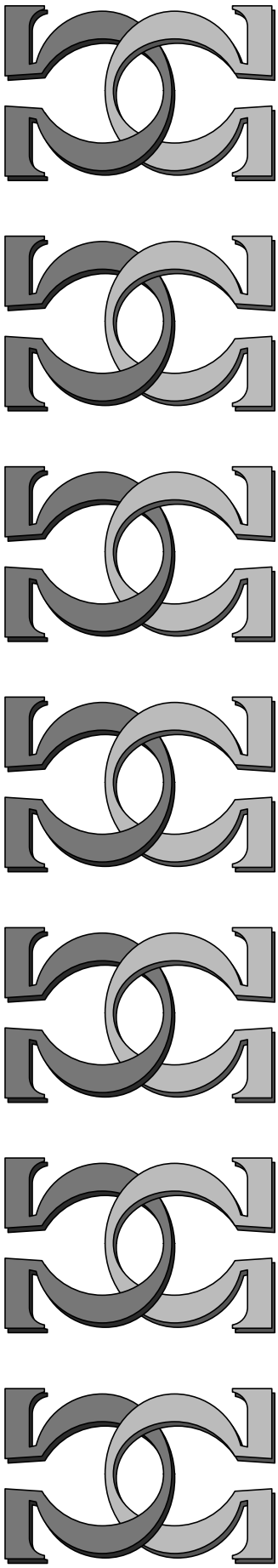


CDMTCS
Research
Report
Series

Computational Methods and New Results for Chessboard Problems

Matthew D. Kearse
Peter B. Gibbons
Department of Computer Science
Univeristy of Auckland, Auckland NZ

CDMTCS-133
May 2000

Centre for Discrete Mathematics and
Theoretical Computer Science

Computational Methods and New Results for Chessboard Problems

Matthew D. Kearse
Peter B. Gibbons
Department of Computer Science
University of Auckland
Private Bag 92019
Auckland
New Zealand
email: p.gibbons@auckland.ac.nz

Keywords: chessboard, queens' graph, kings' graph, domination number, irredundance number, backtracking, local search, reduction.

Abstract

We describe various computing techniques for tackling chessboard domination problems and apply these to the determination of domination and irredundance numbers for queens' and kings' graphs. In particular we show that $\gamma(Q_{15}) = \gamma(Q_{16}) = 9$, confirm that $\gamma(Q_{17}) = \gamma(Q_{18}) = 9$, show that $\gamma(Q_{19}) = 10$, show that $i(Q18) = 10$, improve the bound for $i(Q_{19})$ to $10 \leq i(Q_{19}) \leq 11$, show that $ir(Q_n) = \gamma(Q_n)$ for $1 \leq n \leq 13$, show that $IR(Q_9) = \Gamma(Q_9) = 13$ and that $IR(Q_{10}) = \Gamma(Q_{10}) = 15$, show that $\gamma(Q_{4k+1}) = 2k + 1$ for $16 \leq k \leq 21$, improve the bound for $i(Q_{22})$ to $i(Q_{22}) \leq 12$, and show that $IR(K_8) = 17, IR(K_9) = 25, IR(K_{10}) = 27$, and $IR(K_{11}) = 36$.

1 Introduction

Combinatorial problems on chessboards have been studied by combinatorialists and puzzle-solvers for over 250 years. Investigations centre on the placement of the various types of chess pieces, viz. kings, queens, bishops, rooks, knights and pawns, on generalized $n \times n$ chessboards. For excellent surveys of work and results in this area the reader is referred to [10, 14].

Given a graph $G = (V, E)$, a set of vertices D in V is a *dominating set* if every vertex in $V \setminus D$ is adjacent to at least one vertex in D. The *domination number* of G, denoted $\gamma(G)$, is the minimum cardinality of a dominating set in G. A set of vertices S in V is independent if no two vertices in S are adjacent. The *vertex independence number* of G, denoted $\beta(G)$, is the maximum cardinality of an independent set of vertices in G. The *independent domination*

number of G, denoted $i(G)$, is the minimum cardinality of an independent dominating set in G. The *upper domination number* of G, denoted $\Gamma(G)$, is the maximum cardinality of a minimal dominating set of vertices in G. The *closed neighbourhood* of a set of vertices S, denoted $N(S)$, is the set of vertices, including S, that are adjacent to a vertex in S. A set of vertices S in V is *irredundant* if, for every vertex u in S, the set $N(u) \setminus N(S \setminus \{u\}) \neq \emptyset$. The *irredundance number* of G, denoted $ir(G)$, is the minimum cardinality of a maximal irredundant set of vertices in G. The *upper irredundance number* of G, denoted $IR(G)$, is the maximum cardinality of an irredundant set in G.

These 6 domination, independence and irredundance numbers are related as follows (see [5]):

$$ir(G) \leq \gamma(G) \leq i(G) \leq \beta(G) \leq \Gamma(G) \leq IR(G).$$

Associated with each of the 6 chess pieces one can define a class of graphs. For example the queens' graph Q_n has a set of n^2 vertices corresponding to the squares of an $n \times n$ chessboard with two vertices adjacent if and only if a queen placed on one square can attack the other square in one move. Note that on a chessboard a queen attacks all squares on the same row, column and diagonal. In a similar manner one can define the kings' graph K_n, and graphs for the other chess pieces. Due to the complexities of allowable pawn moves, the grid graph G_n (in which two squares are adjacent if they are adjacent on the same row or column) is normally studied in preference to the pawns' graph.

Determination of the 6 numbers defined above for the 6 classes of graphs gives rise to 36 fundamental problems on chessboard domination. In work to date many of the numbers have been completely determined for all values of n. For example all 6 values have been determined for the rooks' graph. In other cases, values have been determined, generally by computer search, for only small values of n, with bounds established for other values. The development of improved search techniques is important in enabling us to extend the set of known domination, independence and irredundance numbers, thereby providing opportunities for further insight into the complete spectrum of these values.

In this paper we describe probabilistic and exhaustive search techniques for tackling chessboard domination problems, and apply these to the determination of domination and irredundance numbers for the queens' and kings' graphs. In particular we describe previous work on these problems in Section 2. In Section 3 we describe the backtracking method and various refinements that have allowed us to show that $\gamma(Q_{15}) = \gamma(Q_{16}) = 9$, to confirm that $\gamma(Q_{17}) = \gamma(Q_{18}) = 9$, to show that $\gamma(Q_{19}) = 10$, to show that $i(Q18) = 10$, and to improve the bound for $i(Q_{19})$ to $10 \leq i(Q_{19}) \leq 11$. In section 4 we describe the application of reduction techniques chessboard problems, and use it to show that $ir(Q_n) = \gamma(Q_n)$ for $1 \leq n \leq 13$, to show that $IR(K_8) = 17, IR(K_9) = 25, IR(K_{10}) = 27$, and $IR(K_{11}) = 36$, and to show that $IR(Q_9) = \Gamma(Q_9) = 13$ and that $IR(Q_{10}) = \Gamma(Q_{10}) = 15$. Section 5 applies a local search method to generate dominating sets of size $2k + 1$ in Q_{4k+1} and to thereby confirm that $\gamma(Q_{4k+1}) = 2k + 1$ for $1 \leq k \leq 15$, and to extend the result to $k \leq 21$. In addition we establish that $i(Q_{22}) \leq 12$. Some concluding remarks are presented in Section 6. Throughout the paper quoted running times are for algorithms implemented in the language C on an AlphaServer 2100 Model 4/275 running at 275 Mhz under Digital UNIX V4.0D.

2 Background

2.1 Queens Domination Problems

The first eight values of $\gamma(Q_n)$ were given by Ball[1] in 1892. Since then there has been much work on determining values for $\gamma(Q_n)$ for larger n. Many bounds have been discovered for $\gamma(Q_n)$, the most well known being the lower bound due to Spencer (private communication to Cockayne[7]) of $\gamma(Q_n) \geq \frac{n-1}{2}$. Since then, Weakley [22] has obtained the improved bound for n of the form $4k+1$ of $\gamma(Q_{4k+1}) \geq 2k+1$ for all k. In fact it has been suggested but not proven that $\gamma(Q_{4k+1}) = 2k+1$ for all k, and this has been verified for $k \leq 15$. There have also been many upper bounds developed for $\gamma(Q_n)$. Burger, Cockayne and Mynhardt [4] have shown that $\gamma(Q_n) \leq \frac{31}{54}n + O(1)$. Obviously, any lower bound for $\gamma(Q_n)$ is also a lower bound for $i(Q_n)$. However, upper bounds for $\gamma(Q_n)$ do not hold for $i(Q_n)$. The best known upper bound of $i(Q_n) \leq \frac{7}{12}n + O(1)$ was discovered by Eisenstein, Grinstead, Hahne and Van Stone [9]. Recently Weakley [23] has shown that if $n \equiv 1 \pmod 4$ and D is a d-element dominating set of Q_n of a particular, commonly used kind, then $\gamma(Q_k) \leq \frac{d+3}{n+2}k + O(1)$. Also if D is independent, then $i(Q_k) \leq \frac{d+6}{n+2}k + O(1)$.

A list of known values for $\gamma(Q_n)$ and $i(Q_n)$ for small n are given in Table 1.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\gamma(Q_n)$	1	1	1	2	3	3	4	5	5	5	5	6	7	8	≤ 9
$i(Q_n)$	1	1	1	3	3	4	4	5	5	5	5	7	7	8	9

n	16	17	18	19	20	21	22	23	24	25
$\gamma(Q_n)$	≤ 9	9	9	≤ 10	≤ 11	11	≤ 12	≤ 12	≤ 13	13
$i(Q_n)$	9	9	≤ 10	≤ 11	≤ 11	11	≤ 13	≤ 13	≤ 13	13

Table 1: Known values for $\gamma(Q_n)$ and $i(Q_n)$.

