卒業研究報告書

題目

部分解合成法を用いたモンテカルロ法によるNQueenの全解探索

指導教員

石水 隆　助教

報告者

08-1-037-0022

奥川　史樹

近畿大学理工学部情報学科

平成 24 年 1 月 31 日提出

概要

近似解を求める計算手法として乱数を用いたシミュレーションを行うモンテカルロ法がある。モンテカルロ法は解析的に解くことができない問題でも十分多くの解析シミュレーションを繰り返すことにより、近似的に解を求めることができるため、適用範囲が広く問題によっては他の計算手法より簡単に解を得られる。本研究では代表的な組み合わせ最適化問題である NQueen 問題に対し、モンテカルロシミュレーションを用いた解探索を行い、NQueen 問題に対する解探索法としてモンテカルロシミュレーションがどこまで有効かを検証する。また、本研究では、探索範囲を狭めるために部分解合成法を用い、モンテカルロシミュレーションと部分解合成法を組み合わせることにより探索性能がどの程度上昇するかを検証する。

目次

1　序論

1.1 本研究の背景

近似解を求める計算手法として乱数を用いたシミュレーションを行うモンテカルロ法[1]がある。モンテカルロ法は解析的に解くことができない問題でも十分多くの解析シミュレーションを繰り返すことにより、近似的に解を求めることができるため、適用範囲が広く問題によっては他の計算手法より簡単に解を得られる。

モンテカルロ法は 1940 年代にフォン・ノイマンにより提案された[1]。モンテカルロ法は乱数を用いたシミュレーション技法の総称である。物体の移動する方向が確率的に与えられており、時間の経過とともにその動きを追う問題であるランダムウォーク問題[1]や、待ち行列問題[1]などではモンテカルロ法を使用することによって解ことができる。

また、本研究では、探索範囲を狭めるために部分解合成法[2]を用いる。部分解合成法とは問題における部分解を作成し、最終的にその部分解を合成して一つの全体解を作成するといものであり、最終的な解を分割できる問題に対して使用することができる。

1.2 本研究の目的

近年、囲碁[3]や将棋[4]などのゲームプログラミングにおいてモンテカルロ法用いた最適解の探索が研究されている。しかしながらこれらの解探索範囲は膨大で、いかに解の探索範囲を抑えるかが問題となってくる。そこで本研究では組み合わせ最適化問題として問題構造が比較的簡単な NQueen 問題を取り上げ、モンテカルロ法を用いた解探索能力の向上を検証した。

1.3 NQueen 問題

1.3.1　NQueen 問題とは

「8×8 のチェス盤上に、8 つのクイーンを互いに利き筋に当たらないように配置する」という古典的なパズル問題を 8 クイーン問題という。チェスのクイーンは、将棋の飛車と角を合わせた動きをする。つまり、図 1.3.1 に示す通り、上下、左右、それに斜めにどこまでも進むことができる。

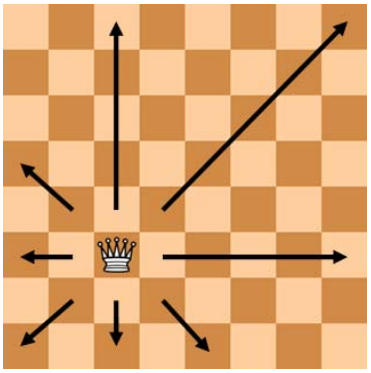

図 1.3.1 クイーンの利き筋

これを一般化した「N×N のマス目上に N 個のクイーンを配置する」という問題 NQueen 問題という。NQueen 問題は N が大きくなると解の量が爆発的に増え、解を求めるのに非常に多くの時間を費やすという特性がある。しかしながら現在のところこの問題を解析的な方法では解くことはできず、盤面に実際に駒を配置して確認しなければならない。

1.3.2 NQueen 問題の既知の結果

今現在 NQueen の探索方法としてよく使用されているのが Jeff Somers 氏のビット演算を用いた探索アルゴリズムである[6]。表 1.3.3.1 に[6]に掲載されている解の探索結果を示す。また表 1.3.3.2 に N=26 までの最新の結果を示す。

表 1.3.3.1 ビット演算を用いた NQueen 問題の解探索結果[6]

問題のサイズ	解の数	実行時間(時間 : 分 : 秒)
1	1	0
2	0	0
3	0	0
4	2	0
5	10	0
6	4	0
7	40	0
8	92	0
9	352	0
10	724	0
11	2680	0
12	14200	0
13	73712	0
14	365596	00:00:01
15	2279184	00:00:04
16	14772512	00:00:23
17	95815104	00:02:38
18	666090624	00:19:26
19	4968057848	02:31:24
20	39029188884	20:35:06
21	314666222712	174:53:45
22	2691008701644	?
23	24233937684440	?
24	?	?

表 1.3.3.2 最新の NQueen 問題における解探索状況[2]

N	公表日	公表機関名	基本プログラム	解の数	文献
24	2004.04.11	電気通信大学	qn24b	227,514,171,973,736	[7]
25	2005.06.11	ProActive	不明	2,207,893,435,808,350	[2]
26	2009.07.11	Tu-dresden	JSomer 版の改良版	22,317,699,616,364,000	[9]

1.4 本報告書の構成

本報告書は以下の通りの構成となる。2 章においてはモンテカルロ法よる NQueen 問題を解くアルゴルズムについて記述する。3 章においては部分解合成法を用いたモンテカルロ法による NQueen 問題を解くアルゴルズムについて記述する。4 章においては本研究の結果と考察について記述し、5 章においては結論と今後の課題について記述する。

2　モンテカルロ法を用いた NQueen 問題の解探索

2.1 モンテカルロ法とは

モンテカルロ法[1]とは、乱数を用いるシミュレーション技法の総称で近似解を求める計算手法としてよく使用される。モンテカルロ法の起源は 18 世紀後半に行われた「Buffon の針」という確率的模擬実験[1]にさかのぼることができる。このときは、針を投げる実験を実際に数千回も行ったが、考え方はモンテカルロ法そのものであったといえる。コンピュータを用いる現代的な意味でのモンテカルロ法は、1940 年代にフォン・ノイマンとウラムが行った「核分裂における中性子の拡散現象」に関する確率的模擬実験[1]が最初であるといわれており、賭博で有名な地＝モンテカルロに由来する名称は、このときに付けられたものである。モンテカルロ法の根幹にあるのは乱数であり、シミュレーションでは多くの乱数を高速に発生させる必要がある。また、シミュレーションは膨大かつ複雑な計算を伴う。

例えばモンテカルロ法を用いて円周率πを求める場合について説明する。この場合、1×1 の正方形の内部に半径 0.5 の円を描き、正方形の中にランダムに点を打っていく。

円の面積はπ×r×r=0.25πであり、正方形の面積は 1 であるから、無限に点を打つと

〈円の中の点の数〉:〈全部の点の数〉

＝〈円の面積〉:〈正方形の面積〉

＝0.25π：1

となり、円の中の点の数の 4 倍÷全部の点の数＝πとなる。正方形内に打つ点の個数が増えるにつれて近似解が、真の値に近づいていく性質がある。例えば正方形内に 10^4 個の点を打った場合、得られる近似解と真の値との誤差は約 0.01 程度となる[5]。このように、実際に乱数シミュレーションを繰り返した結果から近似解を求める方法がモンテカルロ法である。

2.2 モンテカルロ法を用いた NQueen 問題の解探索

本節では、本研究で提案するモンテカルロ法を用いて NQueen 問題を解くアルゴリズムについて述べる。上記で説明した「Buffon の針」において針をクイーンの駒に置換え、針を落とす的をチェス盤のマス目に置き換えてシミュレーションを行うのがモンテカルロ法を用いて NQueen 問題を解く基本戦略である。「Buffon の針」を NQueen 問題に置き換えると、盤面上のマス目にランダムに N 個のクイーンを落とし、得られたクイーンの配置が NQueen 問題の解となるかを判定する。ある配置パターンが NQueen 問題の解となるかの判定は、その配置パターンの競合数を求めることで行える。競合数とは駒が存在する縦横斜めのラインの数の和のことをいう。

