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

email : deepgreen@libra.nifty.jp

あらすじ

NQueens 問題とは

準備（用語の説明）

部分解合成法（**PAA**）の概要

QJH-even 版の実装

QJH-odd 版の概要

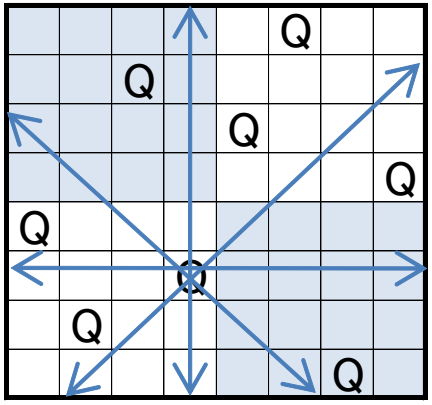
実測結果

まとめ

NQueens 問題とは

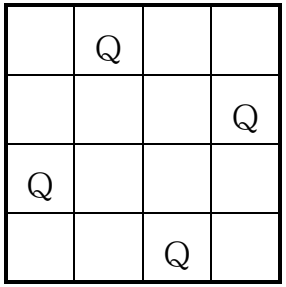
下記の条件を満たすクイーン配置の数を求める問題

- ＮｘＮのチェスボードにＮ個のクイーンを配置
- クイーンはお互いの効き筋をさけて配置



８ｘ８の問題の例

- クイーンの効き筋は縦、横、斜めの４方向
- 反転対称解は存在しない
- 回転対称解は存在する（90°、180°）



最近の状況

すでに解決済の問題

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		

- クイーンの数とともに、急激に解の数も増大（約 10 倍のペース）
- N=27 以降は未解決

解決に必要な CPU 資源

N	公表日	公表機関名	基本ﾌﾟﾛｸﾞﾗﾑ	CPU 資源（延べ）
24	2004.04.11	電気通信大学	qn24b	1,496 CPU・日　[1]
25	2005.06.11	ProActive	不明	18,250 CPU・日　[2]
26	2009.07.11	Tu-dresden	JSomers 版の改良版	168,960 CPU・日　[3]

- 必要な CPU 資源も約 10 倍ずつ増大
- PC クラスタや FGPA を使った並列処理が主流

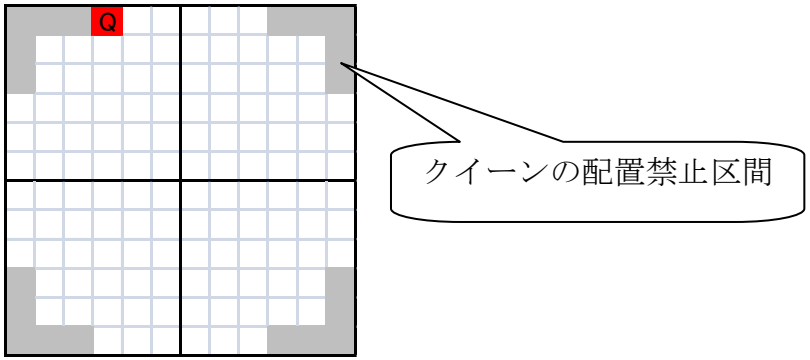
最近の Solver

Solver 名	特徴など
JSomers 版(仮称)[7]	● 巧みなビット演算による高速化 ● 上下反転の解を考慮し、探索を半分に削減 ● Jeff Somers 氏が N=23 の解を求めるときに使用した解法
qn24b [8]	● JSomer 版を改良し、7～14%の性能向上 ● 電通大で N=24 を求める時に使用した解法
takaken 版(仮称)[9]	● JSomer 版を高橋謙一郎氏が改良 ● 対称性に着目して代表解(後述)のみを探索 ● 再帰呼び出しによるプログラム解読性の向上

JSomers 版のビット演算

$$bit = -M_n \ \& \ M_n$$
$$X_{n+1} = X_n \mid bit$$
$$L_{n+1} = (L_n \mid bit) << 1$$
$$R_{n+1} = (R_n \mid bit) >> 1$$
$$M_{n+1} = (2^N - 1) \ \& \ \sim(X_{n+1} \mid L_{n+1} \mid R_{n+1})$$

takaken 版の代表解の判定条件



開発の経緯

□未解決の問題に挑戦するには、約 **10** 倍の性能 **up** が必要

takaken 版の性能 ：**3.2** μ **s /** 解 **1** 個 ≒ **6400** クロック / 解 **1** 個

→ **JSomers** 版のビット演算方式では実現困難

□部分解合成法では？

N=24 の場合

半分の問題（**12 x 24**）は約 **5** 時間で求まるが、全体の問題（**24 x 24**）は **1496 CPU**・日かかる

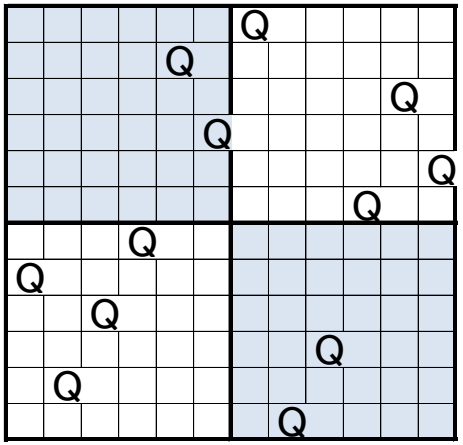
また、半分の問題の解の数は約 **1.5T** 個。フレーム分割（後述）すれば、メモリ常駐可能な範囲