2.2 Queens Lower Irredundance

Very little is known about the queens lower irredundance number $ir(Q_n)$. In fact no case is known where $ir(Q_n) < \gamma(Q_n)$ although Fricke et al[10] suspect that the value of $\gamma(Q_n) - ir(Q_n)$ can be arbitrarily large. Kearse [15] has produced the only published bound of $ir(Q_n) \geq \frac{4-\sqrt{7}}{6}n$. Values of $ir(Q_n)$ for small n (see [14]) are: $ir(Q_1) = ir(Q_2) = ir(Q_3) = 1, ir(Q_4) = 2$.

2.3 Queens Upper Irredundance and Domination

The queens upper domination number $\Gamma(Q_n)$ and upper irredundance number $IR(Q_n)$ have had relatively little study until recently. Weakley [22] was one of the first to investigate $\Gamma(Q_n)$ and to discover that $\Gamma(Q_n) > \beta(Q_n)$ was possible. He also proved that $\Gamma(Q_n) \geq 2n-5$. This bound was later improved by Burger, Cockayne and Mynhardt [4] to $\Gamma(Q_n) \geq \frac{5}{2}n - O(1)$. Kearse ([15]) has further improved this bound to $IR(Q_n) \geq 6n - O(n^{\frac{2}{3}})$.

The best upper bound of $IR(Q_n) \leq \lfloor 6n+6-8\sqrt{n+\sqrt{n}+1} \rfloor$ for $n \geq 6$ is due to Burger, Cockayne and Mynhardt [4], and is a slight improvement on the bound originally given by Cockayne of $IR(Q_n) \leq \lfloor 6n+6-8\sqrt{n+3} \rfloor$ for $n \geq 6$. There is no known upper bound for $\Gamma(Q_n)$ other than that of $IR(Q_n)$. A list of known values for $\Gamma(Q_n)$ and $IR(Q_n)$ is given in Table 2.

n	1	2	3	4	5	6	7	8	9	10
$\Gamma(Q_n)$	1	1	2	4	5	7	9	11		
$IR(Q_n)$	1	1	2	4	5	7	9	11		

Table 2: The known upper irredundance and domination numbers for the queens' graph.

2.4 The Kings' Graph

In the kings' graph two squares are adjacent if they lie on the same or adjacent columns as well as the same or adjacent rows. Unlike the queens' graph, many problems involving the kings' graph have been solved, in that optimal solutions to arbitrary sized boards can be constructed. In particular, in 1954 Akiva and Isaak Yaglom [24] proved among other results that $\gamma(K_n) = i(K_n) = \lfloor \frac{n-2}{3} \rfloor^2$, and that $\beta(K_n) = \lfloor \frac{n+1}{2} \rfloor^2$. However, little is known about solutions to the other three problems. It is known than that $ir(K_n) < \gamma(K_n)$ is possible, and that $\beta(K_n) < \Gamma(K_n)$ is possible. Another result of interest is that for $n > 1$, $\frac{(n-1)^2}{3} \leq IR(K_n) \leq \frac{n^2}{3}$ (due to Fricke *(unpublished)*).

A list of known values of irredundance, domination and packing numbers for the kings' graph is given in Table 3.

n	1	2	3	4	5	6	7	8	9	10
$ir(K_n)$	1	1	1	3	4	4	8	9	9	
$\gamma(K_n)$	1	1	1	4	4	4	9	9	9	16
$i(K_n)$	1	1	1	4	4	4	9	9	9	16
$\beta(K_n)$	1	1	4	4	9	9	16	16	25	25
$\Gamma(K_n)$	1	1	4	4	9	9	16			
$IR(K_n)$	1	1	4	4	9	9	16			

Table 3: The known irredundance, domination and packing numbers for the kings' graph.

2.5 Terminology

In referring to the positions on a chessboard we often identify the squares using x and y coordinates which start from 0 and increase across and down the page respectively. In some cases it is desirable to order the squares and to identify them with a single integer. In this case, we begin with the top left square and proceed to order the squares column-wise. Refer to Figure 1 for an example of these numberings on a 4×4 board.

(0,0) 1	(1,0) 5	(2,0) 9	(3,0) 13
(0,1) 2	(1,1) 6	(2,1) 10	(3,1) 14
(0,2) 3	(1,2) 7	(2,2) 11	(3,2) 15
(0,3) 4	(1,3) 8	(2,3) 12	(3,3) 16

Figure 1: The chessboard square numbering convention used in this paper.

There are two main ways of identifying diagonals in the literature. Diagonals that move down and to the right are called *down* or *positive* diagonals, and similarly diagonals moving up and to the right are *up* or *negative* diagonals.

3 Backtracking and refinements

3.1 The Basic Backtracking Algorithm

Backtracking is a well known exhaustive search method that involves systematically generating partial solutions, and trying to extend each to a full solution. If an extension is not possible then the algorithm *backtracks* to consider the next partial solution in order. The technique dates back for at least 100 years (see [16] for example) and has since been successful in its application to a vast array of problems (see [2]).

In [12] Gibbons and Webb described the application of backtracking to the independent queens domination problem, and were able to determine the values of $i(Q_{14})$, $i(Q_{15})$ and $i(Q_{16})$. This algorithm backtracks on the placement of queens on the chessboard and uses the dihedral group of chessboard symmetries to reject partial solutions that have already been considered earlier in the search. This algorithm can be adapted to solve the queens domination problem.

3.2 Backtracking Based on Unattacked Squares

Rather than backtracking on queen placements, an improvement is to systematically identify undominated squares and to try placing pieces in all positions that attack these squares. For example in Figure 2 two queens have already been placed, and the placement of a third is being considered. An undominated square has been located at X and squares adjacent to X have been marked with an o. The third queen will now be tried systematically in all o squares as well as the X square.

At each stage during the execution of the algorithm, we must decide on which of the many undominated squares to focus the search. The naive method of taking the first such square in order performs quite well. However, the breadth of the search tree is more constrained if

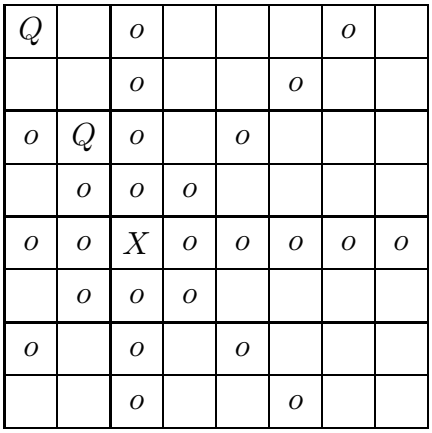


Figure 2: The undominated square X has been selected, and the third queen will be tried in each of the locations marked o, which are adjacent to X.

we take that square with the least number of options available to place the attacking piece. There is a tradeoff involving the cost of finding this square. In practice it is best to the search for the best square early on in the search, but to simply take the first uncovered square in the latter part of the search. The performance of this method is better than that of the first method, but not to the extent that any new results were able to be found. However, it forms the basis for an improved method which is described now.

3.3 Backtracking Based on Intersecting Sets of Squares

Instead of backtracking based on the positions of the queens, this method backtracks based on the set of queens attacking a given position. Consider first the case of placing the first queen. We select the first uncovered square x and assign to this queen the list $N(x)$ of squares that attack this position. We now select the next uncovered square y and investigate separately whether this is to be covered by the first queen or a second (as yet unused) queen. We first investigate the case where the first queen is to cover y. In this case its associated list of squares must be reduced to $N(x) \cap N(y)$. In the case that $N(x) \cap N(y) = \emptyset$ we cannot cover the first and second squares simultaneously using the first queen, and therefore must use the second. Otherwise the first queen can cover both squares, and the procedure is recursively called to investigate coverage of the next uncovered square. In the case that $|N(x) \cap N(y)| = 1$, the first queen is *fixed* and all squares in its neighbourhood are recorded as being covered. Note that squares are not considered covered until they are attacked by a fixed queen.

More generally, at each level of the backtracking search, the following steps must be taken. First, the next uncovered square must be found. Then each unfixed queen is considered in turn with its associated list of positions reduced so that it covers this new square as well as previously assigned ones. If its list becomes empty, then we cease consideration of this queen and move on to the next. Otherwise we recursively call the procedure to investigate coverage of the next uncovered square. Again we record squares attacked by this queen if it becomes fixed. After all pieces have been considered, we then try dominating the latest

square with an as yet unused queen.

This is all demonstrated in more detail in the following algorithm description.

Intersecting Squares Algorithm

Data Structures

$attackCount_i$:	the number of queens attacking square i
$positions_i$:	the set of positions assigned to queen i
k:	the number of queens available to cover the board
$N(X)$:	the neighbourhood of square X

procedure $DominationBacktrack(p, attackCount_{1..number\_of\_squares}, positions_{1..number\_of\_pieces})$
  while $attackCount_p > 0$ **do**
    {Find the next undominated square}
    $p = p + 1$
    if $p > number\_of\_squares$ **then**
      { Any combination of the elements of }
      { $positions_{1..number\_of\_pieces}$ is a solution }
      return
  for $i = 1$ **to** k **do**
    if $|positions_i| > 1$ **then**
      { For each piece already dominating at least one square }
      $newPositions = positions_i \cap N(p)$
      if $|newPositions| > 0$ **then**
        if $|newPositions| = 1$ **then**
          for each $x \in N(newPositions)$ **do**
            $attackCount_x = attackCount_x + 1$
        $DominationBacktrack(p + 1, attackCount,$
          $(positions_1..i - 1, newpositions, positions_{i+1..number\_of\_pieces}))$
        if $|newPositions| = 1$ **then**
          for each $x \in N(newPositions)$ **do**
            $attackCount_x = attackCount_x - 1$
  $i = 1$
  while $|positions_i| > 0$ **do** { Find the first unused piece }
    $i = i + 1$
  if $i \leq number\_of\_pieces$
    $newPositions = N(p)$
    if $|newPositions| = 1$ **then**
      for each $x \in N(newPositions)$ **do**
        $attackCount_x = attackCount_x + 1$
    $DominationBacktrack(p + 1, attackCount,$
      $positions_1..i - 1, newpositions, positions_{i+1..number\_of\_pieces})$
    if $|newPositions| = 1$ **then**
      for each $x \in N(newPositions)$ **do**

$attackCount_x = attackCount_x - 1$

endProcedure

$DominationBacktrack(1, (0, 0, ..., 0), (\{\}, \{\}, ..., \{\}))$

Applying this algorithm yielded some already known solutions, but with significantly longer times than those of either of the two previous algorithms when solving the queens domination problem. However, applying the method only for the placement of the last piece always gave a significant improvent in the execution time. We now discuss a number of enhancements that can be applied to any of the three algorithms and finally a combined method that produces significant speed improvements.