以下にモンテカルロ法を用いて NQueen 問題を解くアルゴリズムを示す。

［モンテカルロ法を用いた NQueen 問題アルゴリズム］

① 1～N の範囲の整数値を取る N 個の乱数 $r[i]$ $(1 \leqq i \leqq N)$を発生する。

② 盤面の i 行 $r[i]$列 $(1 \leqq i \leqq N)$の N 個のマス目に駒を配置する。

③ 配置パターンの競合数を求め、競合数が 0 であれば解であると判定する。

2.3 モンテカルロ法を用いた NQueen 問題の解探索プログラム

本研究では、2.2 で示したアルゴリズムに従い部分解合成法を用いたモンテカルロ法による NQueen 問題の解探索プログラムの作成を行った。付録 A に作成したプログラムを示す。

main.cpp

・main.cpp は各メソッドを実行する部分であり、駒を配置する盤面にあたる変数 a を作成するプログラムである。

init.cpp

・init.cpp は init メソッドにおいて変数 a にランダムに駒を配置するプログラムである。

check.cpp

・check.cpp は check メソッドにおいて変数 a に保存されている駒の配置情報から競合数を判定するプログラムである。もし解が発見されれば vector 型の変数 vector に解を保存する。

print0.cpp

・print0.cpp は ptint0 メソッドで vector 内に保存されている解を表示するプログラムである。

set.h

・set.h はプログラムにおいて扱う設定が記述されているプログラムである。

3　部分解合成法を用いたモンテカルロ法による NQueen 問題の解探索

3.1 部分解合成法とは

部分解合成法[2]とは問題における部分解を作成し、最終的にその部分解を合成して一つの全体解を作成する分割統治法の一種であり、問題をより小さい部分問題に分割できる問題に対して使用することができる。解の探索範囲を分割することで、問題空間を小さくすることができ、高速化の余地が生まれると考えられる。

以下に部分解合成法の基本的なアルゴリズムを示す。

［部分解合成法の基本的なアルゴリズム］

① 問題のサイズ *n* と閾値 M に対して、n≦M であれば、解を作成し、終了する。n>M であれば②～④の処理を行う。

② 問題をサイズ *n*/2 の 2 個の部分問題に分割する。

③ 各部分問題を、部分解合成法アルゴリズムを用いて再帰的に解く。

④ 2 つの部分解を合成し、サイズ *n* の解を作成する。

3.2 部分解合成法を用いたモンテカルロ法による NQueen 問題の解探索

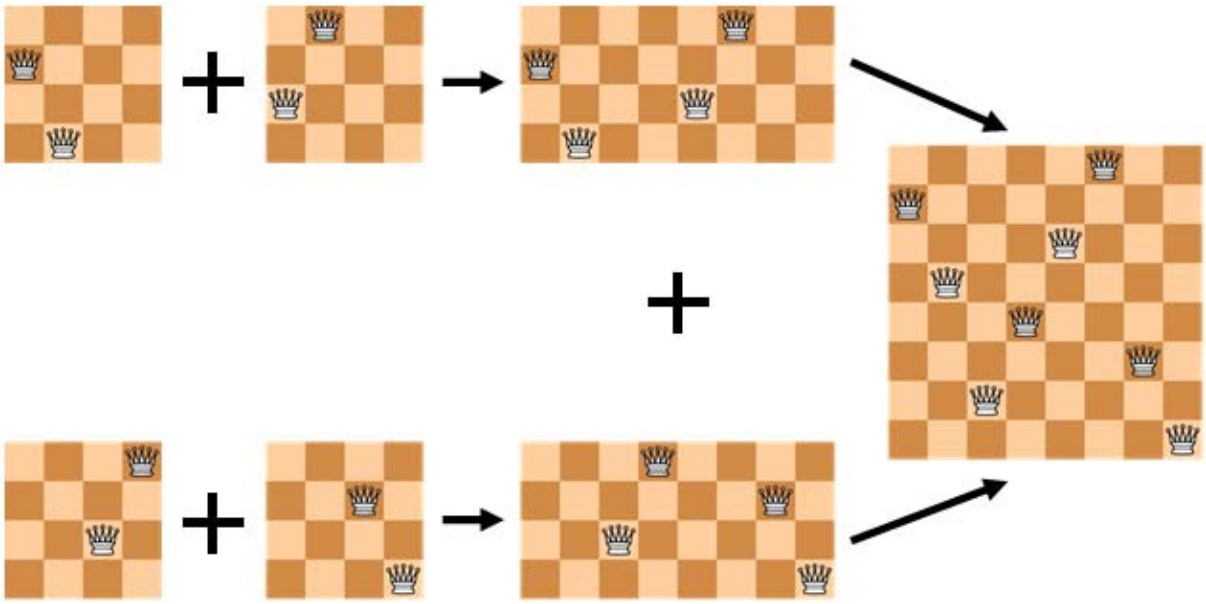

図 3.2.1　部分解合成法による NQueen 問題の部分解の合成例

本節では、本研究で提案する部分解合成法を用いたモンテカルロ法による NQueen 問題を解くアルゴリズムについて述べる。このアルゴリズムは部分解の駒の配置をモンテカルロ法により決定し、その部分解を合成し解を求めるというものである。 NQueen においての部分解は盤面を 4 等分したものとして、最終的にすべての部分解を合成し、解かどうかの判定を行う。図 1 に部分解合成法による NQueen 問題の部分解の合成例を示す。

また、NQueen 問題の解は、その解を反転・回転しても解となる。そこで、部分解を作成したときに、その部分解を反転・回転した部分解も合成の素材として用いることで、作成する部分解数を減らし、計算時間の削減を図る。本研究では、部分解の作成パターンとして 3 つのタイプ TypeA, TypeB，TypeC を提案する。

TypeA は、サイズ N/2×N/2 の 4 つの部分解を作成し、それを合成する。TypeA による部分解の合成例を図 2 に示す。

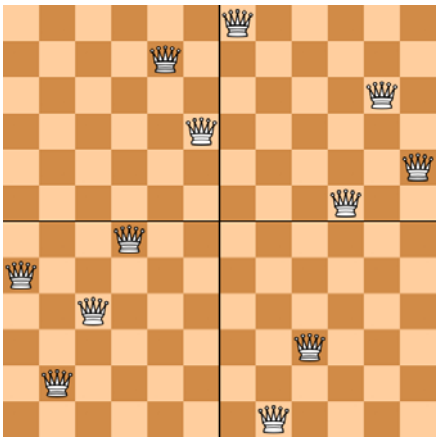

図 3.2.2　TypeA による部分解の合成例

図 2 に示されるように、TypeA により生成される解は対称性がないため、回転や反転を行っても元の解と一致しない。よって 90°,180°,270°の回転と反転操作により、別解としてあと 7 個の解が求まる。

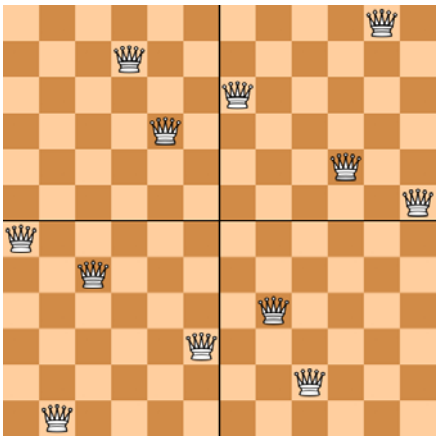

図 3.2.3　TypeB による部分解の合成例

TypeB は、サイズ N/2×N/2 の 2 つの部分解を作成し、それらを 180°回転させてできる 2 つの部分解と合成する。図 3 に TypeB による部分解の合成例を示す。この TypeB により生成される解は 180°回転すると元の解と一致する解である。よって 90°の回転と反転から別解としてあと 3 個の解が求まる。

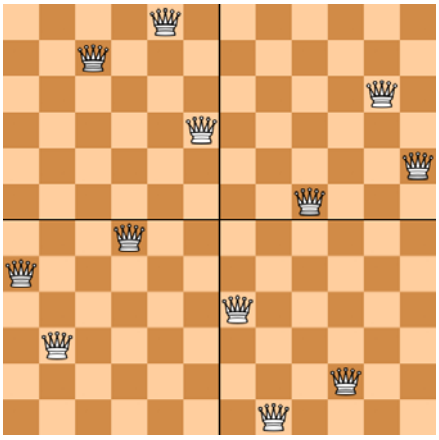

図 3.2.4　TypeC による部分解の合成例

TypeC は、サイズ N/2×N/2 の部分解を 1 つ作成し、それらを 90°,180°,270°回転させてできる 3 つの部分解と合成する。図 4 に TypeC による部分解の合成例を示す。この TypeC により生成される解は 90°回転すると元の解と一致する解である。よって反転操作のみによって 1 個の別解を得ることができる。

