論　文

CVRPに対する量子アニーリング方式の決定変数削減

若谷彰良

甲南大学 知能情報学部
神戸市東灘区岡本８－９－１, 658-8501

(受理日 2024 年 5 月 14 日)

概 要

量子コンピューティングの方式のひとつとして量子アニーリング方式が注目され, 組合せ最適化問題へ実用的な利用が進んでいる, 量子アニーリングはイジングモデル上のスピングラス問題を解くものであるが, 昨今は, その手法を起源としたデジタルマシンも多く考案され, 本稿で用いた GPU ベースの Fixstars Amplify Annealing Engine もその一つである. 本稿では, 組合せ最適化問題の一つである容量制約つき運搬経路問題に対して, 最適解探索に用いる決定変数を削減するための 2 個の手法で実装し, 性能を評価した. ルート数を考慮して決定変数を削減する場合に, 決定変数の一部の値を固定化して削減する方法と決定変数の生成自体を削減する方法を評価し, 前者の方法は決定変数削減による高速化の効果は限定的であるが, 後者の方法は, 訪問都市数に依存するものの, 概ね 2 倍程度の高速化が達成できた.

キーワード: 量子コンピューティング, 組合せ最適化, イジングモデル, QUBO

1 はじめに

ムーアの法則に則った半導体技術の向上に伴い, 計算機の性能は着実の進展してきたが, その限界が視界に入るにつれて新しい計算方式が注目されるようになった. その代表が量子コンピューティングであり, 量子ゲート方式および量子アニーリング方式がある [1], [2]. 量子ゲート方式は一部実現はされているが, 現時点では実用レベルの性能にはなっておらず, 特にノイズをどのように対処するかが大きな課題となっている. 量子アニーリング方式には, 実用機として実現されている D-Wave のマシンだけでなく, 量子アニーリング方式から発想された半導体デジタル技術をもとにした擬似的な量子アニーリングマシンが実現されており, 代表的マシンである Fixstars Amplify の Amplify Engine (以下, Fixstars AE と記す) は高性能 GPU をもとに実装されている.

量子アニーリング方式は全ての計算に適しているものではなく, イジングモデルを基づいて実装されているので組合せ最適化問題を解くことに特化しているといえる. すなわち, QUBO (二次制約無し二値最適化: Quadratic Unconstrained Binary Optimization) に変換できる組合せ最適化問題であれば, 量子アニーリング方式で解ける. 本方式に適した組合せ最適化問題の一つに CVRP (Capacitated Vehicle Routing Problem: 容量制約つき運搬経路問題) がある. CVPR とは, 配送元の都市から複数の都市に貨

物を複数のトラックで運搬するコスト (時間や燃料費等) を最小にする問題で, トラックに搭載できる貨物量に制限があるという制約が課せられるものであり [3], 貨物の運搬だけではなく, 災害時の罹災状況把握の最適経路問題 [4] など様々な問題に適用できる. 後述するように, 最適値を求めるために未決定の情報を決定変数の形で定式化する必要があるが, 決定変数の数は計算時間や計算リソースなどに大きな影響を与えるので, その最少化は高速化のための重要なファクターの一つである.

CVRP に対してイジングマシンでの実現を行っている研究はあり, Fixstars Amplify は参照実装を公開しており, 後述するような決定変数の削減方法を用いている [5]. Harikrishnakumar らは, 量子アニーリングを用いて, CVRP に対する最適となるルーティングを static 決めるだけでなく, 運用をしている途中でカスタマーからのリクエストが新たにあった場合に, dynamic な最適ルーティングを決定する手法について述べているが, 実装例や決定変数の削減については述べていない [6]. また, Masuda らは, 物流事業者に向けた業務支援ツールの一つである配送計画システムをイジングマシンで解く試みをしており, 制約条件に基づく決定変数の値の固定化を行って Fixstars AE での実装を行っている [7].

本稿では, Fixstars AE 上での CVRP の実装の効率化を図るために, 決定変数の削減方法を 2 種類提案し, 性能面での効果を中心に評価を行う. 2 章で CVRP の QUBO 実装について述べ, 3 章で Fixstars AE での実装方法及び決定変数削減方法の提案を行い, 4 章で提案手法の評価について言及し, 5 章でまとめを述べる.

2 CVRP の QUBO 実装

2.1 定式化

本稿で扱う CVRP のターゲット問題を定式化する. すなわち, 0 番の配送元から $(ncity-1)$ 箇所の都市を $nroute$ 台のトラックで順番に貨物を配送する経路のコストの最適化において, 次の制約条件を満たすものを考える.