3.4 Enhancements

Two simple methods were developed for quickly deciding whether to extend from the current partial solution. The first check is for a queen that covers no squares not covered by another queen. If such a queen exists, and we are searching for only a solution with a minimal number of queens, then this partial solution will obviously not lead to it, and the algorithm can backtrack.

For example, in Figure 3 the fourth placed queen, Q_4, has created a configuration where the first queen(Q_1) no longer attacks a unique square and is therefore redundant. The algorithm can immediately consider the next placement of the fourth queen.

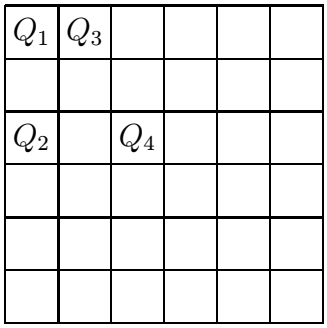


Figure 3: The placement of Q_4 causes Q_1 to no longer be necessary.

The next check is whether there are enough remaining queens to cover the remaining uncovered squares. For an $n \times n$ chessboard a queen can attack at most $4n - 3$ different squares including the one it is placed on. A second queen can attack fewer new squares than this as many of the squares attacked by it are already attacked by the first queen. By using the backtracking methods already presented, we can build a table with the maximum number of new squares that can be attacked for any given number of queens for a fixed size board. If the number of remaining squares to be covered is greater than the maximum than can be covered by the remaining number of queens, then we can backtrack.

Note that actually building up a list of the maximum number of squares that can be dominated for any number of pieces up to the domination number would require as much

work as solving the problem itself. Therefore in practice the number of squares dominated by up to half this number is initially computed, with this information being used for placing the second half of the pieces.

3.5 Combining the Methods

The best method was obtained by applying the enhancements of the previous section to a combination of the three backtracking methods. In particular, optimal performance was achieved by placing the first queen using the first algorithm, placing all but the last 2 or 3 queens using the second algorithm, and placing the remaining 2 or 3 queens using the final method. The optimal number of queens to place using the last method increases with the overall number of queens being placed.

3.6 Isomorph Rejection

Isomorph rejection cannot be easily applied to the unattacked squares method, so once the algorithm switches to this method, isomorph rejection is no longer applied. However, a count of the number of non- isomorphic solutions can still be found by rejecting complete solutions found that are not the first in their isomorphism class.

3.7 Results Obtained

The results of the combined methods are listed in Table 4 (with previously unknown results in **bold**). In particular we showed that $\gamma(Q_{15}) = \gamma(Q_{16}) = 9$ by finding no dominating sets with 8 queens. Additionally, we confirmed the already known results of $\gamma(Q_{17}) = \gamma(Q_{18}) = 9$.

The algorithm can be easily modified to handle the case of the independent queens domination problem by adding the restriction that a queen cannot be placed in a location that attacks another queen. This restriction greatly speeds up the algorithm, and enabled us to show that $i(Q_{18}) = 10$. Also by finding no independent dominating set of 9 queens on a 19 by 19 board we improved previously known bound of $9 \leq i(Q_{19}) \leq 11$ to $10 \leq i(Q_{19}) \leq 11$.

This last result leads to one further result for the queens domination problem. Weakley [22] has proved a theorem which includes the result that if R is a dominating set of $(n-1)/2$ squares of Q_n, then R is independent. It follows from this that if $i(Q_n) > \frac{n-1}{2}$ then $\gamma(Q_n) > \frac{n-1}{2}$. The result that $i(Q_{19}) > 9$ implies that $\gamma(Q_{19}) > 9$. Since it is already known that $\gamma(Q_{19}) \leq 10$ it follows that $\gamma(Q_{19}) = 10$.

4 Reductions

In this section we present a reduction method which is suitable for solving a variety of constraint satisfaction problems. The method is particularly well suited to the kings upper irredundance problem. It is also applied to the queens upper and lower irredundance problems to produce some new results.

Suppose we have a partial configuration and are considering all possible extensions to see whether any of them form a complete solution. In addition suppose that for any such

Queens Domination

Board size	Number of Queens	Number of Non-Isomorphic Solutions	Time
3	1	1	-
4	2	3	-
5	3	37	-
6	3	1	-
7	4	13	-
8	5	638	1.1 secs
9	5	21	0.4 secs
10	5	1	0.7 secs
11	5	1	1.2 secs
12	6	1	34 secs
13	7	41	18 mins
14	**8**	**588**	11 hours
15	**8**	**0**	21 hours
15	**9**	**25872**	230 hours
16	**8**	**0**	31 hours
17	8	0	30 hours
18	8	0	43 hours

Independent Queens Domination

Board size	Number of Queens	Number of Non-Isomorphic Solutions	Time
3	1	1	-
4	3	2	-
5	3	2	-
6	4	17	-
7	4	1	-
8	5	91	-
9	5	1	-
10	5	1	-
11	5	1	0.2 secs
12	7	105	34 secs
13	7	4	45 secs
14	8	55	9.4 mins
15	9	1314	110 mins
16	**9**	**16**	5 hours
17	**9**	**2**	11 hours
18	**9**	**0**	28 hours
18	**10**	**28**	120 hours
19	**9**	**0**	62 hours

Table 4: Running times for the enhanced backtracking algorithm solving the queens and independent queens domination problems.

extension that forms a complete solution, one of the pieces in the partial solution can be moved to another location and the solution preserved. Then there is no point in continuing the search from this partial configuration if the other partial configuration found by moving the piece has already been or will be searched.

Specifically, assume a partial solution $(x_1, x_2, ..., .x_k)$ to a problem with solution space $(X_1, X_2, ..., X_n)$ with $k < n$ is being considered. If (x_1,x_2,...,x_k, x_{k+1},..,x_n) being a solution implies that $(y_1, y_2, ..., y_k, x_{k+1}, ..., x_n)$ is also a solution, then $(x_1, x_2, ..., .x_k)$ need not be extended if all extensions to $(y_1, y_2, ..., y_k)$ have been or will be searched. If such a condition holds, we say that $(x_1, x_2, ..., .x_k)$ is *reduces* to $(y_1, y_2, ..., y_k)$. This is a particularly powerful technique if a large proportion of partial solutions are reducible, as is the case with the kings upper irredundance problem.

4.1 Examples

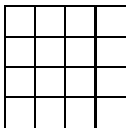
We demonstrate the technique with a simple example for the kings upper irredundance problem. This problem involves producing a configuration of as many kings as possible such that every king has at least one private neighbour. A private neighbour is a square attacked by a unique king, and can include the square the king is on itself.

Figure 4: The king at position (1,1) could move to position (0,0).

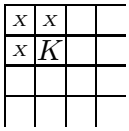
Consider the example in Figure 4. Here a king is being considered for placement near the top-left corner. Three other squares are marked with an X because if a king were placed in any of these three squares then it would have no private neighbour. Because of these restrictions, no matter how the solution is extended, the king already placed is guaranteed to have a private neighbour in the top-left corner. Now observe that if there were a solution involving a king in this position, then it could be moved to the top-left square and this would still be a solution. That is, the partial solution with a king in position (1,1) reduces to the partial solution with a king in position (0,0). In fact it is more likely that a solution would be found if the king were position (0,0) since it would attack fewer additional squares from that position, thereby leaving those squares to be private neighbours of other kings. A consequence of this is that there is really no need to ever search for a maximal irredundant configuration of kings with one in the original position, so before beginning such a search, this square can be immediately ruled out from containing a king.

We can express this sequence of logical conclusions in the following way.

```
The board is initially:
```

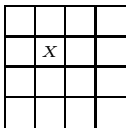


```
If a king were at position (1,1)
       then the following positions would be prohibited:
         (0,0), (0,1), (1,0)
       and the board becomes:
```




```
       and the king at (1,1) could be moved to (0,0).
```

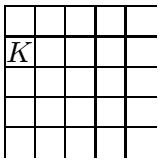
So a king should not be at position (1,1)
and the board becomes:



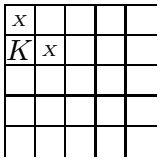
This demonstrates that starting from a board about which no decisions have so far been made we can conclude that there should be no piece in position (1,1). Note that whenever a square is listed as being prohibited, it is done either because a king placed in it would have no private neighbours, or because a king in that square would cause some already placed king to have no private neighbours.

Consider the logic of another example.

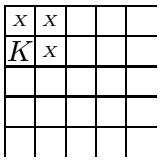
The board is initially:



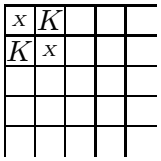
The following positions would be prohibited:
(0,0), (1,1)
and the board becomes:



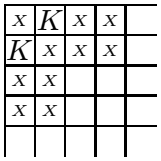
If position (1,0) were prohibited
then the board becomes:



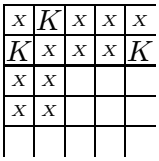
and the king at (0,1) could be moved to (0,0).
So position (1,0) should contain a king
and the board becomes:



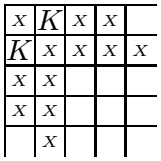
The following positions are prohibited because either of the already placed kings would have no private neighbour:
(2,0), (2,1), (3,0), (3,1), (0,2), (0,3), (1,2), (1,3)
and the board becomes:



If a king were at position (4,1)
then the following position would be prohibited
(4,0)
And the board becomes:

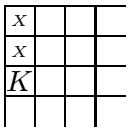


and the king at (4,1) could be moved to (4,0)
So there can be no king at position (4,1).
By symmetry there can be no king at position (1,4)
and the board becomes:

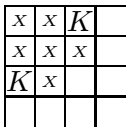


Thus by placing a king at position (0,1), we can conclude that there must also be a king at (1,0) and that 12 other squares cannot contain kings.