上記した NQueen 問題の解の性質を生かし、モンテカルロ法を用いた部分解合成法による NQueen 問題の解法にあたる。

以下に部分解合成法を用いたモンテカルロ法による NQueen 問題探索アルゴリズムを示す。

[部分解合成法を用いたモンテカルロ法による NQueen 問題探索アルゴリズム]

① N/2×N/2 の部分解を 1 個作成する。

② ①で作成した部分解をもとに、各 Type で部分解を作成する

TypeA:盤面の右下にあたる部分の部分解を①で作成した配置数と同じだけランダムに配置することで、全体の 1/2 の部分解を作成する。

TypeB:①で作成した部分解を 180°回転させ、右下部分の部分解とする。①の部分解と合わせて全体で 1/2 の部分解が完成する。

TypeC:①で作成した部分解を 90°回転させ右上部分の部分解に、270°回転させ左下部分の部分解とする。①の部分解とあわせて 3/4 の部分解が完成する。

③ ②で作成した各 Type の部分解の競合数を判定する。もし競合数が 0 であれば、④に進む。それ以外であれば以降の処理は行わない。

④ 各 Type の駒の配置されていない部分に駒を配置していく。

TypeA と TypeB:駒の配置されていない右上部分と左下部分に駒を配置していく。

TypeC:駒の配置されていない右下部分に駒を配置していく。

⑤ 各 Type 別に競合数の判定を行う。もし競合数が 0 ならば駒の状態を保存する。また、Type 別に別解を作成し駒の状態を保存する。

TypeA:回転と反転から 7 個の別解を作成する。

TypeB:回転と反転から 3 個の別解を作成する。

TypeC:回転と反転から 1 個の別解を作成する。

3.3 部分解合成法を用いたモンテカルロ法による NQueen 問題の解探索プログラム

本研究では、3.2 で示したアルゴリズムに従い部分解合成法を用いたモンテカルロ法による NQueen 問題の解探索プログラムの作成を行った。付録 A に作成したプログラムを示す。

本研究では、上記のプログラムを用いて N=(N の範囲)に対する NQueen 問題の解を求めた。また、本研究で用いた計算機のスペックは

CPU：Cor2duo　E8400　3.00GHz

OS：windouws7homepremium64bit

上記の通りである。

main.cpp

・main.cpp は本プログラム実行部分であり、初期配置を保存する配列 Q、および各 Type を保存 A,B,C,の配列を作成しているプログラムである。

init.cpp

・init.cpp は main で作成した Q,A,B,C を配置されている駒の状態が空になるよう初期化を行うプログラムである。

partget.cpp

・partget.cpp は左上部分にあたる 1/4 の部分解を作成するプログラムである。

partgetA.cpp

・partgetA.cpp は右下部分にあたる 1/4 の部分解を作成し、TypeA において全体で 1/2 の部分解の作成するプログラムである。

partgetB.cpp

・partgetB.cpp は初期に配置した駒の配置を 180°回転させ、右下部分にあたる 1/4 の部分解を作成するプログラムである。TypeB において全体で 1/2 の部分解の作成が完了する。

partgetC.cpp

・partgetC.cpp は初期に配置した駒の配置から 90°回転させ右上部分、270°回転させ左下部分の計 2 の部分解を作成するプログラムである。TypeC において全体で 3/4 の部分解の作成が完了する。

checkA.cpp

・checkA.cpp は partgetA で完成した 1/2 の部分解の競合判定を行い、もし競合数が 0 ならば allputA を実行できるようにするプログラムである。

checkB.cpp

・checkB.cpp は partgetB で完成した 1/2 の部分解の競合判定を行い、もし競合数が 0 ならば allputB を実行できるようにするプログラムである。

checkC.cpp

・checkC.cpp は partgetC で完成した 3/4 の部分解の競合判定を行い、もし競合数が 0 ならば allputC を実行できるようにするプログラムである。

allputA.cpp

・allputA.cpp は TypeA の駒が配置されていない残りの場所に駒を配置するプログラムである。

allputB.cpp

・allputB.cpp は TypeB の駒が配置されていない残りの場所に駒を配置するプログラムである。

allputC.cpp

・allputC.cpp は TypeC の駒が配置されていない残りの場所に駒を配置するプログラムである。

fcheckA.cpp

・fcheckA.cpp は allputA にて駒の配置された盤面の競合判定を行うプログラムである。もし競合数が 0 であれば解とし駒の配置を保存する。また、回転と反転を加え、7 の別解をつくりその駒の配置も保存する。

fcheckB.cpp

・fcheckB.cpp は allputB にて駒の配置された盤面の競合判定を行うプログラムである。もし競合数が 0 であれば解とし駒の配置を保存する。また、回転と反転を加え、3 の別解をつくりその駒の配置も保存する。

fcheckC.cpp

・fcheckC.cpp は allputC にて駒の配置された盤面の競合判定を行うプログラムである。もし競合数が 0 であれば解とし駒の配置を保存する。また、反転を加え、1 の別解をつくりその駒の配置も保存する。

set.h

・set.h はプログラムに対する設定を記述しているプログラムである。

表 4.1 プログラムを実行した条件

N の範囲	シミュレート回数	試行回数
4〜10	500000	1000

表 4.2 モンテカルロ法を用いた NQueen の解探索の結果

	発見数(平均値±標準偏差)	全解数[6]	解発見数/全解数
4	2±0	2	100%
5	10±0	10	100%
6	4±0	4	100%
7	20±5	40	50%
8	1±1	92	1%
9	1±1	352	0.2%
10	0	724	0%

4 結果・考察

まずモンテカルロ法を用いた NQueen 探索を行う。およそ実行時間の目安を 1 秒とすると、乱数を発生させる回数を 50 万程度となる。そこで、表 4.1 の条件でシミュレートを行い、その結果および全解に対する解発見割合を表 4.2 に示す。表 4.2 からもわかるように N の値が小さければ問題なく探索できているがサイズが大きくなるととたんに探索能力が落ちている。問題サイズが 10 になると全く探索できなくなってしまった。次に表 4.3 の条件で、部分解合成法を用いたモンテカルロ法で NQueen の解探索を行った結果を表 4.4 に示す。また部分解合成法はプログラムの仕様上、偶数の問題サイズしか扱えない。表 4.4 のグラフからわかるとおり、モンテカルロ法のみを用いた場合では全く解を求めることのできなかったが、部分解合成法を用いたことで全ての解を求めることができた。しかしながらモンテカルロ法のみの場合は乱数を用いるのみでよく、50 万回のシミュレートを行っても 1 秒たらずで済むが、部分解合成法を用いると解判定の部分で時間がかかり、30 秒も実行時間がかかった。そこで今度は部分解合成法の実行時間を 1 秒程度と設定し、解探索を行った結果を表 4.5 に示す。表 4.5 から実行時間においてもモンテカルロ法お用いた場合より多くの解を探索できていることがわかる。

表 4.3 プログラムを実行した条件

N の範囲	シミュレート回数	試行回数
4〜14 の偶数数値のみ	500000	1000

表 4.4 モンテカルロ法を用いた部分解合成法による NQueen の解探索の結果

	発見数(平均値±標準偏差)	全解数[6]	解発見数/全解数
4	2±0	2	100%
6	4±0	4	100%
8	92±0	92	100%
10	652±8	742	87.87%
12	598±8	14200	4.21%
14	332±8	365596	0.09%

表 4.5 モンテカルロ法を用いた部分解合成法による NQueen の解探索の結果

	発見数(平均値±標準偏差)	全解数[6]	解発見数/全解数
4	2±0	2	100%
6	4±0	4	100%
8	92±0	92	100%
10	60±8	742	8.086%
12	58±8	14200	0.408%
14	40±8	365596	0.010%

また、本研究で用いたプログラムは表 1.3.3.1 の実行時間に比べると非常に遅く、時間のコストに対してあまり解が求められていない。しかしながら表 1.3.3.1 の実行プログラムは解の個数を再帰的に求めるものである。このことから、例えばある一定時間での解の結果をみた時に、表 1.3.3.1 のプログラムでは何度やっても同じ最適解しかでないが、本プログラムはその時のランダムな配置によって同じ最適解がでるとはかぎらない。毎回一定の最適解しか生成しない表 1.3.3.1 のプログラムに対し、実行するたびに別の最適解が発生するかもしれない本プログラムを比較した場合、目的が違えば、実行時間での単純比較では比較しきれない部分があると考える。