制約条件:

- トラックに載せられる貨物の容量の上限を $WMAX$
- 配送元以外の各都市には 1 回しか訪れない
- 配送元以外の各都市には 1 回は訪れる

以下の変数を設定し, $q_{r,i,k}$ を決定変数とする.

$d_{i,j}$ ：i 番の都市から j 番の都市へのコスト

w_i ：i 番の都市へ配送する貨物の容量

$q_{r,k,i}$ ：第 r トラックが, i 番の都市に k 番目に配送する時に 1, そうでない時に 0 となる決定変数

第 r トラックが通る経路を第 r 番目のルートと呼び, 0 番目の都市を配送元とする.

以上の問題を定式化すると下記のように表される.

$$q_{r,0,0} = 1 \leftarrow \text{最初は 0 から始まる} \tag{1}$$
$$cost = \sum_{r,i,j} d_{i,j} \times q_{r,k,i} \times q_{r,k+1,j} \tag{2}$$
$$constraint1 = [\sum_{r,k} q_{r,k,i} = 1 \text{ for each } i \neq 0] \tag{3}$$
$$constraint2 = [\sum_{r,k} q_{r,k,0} = (nroute-1) \times ncity + 1] \tag{4}$$
$$constraint3 = [\sum_{k} q_{r,k,i} = 1 \text{ for each } r,i] \tag{5}$$
$$constraint4 = [\sum_{i,k} w_i \times q_{r,k,i} \leq WMAX \text{ for each } r] \tag{6}$$
$$H = cost + C * (constrain1 + constrain2 + constrain3 + constraint4) \tag{7}$$

$cost$ が最小化したい目的関数であり, $constraint1$ は配送元以外の各都市にはちょうど 1 回は訪れる条件で, $constraint2$ は配送元 (0 番の都市) は $((nroute-1) \times ncity + 1)$ 回訪れることになる条件で, $constraint3$ は各ルートにおいて k 番目に訪れる都市は 1 個である条件で, $constraint4$ は各ルートの積載貨物の容量は $WMAX$ 以下である条件を表す. すなわち, C を適切な大きさに設定することで, 全ての制約条件が 0 になることを満たすこととなり, よって, H の最小化により CVRP の最適解が得られることになる.

2.2 問題設定

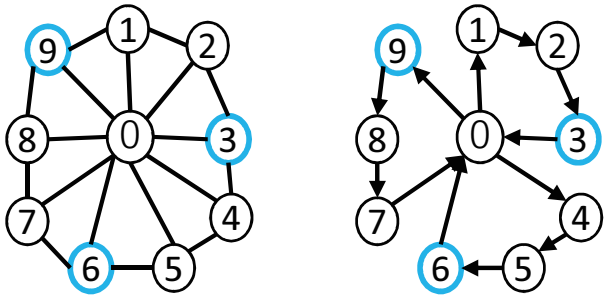


図 1: Example of CVRP

以下の章では, 下記に示すようにパラメータが設定された問題を対象とする. この問題は各ルートにほぼ均等に都市が配置される場合に最適になるもので, 最適解が容易に求められるものになっている.

1. 都市 0 とその他の都市間のコストは 1 とする.
2. 都市 i と都市 $(i+1)\%ncity$ の間のコストは 1 とする.
3. 上記以外の都市間のコストは 60 とする.

4. $mod(n, ntoute) \neq 0$ となる都市への貨物の容量は 1 とする.

5. $mod(n, ntoute) = 0$ となる都市への貨物の容量は $(WMAX - ncity/nroute) - 4$ とする.

6. $WMAX = 30$ とする.

図 1 の左に, 10 都市, 3 ルートの場合を示す. 図の中で実線でつながっている都市間のコストは 1 で, 線が表示されているない都市間, 例えば, 都市 2 と都市 4 の間のコストは 60 となっている. また黒丸の都市への貨物の容量は 1 で, 青丸の都市 (3, 6, 9) への貨物は $(WMAX - ncity/nroute) - 4$ としており, 青丸の都市を 2 箇所以上含むルートは制約条件違反となる. よって, 最適な最適解の一つは図 1 の右のようになることが容易に分かる.

