、どちらも静的で変わらない場合には、マイナーエンベディングを行いチェーンへのマッピングを一度取得すれば使い回すことができます。

一方で、流動的に解きたい問題が変わるといった状況で、グラフのトポロジーが変わる場合には、都度マイナー

エンベディングを行う必要があります[*4][*5][*6] 。

7.1.4 チェーンの強度

図 7.2 では、同じ値を取るように、チェーンに含まれるスピンが同じ方向を向くような力を設定していました。この力の強さを「**チェーンの強度（Chain Strength）**」といいます。

チェーンの強度は、理論上は極めて大きな正の値を設定すれば良いです。この考え方は、**ペナルティ関数** におけるペナルティパラメータと類似しています。チェーンに含まれるスピン同士が、別の方向を向いた場合にペナルティをかけるように設定します。

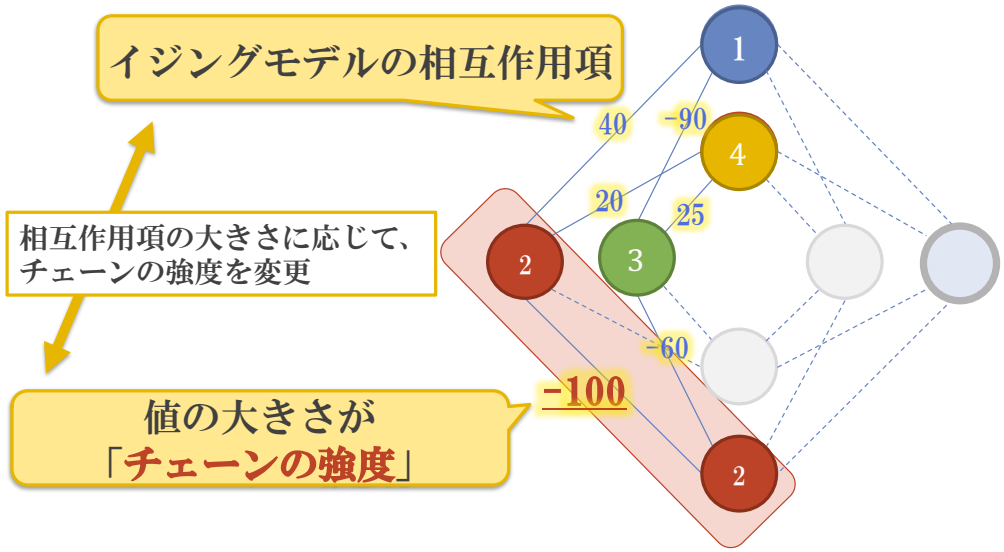


図 7.4　イジングモデルの相互作用項の大きさに応じて、チェーンが破られないよう適度に大きいチェーンの強度を設定する必要がある。

ただし、実際にハードウェアを利用する場合には、ペナルティパラメータ同様、大きすぎる値を設定すると不安定な数値挙動を引き起こす傾向が見られます。また、適切なチェーンの強度は、図 7.4 のように相互作用項の大きさに応じて変わります[*7]。

[*4] D-Wave Systems によって、*minorminer* というマイナーエンベディングの自動化ツールが提供されています。
[*5] GitHub - dwavesystems / minorminer https://github.com/dwavesystems/minorminer
[*6] A practical heuristic for finding graph minors https://arxiv.org/abs/1406.2741
[*7] D-Wave Ocean SDK では `chain_strength` という名前で扱われており、設定する値に関連する機能が提供されています[*8] 。
[*8] dwave-system - Chain Strength https://docs.ocean.dwavesys.com/projects/system/en/stable/reference/embedding.html#module-dwave.embedding.chain_strength

まとめ

- 「Chimera グラフ」「Pegasus グラフ」「King's グラフ」のように、ハードウェアが「疎結合」であり、エッジの結合の仕方に制限が存在するハードウェアが存在する。
- ビットを消費して、イジングモデルの結合を上げ汎用性を上げる「チェーン」
- 求解対象のイジングモデルにも「全結合」「疎結合」があり、ハードウェアのチェーンと対応付けする「マイナーエンベディング」

7.2 階調

実際のハードウェアにイジングモデルを入力する際に、J, h の値を無制限に投入できるのではなく、ハードウェアで設定されている値までしか設定することができません。この入力値として設定できる係数の度合いのことを「**階調（結合精度, Precision）**」と呼びます。

なお、階調には、量子アニーリング／イジングマシンの実装に応じて、いくつかの観点が存在することに注意してください。

- ハードウェアとして、入力パラメータに設定できる意味での階調。
- 良質な解を得られる意味での階調。
- 量子アニーリングの回路上に値を設定する上で生じる誤差[*1]

量子アニーリング／イジングマシンは、ヒューリスティックなアルゴリズムによって近似解を返却しますが、特に投入したイジングモデルの係数などの条件によって、その質の良さが左右する傾向にあります。例えば、**ペナルティ関数** や **マイナーエンベディング** の節で、ペナルティパラメータやチェーン強度をむやみに大きな値にすると、解の質が悪化する傾向にあることを説明しました。

