Vol.2011-GI-26 No.11
2011/7/1

NQueens 問題への新しいアプローチ（部分解合成法）について

萩野谷　一二

1. はじめに

N クイーン問題は、よく知られた 8 パズルを拡張したものであり、クイーンの数(N)が大きくなるにつれて解の数が膨大となることが知られている。基本的な探索方法としては、Jeff Somers 氏のビット演算を用いたエレガントなアルゴリズムが有名である。最近の成果(N=24,26)は、Jeff Somers 氏のアルゴリズムの改良版をベースに、大規模な computing power を投入した並列処理によるものである。しかし、物量作戦もほぼ限界に近く、次の N=27 の解を求めるには基本アルゴリズムの抜本的な改善が必要と思われる。

本稿では、「部分解を合成して全体解を得る」というアプローチにより、既存 Solver の約 10 倍という大幅な性能向上が実現できたことを報告する。

1.1 N クイーン問題

N クイーン問題とは、N x N のチェスボードに N 個のクイーンを配置する場合に、お互いのクイーンが他のクイーンの効き筋に入らないような配置（図１）の数を求めるパズルである。N が大きくなるにつれて、解の数が膨大になることが知られており、最近では、並列処理の研究題材としても扱われている。

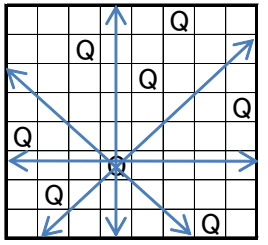


図 1　クイーンの配置例

1.2 最近の状況

既に判明している N クイーン問題の解の数を表 1 に示す。N が増えるにつれて、解の数は 8～10 倍に増加していく。また、N=27 以降は未解決である。

表 1　N クイーン問題の最新状況[1][2][3][4][5]

N	公表日	すべての解の数	unique解の数
24	2004.04.11	227,514,171,973,736	28,439,272,956,934
25	2005.06.11	2,207,893,435,808,350	275,986,683,743,434
26	2009.07.11	22,317,699,616,364,000	2,789,712,466,510,280
27	未解決		

また、解の数の増大につれて、解法に必要な CPU 資源も増大する。（表 2 参照）クイーンの数が 1 つ増えるごとに、約 10 倍の CPU 資源が必要となる。もはや、多数の CPU 資源を長期間に渡って使用しなければ求めることができない領域に入っている。

ⓒ2011 Information Processing Society of Japan

表 2　N クイーン問題の解法に必要な CPU 資源

N	公表日	公表機関名	基本ﾌﾟﾛｸﾞﾗﾑ	CPU 資源
24	2004.04.11	電気通信大学	qn24b	1,496 CPU・日[a]
25	2005.06.11	ProActive	不明	18,250 CPU・日[b]
26	2009.07.11	Tu-dresden	JSomers 版の改良版	168,960 CPU・日[c]

最近の Solver は、Jeff Somers 氏のビット演算を使った探索アルゴリズムをベースにしたものが多い。

JSomer 版(仮称）[7]：Jeff Somers 氏が N=23 の解を求めるときに使用した解法。
巧みなビット演算による高速化が特徴。

qn24b　[8]　：電通大で N=24 を求める時に使用した解法。
JSomer 版を改良し、7～14%の性能向上を実現。

takaken 版(仮称）[9]：JSomer 版を高橋謙一郎氏が改良を図ったもの。
対称性に着目して代表解（後述）のみを探索する高速化と
再帰呼び出しによるプログラム解読性の向上。

2.　部分解合成法（ PAA：Parts Assenbly Approach ）

以下では、クイーンの数が偶数（N=2n）の場合について説明する。

2.1　準備

・演算子記号

#(X)：集合 X の要素の数

～X：X のビットを反転したもの。　X=011 → ～X=100

rBit(X)：X のビットを上位下位反転させたもの。X=011　→ rBit(X)=110

Bit(n)：２の n 乗。　n=4　→ Bit(4)=10000

・解の分類

N クイーン問題の解は、回転・反転操作により、一般に 7 個の別解を得ることができるが、稀なパターンとして回転操作により元の解と一致する解も存在する。

そこで、解の回転対称性に着目し、以下の 3 種類に分類する。(図2参照)