→ 可能性あり

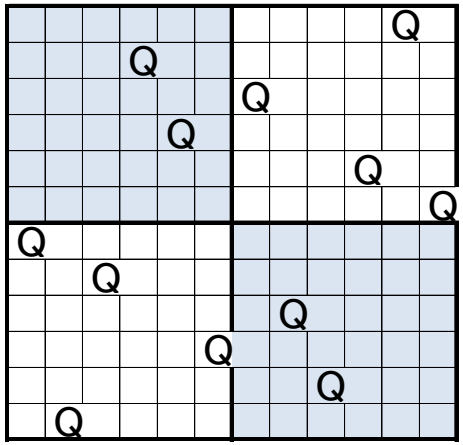
課題はジョイン処理の高速化

解の分類

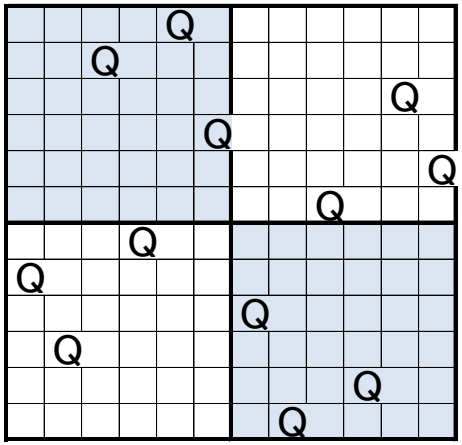
1）type-aの例



2）type-bの例



3）type-cの例



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）

type-b ： 180° 回転対称解（180° 回転すると元の解と一致する解）

type-c ： 90° 回転対称解（90° 回転すると元の解と一致する解）

代 表 解 ： 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

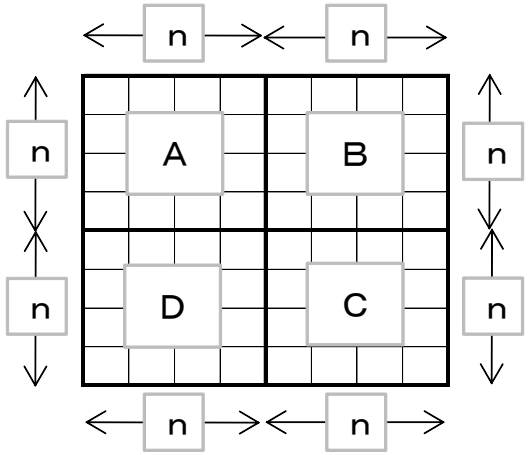
unique 解 ＝ A + B + C （代表解の総和）

全数解 ＝ 8*A + 4*B + 2*C （すべての解をカウント）

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

ボードの分割

チェスボードを縦、横にそれぞれ 2 分割。　　時計回りに A,B,C,D とする



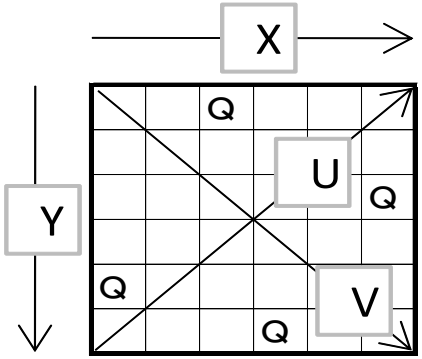
A～D のクイーンの数

#(A) = #(C) = m

#(B) = #(D) = n – m　　ただし、m = 0～n の整数

⇒ 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

特性値

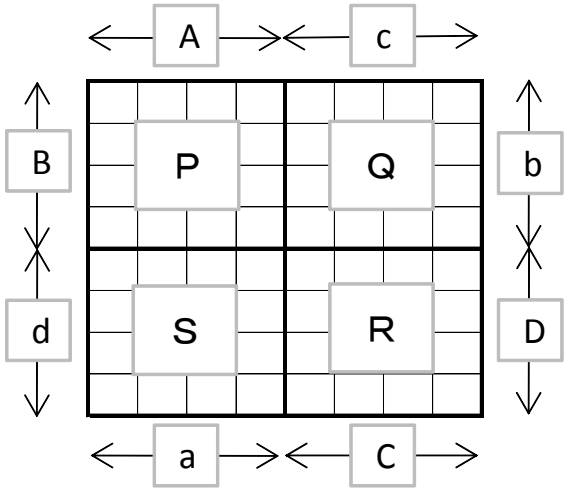


X 特性値 ：	各列(X 方向)のクイーンの配置を表す指標	X=101101
Y 特性値 ：	各行(Y 方向)のクイーンの配置を表す指標	Y=110101
U 特性値 ：	ななめ方向(U 方向)のクイーンの配置を表す指標	U=00110001010
V 特性値 ：	ななめ方向(V 方向)のクイーンの配置を表す指標。	V=00110010100

部分解の配置とフレーム

P，Q，R，S ： 部分解の要素 （P, Q, R, S は、時計まわりに 90 度回転した配置）

A, B, C, D, a, b, c, d ： フレーム値



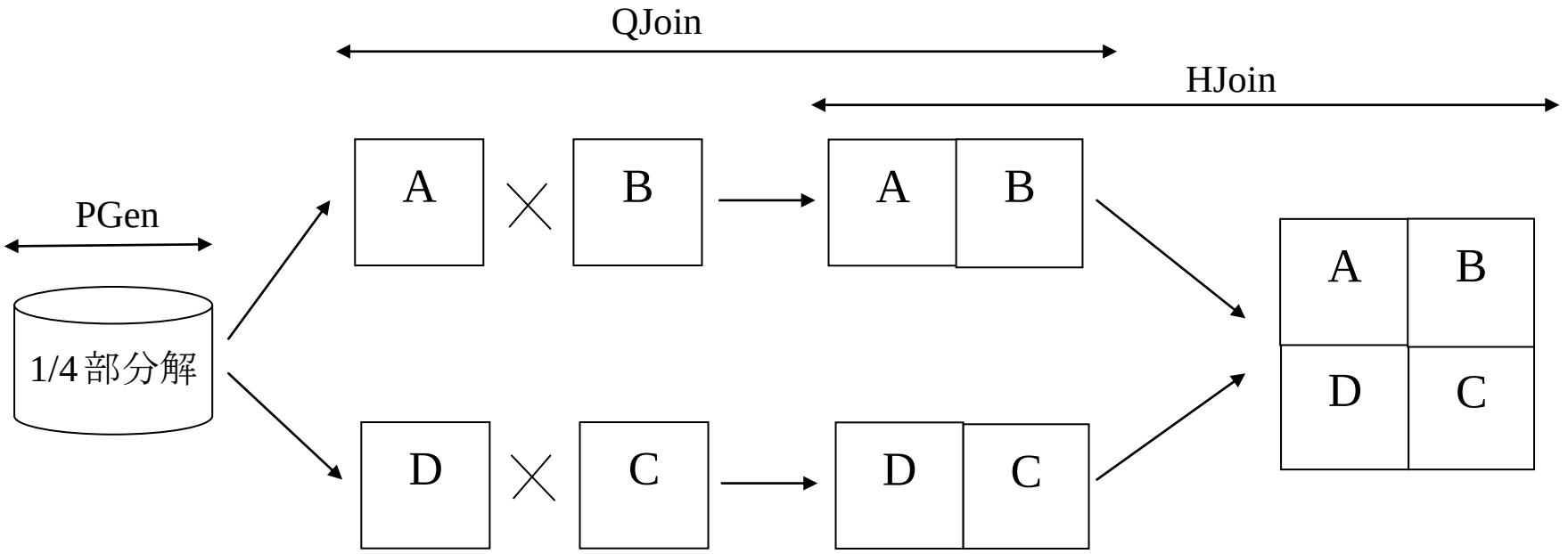
A ：rBit(P の X 特性値)	B ：rBit(P の Y 特性値)
C ：rBit(R の X 特性値)	D ：rBit(R の Y 特性値)
a ：rBit(S の Y 特性値)	b ：rBit(Q の X 特性値)
c ：rBit(Q の Y 特性値)	d ：rBit(S の X 特性値)

