卒業研究報告書

題目

３次元Ｎクイーンの解の存在の検証

指導教員

石水 隆　助教

報告者

07-1-037-0106

前波 大貴

近畿大学理工学部情報学科

平成 24 年 1 月 31 日提出

概要

本研究では、代表的な組み合わせ最適化問題である N クイーン問題を拡張した 3 次元 N クイーン問題において、解であるクイーンの最大配置可能数に法則性があるのかを検証する。また、本研究では、N クイーン問題を解く一般的なバックトラック法を 3 次元 N クイーン問題に則してのアルゴリズム・プログラムの改良も目的とする。

膨大な計算時間が予想される 3 次元 N クイーン問題の解の探索に対し、アルゴリズムでの改良は、現時点での探索で判明しているクイーンの最大配置可能数を用いて、現在の探索手順からを超える新たなクイーンの最大配置可能数の存在の有無の判定を出し、無いのであれば、次の探索手順に移り、現在の探索手順を省くようにした。また、立体チェス盤を反転・回転した場合に同一となる配置パターンを極力抑えて探索することにより計算時間の短縮を図った。プログラムでの改良は、一定時間で探索途中のデータをファイルに書き込み、探索途中で終了してもファイルの書き込み時点からバックトラック法での再探索を可能にして、何らかの原因での探索途中の終了の対策をした。

改良したバックトラック法を用いても 3 次元 N クイーン問題の最大配置可能数に法則性が見出せなかったが、大幅な計算時間の短縮に成功した。今後の課題として、3 次元 N クイーン問題はサイズ N が大きくなると指数的に探索時間が増加するので、バックトラック法に代わる新たな探索方法での探索、または抜本的なバックトラック法の改良、もしくは、実行環境の改善を行うことが挙げられる。また、N と最大配置可能数 m との関係については、得られた結果が少ないため充分な検証を行うことができないため、より大きな N に対する解を求めて N と m の関係の検証を行うことも課題である。

目次

1　序論

1.1 本研究の背景

N クイーン問題[1]は、縦・横・斜めの 8 方向に直進できるチェス駒のクイーンを、N×N マスのチェス盤上に、お互いの駒の移動可能範囲を侵さないように N 個置く問題であり、代表的な組み合わせ最適化問題である。この N クイーン問題を拡張した 3 次元 N クイーン問題は、N×N×N マスの立体のチェス盤上に、2 次元クイーンの移動方向を踏襲した 26 方向に直進できるクイーンを、お互いの駒の移動可能範囲を侵さないように複数個配置する問題である。3 次元 N クイーン問題の解とは N×N×N のチェス盤にクイーンを存在出来る最大数配置した配置パターンである[2]。

2 次元 N クイーン問題では、N≧4 の場合 N×N マスに N 個のクイーンを配置する解が存在する。しかし、3 次元 N クイーン問題では、N^2 個のクイーンを配置する解は一般に存在せず、各 N の値に対してお互いの移動可能範囲を侵さないように配置できるクイーンの最大数 m（$\leqq N^2$）は明らかでは無い。したがって、3 次元 N クイーン問題は、与えられた N の値に対し、クイーンの最大配置可能数 m と、その時のチェス盤上の配置パターンを解とする。

3 次元 N クイーン問題に対しては、N=11 の場合に、最大配置可能数 $m=N^2$ の解があると示唆されている[3]。本研究で用いる探索方法のバックトラック法[1]は、可能である手順から順に解を探索し、その手順で解が求められなければ、一つ前の状態に戻って別の手順から解を求める探索方法である。

1.2 本研究の目的

本研究の目的は各 N に対して 3 次元 N クイーン問題の解を求め、3 次元 N クイーン問題においてクイーンの最大配置可能数 m に法則性があるのかを検証し、より複雑な組み合わせ最適化問題として確立させることである。また、もう 1 つの研究目的として、本研究では、膨大な計算時間が予想される 3 次元 N クイーン問題に対し、何らかの原因から探索途中でプログラムの異常終了が起こった場合の対策と、計算時間の短縮を図る。

1.3 本報告書の構成

本報告書では、まず 2 章において対象とする問題である 3 次元 N クイーン問題について述べる。3 章に本研究で提案する 3 次元 N クイーン問題を解くアルゴリズムおよび本研究で作成した 3 次元 N クイーン問題を解くプログラムについて述べる。3.1 節、3.2 節で計算時間の短縮を図るためのアルゴリズム上の改良点を記載している。3.3 節では何らかの原因から探索途中でプログラムで異常終了が起こった場合のプログラム上の対策について述べ、4 章では研究内容を用いたプログラム・アルゴリズムを実行させての結果および考察ついて述べ、5 章ではまとめおよび今後の課題を記載している。

2　3 次元 N クイーン問題

本章では、本研究が対象とする 3 次元 N クイーン問題を定義する。

まず 2 次元 N クイーン問題について述べる。チェスの駒の一つであるクイーンはサイズ 8×8 のチェス盤上を縦横斜め方向に他の駒あるいは盤端に当たるまで移動できる。8 クイーン問題は、チェス盤上に 8 個のクイーンを互いの移動範囲を妨げないように配置する問題である。N クイーン問題は 8 クイーン問題の拡張であり、サイズ N×N のチェスに N 個のクイーンを互いの移動範囲を妨げないように配置する問題である。N≧4 である N に対する N クイーン問題では、N 個のクイ