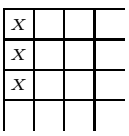
In a similar way, it can be shown that a starting configuration of



must be extended to



As a final example we can show, using similar logic, that there is no point in extending the following partial solution:



¿From the above four examples we can conclude that in a search for a maximal irredundant configuration of kings, we only need to consider the following three possibilities in the top left corner:

- There is a king in position (0,0)
- There is a pair of kings at positions (0,1) and (1,0)
- There is a pair of kings at positions (0,2) and (2,0)

Of course a similar situation arises in all four corners of the board if it is large enough, and such information would greatly speed up any backtracking search for a solution to the problem.

4.2 The Reduction Algorithm

The logic described by way of examples in the previous section can be applied systematically during the search. The procedures applying the checks use the following data structures:

- The number of private neighbours of each piece denoted by $privateNs_{1..number\_of\_pieces}$.
- The current usage of each square is recorded in the array $squareUsage_{1..number\_of\_squares}$. Each square can be unused (UN_USED), occupied ($USED$) or prohibited ($PROHIBITED$). At certain points during the algorithm, the variable $oldSquareUsage$ is used. This is a local variable to each procedure, and is used to backup the current usages for the squares so that it can be restored at a later point.
- The current status of each square is recorded in the array $squareStatus_{1..number\_of\_squares}$. In addition to knowing the number of pieces attacking each square, it is useful to know if this can possibly change deeper within the search. For example a square could be unattacked ($UN\_ATTACKED$) or if all adjacent squares are prohibited then it is guaranteed to be unattacked ($G\_UN\_ATTACKED$). Similarly a square can be a private neighbour of some piece ($P\_NEIGHBOUR$) or a guaranteed private neighbour ($G\_P\_NEIGHBOUR$). If the square is not any of these states, then it is attacked by at least 2 pieces ($ATTACKED$).

Pieces are placed using the procedure $PutPiece(pieceNum, p, ok)$ which places piece $pieceNum$ at position p only if it and all existing pieces still have at least one private neighbour. The flag ok is set to *true* if the piece was placed. Procedure $RemovePiece(pieceNum)$ removes the piece and restores the data structures.

We now consider the conditions that must hold for a piece to allowed to move from one square to an other. Firstly, it must either still attack a guaranteed private neighbour from the new position, or alternatively still attack all the original private neighbours. Secondly, it cannot attack any additional squares that could be private neighbours of some other piece either now or deeper in the search, unless the other piece is guaranteed to have a private neighbour elsewhere. These additional squares can be identified as those for which the status is $UN\_ATTACKED$, $P\_NEIGHBOUR$ or $G\_P\_NEIGHBOUR$, and are referred to as *disallowed squares*. We need to be sure that, if we have ruled out an extension based on moving a piece to another location, then a later solution with the piece at this new location will not be rejected because it could move back to the old position. This is ensured by

checking that the number of disallowed squares which are attacked is smaller in the position moved to. If this is true at some point within the search, then deeper within the search it will not be possible to move a piece back because no squares ever change state from being allowed to disallowed further down the backtracking tree.

The procedure $DisallowedSquares(p)$ returns a list of squares adjacent to p that may not be attacked. The procedure $ComputeSquareStatus()$ computes the status of each square x and stores the result in $squareStatus_x$

The algorithm that analyses the current state of the board to determine if a move for any piece is possible is given below. Note that it also checks for cases where a new piece could be added to the board at no cost to the current partial configuration. Such cases arise when there exists a prohibited square for which all adjacent squares are either attacked by at least two pieces or are guaranteed to be unattacked so that it will not interfere with any existing or future pieces. Also at least one of these squares should be guaranteed to be unattacked so that the new piece has at least one private neighbour itself.

Procedure *Move()*
{ Returns true if any piece could move or be created }
{ at no cost to the current partial configuration. }
ComputeSquareStatus()
for all squares x **do** {Check if a new piece could be created at x }
if $squareUsage_x = PROHIBITED$ **and** $|DisallowedSquares(x)| = 0$ **then**
for all $y \in N(x)$ **do**
if $squareStatus_y = G\_UN\_ATTACKED$ **then**
return *true* { A new piece could be created at x }
for all p **such that** $position_p \neq NULL$ **do** { Check if piece p can be moved }
$pos = position_p$
$PN = \{x : x \in N(pos), |attackList_x| = 1$
{ PN = the set of private neighbours of piece p }
$GPN = \{x : x \in N(pos), squareStatus_x = G\_P\_NEIGHBOUR\}$
{ GPN = the set of guaranteed private neighbours of piece p }
$RemovePiece(p)$ { Temporarily remove the piece }
$ComputeSquareStatus()$
$oldList = DisallowedSquares(pos)$
for all $x \in N(N(pos))$ { For all possible moves for piece p }
{ $N(N(pos))$ is used because it contains the set of possible squares }
{ still attacking neighbours of p. }
$newList = DisallowedSquares(x)$
if $newList \subset oldList$ **then**
{ No new squares that are not allowed are attacked and at least }
{ one of these is no longer attacked }
if $(PN \setminus N(x)) = \emptyset$ **or** $|GPN \cap N(x)| > 0$ **then**
{ Either all private neighbours or at least one guaranteed }
{ private neighbour is still attacked }
$PutPiece(p, pos, ok)$ { Put the piece back in its original place }
return *true* { Piece p could move from pos to x }

```
        PutPiece(p, pos, ok) { Put the piece back in its original place }
    return false
endProcedure
```

In order to be able to make the logical deductions presented earlier, a form of forward checking is required. A simple case involves checking to see if piece k can be placed in each possible position, and if not prohibiting that square. This task is carried out by the procedure *BasicForwardCheck(k)*. The following function *ForwardCheck(k, startSquare)* returns *true* if there is no point in extending the current partial solution. Otherwise it returns *false* and prohibits any square that would result in a move being possible if it contained a piece. Additionally, if it detects that a piece must be placed in a certain square then it calls the recursive backtrack procedure to search the remaining solution space and then returns *true*.

```
Procedure ForwardCheck(k, startSquare)
    BasicForwardCheck(k)
    if Move() then
        return true
    ok = true
    while ok do { As long as something has changed, keep repeating }
        ok = false
        for all squares x do
            if squareUsage_x = UN_USED then
                oldSquareUsage = squareUsage
                canMove = false
                if PutPiece(k, x) then { Check if a piece could be at x }
                    BasicForwardCheck(k + 1)
                    if Move() then
                        canMove = true
                    RemovePiece(k)
                    squareUsage = oldSquareUsage
                if canMove = true then
                    squareUsage_x = PROHIBITED { There must not be a piece at x }
                    ok = true
                else
                    squareUsage_x = PROHIBITED
                    { Check if this square could be prohibited }
                    if Move() then { There must be a piece in square x }
                        PutPiece(k, x)
                        Backtrack(k + 1, startSquare)
                        RemovePiece(k)
                        return true { This case has been fully searched now }
                    squareUsage_x = UN_USED
    return false
endProcedure
```

Finally the fundamental backtracking algorithm is:

Procedure $Backtrack(k, startSquare)$
    if $k > number\_of\_pieces$ **then**
        $position_{1..number\_of\_pieces}$ is a solution
        return
    $oldSquareUsage = squareUsage$
    $abort = ForwardCheck(k, startSquare)$
    $i = startSquare$
    while $i \leq number\_of\_squares$ **and** $abort = false$ **do**
        if PutPiece(k, i) **then**
            $Backtrack(k + 1, i + 1)$
            $RemovePiece(k)$
            $squareUsage_i = PROHIBITED$
            $abort = ForwardCheck(k, startSquare)$
            $i = i + 1$
    squareUsage = oldSquareUsage
endProcedure

$squareUsage_{1..number\_of\_squares} = UN\_USED$
$attackList_{1..number\_of\_squares} = \emptyset$
$position_{1..number\_of\_pieces} = NULL$
$Backtrack(0, 0)$

4.3 Isomorph Rejection

As with the algorithm for the queens domination, isomorph rejection is easily incorporated into this algorithm. It is not included in the given algorithm descriptions to keep them as simple as possible, but the same method used before is employed. That is, at any stage where a square x is prohibited, if the current board configuration maps to itself under some transformation T, then the square that x maps to under T is also prohibited.

4.4 Results

The described algorithm obtained the following new results for the kings upper irredundance problem: $\mathrm{IR}(K_8) = 17$, $\mathrm{IR}(K_9) = 25$, $\mathrm{IR}(K_{10}) = 27$, and $\mathrm{IR}(K_{11}) = 36$. Maximal irredundant configurations of the given sizes were already known to exist, but this program determined that no larger cases exist. Some performance figures for the algorithm are given in Table 5.

4.5 Queens Lower Irredundance

This problem involves finding a minimal placement of queens on a chessboard so that every queen is irredundant, and so that no more irredundant queens can be added to the board while maintaining the property that all queens are irredundant. Currently there is no known

Board size	Number of Kings	Solution Found	Time
6	9	Yes	0.2 secs
6	10	No	0.3 secs
7	16	Yes	1.6 secs
7	17	No	2.1 secs
8	17	Yes	60 secs
8	**18**	**No**	59 secs
9	25	Yes	25 mins
9	**26**	**No**	9 mins
10	27	Yes	8.4 hours
10	**28**	**No**	9.2 hours
11	**37**	**No**	140 hours

Table 5: Running times for backtracking algorithm solving the kings upper irredundance problem.

case for which $\mathrm{ir}(Q_n) < \gamma(Q_n)$, and to date only the first four values for $ir(Q_n)$ have been published. To find if there does exist a configuration in which $ir(Q_n) < \gamma(Q_n)$ we must search an $n \times n$ board for maximal irredundant sets of sizes between 1 and $\gamma(Q_n) - 1$.