type-a　：　対称性なしの解（回転や反転を行っても元の解と一致しない解）
回転・反転操作により、7 個の別解が得られる

type-b　：　180°回転すると元の解と一致する解（180°回転対称解）
回転・反転操作により、3 個の別解が得られる（下記の type-c を除外）

type-c　：　90°回転すると元の解と一致する解（ 90°回転対称解）
反転操作により、1 個の別解が得られる

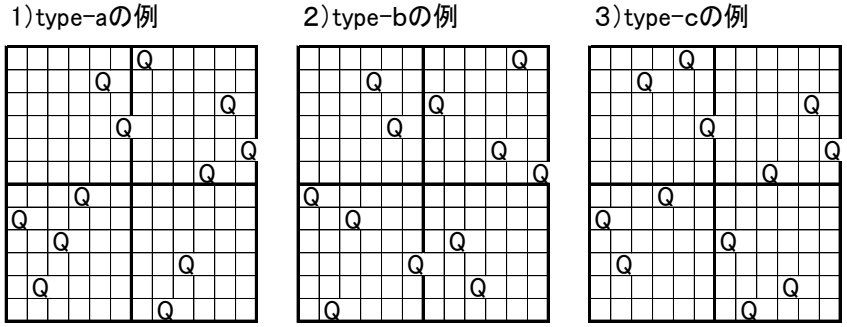


図 2 各タイプの解の例（N=12）

代表解　：回転・反転操作によって生成される解グループの中から 1 つを選んで代表解とする。代表の選び方は任意である。

unique 解　：　代表解の総和

全数解　：　すべての解をカウントする数え方(回転・反転操作による別解も計上)

unique 解　＝　A　　+　B　　+　C

全数解　　＝　8*A　+ 4*B　+　2*C　---------　式 1

ただし、A,B,C は、それぞれ type-a，type-b，type-c の代表解の数とする

・nq(m,n,q), nq(m,n)

m 行 n 列のチェスボードに q 個のクイーンをお互いの効き筋に当たらないように配置する解の集合を nq(m,n,q)で表す。nq(m,n)は、クイーンの数を任意とした場合の解の集合とする。（N クイーン問題は、nq(N,N,N)の要素の数といえる）

・ボードの分割

チェスボードを縦、横にそれぞれ 2 分割した 4 つのサブボード （大きさは n x n）を時計回りに A,B,C,D とする（図 3）。A,B,C,D は、nq(n,n)の要素である。

a ）　文献[3]：68CPU x 22 日より計算
b ）　文献[4]：単一 CPU 換算で 50 年以上より計算
c ）　文献[6]：FGPA の数は 8 枚 * 22 台 = 176 枚。
FGPA 1 枚は、2.5GHz-QuadCore system(4CPU)に相当　（176 * 4CPU = 704CPU）
計算期間 240 日より、704 * 240 = 168,960 CPU・日

ⓒ2011 Information Processing Society of Japan

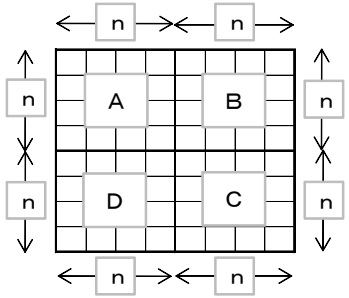


図 3 ボードの分割

簡単な計算により、以下の関係が得られる。

#(A) = #(C) = m
#(B) = #(D) = n - m ——— 式 2

ただし、m = 0～n の整数とする

A のクイーンの数が決まると、B,C,D のクイーンの数も決まる点は注目に値する。

・特性値

部分解の各要素(それぞれの解について）に図 4 の座標系(x,y)を導入する。また、クイーンの配置情報を表す 4 つの特性値を以下のように定義する。

X 特性値 ： 各列にクイーンが配置されているかどうかを表す指標
Y 特性値 ： 各行にクイーンが配置されているかどうかを表す指標
U 特性値 ： ななめ方向(直線 X-Y=定数)にクイーンが配置されているかどうかを表す指標。
V 特性値 ： ななめ方向(直線 X+Y=定数)にクイーンが配置されているかどうかを表す指標。

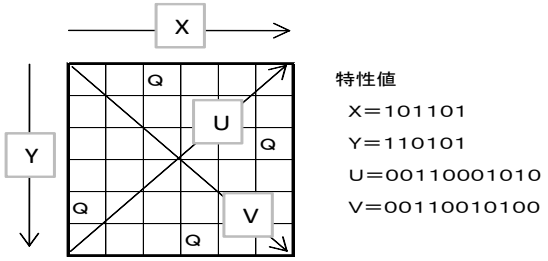


図 4 特性値の定義

この 4 つの特性値により、部分解から順次結合して全体解を得る処理の適否を判定できる。 ただし、4 つの特性値のみでクイーンの配置が唯一に決まるという保証はない。