なお, 以下の章ではアルゴリズムの詳細は数式の形ではなく, Fixstars Amplify のプログラムで表示することとする. また実装は, Google colaboratory 上で Amplyfy SDK 1.0.2 を用いてプログラミングを行い, 実行は Fixstars AE のベーシックプラン内で, タイムアウト 20,000 msec. で実行した[1]. 実行時間の計測は図 2 のコードに示す `times` リストに格納されている値を用いる.

```
result = solve(model,client)
times = [solution.time.total_seconds() for solution in result]
objective_values = [solution.objective for solution in result]
```

図 2: Measurement of elaplsed time

3 決定変数の削減

3.1 Fixstars Amplify による実装

第 r トラック $(0 \leq r < nroute)$ が, i 番の都市 $(0 \leq i < ncity)$ に k 番目 $(0 \leq k < ncity)$ に配送するか否かを表す決定変数 $q_{r,k,i}$ は, 図 3 に示すように生成される.

```
from amplify import VariableGenerator
gen = VariableGenerator()
q = gen.array("Binary", nroute, ncity, ncity)
```

図 3: Genaration of decision variable

これにより, $nroute \times ncity \times ncity$ 個の決定変数が生成される.

さらに式（2）から (6) で示される目的関数及び制約条件式は, 図 4 に示すプログラムで実装される. なお, `cost` を計算するには通常の `for` と加算でも実装可能であるが, Amplify SDK で提供される `sum` 関数を用いて実装するのに比べ, Python の実行時間が数倍から数十倍の程度で低速となる. よって, 利

[1]仕様ではベーシックプランでのタイムアウトの上限は 10,000 msec. であるが, エラーなく実行できたのでそのまま利用した.

用可能であれば, 制約条件式や目的関数の実現には SDK で提供される API を活用することが重要である.

```
from amplify import sum as amplify_sum
cost = amplify_sum(
    range(nroute),
    lambda nr: amplify_sum(
        range(ncity),
        lambda n: amplify_sum(
            range(ncity),
            lambda i: amplify_sum(
                range(ncity), lambda j: distances[i][j] * q[nr][n][i]\
                 * q[nr][(n + 1) % ncity][j]
            ),
        ),
    ),
)
constraint1 = [
    equal_to(sum([q[nr][n][i+1] for n in range(ncity)\
      for nr in range(nroute)] ), 1)
     for i in range(ncity-1)
]
constraint2 = [
    equal_to(sum([q[nr][n][0] for n in range(ncity)\
      for nr in range(nroute)]), (nroute-1)*ncity+1)
]
constraint3 = [
    equal_to(sum([q[nr][n][i] for i in range(ncity)]), 1)\
      for n in range(ncity) for nr in range(nroute)
]
constraint4 = [
    less_equal(sum([q[nr][n][i]*weights[i] for n in range(ncity)\
      for i in range(ncity)]), WMAX) for nr in range(nroute)
]
```

図 4: Object function and constraints

3.2 決定変数の削減

文献 [5] では, 各都市に配送される貨物の容量を合計することによりトラックの貨物容量の上限 ($WMAX$) を越える限界があるので, 1 個のルートで訪問できる都市数には上限があり, それを L とすると, 決定変数の個数は, $nroute \times ncity \times ncity$ ではなく $nroute \times ncity \times L$ $(L \leq ncity)$ であり, 決定変数の削減は可能であるとしている. しかし貨物の容量やトラックの容量の上限の与え方次第では, 必ずしも L は $ncity$ よりも大幅に小さくならない場合もあり, この方式による決定変数の削減効果は限定的である.

そこで全ルートで配送元以外の都市を訪れる回数が 1 回であることを考慮すると, 次のように考えることで効果的な決定変数削減が可能である. もし 1 個のルートで全ての都市に配送することが最適であれば, 0 番目のルートの都市数は $ncity$ にする必要があるが, もし 2 個のルートで全ての都市に配送することができれば, それぞれのルートの都市数は最大で $\lceil \frac{ncity}{2} \rceil$ となる. 同様に, 3 個のルートで全ての都市に配送することができれば, それぞれのルートの都市数は最大で $\lceil \frac{ncity}{3} \rceil$ となる. このように, 決定変数の削減を行わない場合は $N_0 (= nroute \times ncity \times ncity)$ 個の決定変数が必要であり, 文献 [5] の削減を行った場合は $N_1 (= nroute \times ncity \times L)$ 個の決定変数が必要であるが, 提案の削減方法を適用した場合は $N_2 (= nroute \times ncity \times \sum_{n=1}^{ncity} \frac{1}{n})$ 個の決定変数で済むことになる.