The algorithm was run with a number of queens ranging from 1 up to 1 less than the domination number for boards up to size 13×13. In all cases no solution was found, thereby indicating that $ir(Q_n) = \gamma(Q_n)$ for $1 \leq n \leq 13$. The running times are given in Table 6.

Board size	Number of Queens	Solution Found	Time
5	2	**No**	-
6	2	**No**	-
7	3	**No**	-
8	4	**No**	1 sec
9	4	**No**	3 secs
10	4	**No**	9 secs
11	4	**No**	21 secs
12	5	**No**	22 mins
13	6	**No**	26 hours

Table 6: Running times for backtracking algorithm solving the queens lower irredundance problem.

4.6 Queens Upper Irredundance

The reduction techniques developed for the kings' graph were also applied to the queens' graph. However the algorithm is far less suited to the queens graph with the results being not nearly as good. In fact the reduction part of the backtracking algorithm was found to *slow down* the algorithm, with the best results being found with the reduction part removed. The algorithm does run fast enough to find the previously unknown values of $IR(Q_9) = 13$ and $IR(Q_{10}) = 15$. These are in fact equal to Weakley's lower bound of $\Gamma(Q_n) \geq 2n - 5$. Since $IR(Q_n) \geq \Gamma(Q_n)$ this additionally implies that $\Gamma(Q_9) = 13$ and $\Gamma(Q_{10}) = 15$. The running times for these and a number of other cases are given in Table 7.

Board size	Number of Queens	Solution Found	Time
6	7	Yes	0.4 secs
6	8	No	0.2 secs
7	9	Yes	7.3 secs
7	10	No	2.9 secs
8	11	Yes	1 sec
8	12	No	84 secs
9	13	Yes	2 hours
9	**14**	**No**	1 hour
10	15	Yes	40 hours
10	**16**	**No**	46 hours

Table 7: Running times for backtracking algorithm solving the queens upper irredundance problem.

5 Probabilistic Methods

Probabilistic methods have the advantage of being able to generate chessboard configurations of much greater sizes than can be generated by exhaustive search. On the other hand they cannot establish nonexistence of a configuration with the required properties, or to enumerate all such configurations. In this section we describe the use of a simple local search technique which was used to establish the new result that $i(Q_{22}) \leq 12$ by finding an independent dominating configuration of 12 queens on a 22 by 22 board. Previously it has been proven that $\gamma(Q_{4k+1}) = 2k + 1$ for all $k \leq 15$ by exhibiting dominating sets of this size. This is extended here to prove that the result holds for $k \leq 21$.

5.1 Local search

Local search techniques operate within a set Σ of *feasible solutions*. Associated with each $i \in \Sigma$ is a cost $c(i)$, and the task is to find an optimal solution with overall minimum cost. In the problems of this section an optimal solution has cost 0. For each $i \in \Sigma$ we define a set T_i of transformations each of which can be used to change i into a closely related feasible solution j. The set of solutions that can be reached from i by applying a transformation from T_i is called the neighbourhood $D(i)$ of i.

The simplest local search is simply a random walk, starting with an arbitary starting feasible solution and successively moving to a neighbouring feasible solution. The chosen neigbouring solution is selected from a small set of neighbours, and is the one with the lowest cost. The walk finishes successfully when an optimal solution is found, or unsuccessfully when an upper limit of transformations is reached.

The algorithm is as follows:

Procedure $RandomSearch(x, searchSize, stopThreshold)$
{ x a random starting configuration. }
{ $searchSize$ is the number of neighbours generated from the current solution. }
  repeat $stopThreshold$ **times**
    $bestCost = \infty$
    repeat $searchSize$ **times**
      $y = transform(x)$ { Generate a random element in $D(x)$ }
      if $Cost(y) = 0$ **then**

return y { y is a solution }
if $Cost(y) < bestCost$ **then**
$bestCost = Cost(y)$
$bestSolution = y$
$x = bestSolution$
endProcedure

This algorithm was adapted to the queens domination problem using the following *Transform* procedure:

Procedure $Transform(x)$
{ Returns a random transformation from x to a new configuration. }
{ x is a set of ordered pairs (i, j) which indicates that queen i is placed in square j }
Select a random pair $(i, j) \in x$
$y = x \setminus \{(i, j)\}$
Select a random undominated square a from y
Select a random square $b \in D(a)$
return $y \cup \{(i, b)\}$
endProcedure

Despite its simplicity, this method, using *searchSize = 20*, quickly found solutions of size $\gamma(Q_n)$ for all $n \leq 18$. We noted that the majority of solutions found in this way had queens only on even-even squares (i.e. squares (i, j) where $i, j \equiv 0 \pmod 2$). Modifying the previous algorithm to search only for solutions with queens on even-even squares we were able to find optimal solutions for boards from sizes 12 to 25, including Johannes Waldmann's dominating configuration ([20]) of 12 queens on a 23×23 board (see Figure 5). The average running times of this algorithm over a number of trials are given in Table 8.

Queens Domination

Board size	Number of Queens	Search Size	Average Time
12	6	10	0.01 secs
13	7	10	0.07 secs
14	8	11	0.05 secs
15	9	11	0.03 secs
16	9	12	0.16 secs
17	9	13	1.1 secs
18	9	14	9.4 secs
19	10	17	12 secs
20	11	19	2.7 secs
21	11	20	35 secs
22	12	20	22 secs
23	12	20	28 mins
24	13	20	99 secs
25	13	20	27 mins

Table 8: Running times for the probabilistic algorithm solving the queens domination problem.

A similar transformation was used for the queens independent domination problem, but since most solutions to this problem do not have queens on only even-even squares, this restriction was dropped. This algorithm generated the new result of an independent dominating configuration of 12 queens on a 22×22 board (see Figure 5). The algorithm was tested on boards up to 22 by 22, and the results of this are given in Table 9.

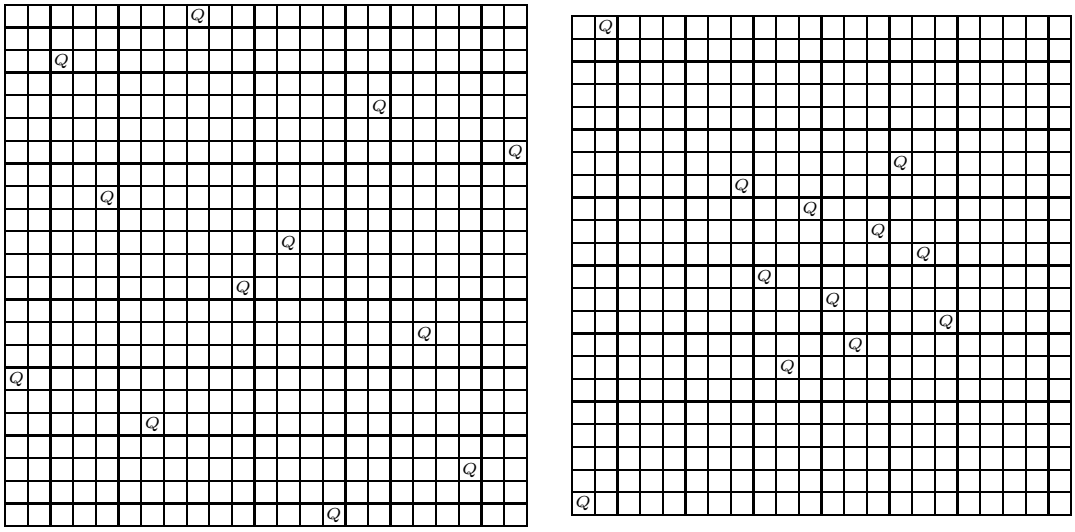


Figure 5: A dominating configuration of 12 queens on a 23 by 23 board and an independent dominating configuration of 12 queens on a 22 by 22 board.

Queens Independent Domination

Board size	Number of Queens	Search Size	Average Time
12	7	11	0.3 secs
13	7	12	11 secs
14	8	14	5 secs
15	9	17	2 secs
16	9	22	17 secs
17	9	22	56 mins
18	10	22	27 mins
19	11	22	101 secs
20	11	22	16 mins
21	11	22	7 hours
22	**12**	22	4 hours

Table 9: Running times for the probabilistic algorithm solving the independent queens domination problem.

5.2 Domination in Q_{4k+1}

Weakley [22] has shown that $\gamma(Q_{4k+1}) \geq 2k+1$ for all integers k, and has conjectured that $\gamma(Q_{4k+1}) = 2k+1$ for all integers k. This conjecture has been confirmed for all $k \leq 15$ by generating dominating sets of size $2k+1$ (see Weakley [22] for $k \leq 6$ and $k = 8$, Burger, Cockayne and Mynhardt [3] for $k = 9, 12, 13, 15$, Gibbons and Webb [12] $k = 7, 10, 11, 14$.

In this section we adapt the above local search algorithm to find independent dominating sets of size $2k+1$ for $k \leq 21$.

The method used in [12] to generate dominating sets of size $2k+1$ was that of simulated annealing. In this paper it was noted that many dominating sets contain queens only on even-even squares. For example see the dominating set for $k = 3$ in Figure 6. In order to check whether such a configuration is dominating, it is necessary to only check that the odd-odd squares are attacked since all others are attacked by a queen in either the same row or column. This leads to a compressed representation of the board where even rows and columns are represented as lines and odd-odd squares form the squares in the compressed representation. Queens are now placed on the intersections of the lines, with one queen per line. Figure 7 shows the compressed board of Figure 6. In this representation all squares must be dominated diagonally by some queen.