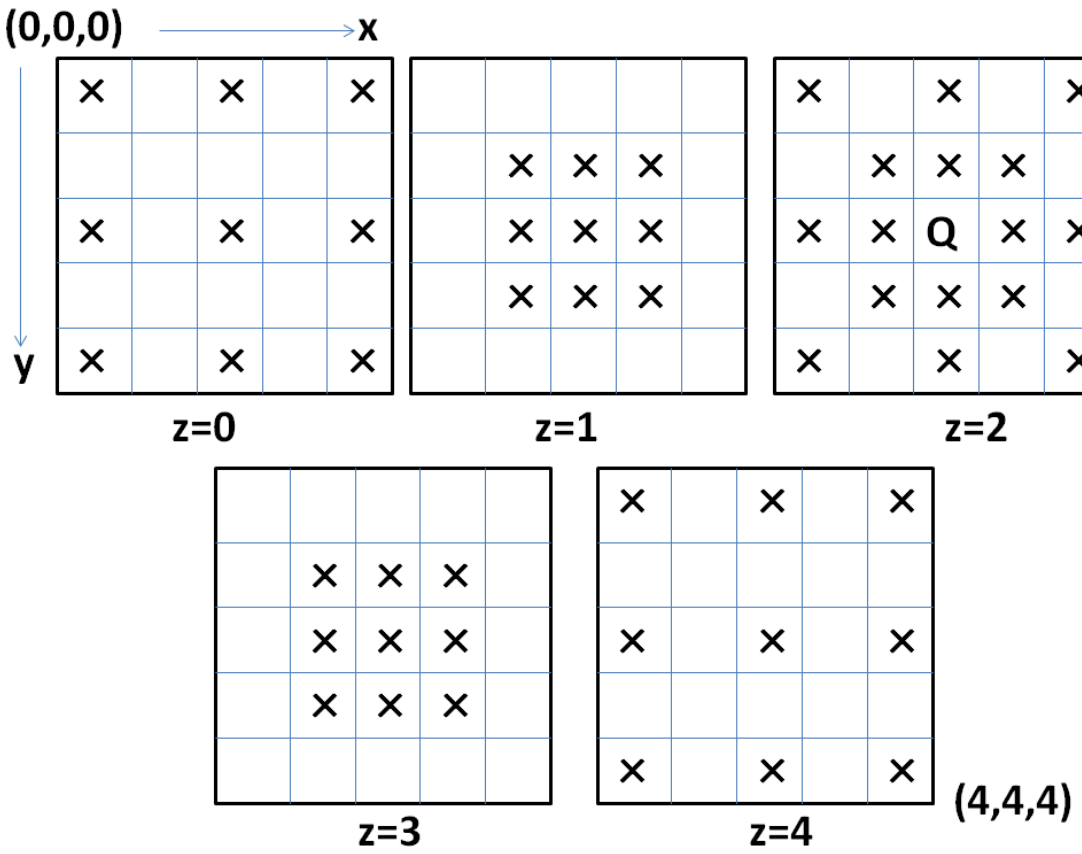

図 1　N=5 の三次元チェス盤の中心座標(2,2,2)に配置したクイーンの移動可能範囲

ーンを配置する解が複数個存在する。

N クイーン問題を計算機上で解く場合の代表的な手法にバックトラック法がある。バックトラック法はある解を求めるときに、可能性のある手順を順に試していき、その手順では解が求められないと判明した時点で一つ前の状態に戻って別の手順を試す方法である。最初の探索出発地点の座標を(0,0,0)とし、x 方向、y 方向、z 方向の優先順に探索をする。3 次元 N クイーン問題は、チェス盤よびクイーンの移動範囲を 3 次元に拡張したものである。3 次元クイーンは、27 個のベクトル(x, y, z) (x, y, z={-1, 0, 1})のうち 0 ベクトル(0, 0, 0)を除く 26 個のいずれか 1 つの方向に盤端に当たるまで移動できる。図 1 に、サイズ 5×5×5 のチェス盤において座標(2, 2, 2)のマスにクイーンがあるとき、クイーンが移動できるマス目を示す。図 1 中の Q はクイーンのあるマス、×はクイーンが移動できるマスを表す。

3　研究内容

本章では、本研究で提案する 3 次元 N クイーン問題を解くアルゴリズムおよび本研究で作成した 3 次元問題を解くプログラムについて述べる。2 章で述べた通り膨大な計算時間が予想される 3 次元 N クイーン問題に対し、本研究で提案するアルゴリズムでは探索範囲を枝切りするための改良を施し、本研究で作成したプログラムではプログラムの異常終了に対する対策を施している。

3.1 次元 N クイーン問題を解くアルゴリズム

3.1.1 基本的なアルゴリズム

本節では 3 次元 N クイーン問題を解く基本的なアルゴリズムについて述べる。本研究では、バックトラック法を用いて 3 次元 N クイーン問題を解く。以下に基本アルゴリズムを示す。

[基本アルゴリズム]

Step1:最初の探索出発地点の座標を(0,0,0)とし、x 方向、y 方向、z 方向の優先順に探索をする。

Step2:全ての探索順路が存在しなくなるまで、以下のステップを繰り返す。

Step2.1:チェス盤から探索順路上にクイーン設置可能マス存在する場合、以下のステップを繰り返す。

Step2.1.1:x 方向、y 方向、z 方向の優先順にクイーン設置可能マスにクイーンを設置する。

Step2.2.2:クイーンの移動可能範囲を設置不可マスにする。

Step2.2: 最後に設置したクイーンを取り除き、設置した次の座標以降から探索を再開する。

3.1.2 クイーン設置個数から判定した現在の探索手順の省略

バックトラック法では、探索範囲を狭めるために枝切りが用いられる。枝切りとは探索手順に必要でない無駄な探索があればそれを切り捨てる方法である。

以下では、3.1 節に挙げた基本アルゴリズムに枝切りを加えて探索範囲を狭める手法について述べる。

3.1 節の基本アルゴリズムでは、探索中クイーンを置けなくなるまで配置可能場所を探して置き、置く場所が無くなればバックトラックを行う。しかし、探索中一部のクイーンを配置したときに、場合によっては残りのクイーンを置く前に、現在の配置からでは解が存在しないことが判定できる。探索中にそれまでに得られたクイーンの最大配置可能個数 t、現時点の探索でチェス盤に配置されているクイーンの個数 q、プログラム上探索予定の空きマスの数値 e に対し t−q ＞ e ならば、現時点のクイーン配置パターンからの探索では今までのクイーンの最大配置可能個数を超える事がない。従って、その配置パターンからクイーンの配置場所を探しても解となる配置パターンには到達し得ないことがわかる。この場合、現在の配置パターンからの探索を打ち切り、次の手順から検索することにより省略分の計算時間の短縮を図っている。