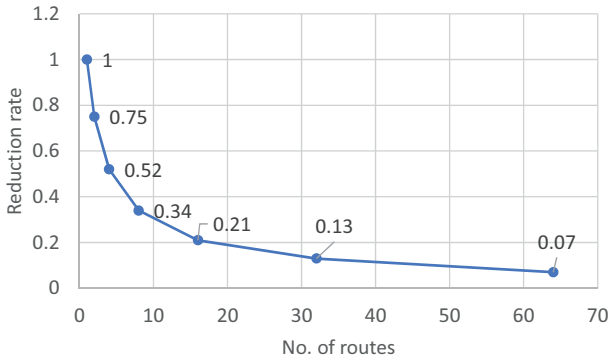


図 5: Reduction of decision variables

$nroute$ の違いによる N_0 と N_2 の比率を図 5 に示す. $nroute = 4$ の時は 0.52 で, $nroute = 16$ の時は 0.21 であり, 大幅な削減になっていることが分かる.

次節以降において, 本削減を実現するために, 1) 決定変数の固定化と 2) 決定変数の生成の削減の 2 個の実現方法について述べる.

3.3 決定変数の固定化

```
1 for nr in range(1,nroute):
2   nsize=1+math.ceil((ncity-1)/(nr+1))
3   for n in range(nsize,ncity):
4     q[nr][n][0]=1
5     for i in range(ncity-1):
6       q[nr][n][i+1]=0
```

図 6: Fix of values of decision variables

前節で紹介したように, $nroute$ の個数に着目した方法により決定変数の削減が有効である. そのために, 生成した決定変数の一部の変数の値を固定化 (初期値を代入) することで実現できる. 決定変数は図 3 で示したプログラムで $nroute \times ncity \times ncity$ 個生成されるが, 最初のルート以外の決定変数は削

減可能であるので, 図6に示すように, その部分の都市0の決定変数の値に1を代入しておくことで決定変数の削減が可能である. なお, 5行目, 6行目で都市0以外の決定変数の値に0を代入しているが, 制約条件から必ずしも代入しなくてもよいが, 解探索の過程で余分な探索がされることにより, 予備評価において実行時間は2倍程度低速になった. したがって, これらの代入は不可欠と考えられる.

以上のように, 固定化した部分の配送先は都市0すなわち配送元になるので, その部分は配送元から動かないことになるので, 制約条件を満たす解であれば, 実質的に他の都市が割り当てられることはない. しかし, D-Waveのような真の量子アニーリングマシンではなく, Fixstars AEのようなデジタル技術を用いた擬似的な量子アニーリングマシンであれば, そのシステムの実装方法によっては, 制約条件を満たさない解もランダムに生成される可能性があり, その場合であれば, 実質的には決定変数の削減にならない可能性があるので, 後述の性能評価を行うまで, 有効性の判断は保留することとする.

3.4 決定変数の生成の削減

前節で述べた削減方法は, 決定変数の生成自体は削減しないが, 一部の決定変数の値を固定化することで実質の決定変数を削減するものであった. ここでは, 実際に生成する変数を削減する方法について述べる. 前節までは, 決定変数 $q_{r,k,i}$ は, 図3に示すようなプログラムで生成されるので, すべてのルートで同じサイズとなっている. そこで, 例えば $nroute = 4$ であれば, 決定変数として $q0$, $q1$, $q2$, $q3$ を, 図7のプログラムで, それぞれ別のサイズで生成することを行なう.

```
1 nncity=[]
2 for nr in range(nroute):
3   nncity.append(1+math.ceil((ncity-1)/(nr+1)))
4 q0 = gen.array("Binary", nncity[0], ncity)
5 q1 = gen.array("Binary", nncity[1], ncity)
6 q2 = gen.array("Binary", nncity[2], ncity)
7 q3 = gen.array("Binary", nncity[3], ncity)
```

図 7: Generation of values of decision variables in different sizes

これにより前節の手法に比べ, 実際の決定変数も削減できるので, デジタルマシン上の実装方法に関わらず決定変数の数は削減可能となる.

```
1 for i in range(nroute):
2   st=f'''
3 q{i} = gen.array("Binary",nncity[{i}],ncity)
4   '''
5   exec(st)
```