・識別子、pid

部分解の集合 nq(n,n,m)の各要素に以下のような unique な番号をつけ、識別子と呼ぶことにする。

識別子＝（X 特性値, Y 特性値, pid)
pid は、（X 特性値, Y 特性値）が同じものを並べたときの相対位置

・部分解の配置とフレーム

部分解をどのように配置して全体解を構成するかを規定する。
図 5 のように、P, Q, R, S を部分解の要素、A, B, C, D, a, b, c, d をフレーム値とする。
P, Q, R, S は、時計まわりに 90 度回転した配置とする。

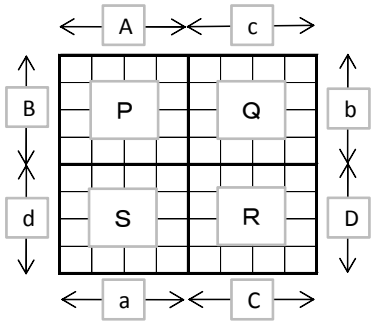


図 5 フレーム

このとき、各フレーム値を以下のように定義する。

A ：rBit(P の X 特性値)、 B ：rBit(P の Y 特性値)
C ：rBit(R の X 特性値)、 D ：rBit(R の Y 特性値)
a ：rBit(S の Y 特性値)、 b ：rBit(Q の X 特性値)
c ：rBit(Q の Y 特性値)、 d ：rBit(S の X 特性値)

フレーム値は、全体解を分割して処理するために導入した概念である。また、
a = ～A, b = ～B, c = ～C, d = ～D という関係が成立する。
補足：初版ではフレーム値＝特性値としていたが、性能面で有利なため変更した。

ⓒ2011 Information Processing Society of Japan

2.2 PAA の概要

N=2n の場合、N x N の N クイーン問題を 4 分割した部品(n x n の部分解)から以下の手順で合成する。(図 3 の表記を用いる)

1) A と B から、2 分割した部品（n x 2n の部分解）AB を生成（上半分の合成）
2) C と D から、2 分割した部品（n x 2n の部分解）CD を生成（下半分の合成）
3) AB と CD から全体解を生成

なお、解の生成は unique 解のみに限定し、全数解は式 1 により求める。
高速化実現のポイントは、

・結合処理の高速化（特に、キャッシュミスの削減）
・代表解の判定条件による集合積演算の軽減

にある。

2.3 結合処理

(1) QJoin（ 2 つの 1/4 解から 1/2 解を生成する ）

1/4 解 A, B およびその結合の結果得られる 1/2 解の特性値を図 6 とする。

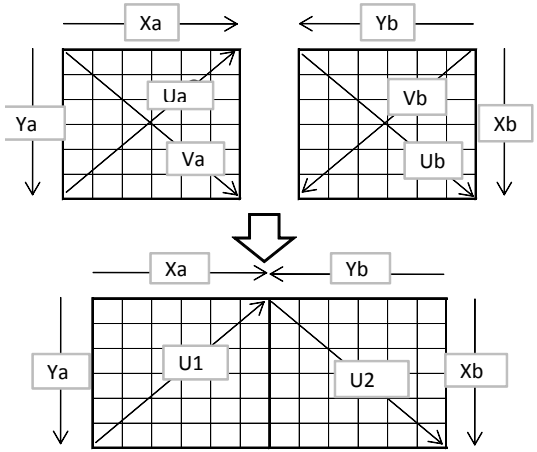


図 6　QJoin の各特性値の定義

この時、結合条件は、

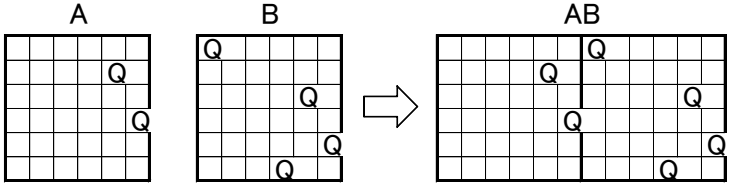
① Ya = ～Xb
② (Ua >> n) & rBit(Vb >> n) = 0
③ Ub & (Va >> n) == 0