a = ～A， b = ～B， c = ～C， d = ～D という関係が成立

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

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

部分解合成法(PAA)の概要

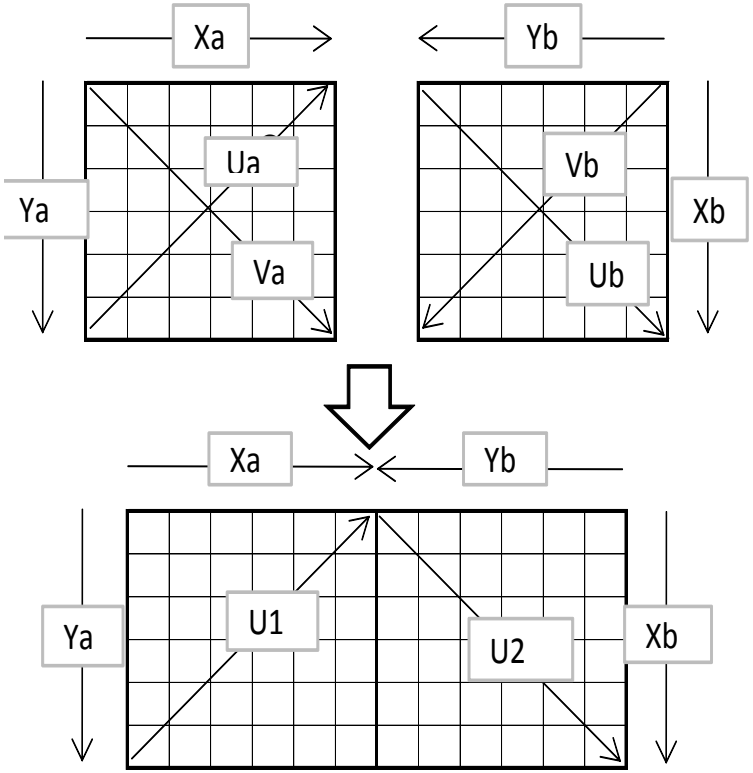


解の生成は unique 解のみに限定

高速化のポイント

- 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

QJoin（ 2 つの 1/4 解から 1/2 解を生成 ）



結合条件

① Ya = ～Xb

② (Ua >> n) & rBit(Vb >> n) = 0

③ Ub & (Va >> n) == 0

結果

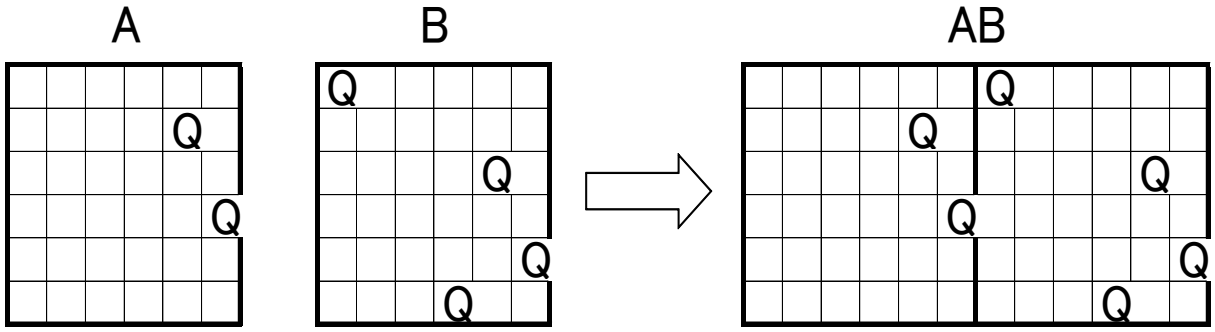
① U1 = Ua | (rBit(Vb>>n) << n)

② U2 = Ub | (Va >> n)