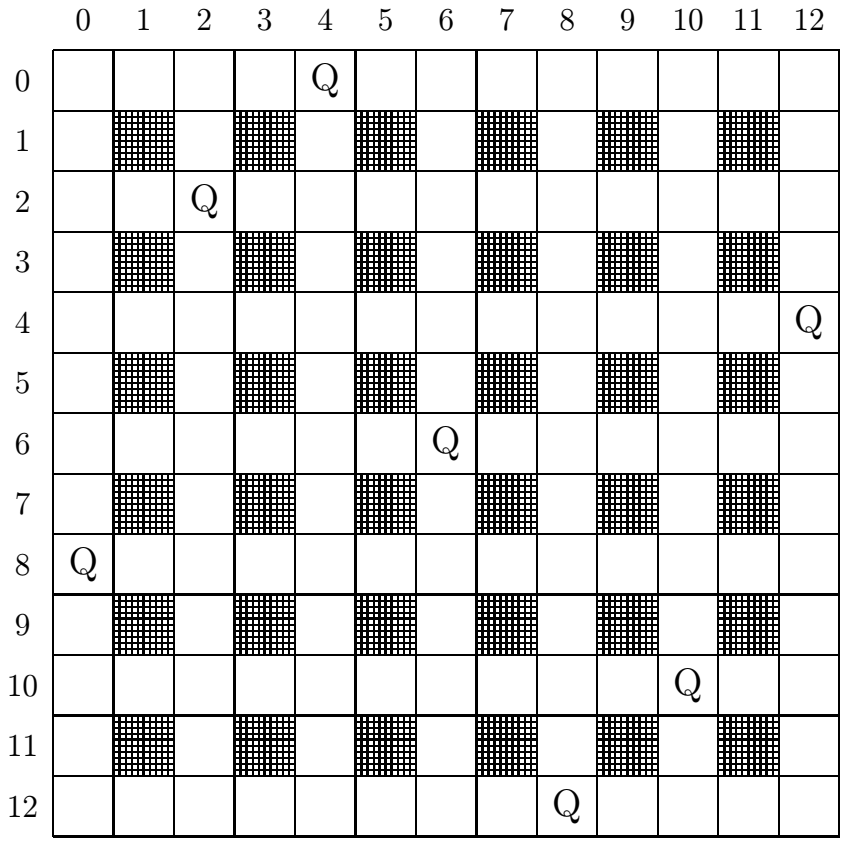


Figure 6: A 13 x 13 board (k=3) dominated by 7 queens. Odd-odd squares are shaded.

An initial configuration for the simulated annealing algorithm was generated by randomly

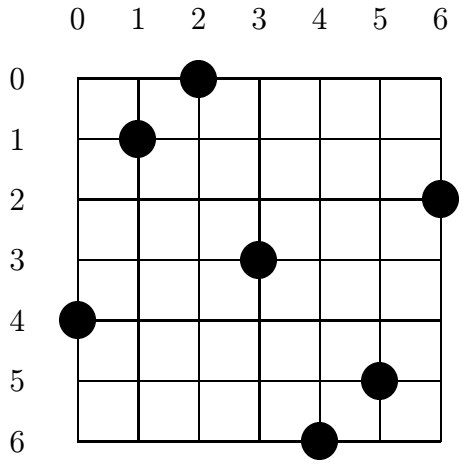


Figure 7: A compressed representation of Figure 6.

placing the $2k+1$ queens with one per row and column of the compressed board. The cost of a configuration of queens was the number of undominated squares. The transformation method was to choose two random queens at positions (x_1,y_1) and (x_2,y_2) and move them to positions (x_1,y_2) and (x_2,y_1). This preserves the one queen per row and column constraint, and allows for the new configuration to possibly dominate different squares along the diagonals.

By observing the dominating configurations found on boards of size $4k+1$ interesting patterns emerge. When looking at the compressed board representation for large boards, it was noted that apart from the edges and the centre of the board, only every second diagonal was occupied in most cases. In fact the board could be dominated if only every second diagonal were occupied as illustrated in Figure 8(a) for the case of $k = 4$. The board could also be dominated by omitting some number of corner diagonals and including more centre diagonals. In general this is denoted as an X/Y set of diagonals, where X (resp. Y) down (resp. up) diagonals have been removed from each of the two appropriate corners and X (resp. Y) pairs of up (resp. down) diagonals added to the centre. Figure 8 gives examples of 0/0, 0/1, 1/1, and 1/2 sets of diagonals for $k = 4$.

These diagonals can be specified formally by first numbering the diagonals. Let the squares be numbered from 0 to $2k-1$ across and down the board. Then the down diagonal passing through square (x, y) is numbered $2k + x - y$ and the up diagonal is numbered $x + y + 1$.

An X/Y dominating set of diagonals then consists of

- Down Diagonals

  $(2 + 2X), (2 + 2X) + 2, ..., (4k - 2 - 2X) - 2, (4k - 2 - 2X)$

  $(2k + 1 - 2Y), (2k + 1 - 2Y) + 2, ..., (2k - 1 + 2Y) - 2, (2k - 1 + 2Y)$

- Up Diagonals

  $(2 + 2Y), (2 + 2Y) + 2, ..., (4k - 2 - 2Y) - 2, (4k - 2 - 2Y)$

  $(2k + 1 - 2X), (2k + 1 - 2X) + 2, ..., (2k - 1 + 2X) - 2, (2k - 1 + 2X)$

These cases all use mostly even numbered diagonals and are referred to as an even diagonal set. It is also possible to dominate the board with an odd diagonal set, which consists mostly odd numbered diagonals. These are

- Down Diagonals

  $(1+2X), (1+2X)+2, ..., (4k-1-2X)-2, (4k-1-2X)$

  $(2k+2-2Y), (2k+2-2Y)+2, ..., (2k-2+2Y)-2, (2k-2+2Y)$

- Up Diagonals

  $(1+2Y), (1+2Y)+2, ..., (4k-1-2Y)-2, (4k-1-2Y)$

  $(2k+2-2X), (2k+2-2X)+2, ..., (2k-2+2X)-2, (2k-2+2X)$

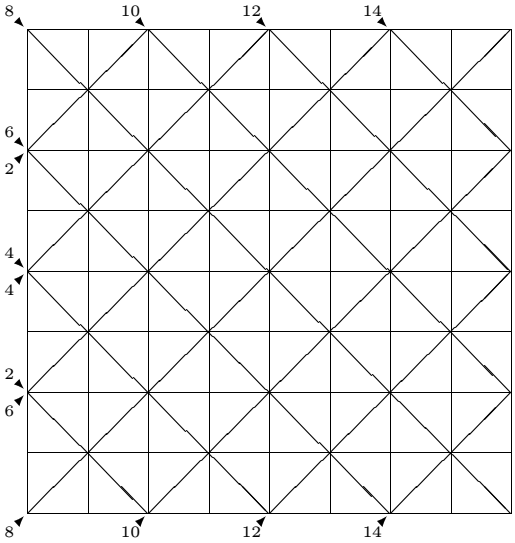
In order for the board to be dominated, there must still be one queen on each vertical and horizontal line and at least one queen occupying each of the specified diagonals. If even cases are restricted to X and Y being less than k, and odd cases are restricted to X and Y being less than $k+1$ and not equal to 0, then there are a total of $2k-1-2X+2Y$ down diagonals and $2k-1-2Y+2X$ up diagonals. There are at most $2k+1$ queens, so at most $2k+1$ diagonals of each type can be occupied, which leads to the constraint that $|X-Y| \leq 1$. Furthermore, the total number of diagonals required overall is $4k-2$ which is 4 less than the maximum that could be attacked by $2k+1$ queens, so in any solution there will be 4 spare diagonals. These could take the form of multiple queens attacking the same diagonal, or redundant diagonals being attacked. Also note that an even X/Y configuration is the same as an odd $(k-Y)/(k-X)$ configuration, hence only cases where either X or Y $\leq \frac{k}{2}$ need be considered.

Solutions to the problem can then be found through a local search for configurations containing a queen in each row and diagonal such that one of these diagonal configurations is dominated. A prescribed set of diagonals is initially chosen, and the transformation of Gibbons and Webb's simulated annealing algorithm is used. The cost of a configuration is much more easily computed as the number of specified diagonals that are not occupied, rather than the number of undominated squares.

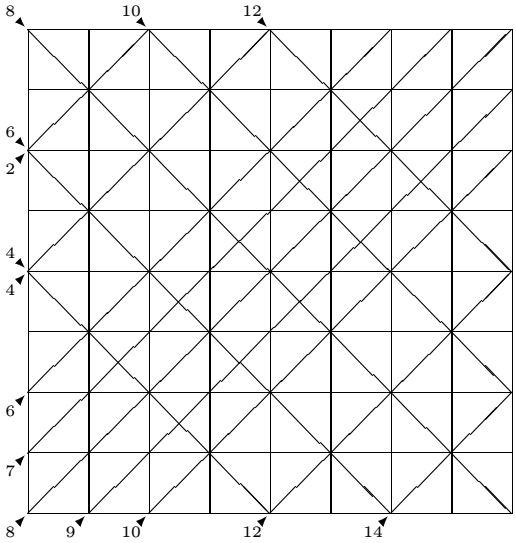
Both even-diagonal and odd-diagonal sets for a range of X/Y values were tested for all values of k up to 23. For all cases with $k \leq 20$ at least one even solution, and with $k \leq 21$ at least one odd solution was found.

The types of solutions found are given in Table 11 in the Appendix. Sample solutions for even cases can be found in Table 12 and odd cases in Table 13, where the nth element in a solution is the row in which the queen in the nth column is placed. An image of the compressed board for a solution with $k = 20$ can be seen in Figure 9.