3.1.3 クイーンの初期配置の限定

あるクイーンの配置パターンが 3 次元 N クイーン問題の解となるとき、そのパターンを 3 次元上で回転、反転して得られるパターンも、やはり解となる。したがって、のようにある配置パターンを回転・配置して得られるパターンを同一のものとして扱うことにより、探索範囲を狭めることができる。本研究では、図 2 に示すように番号付けされた座標位置が同一の番号の座標位置と立体チェス盤の中心から対称となる位置になる。したがって、本研究でのバックトラック法のアルゴリズムより(0,0,0)から x、y、z 方向に進む探索手順から十分な探索を行うために、図 3 に示すように初期クイーンの位置を限定することにより、立体チェス盤を反転・回転した場合に同一となる配置パターンを極力抑えて探索することにより計算時間の短縮を図る。

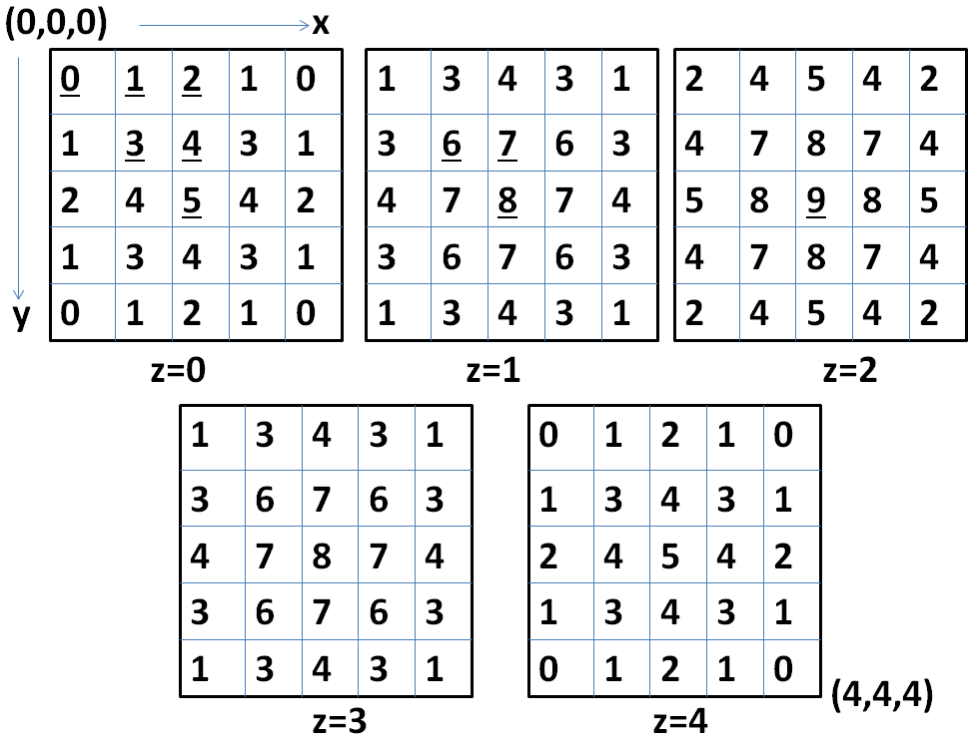

図 2　N=5 の立体チェス盤の中心(2,2,2)から対称となる番号群

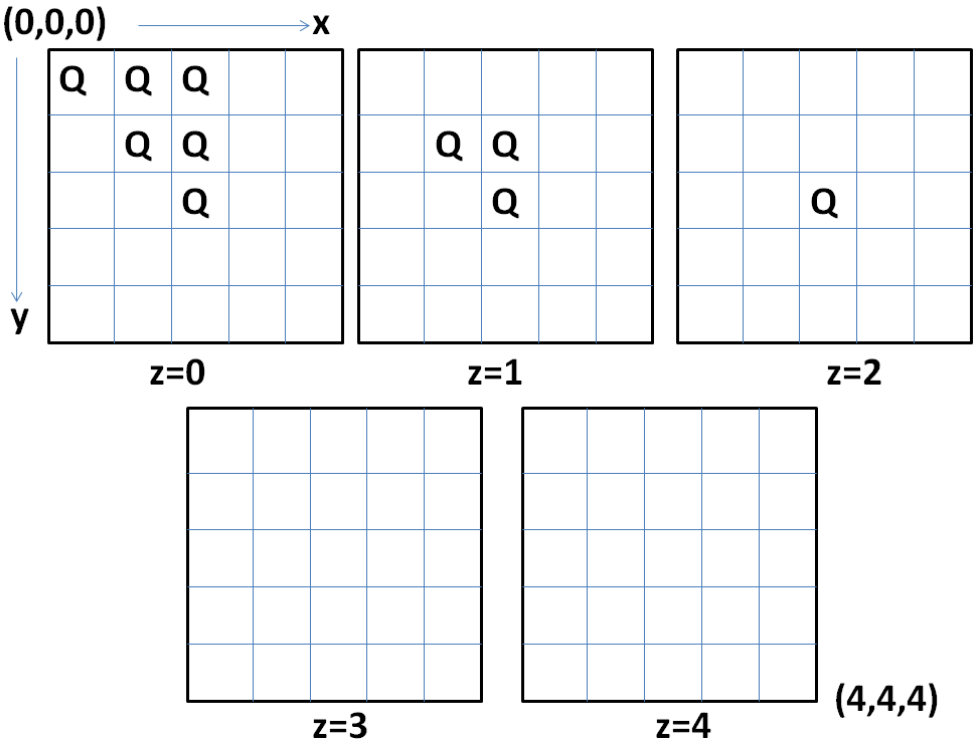

図 3　N=5 の限定クイーン初期配置群

3.1.4 改良アルゴリズム

3.1.1 節の基本アルゴリズムに、3.1.2 節および 3.1.3 節の改良点を加えた改良アルゴリズムを以下に示す。

[改良アルゴリズム]