～　　：ビット反転

rBit()：ビットの上下反転

QJoin の例

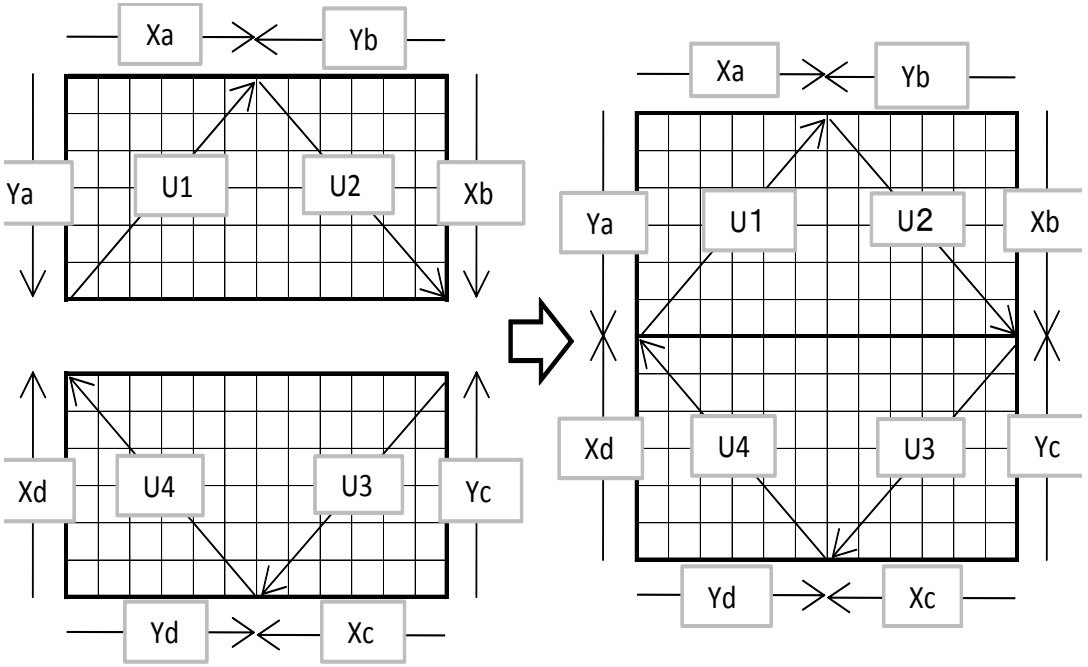


Ya=001010
Ua=00110000000
Va=00100100000

Xb=110101
Ub=01101000001
Vb=00010111000

条件①, ②, ③は成立
U1=01110000000
U2=01101000101

HJoin （2 つの 1/2 解から全体解を生成 ）



結合条件

① Yb = ～Xc

Yd = ～Xa

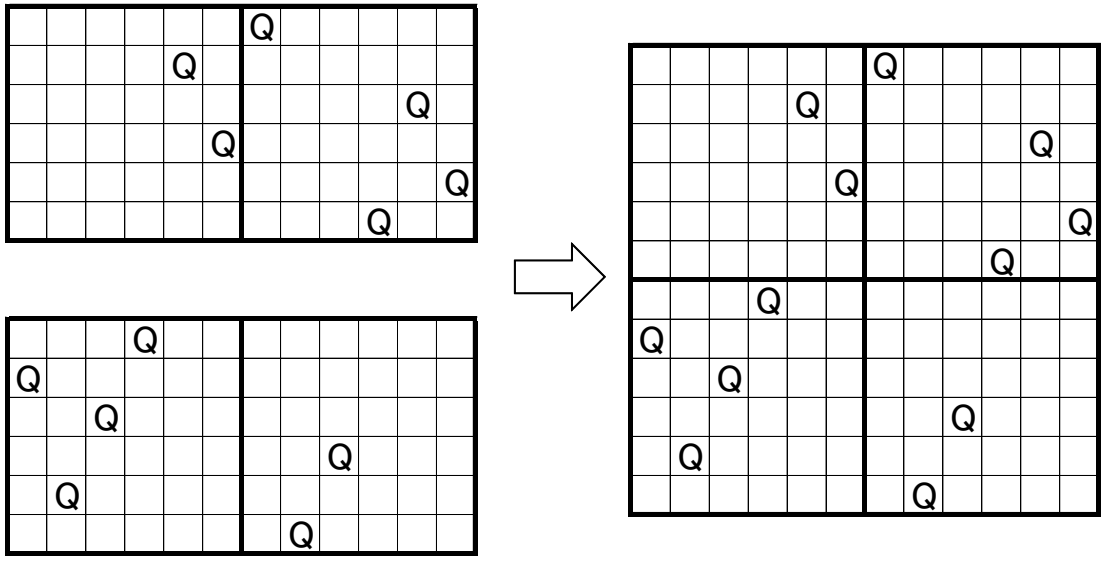
② U1 & rBit(U3) = 0

③ U2 & rBit(U4) = 0

～　　：ビット反転

rBit（）：ビットの上下反転

HJoin の例



Xa=110000
Yb=100111
U1=01110000000
U2=01101000101

Xc=011000
Yd=001111
U3=01101000000
U4=01011100000

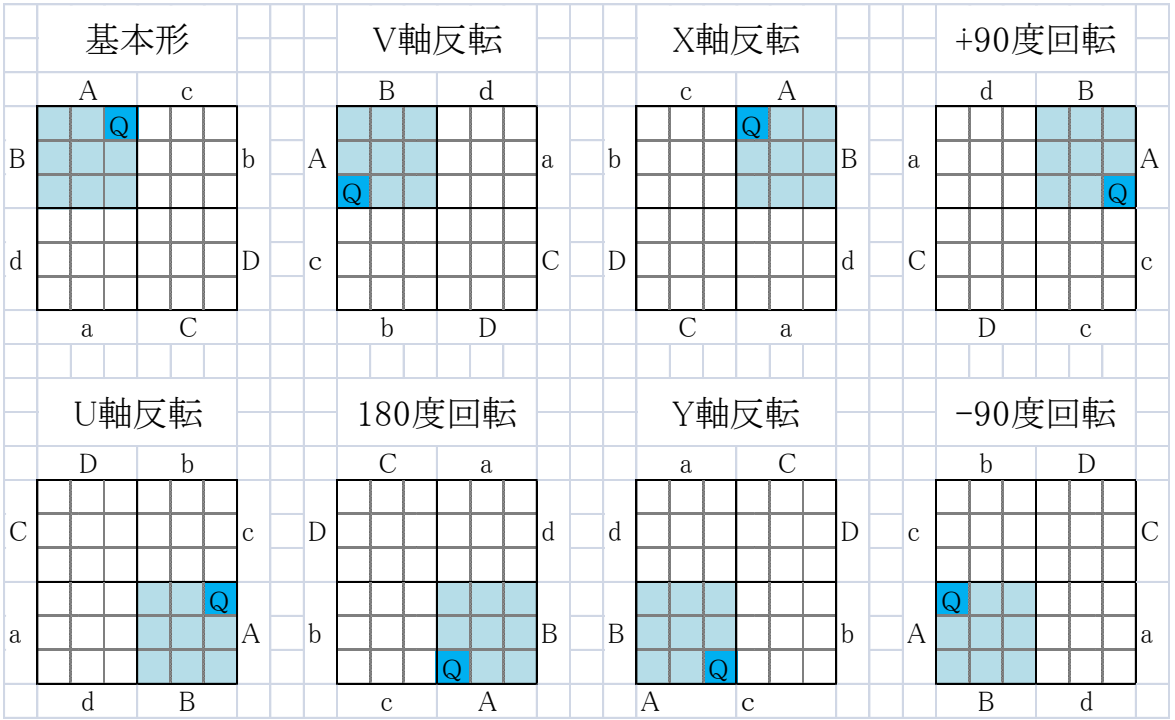
条件①, ②, ③
はすべて成立するので
解として適切である

代表解の判定条件(unique 解の選択方法)