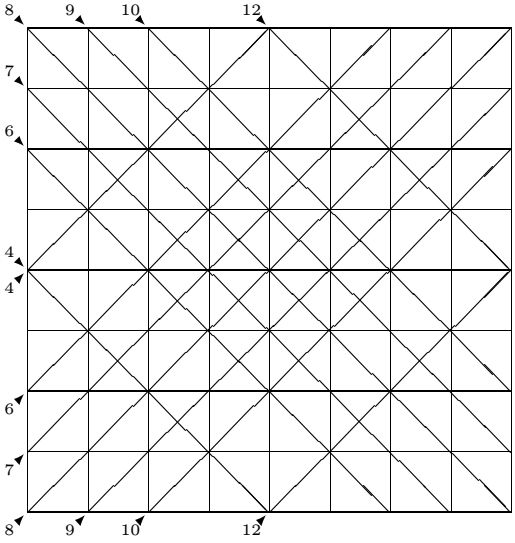
(a) A 0/0 set of dominating diagonals.



(b) A 1/0 set of dominating diagonals.



(c) A 1/1 set of dominating diagonals.



(d) A 2/1 set of dominating diagonals.

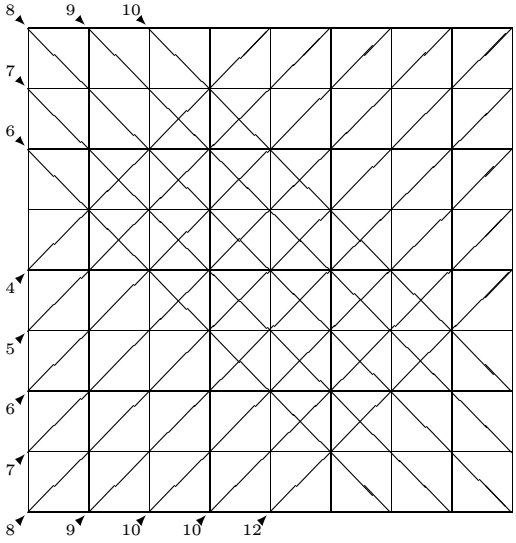


Figure 8: These compressed representations of 17×17 boards ($k = 4$) are dominated by four different sets of numbered diagonals.

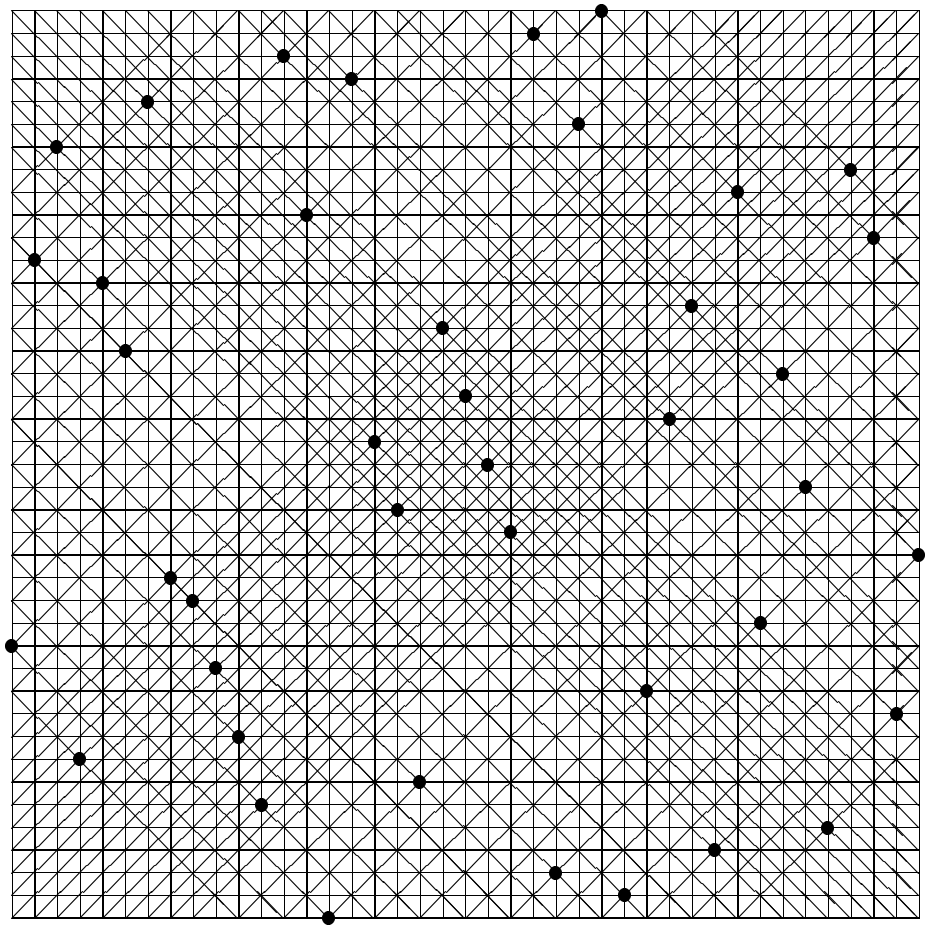


Figure 9: An even 3/4 dominating set for k=20. This corresponds to a set of 41 queens dominating an 81 x 81 board, proving that $\gamma(Q_{81}) = 41$. Note the pair of queens on down diagonals 22 and 30, and also coverage of the redundant down diagonals 51 and 57.

5.3 Further Observations

For all even cases up to k=20 and odd cases up to k=21, an X/(X+1) solution was always found. Furthermore, X increased by 1 for approximately every 4 values of k once k became large enough. Continuing with this pattern, it would be expected that for k=21 to 24, even 4/5 solutions would exist, and for k=22 to 25, odd 5/6 solutions would exist. The algorithm was run extensively on these and other cases with running times totalling several weeks, but no further results emerged for odd cases higher than k=21 or even cases higher than k=20. This could be due to the general increased running time the algorithm requires with moving to higher X/(X+1) solutions.

It is also seen that for even X/(X+1) configurations, all but $2X+2$ queens lie on even/even or odd/odd squares. Furthermore, $2X$ of these must lie within the central rectangle that occupies the intersection central odd diagonals. Similarly, for odd X/(X+1) configurations, all but $2X+1$ queens lie on even/odd or odd/even squares, and $2X-1$ of these must lie in the central even diagonal intersection region. This information could be made use of in the form of a more restricted search space, but attempts to do so have so far been unsuccessful.

6 Concluding Remarks

We have presented a number of computational search techniques that have been applied to various chessboard domination problems. These have been successful in generating new dominating configurations on $n \times n$ chessboards, for small values of n. For example, new optimal dominating and independent dominating sets of queens have been found to produce an updated set of known values for $\gamma(Q_n)$ and $i(Q_n)$ in Table 10.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\gamma(Q_n)$	1	1	1	2	3	3	4	5	5	5	5	6	7	8	**9**
$i(Q_n)$	1	1	1	3	3	4	4	5	5	5	5	7	7	8	9

n	16	17	18	19	20	21	22	23	24	25
$\gamma(Q_n)$	**9**	9	9	**10**	≤ 11	11	≤ 12	≤ 12	≤ 13	13
$i(Q_n)$	9	9	**10**	≤ 11	≤ 11	11	$\leq$ **12**	≤ 13	≤ 13	13

Table 10: Known values for $\gamma(Q_n)$ and $i(Q_n)$. **Bold** values are new results from this paper.

We believe that methods such as the reduction technique could be successfully applied to other chessboard and combinatorial problems.

We have shown that $ir(Q_n) = \gamma(Q_n)$ for $n \leq 13$. Although it is possible for some n that $ir(Q_n) < \gamma(Q_n)$ no such case has yet been found. By examining the difference between the theoretical lower bound of $\gamma(Q_n) \geq \frac{n-1}{2}$ and the actual value of $\gamma(Q_n)$ it is found that for known values this is maximal at $n = 15$, where the difference is 2. This indicates that Q_{15} could be a likely candidate for a situation where $ir(Q_n) < \gamma(Q_n)$ is true. Improvements to the algorithm used in this paper may allow a search for such a case.

The probabilistic algorithm developed here to establish that $\gamma(Q_{4k+1}) = 2k+1$ for $16 \leq k \leq 21$ was very successful. However the transformation method used was rather

simplistic and if improved upon further solutions may be found. There are also additional patterns of solutions that were not taken advantage of during the search.

In conclusion we believe that computational techniques such as the ones described can help shed light on many of the remaining open chessboard problems listed in [10] and [14].

A Appendix

k	Solution Types Found	Average Time
1	Even 0/1 ; Odd 0/1	-
2	Even 0/1 ; Odd 0/1	-
3	Even 0/0, 0/1, 1/2 ; Odd 1/1, 1/2	-
4	Even 0/0, 0/1, 1/2 ; Odd 0/1, 1/1	-
5	Even 0/1, 1/1, 1/2 ; Odd 1/1, 1/2	-
6	Even 0/1, 1/1 ; Odd 1/1, 1/2, 2/3	-
7	Even 1/2 ; Odd 1/2	0.3 secs
8	Even 1/2 ; Odd 1/2	2 secs
9	Even 1/2 ; Odd 1/2	4 secs
10	Even 1/2	3 secs
	Odd 2/3	13 secs
11	Even 1/2	9 secs
	Even 2/3	35 secs
	Odd 2/3	35 secs
12	Even 2/3	150 secs
	Odd 2/3	12 secs
13	Even 2/3	60 secs
	Odd 2/3	80 secs
	Odd 3/3	110 secs
14	Even 2/3	60 secs
	Odd 3/4	240 secs
15	Even 2/3	5 mins
	Odd 3/3	20 mins
	Odd 3/4	24 mins
16	Even 2/3	3 hours
	Odd 3/4	10 mins
17	Even 3/4	3 hours
	Odd 3/4	1 hour
18	Even 3/4	7 hours
	Odd 4/5	28 hours
19	Even 3/4	1 hour
	Odd 4/5	6 hours
20	Even 3/4	4 hours
	Odd 4/5	50 hours
21	Odd 4/5	30 hours

Table 11: Solution types found for $\gamma(Q_{4k+1}) = 2k + 1$.