5 結論・今後の課題

本研究では代表的な組み合わせ最適化問題である NQueen 問題に対し、モンテカルロシミュレ

ーションを用いた解探索を行い、NQueen 問題に対する解探索法としてモンテカルロシミュレーションがどこまで有効かを検証、およびモンテカルロシミュレーションと部分解合成法を組み合わせることにより探索性能がどの程度上昇するかを検証した。

本研究で検討した、部分解法にモンテカルロ法を用いるという手法で、従来のモンテカルロ法を上回る解探索を実現できた。今後は、Type ごとの判定は独立しているので並列化による高速化などが検討できる。

今後の課題として、本研究では開発したモンテカルロ法を用いた部分解合成法による解探索のプログラムは、偶数の問題サイズしか扱えなかった。そこで奇数の問題サイズも扱えるように改良する必要がある。例えば図 5 に示すように、N/2×N/2 の部分解を合成するということは変えず、奇数の場合にはさらに中央の縦と横のラインのどこかにランダムで駒を配置し、そこに N/2×N/2 の部分解を合成していくということが考えられる。

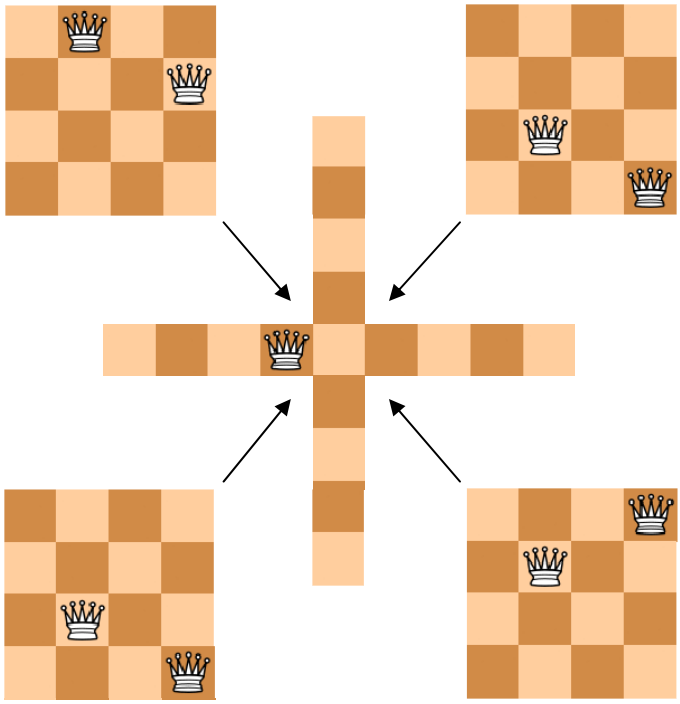

図 5 問題サイズが奇数の場合における部分解合成法の例

謝辞

本研究書を作成するにあたり、ご指導していただきました石水隆先生には多くのご迷惑おかけしましたことのお詫びと、本研究を最後までご指導いただきました感謝をこの場をかりて心より申し上げます。

参考文献

[1] 伊藤俊秀，草薙信照．コンピュータシミュレーション．オーム社，2006.

[2] 萩野谷一二，“NQueen 問題への新しいアプローチ(部分解合成法)について，”情報処理学会報告書，Vol.2011-GI-26, No.11, 2011.

[3] 美添一樹，“モンテカルロ木探索―コンピュータ囲碁に革命を起こした新手法”情報処理学会会誌，Vol.49, No.6, 2008.

[4] 佐藤佳州,高橋大介“モンテカルロ木探索によるコンピュータ将棋”情報処理学会論文誌，Vol.50, No.11, 2009.

[5] 山下宏，モンテカルロ法で囲碁、将棋, 2009, http://www32.ocn.ne.jp/~yss/monte.html

[6] Jeff Somers's N Queens Solutions, http://www.jsomers.com/nqueen_demo/nqueens.html

[7] 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.html

[8] NQueen 問題(解の個数を求める), http://www.ic-net.or.jp/home/takaken/nt/queen/index.html

[9] Queen@TUD, Technische Universitat Dresden, 2009, http://Queens.inf.tu-dresden.de/

付録

・以下に本研究で作成したモンテカルロ法を用いた NQueen 問題の解探索プログラムを示す。

main.cpp

```
#include<iostream>
#include<vector>
#include<string>
#include <ctime>
#include <cstdlib>
#include<windows.h>
#include "set.h"

void main(){

	int a[N];
	std::vector<std::string> vector;

	LARGE_INTEGER freq,time_start,time_end;//周波数、開始時間、終了時間

	void init(int [N]);
	void check(int [N],std::vector<std::string>& vector);
	void print0(int [N],std::vector<std::string> vector);

	srand((unsigned)time(NULL));

	QueryPerformanceFrequency(&freq);
	QueryPerformanceCounter(&time_start);//時間計測開始

	for(int i=0;i<MAX;i++){
		init(a);
		check(a,vector);
	}

	QueryPerformanceCounter(&time_end);//計測時間停止

	print0(a,vector);
	printf("処理時間:%d[ms]¥n",(time_end.QuadPart-time_start.QuadPart)*1000 /
freq.QuadPart);

}
```

init.cpp

```
#include <iostream>
#include "set.h"
/**
*@param a[N] 盤面上に配置されている駒の位置を保存している
*/
```

```
void init(int a[N]){


        for(int i=0;i<N;i++){
                a[i] = rand()%N;
        }
}
```

check.cpp

```
#include<iostream>
#include"set.h"
#include<vector>
#include<string>
#include<sstream>
/**
*@param a[N] 盤面上に配置されている駒の位置を保存している
*@param vector 出現した解を保存している
*/
void check(int a[N],std::vector<std::string>& vector){
        int x=0;
        int p[2*N-1];//右斜め上の配列
        int q[2*N-1];//右斜め下の配列
        int r[N];
        int sum=0;

        std::ostringstream l;


                //配列の初期化
                for(int t=0;t<2*N-1;t++){
                        p[t]=0;
                        q[t]=0;
                }
                for(int i=0;i<N;i++){
                        r[i]=0;
                }

                //a[N]の解判定
                for(int j=0;j<N;j++){
                        x=a[j];//コマの位置情報をxに代入

                        //右上斜め判定
                        if(p[j+x]==0){
                                p[j+x]=1;
                        }
                        else{
```

```
                                        sum+=1;
                                }

                                //右下斜め判定
                                if(q[j-x+(N-1)]==0){
                                        q[j-x+(N-1)]=1;
                                }
                                else{
                                        sum+=1;
                                }

                                //縦判定
                                if(r[x]==0){
                                        r[x]=1;
                                }
                                else{
                                        sum+=1;
                                }

                        }

                        //解を発見したらstring型でvectorに保存
                        if(sum==0){
                                for(int i=0;i<N;i++){
                                        l<<a[i];
                                }
                                vector.push_back(l.str());

                                //重複解を見つけて取り出す
                                int u = vector.size();
                                for(int j=0;j<u-1;j++){
                                        if(vector[j]==vector[u-1]){
                                                vector.pop_back();
                                                break;
                                        }
                                }
                        }
                        sum=0;
}
```

print0.cpp

```
#include<iostream>
#include<vector>
#include<string>
#include"set.h"
using namespace std;
```

```
/**
*@param a[N] 盤面上に配置されている駒の位置を保存している
*@param vector 出現した解を保存している
*/
void print0(int a[N],std::vector<std::string> vector){

	int w = vector.size();
	for(int o=0; o<w;o++){
		cout <<"解:" <<o+1 <<"番目は" <<vector[o] <<endl;
	}

}
```

set.h

```
#define N 8
#define MAX 1000
```

・部分解合成法を用いたモンテカルロ法による NQueen 問題の解探索

main.cpp

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <windows.h>
#include<vector>
#include<string>

#include "set.h"


void main(){

	int Q[N/2];//1/4部分解作成用
	int A[N];//TypeA判定用配列
	int B[N];//TypeB判定用配列
	int C[N];//TypeC判定用配列
	int judgeA=0;//判定用変数
	int judgeB=0;//判定用変数
	int judgeC=0;//判定用変数
	int qcount=0;//初期に配置したクイーンの数
	std::vector<std::string> vector;//解保存用配列
	void init(int [N/2],int [N],int [N],int [N]);
	void partget(int [N/2],int [N],int [N],int [N],int &);
	void partgetA(int [N],int &);
	void partgetB(int [N/2],int [N]);
	void partgetC(int [N/2],int [N]);
```