考え方：　①　クイーンの数の比較
②　フレーム値の比較
③　**pid** の比較

代表解の条件：条件 **0**～条件 **8** をすべて満足すること

ただし、条件 **4**～条件 **8** は、**#(P) = #(Q)**のときのみ



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 < （p’, r’）

条件 2 ： U 条件　　U 軸反転解との比較

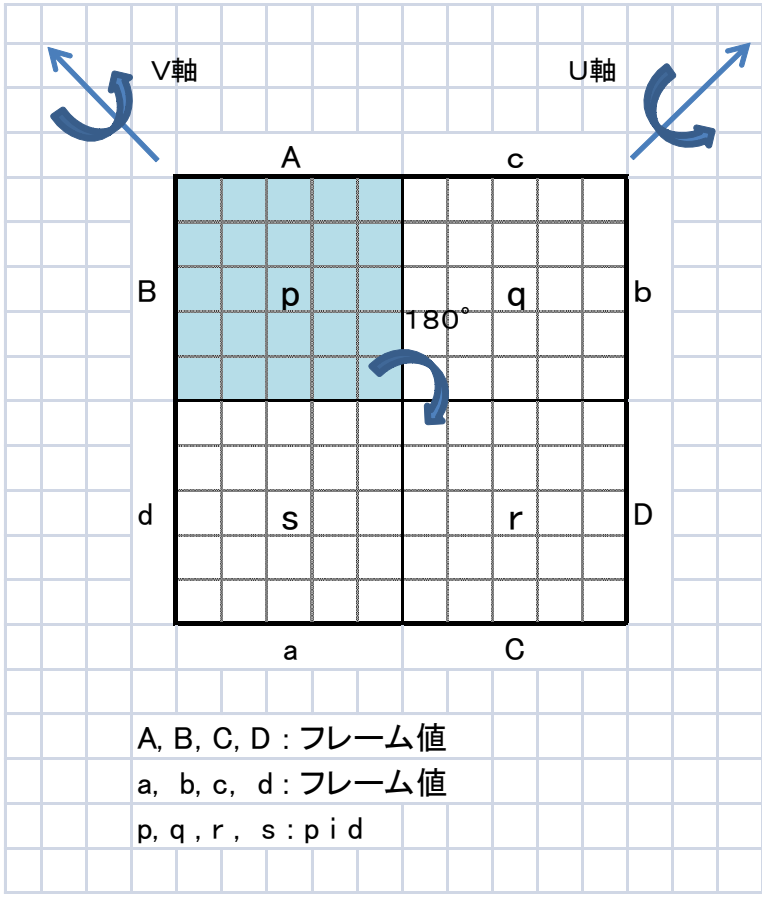
(A, B) < (D, C) or

(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)



以下の条件 4～条件 8 は、#(P) = #(Q)のときのみ

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

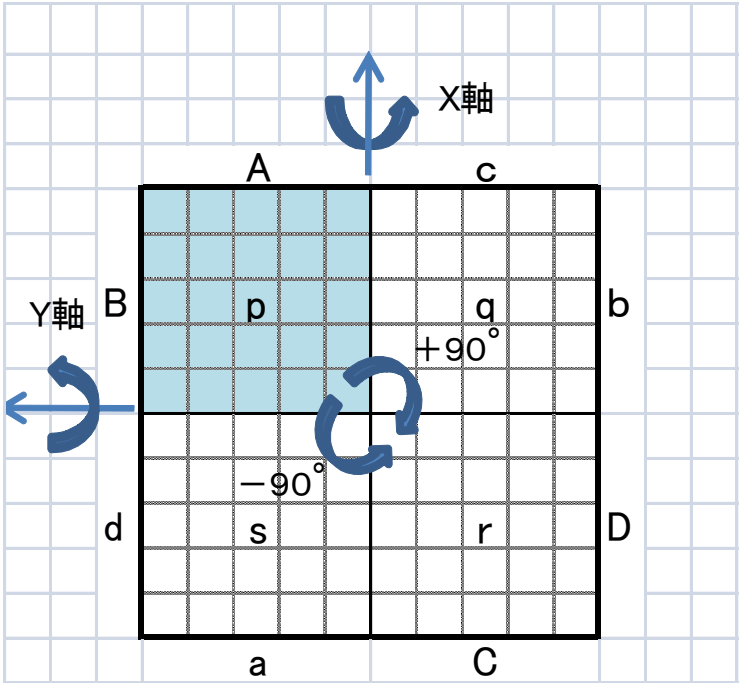
(A, B) ＜ (b, c) or

(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

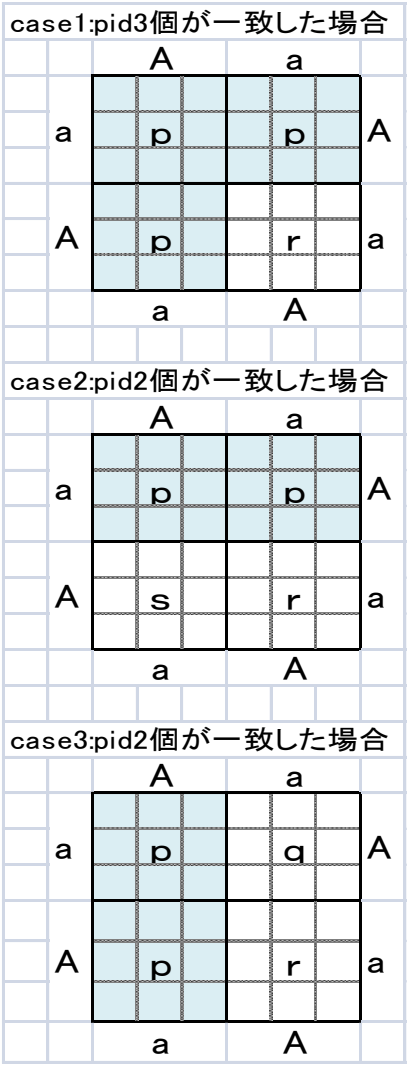
p≠ r　and　r ＜ s

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)