Step1:最初の探索出発地点の座標を(0,0,0)とし、x 方向、y 方向、z 方向の優先順に探索をする。

Step2:全ての探索順路が存在しなくなるまで、以下のステップを繰り返す。

Step2.1: (0,0,0),(N/2-1,0,0),(N/2-1,N/2-1,0),(N/2-1,N/2-1,N/2-1)を頂点とした(N/2 は切り上げ)三角錐状の限定クイーン初期配置群内に探索順にクイーンを初期配置する。

Step2.2:初期配置したクイーンから派生する探索順路が存在しなくなるまで、以下のステップを繰り返す。

Step2.2.1: 3.1.2 より t-q<e の条件を満たす限り、以下のステップを繰り返す。

Step2.2.1.1:x 方向、y 方向、z 方向の優先順にクイーン設置可能マスにクイーンを設置する。

Step2.2.1.2:クイーンの移動可能範囲を設置不可マスにする。

Step2.3:クイーンが今までの最大配置数を超えた場合、その解を記録する。

Step2.4:最後に設置したクイーンを取り除き、設置した次の座標以降から探索を再開する。

3.2 3 次元 N クイーン問題を解くプログラム

本研究では、3.1.4 節で示した改良アルゴリズムを(言語)を用いて実装した。しかし、3.1.2 節および 3.1.3 節で述べた改良を加えても N が大きい場合は探索範囲が広くなり、プログラムの実行時間が長くなる。プログラムを長時間の実行すると、計算機の熱暴走や停電など、予期せぬ事態によりプログラムが中断される確率が高くなってしまう。そこで本研究では作成したプログラムは、長時間の実行時にプログラムが異常終了する事態に備えて一定時間毎に探索途中のデータをファイルに記録する。実行再開時はそのファイルデータより記録した時点から再探索する機能を設けた。

付録に本研究で作成した 3 次元 N クイーン問題を解くプログラムを示す。

以下に基本的な探索を行うための関数を示す。

N ： int 型のチェス盤 N のサイズ

pos[N][N][N] ： 探索に使う仮想チェス盤の三次元配列

maxpos[N][N][N] ： 存在するクイーン最大個数設置の解を納める三次元配列

各三次元のチェス盤はサイズは N×N×N。最上段、二次元上の左上を 0 として pos[z][y][x]。

x が二次元上の左右、y が二次元上の上下、z が三次元の深さ。収納されている数字の意味は、1 はクイーン設置マス、0 はクイーン設置可能マス、-1 はクイーン設置不可マス。

Q ： int 型の探索現在のクイーンの数

QMax ： int 型の判明している最大の Q の個数

startT,nowT,saveT,endT ： time_t 型の startT は探索開始時,nowT,saveT は現在の探索データをファイルに書き込み時 、endT は探索終了時のための計算時間測定用変数

Check() ：void 型 全てのクイーンの可動範囲を-1 にする関数

Search(int a,int b,int c) ： int 型の(c,b,a)からの探索手順に空きマスを数える関数。空きマスの値を返す。

Show() ： void 型の探索現在の盤様子を表示する関数

MaxShow() ： void 型のクイーン最大個数設置の解(設置パターン)を表示する関数

Save() ： void 型の QSet()起動時に一定時間を経過すると現在の探索状況を報告をし、ファイルに現在の探索データを書き込む関数。nowT で現在の時間、saveT で前回のセーブ時間で一定時間を計測する。

MaxQSet() ： void 型のクイーン最大個数設置パターン(盤)を収納する関数

QSet(int a,int b,int c) ： void 型の再帰的呼び出しで(c,b,a)からクイーンを設置する関数。空きマスにクイーンを設置した後にその座標(c,b,a)から Qset()を起動する。

Clear() ： void 型の 盤(pos)をリセット(0 に)する関数
Start() ： void 型のクイーン初期位置を決定し、Qset()に配置探索させる関数

前回の探索データから再探索を行うことに関する関数を示す。

loadcount: int 型前回の各クイーンの位置から Qset()を再帰的に起動させるための機能に使うカウント。前回の探索中のクイーンの数と同値。Restart()と ReQSet()で前回の再帰的起動の再現をさせる。
savetime ： long int 型で前回までの計算時間を納める。
ReSave() ： void 型の一定時間を超えると現在報告をするマスする関数。savetime で前回の計算時間を拡張させた。
ReQSet(int a,int b,int c) ： void 型の再探索の機能を拡張した再帰的呼び出しで(c,b,a)からクイーンを設置する関数。前回のクイーン配置位置から loadcount の数値分 ReQSet()を再帰的起動させ、疑似的に前回までの実行状況を再現させる。
ReStart() ： void 型の再探索の機能を拡張したクイーン初期位置を決定し、ReQset()に探索させる関数。
load() ：ファイルからデータを読み取る void 型の関数。前回のチェス盤の探索状況を pos[N][N][N]に、前回までの計算時間を savetime に、前回の探索で判明した解を pos[N][N][N]に収納する。

4 結果と考察

本章では、付録で示したプログラムを実行させて得られた結果について述べる。
表 1 に、サイズ 4,5,6 の 3 次元チェス盤上で N クイーン問題を解いたときの、クイーンの最大配置可能数および探索時間を示す。表 1 より探索時間が指数的に増えているのがわかる。2 次元 N クイーン問題[2]と比較しても、3 次元 N クイーン問題の探索時間は前回の探索時間の増加量が大きいと予想される。少なくとも前回から 100 倍以上の探索時間を要するため、次の探索対象の N=7 の探索時間は、今回改良したバックトラック法を用いても年単位の探索時間が予想される。
また N=5 については、基本アルゴリズムで 3300 秒の探索時間が掛った。3.1.2 の探索手順の省略方法を利用することにより 660 秒まで短縮し、加えて 3.1.3 のクイーン限定初期配置の方法を用いることで、224 秒まで探索時間の短縮に成功した。

表 1 クイーン最大数mおよび探索時間

N	m	探索時間
4	7	1 秒
5	13	224 秒
6	21	300 時間以上

5 今後の課題

本研究では、3 次元 N クイーン問題を解くプログラムを作成した。本研究で作成したプログラムはバックトラック法を用いているため､サイズ N が大きくなると指数的に探索時間が増加する。今後の課題としては、バックトラック法に代わる新たな探索方法での探索、または抜本的なバックトラック法の改良、もしくは、実行環境の改善を行うことである。また、N と最大配置可能数 m との関係については、得られた結果が少ないため充分な検証を行うことができないため、より大きな N に対する解を求めて N と m の関係の検証を行うことも課題である。

謝辞

本研究を完成するにあたって、御指導下さった石水隆先生と支援を受けた情報論理工学研究室の皆様に対し深く感謝を申し上げます。

参考文献

[1] 柴田望洋：明解 Java によるアルゴリズムとデータ構造, pp.168-179, (2007).