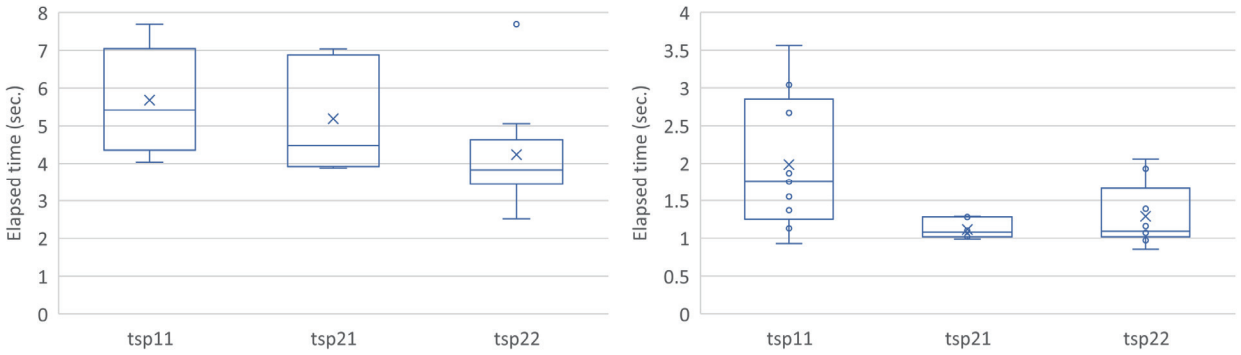
図 8: Generation of values of decision variables in different sizes using exec function

しかし, コストや制約条件の記述も従来の $q_{r,k,i}$ を使った記述ではなく, $q0_{r,k,i}$, $q1_{r,k,i}$, $q2_{r,k,i}$, $q3_{r,k,i}$ を使った記述になるので, 約4倍のプログラム量になり, $nroute$ の値に比例して記述量が増加し, さらに $nroute$ の値毎に異なるプログラムとなってしまう. そこで, 図8に示すように, Python言語の `exec` 関

数を用いることで, *nroute* の値に関わらず同一の記述とすることができる. ただし, この書き方が若干読みにくい点がデメリットと考えられるが, 実行性能としてはどちらの記述方法でも同一であると考えられる.

4 評価

章 2 の節 2.2 で策定した問題において, $nroute = 4$ で, $ncity = 16$ および $ncity = 32$ の場合に対し, 最適解を探索するタスクを, Fixstars AE において 10 回実行した実行時間の分布を, 箱ひげ図の形式で図 9 に示す. 図の中で, tsp11 は章 3 の節 3.1 で述べた決定変数の固定も削減をしない場合を, tsp21 は章 3 の節 3.2 で述べた一部の決定変数の値を固定した場合を, tsp22 は章 3 の節 3.3 で述べた決定変数を生成を削減した場合を示す.



(a) 32 cities tour with 4 routes (b) 16 cities tour with 4 routes

図 9: Comparison of elapsed time of CVRP on Fixstars AE

図に示すように, $ncity = 32$ の場合, tsp22 は tsp11 よりも高速になっている. 4 ルートの場合, tsp22 の決定変数の数は tsp11 の約半分であるので, 探索範囲が削減されていることが理由と思われる. ただし, 図に示しているように, 計測時間が 7.9 秒のように, 数回実行する毎に外れ値が計測されている. おそらく, 今回のデータであれば, 各ルートでほぼ同等の訪問先になるので, tsp11 であれば, 決定変数の決定方法に偏りがないが, tsp22 の場合は, 元々決定変数の割当が少ないルートがあり, そのルートの上の決定変数は 1 になりやすく, 探索方法がランダムであれば, 探索に時間がより多く場合があると考えられ, tsp11 よりも遅くなる場合が発生している.

tsp21 は tsp11 よりもやや高速であるが. tsp22 よりは低速である. 決定変数の数は tsp11 と同じであるが, 一部の決定変数を固定化しているので, 最適値を探索する際には探索対象から除外していれば高速になるはずであるが, 計測結果からは大幅な削減にはなっていない.

一方, $ncity = 16$ の場合は, tsp21 および tsp22 の tsp11 の優位性は小さくなっている. 決定変数による探索の時間の削減より, 求解のためのオーバーヘッドの方が多いのが原因と思われる. よって, より大きい問題では, 決定変数削減の効果はより顕著になると考えられる.