解区分の判定

- 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）

QJH-even 版の実装

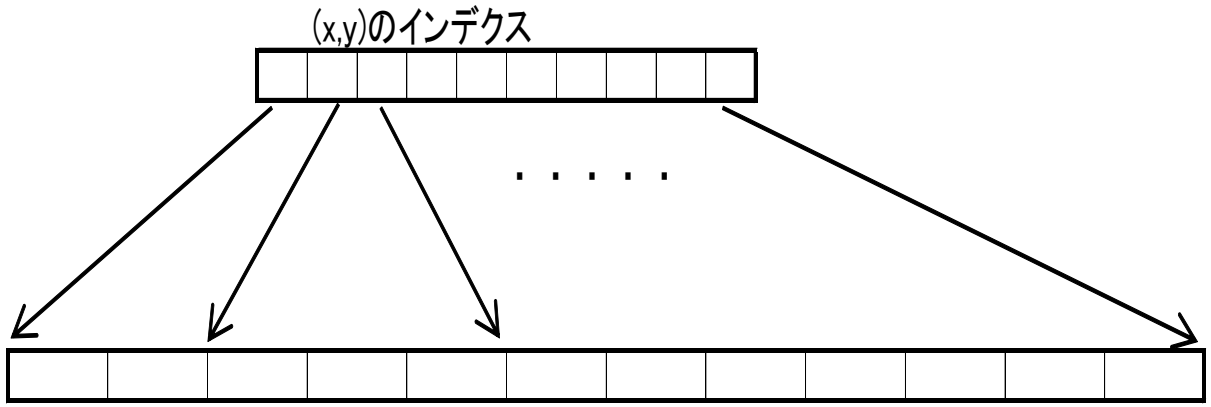
```
for M = (1～n/2)          //Q条件より
                                          //MはサブボードAのQueenの数
  PGen(M)                                 //部分解の作成
  PGen(n-M)                               //部分解の作成
  for frame.A = (1～Bit[n]-1)
    for frame.C = (frame.A～Bit[n]-1)   //R180条件より
      QJoin(A,B)                          // A x B --> AB
      QJoin(C,D)                          // C x D --> CD

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

- 代表解の判定条件は、なるべく早期に適用（絞り込み）
  ほとんどは、フレーム値までの判定で決定
- フレーム A,C による解の分割処理　→　部分解のメモリ常駐化

PGen（1/4 部分解の生成）

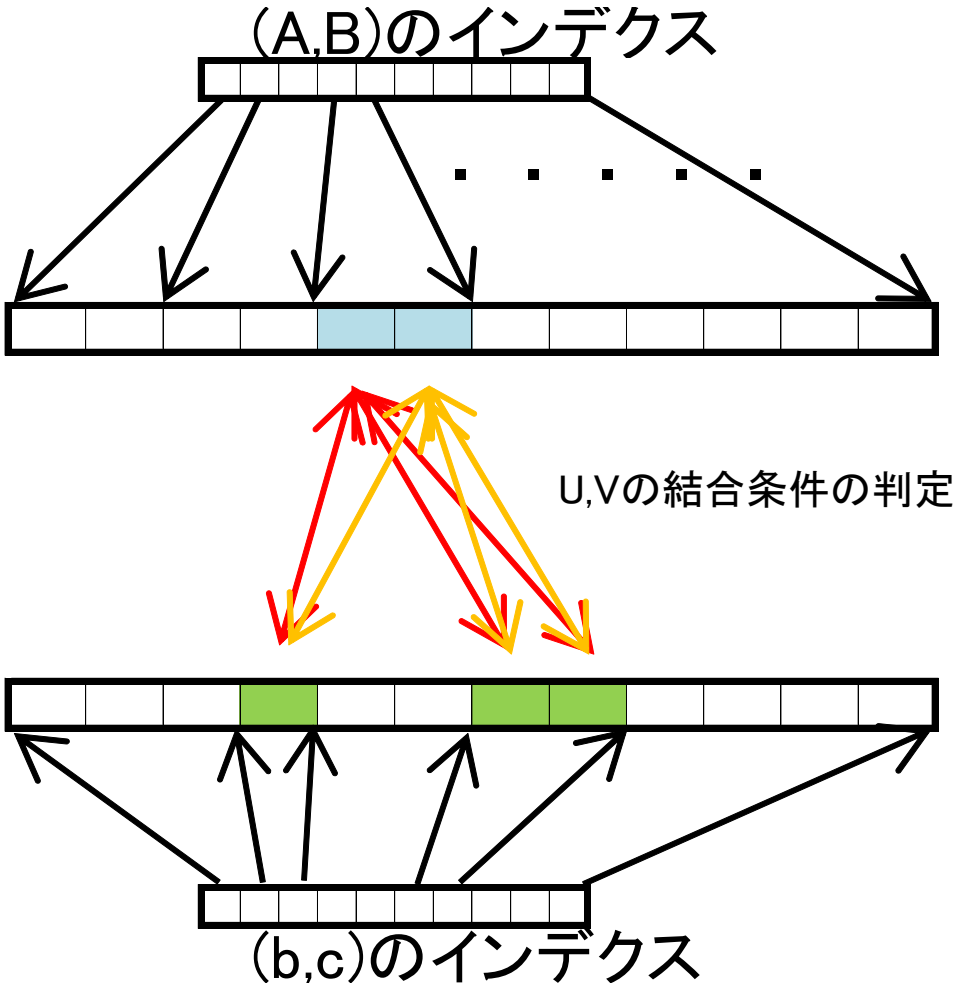
- 基本アルゴリズムは、Jeff Somers 氏のビット演算方式を採用
- 部分解は、特性値（x,y）をキーとするインデクスで管理

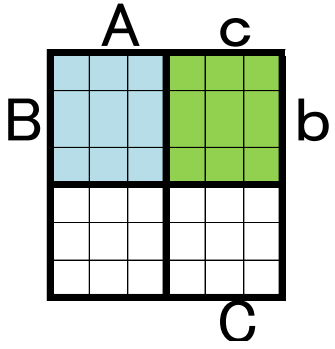


データ(個々の部分解)は、(x、y)の順に並べる

部分解の構成要素：｛ U, V, p, r｝ 12B
p：自身のpid
r：自身のV軸反転解のpid

QJoin





A, C は, フレーム値で固定
B（b） は, 可変要素 A～Bit(n)−1の範囲

（A, B） は、連続エリア

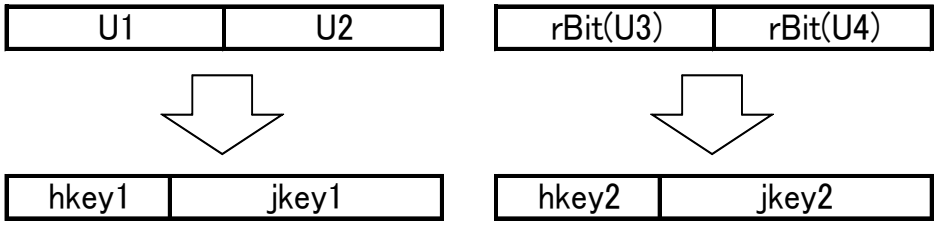
（b,c） は、とびとびのエリア

(b. c)と（c. b）の対称性を利用して、
(c,b) の連続エリアをアクセス

H Join

全要素同士の集合積からの絞り込みは、性能面で不可 → **Hash Join**

Hash Key の生成



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