```
	void checkA(int [N],int &);
	void fcheckA(int [N],std::vector<std::string>&);
	void checkB(int [N],int &);
	void fcheckB(int [N],std::vector<std::string>&);
	void checkC(int [N],int &);
	void fcheckC(int [N],std::vector<std::string>&);
	void allputA(int [N]);
	void allputB(int [N]);
	void allputC(int [N]);
	void print0(int [N/2],int [N]);

	srand((unsigned)time(NULL));

	LARGE_INTEGER freq,time_start,time_end;//周波数、開始時間、終了時間

	QueryPerformanceFrequency(&freq);
	QueryPerformanceCounter(&time_start);//時間計測開始

	for(int i=0;i<MAX;i++){
		init(Q,A,B,C);//1/4部分解のおよび各Typeの初期化
		partget(Q,A,B,C,qcount);//1/4部分解の作成
		partgetA(A,qcount);//TypeA1/2まで作成
		partgetB(Q,B);//TypeBを1/2まで作成
		partgetC(Q,C);//TypeCを1/2まで作成
		checkA(A,judgeA);
		checkB(B,judgeB);
		checkC(C,judgeC);
		if(judgeA!=0){
			allputA(A);
			fcheckA(A,vector);
		}
		if(judgeB!=0){
			allputB(B);
			fcheckB(B,vector);
		}
		if(judgeC!=0){
			allputC(C);
			fcheckC(C,vector);
		}

	}

	QueryPerformanceCounter(&time_end);//計測時間停止

	for(int i=0;i<vector.size();i++){
		std::cout << "解"<< i+1  << " : "<< vector[i] << std::endl;
	}
```

```
        printf("解の数:%d¥n",vector.size());
        printf("処理時間:%d[ms]¥n",(time_end.QuadPart-time_start.QuadPart)*1000 /
freq.QuadPart);
}
```

init.cpp

```
#include <iostream>
#include "set.h"
/**
*@param Q[N/2] 左上部分にあたる1/4の部分解の駒の位置を保存している
*@param A[N] TypeAの駒の配置を保存している
*@param B[N] TypeBの駒の配置を保存している
*@param C[N] TypeCの駒の配置を保存している
*/
void init(int Q[N/2],int A[N],int B[N],int C[N]){

        for(int i=0;i<N/2;i++){
                Q[i]=N+1;
        }

        for(int i=0;i<N;i++){
                A[i]=N+1;
                B[i]=N+1;
                C[i]=N+1;
        }

}
```

partget.cpp

```
/*
* 1/4の部分解作成メソッド
*/
#include <iostream>
#include<vector>//リスト構造を持つ配列vectorを使用するため
#include<string>//stringを使用するため
#include<sstream>//int型の変数をstring型に変換するために使用
#include "set.h"
/**
*@param Q[N/2] 左上部分にあたる1/4の部分解の駒の位置を保存している
*@param A[N] TypeAの駒の配置を保存している
*@param B[N] TypeBの駒の配置を保存している
*@param C[N] TypeCの駒の配置を保存している
*@param qcount 最初に配置した駒の数を保存している
*/
void partget(int Q[N/2],int A[N],int B[N],int C[N],int &qcount){
```

```
int d=0;//駒配置の数の変数

std::vector<int> vector;

//配置する個数を決定する。
do{
        d = rand()%(N/2);
}
while((N/4) >  d);

qcount =d;

//1/4部分解作成(左上部分の作成)
//vectorに0～N/2までの数を代入
for(int i=0;i<N/2;i++){
        vector.push_back(i);
}

//vector内でランダムに並び替え
int size = vector.size();
for(int j=0;j<size;j++){
        int w = vector[j];
        int r = rand()%(size);
        vector[j] = vector[r];
        vector[r] = w;
}

//vector全体から駒の数だけ残し、あとはからにする
size = vector.size() - d;
for(int i=0;i<size;i++){
        vector.pop_back();
}
//配置なしのN+1を入れる
for(int i=0;i<size;i++){
        vector.push_back(N+1);
}

//vector内でランダムに並び替え
size = vector.size();
for(int j=0;j<size;j++){
        int w = vector[j];
        int r = rand()%(size);
        vector[j] = vector[r];
        vector[r] = w;
}
```

```
        //vector内の数をQ[N/2]に代入することで駒の配置を実現
        //左上の1/4の部分解の完成
        for(int c=0;c<N/2;c++){
                Q[c]=vector[c];
                A[c]=vector[c];
                B[c]=vector[c];
                C[c]=vector[c];
        }

}
```

partgetA.cpp

```
/*
* TypeAの1/2の部分解作成メソッド
*/
#include <iostream>
#include<vector>//リスト構造を持つ配列vectorを使用するため
#include "set.h"
/**
*@param A[N] TypeCの駒の配置を保存している
*@param qcount 最初に配置した駒の数を保存している
*/
void partgetA(int A[N],int &qcount){

        std::vector<int> vector;

        //1/4部分解作成(右下部分の作成)
        //vectorにN/2～Nまでの数を代入
        for(int i=N/2;i<N;i++){
                vector.push_back(i);
        }

        //vector内でランダムに並び替え
        int size = vector.size();
        for(int j=0;j<size;j++){
                int w = vector[j];
                int r = rand()%(size);
                vector[j] = vector[r];
                vector[r] = w;
        }

        //vector全体から駒の数だけ残し、あとはからにする
        size = vector.size() - qcount;
        for(int i=0;i<size;i++){
                vector.pop_back();
```

```
        }
        //配置なしのN+1を入れる
        for(int i=0;i<size;i++){
                vector.push_back(N+1);
        }
        //vector内でランダムに並び替え
        size = vector.size();
        for(int j=0;j<size;j++){
                int w = vector[j];
                int r = rand()%(size);
                vector[j] = vector[r];
                vector[r] = w;
        }

        //右下の1/4の部分解の完成
        for(int c=0;c<N/2;c++){
                A[c+(N/2)]=vector[c];
        }
}
```

partgetB.cpp

```
/*
* TypeBの1/2の部分解作成メソッド
*/
#include <iostream>
#include "set.h"
/**
*@param Q[N/2] 左上部分にあたる1/4の部分解の駒の位置を保存している
*@param B[N] TypeBの駒の配置を保存している
*/
void partgetB(int Q[N/2],int B[N]){

        int newb1[N/2];//90° 回転した駒の配置を保存
        int newb2[N/2];//180° 回転した駒の配置を保存

        //1/4部分解作成(右下部分の作成)

        //90° 回転させる
        for(int j=0;j<N/2;j++){
                for(int k=0;k<N/2;k++){
                        if(Q[k]==j){
                                newb1[j]=((N/2)-1)-k;
                                break;
                        }
                        else{
                                newb1[j]=N+1;
```

```
                        }
                }
        }
        //90° 回転させる
        for(int e=0;e<N/2;e++){
                for(int v=0;v<N/2;v++){
                        if(newb1[v]==e){
                                newb2[e]=(N/2-1)-v;
                                break;
                        }
                        else{
                                newb2[e]=N+1;
                        }
                }
        }

        for(int i=0;i<N/2;i++){
                if(newb2[i] != N+1){
                        B[i+(N/2)]=newb2[i]+(N/2);
                }
        }
}
```

partgetC.cpp

```
/*
* TypeCの1/2の部分解作成メソッド
*/
#include <iostream>
#include "set.h"
/**
*@param Q[N/2] 左上部分にあたる1/4の部分解の駒の位置を保存している
*@param C[N] TypeCの駒の配置を保存している
*/
void partgetC(int Q[N/2],int C[N]){

        int newc1[N/2];//90° 回転した駒の配置を保存
        int newc2[N/2];//180° 回転した駒の配置を保存
        int newc3[N/2];//270° 回転した駒の配置を保存

        //1/4部分解作成(右上部分の作成)

        //90° 回転させる
        for(int j=0;j<N/2;j++){
                for(int k=0;k<N/2;k++){
                        if(Q[k]==j){
                                newc1[j]=((N/2)-1)-k;
```