今回の実験では, 最適な経路は, 各ルートでほぼ均等な数の都市を訪問することが最適解になっているものであり, tsp11 と tsp22 にしたがってランダム探索することで最適解を発見できる確率を比較す

る. tsp11 でランダム探索によって最適な配置を選択できる確率は, $ncity \times nroute$ から $ncity$ を選ぶ場合の数を分母とし, $ncity$ から連続した $\frac{ncity}{nroute}$ 個の訪問先を選ぶ場合の数を $nroute$ 乗した値を分子とする値で表される. 一方, tsp22 での確率は, $ncity \times \sum_{n=1}^{ncity} \frac{1}{n}$ から $ncity$ 選ぶ場合の数を分母とし, $ncity$ から連続した $ncity/nroute$ 個の訪問先を選ぶ場合の数, $ncity/2$ から連続した $ncity/nroute$ 個の訪問先を選ぶ場合の数, $ncity/3$ から連続した $ncity/nroute$ 個の訪問先を選ぶ場合の数... $ncity/nroute$ から連続した $ncity/nroute$ 個の訪問先を選ぶ場合の数をかけたものを分子とする値で表される. $nroute = 3, ncity = 7$ および $nroute = 4, ncity = 13$ の場合を表 1 に示す.

表 1: Estimates of probability in the case of a random search

pattern	tsp11	tsp22	rate
$nroute = 3, ncity = 7$	0.067	0.216	3.215
$nroute = 4, ncity = 13$	1.43E-07	1.54E-5	107.2

表より, ランダム探索の場合であっても, 決定変数が均一な tsp11 の方よりも tsp22 よりの方が高い確率で発見でき, 特に問題サイズが大きくなるにつれてその傾向が高まる. よって, Fixstars AE での実験結果からも tsp22 の方が高速であることから, 決定変数の削減の効果は大きいと考えられる.

5 おわりに

量子コンピューティングの方式のひとつとして量子アニーリング方式が注目され, 組合せ最適化問題へ実用的な利用が進んでいる. 本稿では, CVRP に対して決定変数の削減の効果を 2 個の手法で実装し, Fixstars AE で性能を評価した. ルート数を考慮して決定変数を削減する場合に, 決定変数の一部の値を固定化して削減する方法と決定変数の生成自体を削減する方法であれば, 前者の方法は決定変数削減による高速化の効果は限定的であるが, 後者の方法は, 訪問都市数に依存するものの, 概ね 2 倍程度の高速化が達成できた. しかし, 後者の方法のプログラムの記述方法自体が煩雑なので, API などのプログラミング環境の改善が望まれる.

今後は, CVRP 以外の組合せ問題に対して, 決定変数の削減などを含む高速化手法の検討や実用レベルのアプリケーションでの高速化および D-Wave のような実マシンでの実装などをすることが必要になる.

謝辞

本研究の一部は JSPS 科学研究費 (基盤研究 (C) 23K02671 (2023-2025)) 及び私立大学等経常費補助金特別補助「大学間連携等による共同研究」による.

参考文献

[1] S. S. Gill, A. Kumar, H. Singh, M. Singh, K. Kaur, M. Usman and R. Buyya. “Quantum computing: A taxonomy, systematic review and future directions,” https://arxiv.org/ftp/arxiv/papers/2010/2010.15559.pdf (As of 2024/02/21).

[2] “https://amplify.fixstars.com/en/,” (As of 2024/02/21).

[3] T.K. Ralphs, L. Kopman, W.R. Pulleyblank and L.E. Trotter, Jr, “On the capacitated vehicle routing problem,” *Mathematical Programming*. vol. 94, no. 2, pp. 343-359, 2003.

[4] N. Mori and S. Furukawa, “Quantum annealing for the adjuster routing problem,” https://doi.org/10.3389/fphy.2023.1129594, *Frontiers in Physics*, vol. 11, 13 pages, 2023.

[5] Fixstars Amplify, “Capacitated Vehicle Routing Problem (CVRP),” https://amplify.fixstars.com/en/demo/cvrp (As of 2024/02/21).

[6] R. Harikrishnakumar, S. Nannapaneni, N. H. Nguyen, J. E. Steck and E. C. Behrman, “A quantum annealing approach for dynamic multi-depot capacitated vehicle routing problem,” https://arxiv.org/pdf/2005.12478, 2020.

[7] K. Masuda, Y. Tsuyumine, T. Kitada, T. Hachikawa and Tsuyoshi HAGA, “Optimization of delivery plan by quantum computing,” *Sumitomo Electric Technical Review*, no. 96, pp. 85-88, 2023.