---------- 式 3

結果として、

U1 = Ua | (rBit(Vb>>n) << n)
U2 = Ub | (Va >> n)

が得られる。具体例を図 7 に示す。



Ya=001010　　Xb=110101　　条件①, ②, ③は成立
Ua=00110000000　Ub=01101000001　U1=01110000000
Va=00100100000　Vb=00010111000　U2=01101000101

図 7　QJoin の例

(2) HJoin （2 つの 1/2 解から全体解を生成する ）

1/2 解、および全体解の特性値を図 8 のように定める。

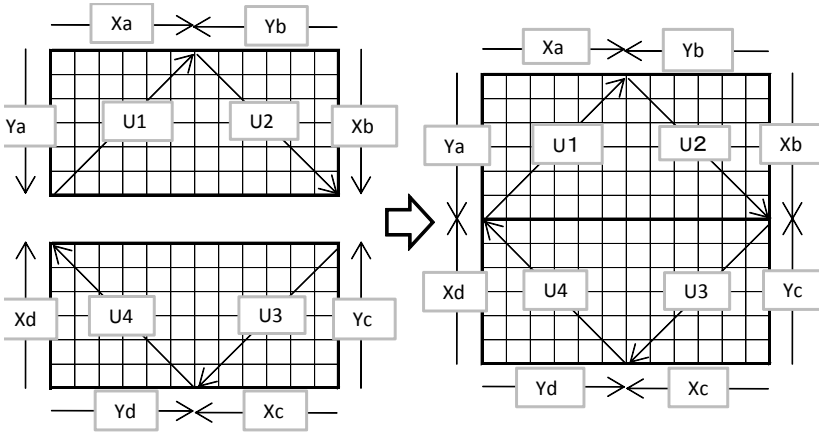


図 8　HJoin の各特性値の定義

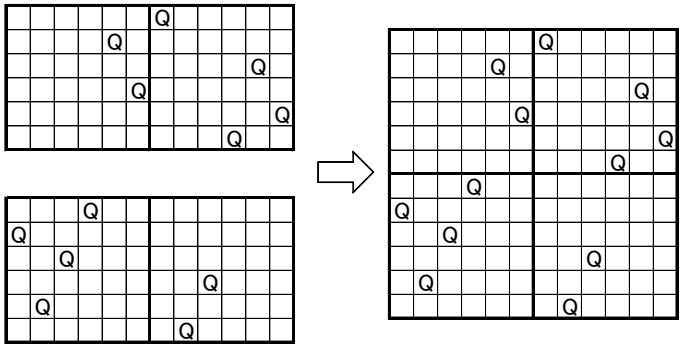
ⓒ2011 Information Processing Society of Japan

このときの結合条件は、

① Yb = ～Xc
　Yd = ～Xa
② U1 & rBit(U3) = 0
③ U2 & rBit(U4) = 0

---------- 式 4

となる。具体例を図 9 に示す。



Xa=110000　Xc=011000　条件①, ②, ③
Yb=100111　Yd=001111　はすべて成立するので
U1=01110000000　U3=01101000000　解として適切である
U2=01101000101　U4=01011100000

図 9　HJoin の例

2.4　代表解の判定条件

unique 解のみを求めるためには、代表解の判定条件を定める必要がある。説明のため、図 5 の記号に加えて、以下の記号を定める。

p, q, r, s　: P, Q, R, S に対応する pid
　また、V 軸反転解の pid を’で表示することとする
　ex.　p’ は、P の V 軸反転解の pid を示す