[2] 吉瀬謙二,「N-queens Homepage in Japanese 」： 電気通信大学, 2004, http://www.arch.cs.titech.ac.jp/~kise/nq/index.htm

[3] 岡田章三：m 次元 n クイーン問題, 岐阜高専紀要 第 37 号, pp.13-16, (2002).

[4] 岡田章三：m 次元 n クイーン問題に関する計算例と予測, 岐阜高専紀要 第 38 号, pp11－14, (2003).

[5] 岡田章三：m 次元 n クイーン問題に関する研究, 岐阜高専紀要 第 39 号, pp7－9, (2004).

[6] 岡田章三：m 次元 n クイーン問題に関する報告, 岐阜高専紀要 第 40 号, pp1－3, (2005).

付録

本研究で作成したプログラムのソースファイルを以下に示す。

付録1. (0,0,0)から3次元Nクイーン問題の解を探索するプログラム

先ず、アルゴリズムの改良とデータをファイルの書き込む機能を施した、最初の(0,0,0)から探索するプログラムを示す。

```
/* バックトラック法を改良し時間短縮を図った3時限Nクイーン問題を解くアルゴリズムのプログラム */

#include <iostream>
#include <fstream>
#include <strstream>
#include <string>
#include<time.h>

#define N 5 // チェス盤Nのサイズ

using namespace std;

int pos[N][N][N];        // 探索に使う仮想チェス盤の三次元配列
int maxpos[N][N][N];    // 存在するクイーン最大個数設置の解を納める三次元配列

/* 三次元のチェス盤。サイズはN×N×N。最上段、二次元上の左上を0としてpos[z][y][x]。
   xが二次元上の左右、yが二次元上の上下、zが三次元の深さ。

(例)N=3ならば
最上段(z=0)
(0,0,0)(0,0,1)(0,0,2)
(0,1,0)(0,1,1)(0,1,2)
(0,2,0)(0,2,1)(0,2,2)
z=1
(1,0,0)(1,0,1)(1,0,2)
(1,1,0)(1,1,1)(1,1,2)
(1,2,0)(1,2,1)(1,2,2)
最下段(z=2)
(2,0,0)(2,0,1)(2,0,2)
(2,1,0)(2,1,1)(2,1,2)
(2,2,0)(2,2,1)(2,2,2)
```

```
収納されている数字の意味は、
1 はクイーン設置マス、0 はクイーン設置可能マス、-1 はクイーン設置不可マス。
*/

int Q = 0;    // 探索現在のクイーンの数
int QMax = 0;    // 判明している最大の Q の個数

/*
startT は探索開始時、nowT、saveT は現在の探索データをファイルに書き込み時、
endT は探索終了時のための計算時間測定用変数
*/
time_t startT,nowT,saveT,endT;



/**
 * 全てのクイーンの可動範囲を-1 にする関数
 */
void Check(){
    // コマのないマスをクリア
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            for(int k=0;k<N;k++){
                if(pos[i][j][k]!=1) pos[i][j][k] = 0; //クイーン設置マス以外を 0 に
            }
        }
    }
    // 各クイーンの可動範囲を-1 に
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            for(int k=0;k<N;k++){
                if(pos[i][j][k]==1){
                    for(int l=0;l<N;l++){
                        /* クイーンの各移動可能方向に-1 を入れる。*/
                        // x 軸方向
                        if(pos[i][j][l]!=1)pos[i][j][l]=-1;
                        // y 軸方向
                        if(pos[i][l][k]!=1)pos[i][l][k]=-1;
                        // z 軸方向
```

```
if(pos[l][j][k]!=1)pos[l][j][k]=-1;
// 二次元左上方向。(x,0,0)方向。
if(j-l>=0&&k-l>=0 && pos[i][j-l][k-l]!=1)pos[i][j-l][k-l]=-1;
// 二次元右上方向。(x,0,N-1)方向。
if(j-l>=0&&k+l<N&&pos[i][j-l][k+l]!=1)pos[i][j-l][k+l]=-1;
// 二次元右下方向。(x,N-1,N-1)方向。
if(j+l<N&&k+l<N&&pos[i][j+l][k+l]!=1)pos[i][j+l][k+l]=-1;
// 二次元左下方向。(x,N-1,0)方向。
if(j+l<N&&k-l>=0&&pos[i][j+l][k-l]!=1)pos[i][j+l][k-l]=-1;
// (0,0,x)方向。
if(i-l>=0&&j-l>=0&&pos[i-l][j-l][k]!=1)pos[i-l][j-l][k]=-1;
// (0,x,N-1)方向。
if(i-l>=0&&k+l<N&&pos[i-l][j][k+l]!=1)pos[i-l][j][k+l]=-1;
// (0,N-1,x)方向。
if(i-l>=0&&j+l<N&&pos[i-l][j+l][k]!=1)pos[i-l][j+l][k]=-1;
// (0,x,0)方向。
if(i-l>=0&&k-l>=0&&pos[i-l][j][k-l]!=1)pos[i-l][j][k-l]=-1;
// (N-1,0,x)方向。
if(i+l<N&&j-l>=0&&pos[i+l][j-l][k]!=1)pos[i+l][j-l][k]=-1;
// (N-1,x,N-1)方向。
if(i+l<N&&k+l<N&&pos[i+l][j][k+l]!=1)pos[i+l][j][k+l]=-1;
// (N-1,N-1,x)方向。
if(i+l<N&&j+l<N&&pos[i+l][j+l][k]!=1)pos[i+l][j+l][k]=-1;
// (N-1,x,0)方向。
if(i+l<N && k-l>=0&&pos[i+l][j][k-l]!=1)pos[i+l][j][k-l]=-1;
// (0,0,0)方向。
if(i-l>=0 && j-l>=0 && k-l>=0 && pos[i-l][j-l][k-l]!=1) pos[i-l][j-l][k-l] = -1;
// (0,0,N-1)方向。
if(i-l>=0 && j-l>=0 && k+l<N && pos[i-l][j-l][k+l]!=1) pos[i-l][j-l][k+l] = -1;
// (0,N-1,N-1)方向。
if(i-l>=0 && j+l<N && k+l<N && pos[i-l][j+l][k+l]!=1) pos[i-l][j+l][k+l] = -1;
// (0,N-1,0)方向。
if(i-l>=0 && j+l<N && k-l>=0 && pos[i-l][j+l][k-l]!=1) pos[i-l][j+l][k-l] = -1;