```
                                        break;
                                }
                                else{
                                        newc1[j]=N+1;
                                }
                        }
                }
                //90° 回転させる
                for(int j=0;j<N/2;j++){
                        for(int k=0;k<N/2;k++){
                                if(newc1[k]==j){
                                        newc2[j]=((N/2)-1)-k;
                                        break;
                                }
                                else{
                                        newc2[j]=N+1;
                                }
                        }
                }

                //90° 回転させる
                for(int j=0;j<N/2;j++){
                        for(int k=0;k<N/2;k++){
                                if(newc2[k]==j){
                                        newc3[j]=((N/2)-1)-k;
                                        break;
                                }
                                else{
                                        newc3[j]=N+1;
                                }
                        }
                }

                for(int i=0;i<N/2;i++){
                        if(newc1[i] != N+1){
                                C[i]=newc1[i]+(N/2);
                        }
                }
                for(int i=0;i<N/2;i++){
                        if(newc2[i] != N+1){
                                C[i+(N/2)] = newc3[i];
                        }
                }
}
```

checkA.cpp

```
#include<iostream>
#include"set.h"
/**
*@param A[N] TypeAの駒の配置を保存している
*@param judgeA TypeAの判定を行うかの判定用変数が格納されている
*/
void checkA(int A[N],int &judgeA){
	int x=0;
	int p1[2*N-1];//右斜め上の配列
	int q1[2*N-1];//右斜め下の配列
	int sum=0;

		//配列の初期化
		for(int t=0;t<2*N-1;t++){
			p1[t]=0;
			q1[t]=0;
		}

		//a[N]の解判定
		for(int j=0;j<N;j++){
			x=A[j];//コマの位置情報をxに代入
			if(x != N+1){
				//右上斜め判定
				if(p1[j+x]==0){
					p1[j+x]=1;
				}
				else{
					sum+=1;
				}

				//右下斜め判定
				if(q1[j-x+(N-1)]==0){
					q1[j-x+(N-1)]=1;
				}
				else{
					sum+=1;
				}
			}
		}

		if(sum==0){
			judgeA=1;
		}
		sum=0;
}
```

checkB.cpp

```
#include<iostream>
#include"set.h"
/**
*@param B[N] TypeBの駒の配置を保存している
*@param judgeB TypeBの判定を行うかの判定用変数が格納されている
*/
void checkB(int B[N],int &judgeB){
	int x=0;
	int p1[2*N-1];//右斜め上の配列
	int q1[2*N-1];//右斜め下の配列
	int sum=0;

		//配列の初期化
		for(int t=0;t<2*N-1;t++){
			p1[t]=0;
			q1[t]=0;
		}

		//a[N]の解判定
		for(int j=0;j<N;j++){
			x=B[j];//コマの位置情報をxに代入
			if(x != N+1){
				//右上斜め判定
				if(p1[j+x]==0){
					p1[j+x]=1;
				}
				else{
					sum+=1;
				}

				//右下斜め判定
				if(q1[j-x+(N-1)]==0){
					q1[j-x+(N-1)]=1;
				}
				else{
					sum+=1;
				}
			}
		}

		if(sum==0){
			judgeB=1;
		}
		sum=0;
}
```

checkC.cpp

```
#include<iostream>
#include"set.h"
/**
*@param C[N] TypeCの駒の配置を保存している
*@param judgeC TypeCの判定を行うかの判定用変数が格納されている
*/
void checkC(int C[N],int &judgeC){
	int x=0;
	int p1[2*N-1];//右斜め上の配列
	int q1[2*N-1];//右斜め下の配列
	int r1[N];
	int sum=0;

		//配列の初期化
		for(int t=0;t<2*N-1;t++){
			p1[t]=0;
			q1[t]=0;
		}
		for(int s=0;s<N;s++){
			r1[s]=0;
		}


		//a[N]の解判定
		for(int j=0;j<N;j++){
			x=C[j];//コマの位置情報をxに代入
			if(x != N+1){
				//右上斜め判定
				if(p1[j+x]==0){
					p1[j+x]=1;
				}
				else{
					sum+=1;
				}

				//右下斜め判定
				if(q1[j-x+(N-1)]==0){
					q1[j-x+(N-1)]=1;
				}
				else{
					sum+=1;
				}

				//縦判定
				if(r1[x]==0){
					r1[x]=1;
```

```
                                        }
                                        else{
                                                sum +=1;
                                        }

                                }
                        }

                        if(sum==0){
                                judgeC=1;
                        }
                        sum=0;
}
```

allputA.cpp

```
/*
* TypeAの残りの駒の配置をすべておくメソッド
*/
#include <iostream>
#include<vector>//リスト構造を持つ配列vectorを使用するため
#include<string>//stringを使用するため
#include<sstream>//int型の変数をstring型に変換するために使用
#include "set.h"
/**
*@param A[N] TypeAの駒の配置を保存している
*/
void allputA(int A[N]){

        int r[N];//縦判定用配列
        int p[2*N-1];//右斜め上判定用配列
        int q[2*N-1];//右斜め下判定用配列
        int putcount=0;//配置できる部分の数を入れる変数
        int putprace=0;//配置できる部分を保存するための変数

        //判定用配列の初期化
        for(int x=0;x<N;x++){
                r[x]=0;
        }
        for(int y=0;y<2*N-1;y++){
                p[y]=0;
                q[y]=0;
        }

        //すでにある駒の利き筋を保存
        for(int s=0;s<N;s++){
                int t=A[s];//コマの位置情報をtに代入
```

```
                if(t != N+1){
                //右上斜め判定
                p[s+t]=1;
                //右下斜め判定
                q[s-t+(N-1)]=1;
                //縦判定
                r[t]=1;
                }
        }

        //すべての駒の配置を確認するまで続ける
        for(int e=0;e<N;e++){
                //すべての駒の配置をうめる
                for(int i=0;i<N;i++){

                        //駒がおいてないところがある場合
                        if(A[i]==N+1){

                                //右上部分にあたる時
                                if(i<N/2){
                                        //駒がおけるかどうか確認する
                                        for(int j=N/2;j<N;j++){

                                                //配置場所が利き筋でないかどうか判定
                                                if(r[j]==0){
                                                        if(p[i+j]==0){
                                                                if(q[i-j+(N-1)]==0){
                                                                        //すべて0なら配
置カウント+1

                                                                        putcount +=1;
                                                                        putprace=j;
                                                                }
                                                        }
                                                }
                                        }
                                        //配置カウントが1なら駒の配置
                                        if(putcount==1){
                                                A[i] = putprace;
                                                r[putprace]=1;
                                                p[i+putprace]=1;
                                                q[i-putprace+(N-1)]=1;
                                        }

                                }

                                //左下部分にあたる時
                                else{
```

```
                                        //駒がおけるかどうか確認する
                                        for(int g=0;g<N/2;g++){
                                                //配置場所が利き筋でないかどうか判定
                                                if(r[g]==0){
                                                        r[g]=1;
                                                        if(p[i+g]==0){
                                                                p[i+g]=1;
                                                                if(q[i-g+(N-1)]==0){
                                                                        //すべて0なら配
置カウント+1し、場所を保存
                                                                        putcount +=1;
                                                                        putprace=g;
                                                                }
                                                                else{
                                                                        r[g]=0;
                                                                        p[i+g]=0;
                                                                }
                                                        }
                                                        else{
                                                                r[g]=0;
                                                        }
                                                }
                                        }
                                        //配置カウントが1なら駒の配置
                                        if(putcount==1){
                                                A[i] = putprace;
                                                r[putprace]=1;
                                                p[i+putprace]=1;
                                                q[i-putprace+(N-1)]=1;

                                        }
                                }
                        }
                        //カウントと場所を初期化する
                        putcount=0;
                        putprace=0;
                }
        }
}
```

allputB.cpp

```
/*
* TypeBの残りの駒の配置をすべておくメソッド
*/
#include <iostream>
#include<vector>//リスト構造を持つ配列vectorを使用するため
```

```
#include<string>//stringを使用するため
#include<sstream>//int型の変数をstring型に変換するために使用
#include "set.h"
/**
*@param B[N] TypeBの駒の配置を保存している
*/
void allputB(int B[N]){

	int r[N];//縦判定用配列
	int p[2*N-1];//右斜め上判定用配列
	int q[2*N-1];//右斜め下判定用配列
	int putcount=0;//配置できる部分の数を入れる変数
	int putprace=0;//配置できる部分を保存するための変数