Hash Join の結合条件

hkey1 & hkey2 = 0 …… 条件1
jkey1 & jkey2 = 0 …… 条件2

① hkey1 より、hkey2 をビット演算（条件1）により算出

hkey1=01011 → hkey2= x0x00 = { 00000,00100,10000,10100 }

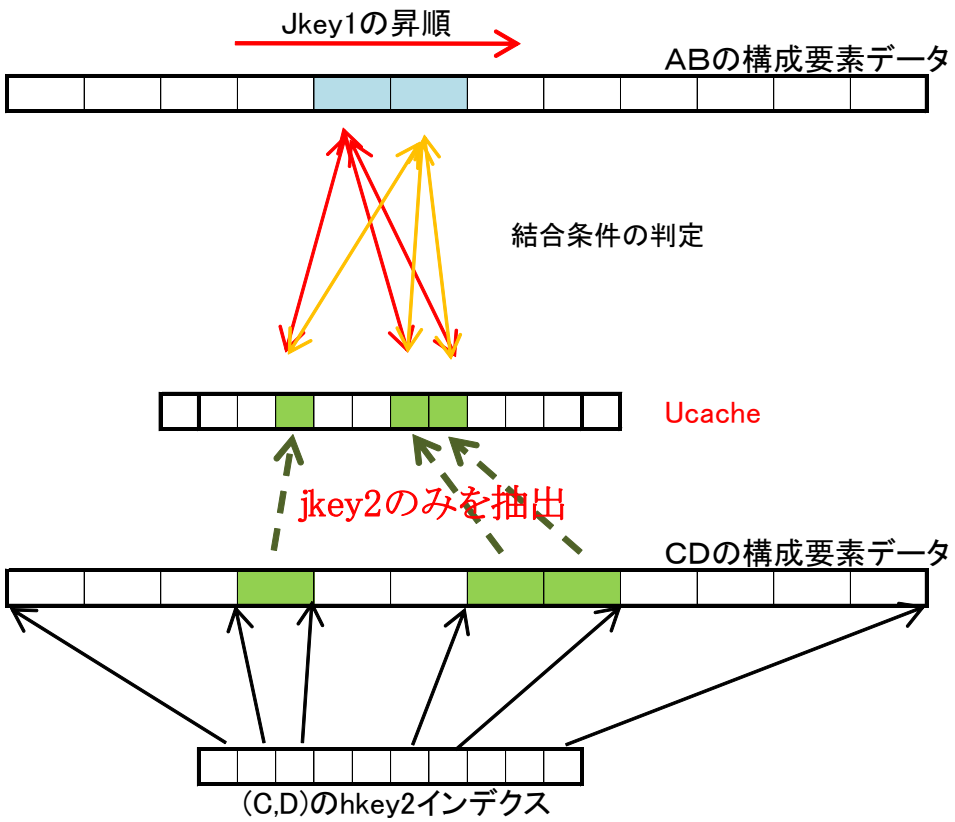
② ハッシュキーのインデクスを検索して、hkey1 → jkey1, hkey2 → jkey2 を得る

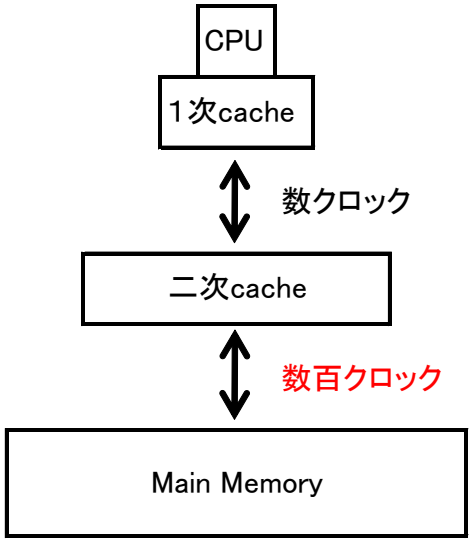
③ jkey1,jkey2の集合積より条件 2 を満たすものを選択

④ 解区分が未定の場合：代表解の判定条件のチェック

ソフト Cache

- メモリアクセスのキャッシュミス削減
- 結合条件の判定に必要な **jkey2** のみを抽出（**22B** → **4B**）

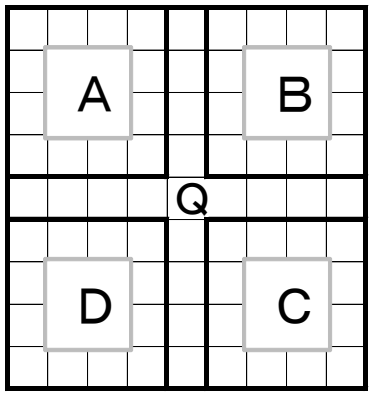




CDの構成要素データ ： 22B
{ X,Y,X2,sym,hkey,jkey,p,q,r }

QJH-odd 版（概略）

type Ⅰ



type Ⅰの留意点：クイーンが中央に１つ配置された形

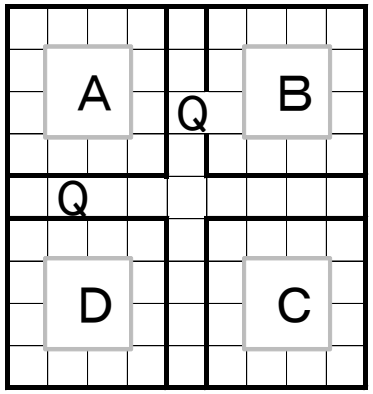
①代表解の判定条件は even 版と同様

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

この場合、even 版とほとんど同じ処理となる

type Ⅱ



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）の位置関係は、

0≦I < J < n とする。また、type Ⅱでは回転対象解は存在しない（type-a ）

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

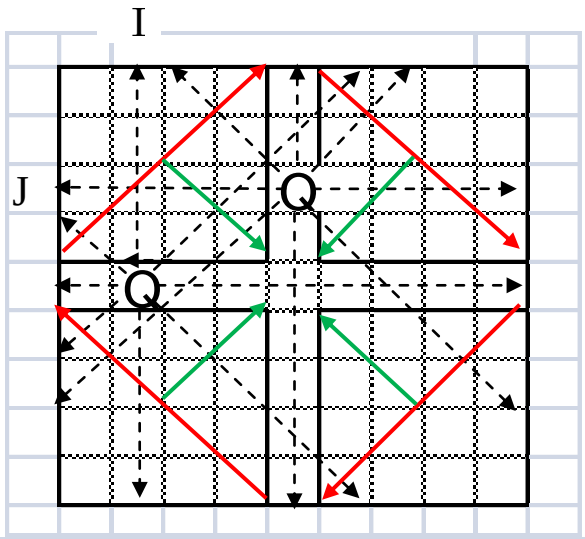
③サブボードのクイーンの数は、以下のようになる

#(A) = m - 1

#(B) = #(D) = n – m　　ただし、1≦m≦n とする

#(C) = m

部分解 nq(n,n)からの抽出条件



U 特性値

V 特性値

		A	B	C	D
typeⅠの場合	U特性値	U(n)=0	U(n)=0	U(n)=0	U(n)=0
typeⅡ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			

実測結果