// (N-1,0,0)方向。
if(i+l<N && j-l>=0 && k-l>=0 && pos[i+l][j-l][k-l]!=1) pos[i+l][j-l][k-l] = -1;
// (N-1,0,N-1)方向。
if(i+l<N && j-l>=0 && k+l<N && pos[i+l][j-l][k+l]!=1) pos[i+l][j-l][k+l] = -1;
// (N-1,N-1,N-1)方向。
if(i+l<N && j+l<N && k+l<N && pos[i+l][j+l][k+l]!=1) pos[i+l][j+l][k+l] = -1;
```

```
                        // (N-1,N-1,0)方向。
                        if(i+l<N && j+l<N && k-l>=0 && pos[i+l][j+l][k-l]!=1) pos[i+l][j+l][k-l] = -1;
                    }
                }
            }
        }
    }
}


/**
 * (c,b,a)からの空きマスを数える関数
 * @param mas:空きマスの数
 * @return (c,b,a)からの探索手順上に存在する空きマス
 */
int Search(int a,int b,int c) {
    int mas=0; // 空きマスの数
    bool first = true; // 最初の探索座標を(c,b,a)にするための判定
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            for(int k=0;k<N;k++){
                // 最初の探索座標を(c,b,a)にし以降を x.y,z 方向の順に空きマスを探索
                if(first){
                    i= a;
                    j= b;
                    k= c;
                    first = false; //最初の探索座標を(c,b,a)にしたので false
                }
                if(pos[i][j][k]==0) mas++;
            }
        }
    }
    return mas;
}

/**
 * 探索現在の盤様子を表示する関数
 */
void Show(){
```

```
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            for(int k=0;k<N;k++){
                if(pos[i][j][k]==1){    // クイーンがあるマス
                    cout << "◎";
                }else if(pos[i][j][k]==0){    // クイーンがないマス
                    cout << "□";
                }else{    // クイーンを置くことができないマス
                    cout << "×";
                }
            }
            cout << endl;
        }
        cout << endl;
    }
    cout << "＊＊＊＊＊＊＊＊＊＊" << endl << endl;
}

/**
 * クイーン最大個数設置の解(設置パターン)を表示する関数
 */
void MaxShow(){
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            for(int k=0;k<N;k++){
                if(maxpos[i][j][k]==1){    // クイーンがあるマス
                    cout << "◎";
                }else if(maxpos[i][j][k]==0){    // クイーンがないマス
                    cout << "□";
                }else{    // クイーンを置くことができないマス
                    cout << "×";
                }
            }
            cout << endl;
        }
        cout << endl;
    }
}
```

```
/*  */
/**
 * QSet()起動時に一定時間を経過すると現在の探索状況を報告をし、ファイルに現在の探索データを書き込む関数
 * @param ST:前の報告時間 saveT から現在の時間 nowT まで経過時間
*/
void Save() {
    time(&nowT); // 現在の時間を計測
    int ST = (int) difftime(nowT,saveT); // 前の報告時間 saveT から現在の時間 nowT まで経過時間を計測
    // 一定時間を経過した場合に処理する。
    if (ST>=900) {
        //  現在の探索状況を報告
        cout << endl;
        cout << "現在の 3D-N-Queen 問題(N=" << N << ")においての探索様子" << endl;
        cout << "現時点での計算時間は" << difftime(nowT,startT) << "秒" << endl;
        cout << "現在のボード全体図" << endl;
        Show();
        cout << "現時点での解の存在する最大の Queen の個数は" << QMax << "個" << endl;
        cout << "現時点での Queen を可能な限り最大個数で設置した場合のボード全体図" << endl;
        MaxShow();
        //ファイルに現在のデータを書き込む
        // 現在の配列情報
        ofstream ofsA( "saveboad.txt" );
        for(int i=0;i<N;i++) for(int j=0;j<N;j++) for(int k=0;k<N;k++) ofsA << pos[i][j][k] << endl;
        // 計算時間情報
        ofstream ofsB( "savetime.txt" );
        ofsB << difftime(nowT,startT) << endl;
        // 最大解の配列情報
        ofstream ofsC( "savemaxboad.txt" );
        for(int i=0;i<N;i++) for(int j=0;j<N;j++) for(int k=0;k<N;k++) ofsC << maxpos[i][j][k] << endl;
        //現状報告、ファイル書き込みされた現在の時間の記録
        time(&saveT);
    }
}

/**