(6.3) において、$\lambda = 10000000$ としましょう。

$$\underset{\mathbf{x}\in\{-1,1\}^3}{\text{minimize}} \quad x_1 + 10000000\,(x_1 + x_2 + x_3 - 1)^2$$

もともとの目的関数は、x_1 の最小化でしたが、ペナルティパラメータがあまりに大きすぎるために、ハードウェアに実装されているヒューリスティックなアルゴリズムの数値挙動上、x_1 の情報が相対的に薄められて消えてしまい、良い解が得られなくなることがあります。

7.3 アニーリングタイム

4.4 章 **量子アニーリングのイメージ** では、計算を収束させるために「ゆっくりと横磁場項を弱める」という説明をしていました。理論的には、十分に長い時間ゆっくりと横磁場項を弱めていくことによって、イジングモデルの基底状態、つまり最適解に収束することが知られています。

実際のハードウェアで計算する場合には、計算し続けるわけにはいかないため、現実的な時間で計算を打ち切る必要があります。D-Wave システムでは、量子アニーリングを行う計算時間のことを、「**アニーリングタイム（Annealing Time）**」と呼んでおり、パラメータ名は `annealing_time` で与えられています[*1][*2][*3]。デフォルトでは指定した分だけ線形に時間をかけて量子アニーリングを行うように設定されていますが、時間の経過方法を変更することもでき、問題固有で調整することによって解の質が向上することがあると報告されています[*4]。

[*1] ICE: Dynamic Ranges in h and J Values https://docs.dwavesys.com/docs/latest/c_qpu_1.html

[*1] D-Wave システムのアニーリングタイムの設定可能な範囲は、Terminal 上で `dwave solvers` と入力した際に表示される `annealing_time_range` というパラメータで確認できます。

[*2] Ocean Documentation - dwave CLI https://docs.ocean.dwavesys.com/en/stable/docs_cli.html#cli-example-solvers

[*3] D-Wave System Documentation - QPU (Hardware) Solvers https://docs.dwavesys.com/docs/latest/c_solver_1.html

[*4] D-Wave QUBITS 2018 - Performance Tuning for D-Wave Quantum Processors https://www.dwavesys.com/sites/default/files/2_Wed_Am_PerfTips.pdf

なお、イジングマシンのハードウェアの場合には、実装されているアルゴリズムによってアルゴリズムと収束性などに違いが存在するものの、アニーリングタイムに相当するパラメータが存在することが多いです。個々のハードウェアによって異なるものの、**シミュレーテッド・アニーリング** をベースとしている場合には、ゆっくり時間をかけるほど最適な解が得られる可能性が高まります。

7.4 解のサンプリング

量子アニーリング／イジングマシンに実装されているアルゴリズムはヒューリスティックであるため、得られた解が最適解であることは保証されていません。そのため、解きたい組合せ最適化に対する定式化の方法や、ハイパーパラメータなどの実験条件によっては、質の悪い近似解が出力される可能性があります。

一方で、量子アニーリング／イジングマシンの中には、1 回の求解リクエストの中で、大量の解の候補を取得することが得意なハードウェアも存在します。その場合、1 回のリクエストの中で、大量の解の候補を算出し、その中でも最も良質な解を計算結果として採用するアプローチを取ることができます。この方法に対する特定の名前は定まっていませんが、D-Wave システム上では「**num_reads**」というパラメータで定められています[*1] 。

多くの解を大量に取得すると、優れた解が混ざっている確率も上昇するため、`num_reads` は大きくするほど、結果的に得られる解の質が向上する傾向にあります。ただし、マシンの計算リソースを多く消費するリスクが存在するため、注意が必要です。

また、イジングモデルに対する大量の解の候補を取得することを、特定の確率分布に従っているようなデータを取得していると見なして、「**サンプリング（Sampling）**」を行っていると呼ぶこともあります。

サンプリングは組合せ最適化とも関連性はあるものの、また別の領域での活用分野であるといえます。「**マルコフ連鎖モンテカルロ法（Markov Chain Monte Carlo, MCMC）**」などをはじめとする古典コンピュータ上のアルゴリズムも存在しますが、組合せ最適化同様に、非常に重い計算を必要とする分野として知られています。本ガイドラインでは組合せ最適化の概要と量子アニーリング／イジングマシンの関連性を主眼としましたが、一度に数多くのサンプリングができる特性に着目して、組合せ最適化以外の用途でも活用できないかという研究も進められています。

[*1] D-Wave System Documentation - QPU (Hardware) Solvers https://docs.dwavesys.com/docs/latest/c_solver_1.html

2021年1月28日発行

株式会社NTTデータ
株式会社NTTデータ　技術革新統括本部 技術開発本部
量子コンピュータ／次世代アーキテクチャ・ラボ
執筆　：　香月 諒大
お問い合わせ先：矢実　貴志　／　田端　佑介　／　尾崎　史朗　／　香月 諒大　／　川又 裕也
E-mail: qcomputer@kits.nttdata.co.jp