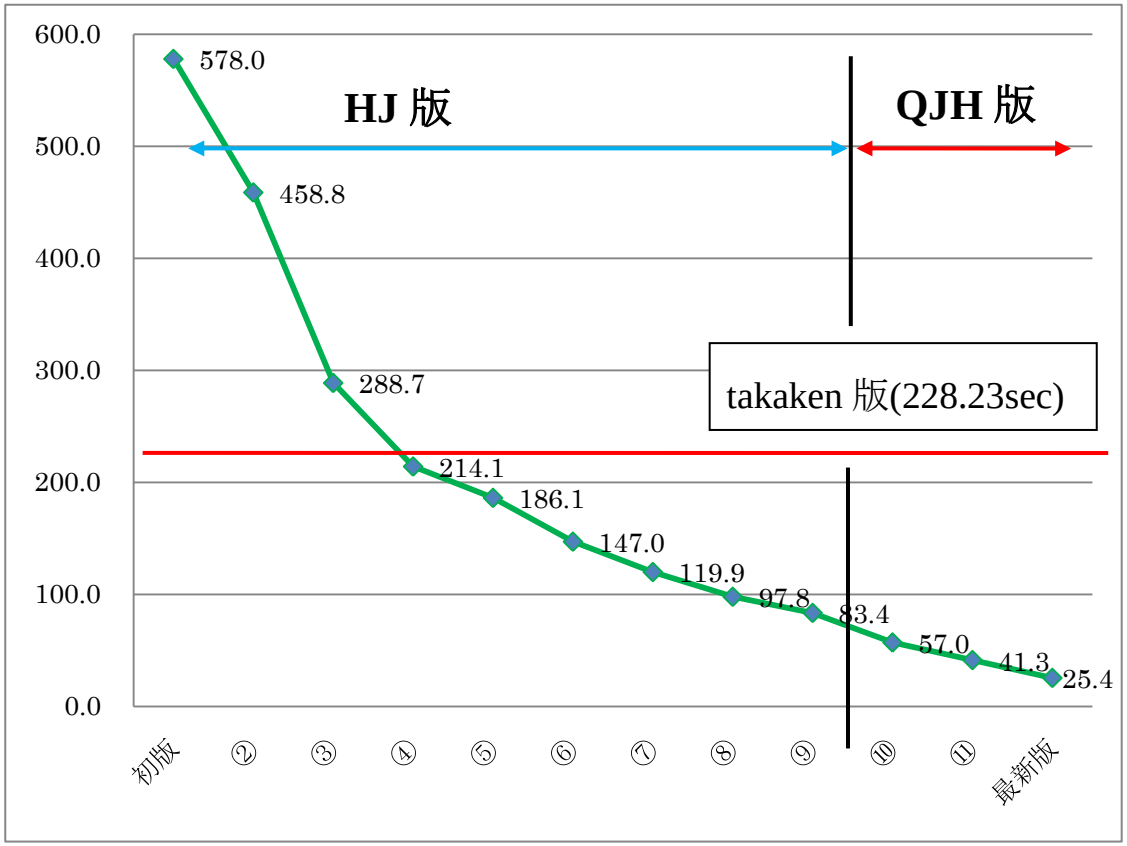
測定環境（ 使用 PC：FMV-BIBLO NF/B50 ）
C　P　U：　Intel Core2 DUO processor T8100　2.1GHz
キャッシュ：　3MB
メ　モ　リ：　4GB
O　　S：　Windows Vista

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	

- takaken 版は、電通大版(qn24b)の約 2 倍
- QJH は、takaken 版の約 10 倍
- QJH の相対比は、徐々に向上

QJH の開発経過(N=18 の探索時間)

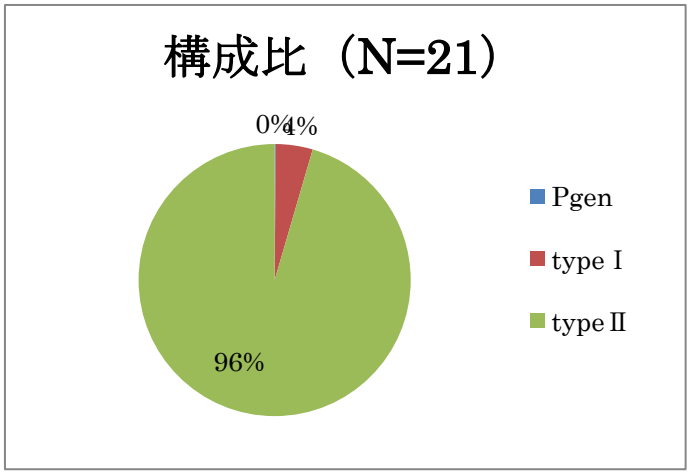
- シンプルな HJ 版で基本構造の確立
- 徐々に性能改善を織り込む



①：初版（HJ 版）
②：PGen の 1 パス化
③：Hash Key のチューニング
④：loop 処理の改善
⑤：U,U2 のソフト cache 化
⑥：Hash　Key の変更
⑦：フレームのビット反転
　　フレーム A≦B の絞り込み
⑧：U,U2 の判定順序入れ替え
⑨：U,U2 を hkey,　jkey 分割
⑩：QJH 版(1/4Parts 生成)
⑪：Qcache 追加
⑫：最新版(SymCheck-pid 化)

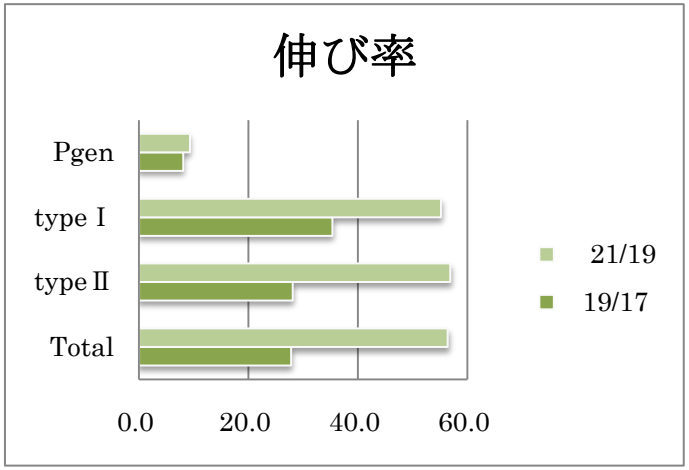
QJH-odd 版の処理時間詳細

	17	19	21
PGen	0.187	1.513	14.087
type Ⅰ	0.250	8.830	486.860
type Ⅱ	6.645	186.498	10,599.576
Total	7.082	196.841	11,100.523



- type Ⅱ の処理時間が殆ど
  (typeI は、全体の約 1/20)

- 伸び率(21/19)では、type Ⅰ ,type Ⅱ とも大差なし
  しかし、type Ⅱ の伸び率が急増。



まとめ

成果

- **既存 Solver の約 10 倍の性能向上**
- 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
- 反面、プログラムが複雑になることが難点

  →　しかし、効果は大。適用例を増やして検証

課題

QJH の並列化（メモリを多用する構造であるため、並列化した場合の効果が未知数)

ソース 公開（準備中)

http://deepgreen.game.coocan.jp

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

ご清聴ありがとうございました。