 *クイーン最大個数設置パターン(盤)を収納する関数
 */
void MaxQSet(){
    for(int i=0;i<N;i++) for(int j=0;j<N;j++) for(int k=0;k<N;k++) maxpos[i][j][k] = pos[i][j][k];
```

```
}

/**
 *再帰的呼び出しで(c,b,a)からクイーンを設置する関数
 * @param first:最初の探索座標を(c,b,a)にするための判定
 */
void QSet(int a,int b,int c){
    bool first = true; // 最初の探索座標を(c,b,a)にするための判定
    // x.y,z 方向の順に空きマスを探索
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            for(int k=0;k<N;k++){
                if(first){
                    i= a;
                    j= b;
                    k= c;
                    first = false; //最初の探索座標を(c,b,a)にしたので false
                }
                // 空きマス発見後の処理
                if(pos[i][j][k]==0){
                    pos[i][j][k] = 1; // 座標にクイーンを置く
                    Q++; // クイーンを設置したので現在探索中のクイーンの数が増える
                    Check(); // 配置したクイーンの可動範囲を-1 に
                    // 現在のクイーンの数がこれまでのクイーンの最大設置可能数を超えていたら処理
                    if(Q>QMax){
                        QMax = Q; // クイーンの最大設置可能数を更新
                        Show();
                        //空きマスがなかったら、クイーンの最大設置可能数の可能性が濃厚なので解を maxpos に収納
                        if(Search(0,0,0)==0) MaxQSet();
                    }
                    // これから置く空きマスの数で現在のクイーン最大設置可能数を更新の可能性があるなら処理
                    if((QMax-Q)<Search(i,j,k)){
                        Save(); // QSet()を再帰的起動する前に報告するか確認
                        QSet(i,j,k); //再帰的に起動
                    }
                    pos[i][j][k] = 0; // 座標のコマを除外
                    Q--; // クイーンの数を減らす。
                    Check(); // 除いたクイーンの可動範囲を空きマスに戻す。
                }
```

```
            }
        }
    }
}

/**
 *盤(pos)をリセット(0に)する関数
 */
void Clear() {
    for(int i=0;i<N;i++) for(int j=0;j<N;j++) for(int k=0;k<N;k++) pos[i][j][k] = 0; //初期化
}

/**
 *クイーン初期位置を決定し、Qset()に探索させる関数
 */
void Start() {
    Clear();
    for(int i=0;i<N;i++) for(int j=0;j<N;j++) for(int k=0;k<N;k++) maxpos[i][j][k] = 0; //maxpos を初期化
    /*
       (0,0,0),(N/2-1,0,0),(N/2-1,N/2-1,0),(N/2-1,N/2-1,N/2-1)を頂点とした (N/2 は切り上げ)
       三角錐状のクイーン初期配置群からクイーンを配置し探索する。
    */
    int n = (N*5+5)/10; // N/2(切り上げ)を求める
    for(int i=0;i<n;i++) {
        for(int j=i;j<n;j++) {
            for(int k=j;k<n;k++) {
                pos[i][j][k] = 1;
                Q = 1; // クイーンを初期配置したので 1
                cout <<"現在、クイーン初期配置座標(" << k <<","<< j <<"," << i << ")より探索開始" << endl;
                Check(); // 配置したクイーンの可動範囲を-1 に
                if(Q>QMax){
                        QMax = Q;
                        Show();
                        if(Search(0,0,0)==0) MaxQSet();
                }
                QSet(i,j,k); // Qset()に再帰的探索をさせる
                Clear(); // クイーンは 1 個だけなので盤をリセットさせる
                Q = 0; // クイーンの数を 0 にする
            }
```

```
        }
    }
}

void main(){
    time(&startT); // 探索時間の開始
    Start(); // (0,0,0)から探索開始
    time(&endT); // 探索時間の終了

    //最終結果をファイルに書き込み
    // 計算時間情報
    ofstream ofstime( "endtime.txt" );
    ofstime << difftime(endT,startT) << endl;
    // 最大解の配列情報
    ofstream ofsmax( "endmaxboad.txt" );
    for(int i=0;i<N;i++) for(int j=0;j<N;j++) for(int k=0;k<N;k++) ofsmax << maxpos[i][j][k] << endl;

    // 最終報告
    cout << endl;
    cout << "3D-N-Queen 問題(N=" << N << ")において Queen を可能な限り最大個数で設置した時のボード全体図" << endl;
    MaxShow();
    cout << "3D-N-Queen 問題(N=" << N << ")において解の存在する最大の Queen の個数は" << QMax << "個" << endl;
    cout << "計算時間は" << difftime(endT,startT) << "秒" << endl;
}
```