	//判定用配列の初期化
	for(int x=0;x<N;x++){
		r[x]=0;
	}
	for(int y=0;y<2*N-1;y++){
		p[y]=0;
		q[y]=0;
	}

	//すでにある駒の利き筋を保存
	for(int s=0;s<N;s++){
		int t=B[s];//コマの位置情報をtに代入
		if(t != N+1){
		//右上斜め判定
		p[s+t]=1;
		//右下斜め判定
		q[s-t+(N-1)]=1;
		//縦判定
		r[t]=1;
		}
	}

	//すべての駒の配置を確認するまで続ける
	for(int e=0;e<N;e++){
		//すべての駒の配置をうめる
		for(int i=0;i<N;i++){

			//駒がおいてないところがある場合
			if(B[i]==N+1){

				//右上部分にあたる時
				if(i<N/2){
					//駒がおけるかどうか確認する
```

```
                                        for(int j=N/2;j<N;j++){

                                                //配置場所が利き筋でないかどうか判定
                                                if(r[j]==0){
                                                        if(p[i+j]==0){
                                                                if(q[i-j+(N-1)]==0){
                                                                        //すべて0なら配
置カウント+1
                                                                        putcount +=1;
                                                                        putprace=j;
                                                                }
                                                        }
                                                }
                                        }
                                        //配置カウントが1なら駒の配置
                                        if(putcount==1){
                                                B[i] = putprace;
                                                r[putprace]=1;
                                                p[i+putprace]=1;
                                                q[i-putprace+(N-1)]=1;
                                        }

                                }

                                //左下部分にあたる時
                                else{
                                        //駒がおけるかどうか確認する
                                        for(int g=0;g<N/2;g++){
                                                //配置場所が利き筋でないかどうか判定
                                                if(r[g]==0){
                                                        r[g]=1;
                                                        if(p[i+g]==0){
                                                                p[i+g]=1;
                                                                if(q[i-g+(N-1)]==0){
                                                                        //すべて0なら配
置カウント+1し、場所を保存
                                                                        putcount +=1;
                                                                        putprace=g;
                                                                }
                                                                else{
                                                                        r[g]=0;
                                                                        p[i+g]=0;
                                                                }
                                                        }
                                                        else{
                                                                r[g]=0;
                                                        }
```

```
                                                }
                                        }
                                        //配置カウントが1なら駒の配置
                                        if(putcount==1){
                                                B[i] = putprace;
                                                r[putprace]=1;
                                                p[i+putprace]=1;
                                                q[i-putprace+(N-1)]=1;

                                        }
                                }
                        }
                        //カウントと場所を初期化する
                        putcount=0;
                        putprace=0;
                }
        }
}
```

allputC.cpp

```
/*
* TypeCの残りの駒の配置をすべておくメソッド
*/
#include <iostream>
#include<vector>//リスト構造を持つ配列vectorを使用するため
#include<string>//stringを使用するため
#include<sstream>//int型の変数をstring型に変換するために使用
#include "set.h"
/**
*@param C[N] TypeCの駒の配置を保存している
*/
void allputC(int C[N]){

        int r[N];//縦判定用配列
        int p[2*N-1];//右斜め上判定用配列
        int q[2*N-1];//右斜め下判定用配列

        int putcount=0;//配置できる部分の数を入れる変数
        int putprace=0;//配置できる部分を保存するための変数

        //判定用配列の初期化
        for(int x=0;x<N;x++){
                r[x]=0;
        }
        for(int y=0;y<2*N-1;y++){
                p[y]=0;
```

```
                q[y]=0;
        }

        //すでにある駒の利き筋を保存
        for(int s=0;s<N;s++){
                int t=C[s];//コマの位置情報をtに代入
                if(t != N+1){
                //右上斜め判定
                p[s+t]=1;
                //右下斜め判定
                q[s-t+(N-1)]=1;
                //縦判定
                r[t]=1;
                }
        }

        //すべての駒の配置を確認するまで続ける
        for(int e=0;e<N;e++){
                //すべての駒の配置をうめる
                for(int i=0;i<N;i++){

                        //駒がおいてないところがある場合
                        if(C[i]==N+1){

                                //右上部分にあたる時
                                if(i<N/2){
                                        //駒がおけるかどうか確認する
                                        for(int j=N/2;j<N;j++){

                                                //配置場所が利き筋でないかどうか判定
                                                if(r[j]==0){
                                                        if(p[i+j]==0){
                                                                if(q[i-j+(N-1)]==0){
                                                                        //すべて0なら配
置カウント+1

                                                                        putcount +=1;
                                                                        putprace=j;
                                                                }
                                                        }
                                                }
                                        }
                                        //配置カウントが1なら駒の配置
                                        if(putcount==1){
                                                C[i] = putprace;
                                                r[putprace]=1;
                                                p[i+putprace]=1;
                                                q[i-putprace+(N-1)]=1;
```

```
                                        }

                                }

                                //左下部分にあたる時
                                else{
                                        //駒がおけるかどうか確認する
                                        for(int g=0;g<N/2;g++){
                                                //配置場所が利き筋でないかどうか判定
                                                if(r[g]==0){
                                                        r[g]=1;
                                                        if(p[i+g]==0){
                                                                p[i+g]=1;
                                                                if(q[i-g+(N-1)]==0){
                                                                        //すべて0なら配置カウント+1し、場所を保存
                                                                        putcount +=1;
                                                                        putprace=g;
                                                                }
                                                                else{
                                                                        r[g]=0;
                                                                        p[i+g]=0;
                                                                }
                                                        }
                                                        else{
                                                                r[g]=0;
                                                        }
                                                }
                                        }
                                        //配置カウントが1なら駒の配置
                                        if(putcount==1){
                                                C[i] = putprace;
                                                r[putprace]=1;
                                                p[i+putprace]=1;
                                                q[i-putprace+(N-1)]=1;

                                        }
                                }
                        }
                        //カウントと場所を初期化する
                        putcount=0;
                        putprace=0;
                }
        }
}
```

fcheckA.cpp

```
#include<iostream>
#include"set.h"
#include<vector>
#include<string>
#include<sstream>
/**
*@param A[N] TypeAの駒の配置を保存している
*@param vector 解となった駒の配置情報を保存する
*/
void fcheckA(int A[N],std::vector<std::string>& vector){
	int x=0;
	int p[2*N-1];//右斜め上の配列
	int q[2*N-1];//右斜め下の配列
	int sum=0;
	int size=0;
	int no=0;

	int newa[N];
	int newb[N];
	int newc[N];

	std::ostringstream l;
	std::ostringstream m;
	std::ostringstream m1;
	std::ostringstream n;
	std::ostringstream h1;
	std::ostringstream h2;
	std::ostringstream o1;
	std::ostringstream o2;
	std::ostringstream z;


	//配列の初期化
	for(int t=0;t<2*N-1;t++){
		p[t]=0;
		q[t]=0;
	}

	//配置なしの場合を確認
	for(int i=0;i<N;i++){
		if(A[i]==N+1){
			no=1;
			break;
		}
	}
```

```
if(no == 0){
        //B[N]の解判定
        for(int j=0;j<N;j++){
                x=A[j];//コマの位置情報をxに代入

                //右上斜め判定
                if(p[j+x]==0){
                        p[j+x]=1;
                }
                else{
                        sum+=1;
                }

                //右下斜め判定
                if(q[j-x+(N-1)]==0){
                        q[j-x+(N-1)]=1;
                }
                else{
                        sum+=1;
                }
        }

        //解を発見したらstring型でvectorに保存
        if(sum==0){
                for(int i=0;i<N;i++){
                        l<<A[i];
                }
                vector.push_back(l.str());

                //重複解を見つけて取り出す
                size = vector.size();
                for(int j2=0;j2<size-1;j2++){
                        if(vector[j2]==vector[size-1]){
                                vector.pop_back();
                                break;
                        }
                }


                //右に90度回転した結果を入れる
                for(int j3=0;j3<N;j3++){
                        for(int k=0;k<N;k++){
                                if(A[k]==j3){
                                        newa[j3]=(N-1)-k;
                                        break;
                                }
```

```
                }
        }
        for(int i=0; i<N; i++){
                m << newa[i];
        }
        vector.push_back(m.str());

        //重複解を見つけて取り出す
        size = vector.size();
        for(int j4=0;j4<size-1;j4++){
                if(vector[j4]==vector[size-1]){
                        vector.pop_back();
                        break;
                }
        }