代表解の条件は、以下の条件 0～条件 8 をすべて満足することである。
ただし、条件 4～条件 8 は、#(P) = #(Q)のときのみ要請される条件である。

条件 0　: Q 条件　　クイーンの数の比較
　#(P) ≦ #(Q)
条件 1　:　V 条件　　V 軸反転解との比較
　(A, C)　<　(B, D)　or
　(A, C)　=　(B, D)　and　(p, r) <　(p’, r’)
条件 2　:　U 条件　　U 軸反転解との比較
　(A, B)　<　(D, C)　or
　(A, B)　=　(D, C)　and　p　<　r’
条件 3　: R180 条件　180 度回転解との比較
　(A, B)　<　(C, D)　or
　(A, B)　=　(C, D)　and　(p, q) ≦ (r, s)
条件 4　:　X 条件　　X 軸反転解との比較
　(A, B)　<　(c, b)
条件 5　:　Y 条件　　Y 軸反転解との比較
　A　<　a
条件 6　: R+90 条件　+90 度回転解との比較
　(A, B)　<　(b, c)　or
　(A, B)　=　(b, c)　and　p ≦ q
条件 7　: R-90 条件　-90 度回転解との比較
　(A, B)　<　(d, a)　or
　(A, B)　=　(d, a)　and　p ≦ s
条件 8　: (A,B) = (b,c) and (A,B) = (d,a)の場合の特殊条件
　(i.e　A=C　and　B=D　and　A=~B のとき)
　case 1 : p = q　and　p = s の場合
　　p = r なら　type-c
　　p≠ r なら　type-a
　case 2 : p = q　and　p ≠ s の場合
　　p≠ r　and　r　<　s
　case 3 : p ≠ q　and　p = s の場合
　　p≠ r　and　q ≦ r
　case 4 : p ≠ q　and　p ≠ s の場合
　　(p, q)　≦　(r, s)

なお、条件 3 で等号が成立する場合は、180 度回転対称解である。（type-b）
さらに条件 8 の case1 で等号で成り立つ場合は、90 度回転対称解である。（type-c）
上記以外は、回転対称解ではない。（type-a）

ⓒ2011 Information Processing Society of Japan

3. QJH-even 版の実装

以下では、クイーンの数が偶数の場合に PAA を実装したプログラム（QJH-even 版）の基本的な構造とその簡単な説明を行う。

```
for M =（1～n/2）          //Q条件より
                           //Mはサブボードの Queenの数
  PGen(M)                  //部分解の作成
  PGen(n-M)                //部分解の作成
  for frame.A = (1～Bit[n]-1)

    for frame.C = (frame.A～Bit[n]-1)  //R180条件より

      QJoin(A,B)           // A x B --> AB
      QJoin(C,D)           // C x D --> CD

      HJoin()              // AB x CD --> ABCD
```

図 10　QJH-even 版の基本構造

(1) 代表解の判定条件の実装

代表解の判定条件は、大抵の場合フレーム値で決まってしまうが、稀に pid まで検査しないと決まらない場合がある。例えば、frame.A > frame.B の場合は代表解とはならないことがＶ条件より直ちに判明する。一方、frame.A = frame.B = frame.C = frame.D の場合は、pid の検査が必要となる。　代表解の判定条件は、なるべく処理の早い段階で部分的にでも適用した方が得策である。（結合処理の対象を少なくする事が性能面では有利であるから)

(2) frame.A, C による分割処理

frame.A, C による分割処理をするもう一つの理由は、HJoin のすべての入力データをメモリ上に保持できないことにある。例えば、N=24 の場合、nq(n,2n)の解は 1T（テラ）個のオーダとなり、どうしても無理となる。frame.A, C を固定すると、自動的に frame.a, c が決まるので、A, c をフレームにもつ解と a, C をフレームにもつ解との合成となり、メモリに保持できる範囲となる。

(3) PGen(M)、PGen(n-M)：部分解 nq(n, n, M)と nq(n, n, n-M)を求める処理

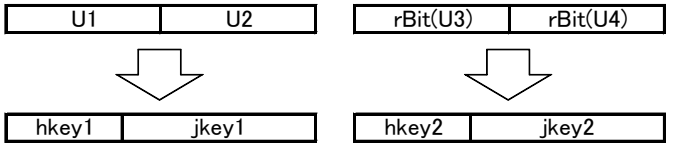
基本アルゴリズムとして、Jeff Somer 氏のビット演算方式を採用した。
得られた部分解は、特性値(x,y)をキーとするインデクスで管理する。また、各エントリのデータはキー順に並べ、QJoin 処理におけるキャッシュミスを少なくする。
また、代表解の判定条件で必要となる pid,pid'も決定する。

(4) QJoin()：1/4 解の結合により 1/2 解を求める処理

PGen で生成したインデクスを活用して、frame.B の 1 ～Bit(n)-1 の各値について結合処理を行う。具体的には、(frame.A,frame.B)と(frame.b,frame.c)の結合処理を式 3 により行う。ただし、代表解の判定条件（の一部）を適用して不適格と判定されたものは棄却する。結果は、後述の図 11 で示すハッシュキーにより、インデクスで管理し、PGen の処理と同様に、各エントリのデータはキー順に並べ、HJoin 処理におけるキャッシュミスを少なくする。