ファイルの書き込み時点から再探索するプログラム

ファイルの書き込みをした時点の配置状況から再探索を行うプログラムソース部分を示す。

```
/* 前回のファイル書き込み時の時点から再探索を行うアルゴリズム。 */

int loadcount = 0;
/*
   前回の各クイーンの位置から Qset()を再帰的に起動させるための機能に使う。。
   前回の探索中のクイーンの数と同じ
*/

long int savetime = 0; // 前回までの計算時間を納める。

/*
 * 一定時間を超えると現在報告をするマスする関数
 * @param ST:前の報告時間 saveT から現在の時間 nowT まで経過時間
 */
void ReSave() {
    time(&nowT);
    int ST = (int) difftime(nowT,saveT);
    if (ST>=180) {
        cout << endl;
        cout << "現在の 3D-N-Queen 問題(N=" << N << ")においての探索様子" << endl;
        cout << "現時点での計算時間は" << difftime(nowT,startT) << "秒" << endl;
        cout << "合計計算時間は" << difftime(nowT,startT) + savetime << "秒" << endl;
        cout << "現在のボード全体図" << endl;
        display();

        cout << "現時点での解の存在する最大の Queen の個数は" << QMax << "個" << endl;
        cout << "現時点での Queen を可能な限り最大個数で設置した場合のボード全体図" << endl;
        maxdisplay();
        cout << "＊＊＊＊＊＊＊＊＊＊" << endl << endl;
        //ファイルに現在のデータを書き込む
        // 現在の配列情報
        ofstream ofsA( "saveboad.txt" );
        for(int i=0;i<N;i++) for(int j=0;j<N;j++) for(int k=0;k<N;k++) ofsA<< pos[i][j][k] << endl;
        // 計算時間情報
        ofstream ofsB( "savetime.txt" );
        // savetime と足した数値を書き込む
```

```
        ofsB << difftime(nowT,startT) + savetime << endl;
        // 最大解の配列情報
        ofstream ofsC( "savemaxboad.txt" );
        for(int i=0;i<N;i++) for(int j=0;j<N;j++) for(int k=0;k<N;k++) ofsC << maxpos[i][j][k] << endl;

        time(&saveT);
    }
}

/**
 * 再帰的呼び出しで(c,b,a)からクイーンを設置する関数
 * @param first:最初の探索座標を(c,b,a)にするための判定
 * @param nextQ:同位置のクイーンを認識させないためのもの。
 */
void ReQSet(int a,int b,int c){
    bool first = true;
    bool nextQ = true; // 同位置のクイーンを認識させないためのもの。
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            for(int k=0;k<N;k++){
                if(first){
                    i= a;
                    j= b;
                    k= c;
                    first = false;
                }
                // 前回で認識したクイーンを再認識させないようにしている
                if(pos[i][j][k]==1&&loadcount>0&&nextQ) nextQ=false;
                // 前回と同じクイーン設置座標位置から Qset()を再帰的に起動させる。
                else if(pos[i][j][k]==1&&loadcount>0) {
                    loadcount--; // クイーンを認識したので減少させる。
                    ReQSet(i,j,k);
                    pos[i][j][k] = 0;    // 座標のコマを除外
                    Q--;
                    Check();
                // 通常時の探索をする。
                }else if(pos[i][j][k]==0&&loadcount<=0){
                    pos[i][j][k] = 1;    // 座標にコマを置く
                    Q++;
```

```
                    Check();    // そのコマの可動範囲を-1 に
                    if(Q>QMax){
                        QMax = Q;
                        Show();
                        if(search(0,0,0)==0) MaxQSet();
                    }
                    if((QMax-Q)<Search(i,j,k)) {
                        ReSave();
                        ReQSet(i,j,k);
                    }
                    pos[i][j][k] = 0;    // 座標のコマを除外
                    Q--;
                    Check();
                }
            }
        }
    }
}

/*
 * クイーン初期位置を決定し、ReQset()に探索させる関数
 * @param n:N/2(切り上げ)を求める引数
 */
void ReStart() {
    Clear();
    int n = (N*5+5)/10; // N/2(切り上げ)を求める
    bool first = true;
    for(int i=0;i<n;i++) {
        for(int j=i;j<n;j++) {
            for(int k=j;k<n;k++) {
                if(first){
                    i= A;
                    j= B;
                    k= C;
                    first = false;
                }
                // 前回と同じクイーン設置座標位置から Qset()を再帰的に起動させる。
                if(pos[i][j][k]==1&&loadcount>0) loadcount--;
                else {
```

```
                    pos[i][j][k] = 1;
                    Q = 1;
                    Check();
                }
                cout <<"現在、クイーン初期配置座標(" << k <<","<< j <<"," << i << ")より探索開始" << endl;
                if(Q>QMax){
                    QMax = Q;
                    MaxShow();
                    if(search(0,0,0)==0) maxqset();
                }
                ReQSet(i,j,k);
                Clear();
                Q = 0;
            }
        }
    }
}

/* ファイルからデータを読み取る関数*/
void load(){
    //最近の配列情報ファイル読み取り
    ifstream ifsA("saveboad.txt");
    string buf;
    // 対応した座標にマス情報を配列に取り込む
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            for(int k=0;k<N;k++){
                if(ifsA && getline(ifsA, buf)){
                    istrstream ibuf(buf.data());
                    ibuf >> pos[i][j][k];
                    // クイーンを認識した時の処理
                    if(pos[i][j][k]==1) {
                        //前回のクイーン配置個数を取り込む。
                        Q++;
                        // クイーンの初期位置を設定
                        if(Q==1) {
                            A = i;
                            B = j;
                            C = k;
```

```
                    }
                }
            }
        }
    }
    loadcount=Q; // 前回の配置されたクイーンの個数を読み取る。

    //計算時間情報ファイル読み取り
    ifstream ifsB("savetime.txt");
    string stime;
    if(ifsB && getline(ifsB, stime)){
        istrstream istime(stime.data());
        istime >> savetime; // 前回の計算時間を savetime に取り込む
    }

    //最大解の配列情報ファイル読み取り
    ifstream ifsC("savemaxboad.txt");
    string mbuf;
    // 対応した座標にマス情報を配列に取り込む
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            for(int k=0;k<N;k++){
                if(ifsC && getline(ifsC, mbuf)){
                    istrstream imbuf(mbuf.data());
                    imbuf >> maxpos[i][j][k];
                    // 前回のクイーンの最大可能設置個数を読み込む
                    if(maxpos[i][j][k]==1) QMax++;
                }
            }
        }
    }
}

void main(){
    load(); // 前回データの読み取り

    // 前回データの表示
    cout << endl;
```

```
    cout << "現在の 3D-N-Queen 問題(N=" << N << ")においての探索様子" << endl;
    cout << "今までの合計計算時間は" << savetime << "秒" << endl;
    cout << "現在のボード全体図" << endl;
    Show();
    cout << "現時点での解の存在する最大の Queen の個数は" << QMax << "個" << endl;
    cout << "現時点での Queen を可能な限り最大個数で設置した場合のボード全体図" << endl;
    MaxShow();
    cout << "＊＊＊＊＊＊＊＊＊＊" << endl << endl;

    //再探索開始
    time(&saveT);
    time(&startT);
    ReStart();
    time(&endT);

    cout << endl;
    cout << "3D-N-Queen 問題(N=" << N << ")において Queen を可能な限り最大個数で設置した時のボード全体図" << endl;
    MaxShow();
    cout << "3D-N-Queen 問題(N=" << N << ")において解の存在する最大の Queen の個数は" << QMax << "個" << endl;
    // savetime を足したものを表示
    cout << "合計計算時間は" << difftime(endT,startT) + savetime << "秒" << endl;
    }
}
```