        //右に180度回転した結果を入れる
        for(int e=0;e<N;e++){
                for(int v=0;v<N;v++){
                        if(newa[v]==e){
                                newb[e]=(N-1)-v;
                                break;
                        }
                }

        }
        for(int i=0; i<N; i++){
                z << newb[i];
        }
        vector.push_back(z.str());

        //重複解を見つけて取り出す
        size = vector.size();
        for(int j=0;j<size-1;j++){
                if(vector[j]==vector[size-1]){
                        vector.pop_back();
                        break;
                }
        }

        //右に270度回転した結果を入れる
        for(int j3=0;j3<N;j3++){
                for(int k=0;k<N;k++){
                        if(newb[k]==j3){
                                newc[j3]=(N-1)-k;
                                break;
                        }
```

```
            }
        }
        for(int i=0; i<N; i++){
            m1 << newc[i];
        }
        vector.push_back(m1.str());

        //重複解を見つけて取り出す
        size = vector.size();
        for(int j4=0;j4<size-1;j4++){
            if(vector[j4]==vector[size-1]){
                vector.pop_back();
                break;
            }
        }


        //左右反転した結果を入れる
        for(int x=0;x<N;x++){
            n << (N-1)-A[x];
        }
        vector.push_back(n.str());

        //重複解を見つけて取り出す
        size = vector.size();
        for(int j=0;j<size-1;j++){
            if(vector[j]==vector[size-1]){
                vector.pop_back();
                break;
            }
        }

        //上下反転した結果を入れる
        for(int j=0;j<N;j++){
            h1 <<A[(N-1)-j];
        }
        vector.push_back(h1.str());

        //重複解を見つけて取り出す
        size = vector.size();
        for(int j=0;j<size-1;j++){
            if(vector[j]==vector[size-1]){
                vector.pop_back();
                break;
            }
        }
```

```
                //90° 回転して左右反転した結果を入れる
                for(int x=0;x<N;x++){
                        o1 << (N-1)-newa[x];
                }
                vector.push_back(o1.str());

                //重複解を見つけて取り出す
                size = vector.size();
                for(int j5=0;j5<size-1;j5++){
                        if(vector[j5]==vector[size-1]){
                                vector.pop_back();
                                break;
                        }
                }

                //180° 回転して左右反転した結果を入れる
                for(int x=0;x<N;x++){
                        o2 << (N-1)-newb[x];
                }
                vector.push_back(o2.str());

                //重複解を見つけて取り出す
                size = vector.size();
                for(int j5=0;j5<size-1;j5++){
                        if(vector[j5]==vector[size-1]){
                                vector.pop_back();
                                break;
                        }
                }

                //90° 回転させ上下反転した結果を入れる
                for(int j=0;j<N;j++){
                        h2 <<newa[(N-1)-j];
                }
                vector.push_back(h2.str());

                //重複解を見つけて取り出す
                size = vector.size();
                for(int j=0;j<size-1;j++){
                        if(vector[j]==vector[size-1]){
                                vector.pop_back();
                                break;
                        }
                }

        }
```

```
                sum=0;

        }
        no=1;
}
```

fcheckB.cpp

```
#include<iostream>
#include"set.h"
#include<vector>
#include<string>
#include<sstream>
/**
*@param B[N] TypeBの駒の配置を保存している
*@param vector 解となった駒の配置情報を保存する
*/
void fcheckB(int B[N],std::vector<std::string>& vector){
        int x=0;
        int p[2*N-1];//右斜め上の配列
        int q[2*N-1];//右斜め下の配列
        int sum=0;
        int size=0;
        int no=0;

        int newb[N];

        std::ostringstream l;
        std::ostringstream m;
        std::ostringstream n;
        std::ostringstream o;


        //配列の初期化
        for(int t=0;t<2*N-1;t++){
                p[t]=0;
                q[t]=0;
        }

        //配置なしの場合を確認
        for(int i=0;i<N;i++){
                if(B[i]==N+1){
                        no=1;
                        break;
                }
        }
```

```
if(no == 0){
        //B[N]の解判定
        for(int j=0;j<N;j++){
                x=B[j];//コマの位置情報をxに代入

                //右上斜め判定
                if(p[j+x]==0){
                        p[j+x]=1;
                }
                else{
                        sum+=1;
                }

                //右下斜め判定
                if(q[j-x+(N-1)]==0){
                        q[j-x+(N-1)]=1;
                }
                else{
                        sum+=1;
                }
        }

        //解を発見したらstring型でvectorに保存
        if(sum==0){
                for(int i=0;i<N;i++){
                        l<<B[i];
                }
                vector.push_back(l.str());

                //重複解を見つけて取り出す
                size = vector.size();
                for(int j2=0;j2<size-1;j2++){
                        if(vector[j2]==vector[size-1]){
                                vector.pop_back();
                                break;
                        }
                }


                //右に90度回転した結果を入れる
                for(int j3=0;j3<N;j3++){
                        for(int k=0;k<N;k++){
                                if(B[k]==j3){
                                        newb[j3]=(N-1)-k;
                                        break;
                                }
```

```
                }
        }
        for(int i=0; i<N; i++){
                m << newb[i];
        }
        vector.push_back(m.str());

        //重複解を見つけて取り出す
        size = vector.size();
        for(int j4=0;j4<size-1;j4++){
                if(vector[j4]==vector[size-1]){
                        vector.pop_back();
                        break;
                }
        }

        //左右反転した結果を入れる
        for(int x=0;x<N;x++){
                n << (N-1)-B[x];
        }
        vector.push_back(n.str());

        //重複解を見つけて取り出す
        size = vector.size();
        for(int j=0;j<size-1;j++){
                if(vector[j]==vector[size-1]){
                        vector.pop_back();
                        break;
                }
        }

        //90° 回転して左右反転した結果を入れる
        for(int x=0;x<N;x++){
                o << (N-1)-newb[x];
        }
        vector.push_back(o.str());

        //重複解を見つけて取り出す
        size = vector.size();
        for(int j5=0;j5<size-1;j5++){
                if(vector[j5]==vector[size-1]){
                        vector.pop_back();
                        break;
                }
        }
```

```
                }
                sum=0;

        }
        no=1;
}
```

fcheckC.cpp

```
#include<iostream>
#include"set.h"
#include<vector>
#include<string>
#include<sstream>
/**
*@param C[N] TypeCの駒の配置を保存している
*@param vector 解となった駒の配置情報を保存する
*/
void fcheckC(int C[N],std::vector<std::string>& vector){
        int x=0;
        int p[2*N-1];//右斜め上の配列
        int q[2*N-1];//右斜め下の配列
        int sum=0;
        int size=0;
        int no=0;

        std::ostringstream l;
        std::ostringstream n;

        //配列の初期化
        for(int t=0;t<2*N-1;t++){
                p[t]=0;
                q[t]=0;
        }

        //配置なしの場合を確認
        for(int i=0;i<N;i++){
                if(C[i]==N+1){
                        no=1;
                        break;
                }
        }


        if(no == 0){
                //B[N]の解判定
                for(int j=0;j<N;j++){
```

```
                x=C[j];//コマの位置情報をxに代入

                //右上斜め判定
                if(p[j+x]==0){
                        p[j+x]=1;
                }
                else{
                        sum+=1;
                }

                //右下斜め判定
                if(q[j-x+(N-1)]==0){
                        q[j-x+(N-1)]=1;
                }
                else{
                        sum+=1;
                }
        }

        //解を発見したらstring型でvectorに保存
        if(sum==0){
                for(int i=0;i<N;i++){
                        l<<C[i];
                }
                vector.push_back(l.str());

                //重複解を見つけて取り出す
                size = vector.size();
                for(int j2=0;j2<size-1;j2++){
                        if(vector[j2]==vector[size-1]){
                                vector.pop_back();
                                break;
                        }
                }

                //左右反転した結果を入れる
                for(int x=0;x<N;x++){
                        n << (N-1)-C[x];
                }
                vector.push_back(n.str());

                //重複解を見つけて取り出す
                size = vector.size();
                for(int j=0;j<size-1;j++){
                        if(vector[j]==vector[size-1]){
                                vector.pop_back();
                                break;
```

```
                                        }
                                }
                        }
                        sum=0;
                }
                no=1;
}
```

set.h

```
#define N 8
#define MAX 1000
```