(5) HJoin()　：1/2 解の結合により全体解を求める処理

フレーム分割により 1/2 解の要素が少なくなった（N=24 では M(メガ)個のオーダ）とはいえ、全要素の集合積から結合条件の式 4 に適う解を抽出するのは、性能面で耐えられない。そこで、ハッシュを用いたジョイン処理を行う。
まず、図 11 のようにハッシュキーとジョインｷｰとに分割する。ハッシュキーの大きさ（ビット幅）やどの部分をハッシュキーとするかは、重要なチューニング要素である。



hkey1: U1，U2の一部を抽出
jkey1: U1，U2からhkey1を除いた残りのビット
hkey2: U3，U4の一部を抽出
jkey2: U3，U4からhkey1を除いた残りのビット

図 11　hash key の生成

ⓒ2011 Information Processing Society of Japan

この分割により、結合条件（式 4）は、以下のようになる。

hkey1 & hkey2 = 0 --------- 式 5
jkey1 & jkey2 = 0 --------- 式 6

hkey1 が与えられた時、式 5 を満たす hkey2 はビット演算により求めることができる。例えば、hkey1=01011 の時、hkey2= { 00000,00100,10000,10100 }となる。
次に、ハッシュキーのインデクスを使って、hkey1 → jkey1, hkey2 → jkey2 を求め、式 6 の条件を満たすものを選択する。まだ、解の分類が確定していない場合は、代表解の判定条件のチェックを行い、解の区分を決定する。

4. QJH-odd 版

クイーンの数が奇数（N=2n+1）の場合に PAA を実装したのが、QJH-odd 版である。N が奇数の場合、図 12 のように中央の行と列を境界として、4 つの n x n のサブボードに分割する。そして、中央の行・列に配置されたクイーンの位置により、type Ⅰ, type Ⅱに分類する。

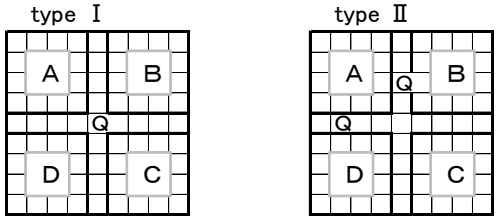


図 12 奇数の場合の分割

この分割により even 版とほぼ同様の考え方が適用できるので、odd 版については留意点のみとし、詳細は割愛する。

・type Ⅰの留意点：クイーンが中央に 1 つ配置された形
①代表解の判定条件は even 版と同様である。
②結合処理では、U 特性値、V 特性値が 1 ビットずれている考慮が必要。
③サブボードのクイーンの数は、even 版と同様である。
この場合、even 版とほとんど同じ処理となる。

・type Ⅱの留意点：２つのクイーンが配置されている形
①代表解の判定条件は、２つのクイーン（n, I)、(J, n）の位置関係により、
0≦I < J < n とする。また、type Ⅱでは回転対象解は存在しない。
②結合処理では、U 特性値、V 特性値が 1 ビットずれている考慮が必要。
③サブボードのクイーンの数は、以下のようになる。
#(A) = m - 1
#(B) = #(D) = n - m
#(C) = m
ただし、1≦m≦n とする

4 つの部分解 A, B, C, D は、中央の行・列に配置されたクイーンと競合しない解のみが有効となるため、nq(n, n)のサブセットになる。nq(n, n)からの抽出条件を表 3 に示す。

表 3 部分解 nq(n,n)からの抽出条件

		A	B	C	D
type Ⅰの場合	U特性値	U(n)=0	U(n)=0	U(n)=0	U(n)=0
type Ⅱの場合	X特性値	X(I)=0	X(J)=0		
	Y特性値	&Y(J)=0			Y(I)=0
	U特性値	&U(I-1)=0	&U(I-1)=0	U(2n-I-1)=0	&U(2n-I-1)=0
		&U(2n-J-1)=0	&U(J-1)=0	&U(J-1)=0	&U(2n-J-1)=0
	V特性値	&V(I)=0	&V(J)=0		&V(I)=0
		&V(J)=0			