k	Solution Type	Sample Solution
1	Even 0/1	(0,2,1)
2	Even 0/1	(1,3,0,2,4)
3	Even 0/1	(2,5,1,4,0,3,6)
4	Even 0/1	(1,7,2,5,8,4,0,3,6)
5	Even 0/1	(8,7,2,4,3,9,0,5,10,1,6)
6	Even 0/1	(10,5,2,11,3,6,12,9,0,7,4,1,8)
7	Even 1/2	(6,3,10,1,14,9,2,8,5,12,0,11,4,7,13)
8	Even 1/2	(12,7,3,11,2,1,16,10,8,6,0,15,14,5,13,9,4)
9	Even 1/2	(8,15,12,6,3,1,18,11,2,10,7,17,0,13,16,5,14,9,4)
10	Even 1/2	(10,15,8,3,20,17,7,5,0,12,9,19,16,1,4,11,18,14,2,13,6)
11	Even 1/2	(16,19,6,3,18,13,7,17,2,12,22,1,11,10,0,21,14,9,20,15,8,5,4)
12	Even 2/3	(18,11,6,19,0,5,8,4,20,12,24,16,13,21,9,1,22,15,2,7,10,3,14, 17,23)
13	Even 2/3	(12,7,22,4,18,15,6,25,2,5,26,14,17,10,13,1,24,21,0,11,8,23, 20,3,19,9,16)
14	Even 2/3	(8,19,12,17,26,21,3,5,28,9,0,14,4,18,15,27,11,23,20,13,2,25, 6,1,10,24,22,7,16)
15	Even 2/3	(12,15,20,7,26,29,2,27,11,19,4,8,17,14,30,3,24,16,13,1,6,21, 28,25,0,23,10,5,18,9,22)
16	Even 2/3	(8,21,24,27,20,15,32,7,4,23,0,16,6,31,17,1,13,18,30,14,19, 25,28,3,12,9,2,29,26,11,22,5,10)
17	Even 3/4	(2,23,28,15,8,5,14,31,6,25,11,29,4,33,21,18,13,27,32,12,19, 16,0,3,30,7,10,1,22,34,26,17,20,9,24)
18	Even 3/4	(7,21,28,25,14,9,20,35,2,11,34,29,0,12,17,1,32,22,19,16,13, 3,30,18,36,31,6,23,26,5,8,15,4,33,24,27,10)
19	Even 3/4	(26,13,32,10,14,5,24,15,28,37,2,9,36,3,30,20,38,16,19,33, 23,18,6,1,17,7,21,29,0,35,34,11,8,27,4,31,22,25,12)
20	Even 3/4	(28,9,22,13,8,29,32,39,14,5,40,33,30,7,4,20,15,24,21,11,36, 16,23,37,2,18,38,27,12,3,6,1,0,35,31,17,34,25,10,19,26)

Table 12: Even solutions found for $\gamma(Q_{4k+1}) = 2k + 1$.

k	Solution Type	Sample Solution
1	Odd 0/1	(1,0,2)
2	Odd 0/1	(3,0,2,4,1)
3	Odd 1/2	(2,0,5,3,1,6,4)
4	Odd 0/1	(7,6,3,0,4,8,5,2,1)
5	Odd 1/2	(7,3,1,6,9,0,4,8,5,2,10)
6	Odd 1/2	(9,0,3,10,6,2,11,5,8,12,1,4,7)
7	Odd 1/2	(7,12,3,8,4,14,1,9,6,0,13,10,5,2,11)
8	Odd 1/2	(9,4,13,16,3,12,7,5,8,2,15,0,14,10,1,6,11)
9	Odd 1/2	(15,4,7,16,13,6,17,0,10,9,8,18,1,12,5,2,11,14,3)
10	Odd 2/3	(0,10,13,16,5,20,1,14,12,6,8,2,17,9,3,18,15,19,11,4,7)
11	Odd 2/3	(11,5,17,20,13,2,6,14,1,22,12,9,3,0,10,18,21,16,19,8,15,4,7)
12	Odd 2/3	(7,14,11,22,5,9,23,4,19,24,12,16,1,13,10,6,3,0,21,18,15,2,20, 8,17)
13	Odd 2/3	(7,20,11,8,5,26,23,18,10,24,14,4,25,13,3,2,12,15,21,0,17,6,1, 22,19,16,9)
14	Odd 3/4	(19,8,13,5,21,18,1,28,23,2,14,17,9,0,3,11,16,26,12,22,25,6,20, 24,15,4,7,10,27)
15	Odd 3/4	(3,22,25,12,9,24,19,4,1,13,27,26,18,15,5,2,10,17,14,30,21,0, 29,8,23,20,7,16,11,6,28)
16	Odd 3/4	(13,18,25,8,19,24,31,28,1,4,11,7,16,2,20,17,14,0,29,26,5,15, 21,32,27,12,23,6,3,10,30,22,9)
17	Odd 3/4	(9,24,13,10,31,7,27,22,33,2,11,16,18,30,3,34,20,15,1,32,14, 17,25,0,21,12,5,8,28,4,29,26,23,6,19)
18	Odd 4/5	(7,12,19,30,27,10,31,1,11,2,25,6,18,34,22,17,35,21,5,15,33,4, 20,32,16,24,3,0,29,14,9,28,13,8,23,26,36)
19	Odd 4/5	(27,12,21,28,6,10,35,4,11,16,33,0,5,19,22,32,29,38,3,23,18, 13,37,17,20,34,9,26,1,8,31,2,7,24,15,30,36,14,25)
20	Odd 4/5	(27,16,13,3,29,36,7,30,37,12,1,28,17,34,9,23,18,2,22,25,39,4, 14,19,33,6,20,40,35,26,5,0,21,32,11,8,15,24,31,10,38)
21	Odd 4/5	(13,28,19,26,37,12,9,18,3,34,41,17,11,42,35,4,31,25,20,38,24, 15,7,23,16,2,22,6,33,10,39,0,5,30,1,36,29,8,21,14,27,32,40)

Table 13: Odd solutions found for $\gamma(Q_{4k+1}) = 2k + 1$.

References

[1] W. W. Rouse Ball, Mathematical Recreation and Problems of Past and Present Times, *MacMillan, London* (1892).

[2] J. R. Bitner and E. M. Reingold, Backtrack Programming Techniques, *Comm. ACM*, 18 (1975), 651-656.

[3] A. P. Burger, E. J. Cockayne, and C. M. Mynhardt, Domination numbers for the Queens' graph, *Bull. Inst. Combin. Appl. 10* (1994) 73-82.

[4] A. P. Burger, E. J. Cockayne, and C. M. Mynhardt, Domination and Irredundance in the Queens' graph, *Discrete Mathematics* 163(1997) 47-66.

[5] E.J. Cockayne, S.T. Hedetniemi, and D.J. Miller, Properties of hereditary hypergraphs and middle graphs. *Canad. Math. Bull.*, 21 (4), 461 - 468, 1978.

[6] E. J. Cockayne, Irredundance in the Queens' Graph, *submitted.*

[7] E. J. Cockayne, Chessboard Domination Problems, *Discrete Mathematics 86* (1990) 13-20.

[8] E. J. Cockayne and P. H. Spencer, On the Independent Queens Covering Problem, *Graphs and Combinatorics 4* (1988) 101-110.

[9] M. Eisenstein, C. M. Grinstead, B. Hahne, D. Van Stone The Queen Domination Problem *Congr. Numer. 91* (1992) 189-193.

[10] G. H. Fricke, S. M. Hedetniemi, S. T. Hedetniemi, A. A. McRae, C. K. Wallis, M.S. Jacobson, H. W. Martin, and W. D. Weakley, Combinatorial Problems on Chessboards: a Brief Survey, in: *Alavi and Schwenk eds., Graph Theory, Combinatorics and Applications, vol. 1, Proc. Seventh Quadrennial Conf. on the Theory and Applications of Graphs, Western Michigan University* (Wiley, 1995).

[11] P. B. Gibbons, Computational Methods in Design Theory, in *The CRC Handbook of Combinatorial Designs, C. J. Colbourn and J. H. Dinitz (eds.),CRC Press* (1996) Chapter 9, Section VI.

[12] P.B. Gibbons and J. A. Webb, Some New Results for the Queens Domination Problem, *Australasian Journal of Combinatorics 15* (1997) 145-160.

[13] C. M. Grinstead, B. Hahne, D. Van Stone, On the Queen Domination Problem, *Discrete Mathematics 86* (1991) 21-26.

[14] Sandra M. Hedetniemi, Stephen T. Hedetniemi, and Robert Reynolds, Combinatorial Problems on Chessboards: II, in: *Domination in Graphs, Advanced Topics* (1997) Chapter 6.

[15] Matthew D. Kearse, Lower Bounds on Iredundance in the Queens' Graph, to appear.

[16] E. Lucas, Récréations Mathématiques, *Gauthier-Villars, Paris* (1891).

[17] Brendan D. McKay, Isomorph-Free Exhaustive Generation, *Journal of Algorithms 26* (1998) 306-324.

[18] Patrick Prosser, Hybrid Algorithms for the Constraint Satisfaction Problem, *Computational Intelligence 9* (1993) 268-299.

[19] J. D. Swift, Isomorph Rejection in Exhaustive Search Techniques, *Proc. A. M. S. Symp. Apple. Math. 10* (1958), 195-200.

[20] Johannes Waldmann, Private correspondence (1998).

[21] R. J. Walker, An enumerative Technique for a Class of Combinatorial Problems, in *Combinatorial Analysis (Proceedings of Symposia in Applied Mathematics, Vol. X), American Mathematical Society, Providence, R. I.* (1960).

[22] W. D. Weakley, Domination in the Queen's Graph, *Graph Theory, Combinatorics, and Applications 2* (1995) 1223-1232.

[23] W. D. Weakley, Upper Bounds for Domination Numbers of the Queens Graph, preprint.

[24] A. M. Yaglom and I. M. Yaglom, Challenging Mathematical Problems with Elementary Solutions, *volume 1:Combinatorial Analysis and Probability Theory.* Holden-Day, San Francisco. (1964).