5. 実測結果

QJH、takaken 版、電通大版(qn24b)を下記の測定環境において実測した結果について報告する。

測定環境（ 使用 PC：FMV-BIBLO NF/B50 ）
CPU : Intel Core2 DUO processor T8100 2.1GHz
キャッシュ： 3MB
メモリ : 4GB
OS : Windows Vista

ⓒ2011 Information Processing Society of Japan

表 4 は、他 Solver との比較を表している。この比較より、

・takaken 版は、電通大版(qn24b)の約 2 倍

・QJH は、takaken 版の約 10 倍

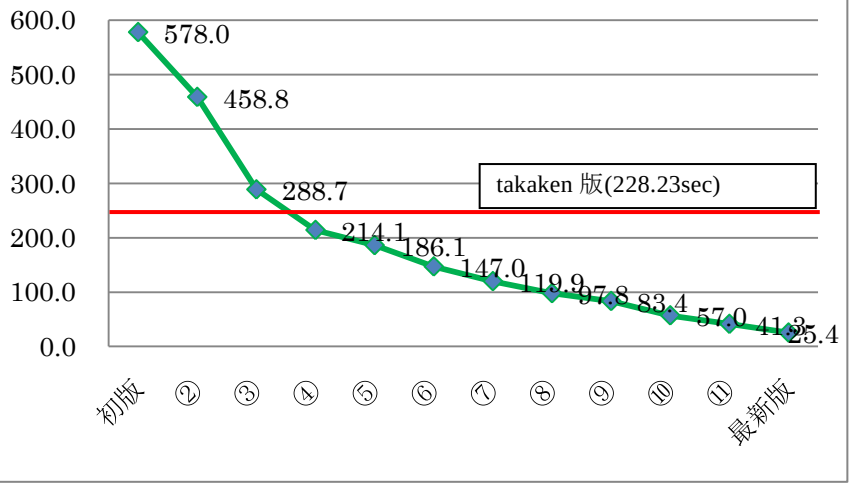
高速であることがわかる。

表 5 は、開発過程の性能値をプロットしたものである。初版は遅かったが、処理論理の改善やメモリシステム（キャッシュミス）を意識したチューニング等により最新版に至っている。

表 4　他 Solver との性能比較

N	takaken版		電通大版(qn24b)		QJH版	
	実測値	相対比	実測値	相対比	実測値	相対比
18	3:48.23	1.0	7:00.717	1.84	25.37	0.111
19	29:22.0	1.0	57:16.602	1.95	03:17.6	0.112
20	3:54:10.34	1.0	7:19:24.765	1.88	24:07.4	0.103
21	33:14:19.48	1.0			3:05:28.2	0.093
22					27:08:40.5	

表５　QJH開発経過データ(N=18の探索時間)



6. まとめ

N クイーン問題の場合、部品の結合が outer join（≠結合）のため、最適の対象とは言えないが、それでも既存 Solver の約 10 倍の性能向上を図ることができた。もし、inner join（＝結合）の問題であれば、さらに強力な効果が期待できるであろう。

一方、部分解合成法はプログラムが複雑になることが難点である。しかし、それに見合った大きな成果が得られることで十分報われると考えている。この点に関しては、他のパズルへの適用例を増やして実証していきたい。

また、QJH は、メモリを大量に使用するアルゴリズムであるため、並列化処理を行った場合に十分な性能が実現できるかどうかは未知数である。是非、検証したい課題と考える。

参考文献

1) Eight queens puzzle , http://en.wikipedia.org/wiki/Eight_Queens_puzzle
2) Number of ways of placing n nonattacking queens on n X n board., http://www.research.att.com/~njas/sequences/A000170
3) N-QUEENS の世界記録樹立，6 年分の計算を並列処理により 22 日に短縮 http://www.arch.cs.titech.ac.jp/~kise/nq/press-2004-10-05.txt
4) ProActive Parallel Siute, http://proactive.inria.fr/index.php?page=nqueen25 (2010.04.24)
5) Queens@TUD, http://Queens.inf.tu-dresden.de/
6) Queens@TUD, RESULTS, http://Queens.inf.tu-dresden.de/?l=en&n=r
7) The N Queens Problem, http://www.jsomers.com/nqueen_demo/nqueens.html
8) N-queens Homepage in Japanese , http://www.arch.cs.titech.ac.jp/~kise/nq/index.htm
9) ＮQueen 問題（解の個数を求める）, http://www.ic-net.or.jp/home/takaken/nt/queen/index.html

ⓒ2011 Information Processing Society of Japan