Solving the N-Queens problem by means of a genetic algorithm with different strategies

Martí Municoy & Daniel Salgado

Master's degree in Modelling for Science and Engineering
Optimisation and Research & Innovation, 2017/2018.

Abstract

In this report, we describe a mathematical model to implement and solve the famous N-queens problem by means of a Genetic Algorithm (G.A.). A theoretical study about the basic G.A. ideas is done to scope out the main tools that are going to be useful to solve this problem.

These include different algorithms and genetic operators to perform selection of individuals, crossovers and mutations. An analysis of different genetic parameters is also performed in order to achieve either a more exploratory algorithm or one more exploitatory.

Moreover, we introduce a program developed by us which uses these mathematical tools to model a genetic algorithm. It is, in principle, able to solve systems composed by up to 2000 queens. Another implementation based on the same approaches is able to find all the different solutions to the N-queens problem, at least for $N \leq 11$.

Our G.A. implementation has been proved to be a reliable tool to solve the N-queens problem. It is another example of how G.A. can be applied to heterogeneous and complex systems and get successful results in an efficient way. In the Appendix Section, some of the results obtained by means of this implementation are included.

Universitat Autònoma de Barcelona

CENTRE DE RECERCA MATEMÀTICA

Contents

1 Introduction

Genetic algorithms (G.A.) are adaptive methods which can be used to solve search and optimisation problems. They are mainly based on the mechanism with which populations of biological species evolve, the *natural selection* and “survival of the fittest”, introduced by Charles Darwin (*The Origin of Species, 1872*), [2].

In a given population of individuals, those who are most successful surviving and competing with others (*fitter*), are more likely to have more children, and thus to hereat their genetic properties to the next generation of individuals. Thus, population individuals tend to be fitter and fitter over generations.

Genetic algorithms use a direct analogy to this natural behaviour. They work with a *population* of *individuals*, each representing a possible solution to a given problem [6], and each individual has assigned a *fitness scorer* (given by some **fitness** function) according to how good it represents a solution for the problem. Then, according to three basic biological principles such as are **selection** of fitter individuals to reproduce (natural selection), **crossover** (reproduction between individuals) and **mutation** (random changes in the genetic information carried by an individual), a genetic algorithm is expected to make evolve the population of individuals over generations, in such a way that the average *fitness score* tends to the *optimal fitness score*, i.e., so that the individuals tend to the optimal solution of the problem.

These concepts of fitness, selection, crossover and mutation, allow us to describe the basic structure of a genetic algorithm as follows [8]:

1) A random population of individuals (potential solutions) is created. All individuals are evaluated using a fitness function,

2) Certain number of individuals that will survive into next generation is selected using selection operator. Selection is somewhat biased, favouring “better” individuals,

3) Selected individuals act as parents that are combined using crossover operator to create children.

4) A mutation operator is applied on new individuals. It randomly changes few individuals (mutation probability is usually low),

5) Children are also evaluated. Together with parents they form the next generation.

Steps 2) to 5) are repeated until a given number of iterations have been run, solution improvement rate falls below some threshold, or some other stop or convergence condition has been satisfied.

2 The N-Queens problem

The eight queens problem consists of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other. In particular, a solution requires that no two queens share the same row, column, or diagonal, [3]. It is known that there exist exactly 92 different solutions. A possible solution is displayed in Figure 2.1

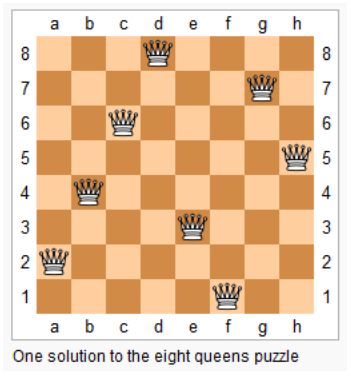


Figure 2.1: A possible solution to the 8-queens problem.

The N-queens problem is a generalisation of the 8-queens problem, where one considers a NxN chessboard and the goal is to place N non-attacking queens. It is known that solutions exist for all natural numbers $N \in \mathbb{N} \backslash \{2, 3\}$.

2.1 Individuals

To define which individuals are valid and solution candidates for our problem, we follow a quite standard approach (see for example [8]).

A necessary condition for an individual to be a solution is that no two queens share the same row or column. Thus, we can assume that the i-th queen, Q_i, is placed in the i-th column, and the q_i-th row (so we identify $Q_i \equiv (i, q_i) \equiv$ (`column, row`)), where $q_i \in \{1, ..., N\}$ and $q_i \neq q_j \ \forall i \neq j$. In other words, all solutions to the N-queens problem can be represented as N-tuples $(q_1, q_2, ..., q_N)$ [1] that are permutations of the N-tuple $(1, 2, ..., N)$, and for these reason we consider as valid individuals those that can be represented by such N-tuple. Thus, the index position (in ascendant order) of a tuple gives the column, and the value of the tuple at that position represents the row.

In Figure 2.2 we show a graphical representation of the N-tuple notation for the case $N = 4$.

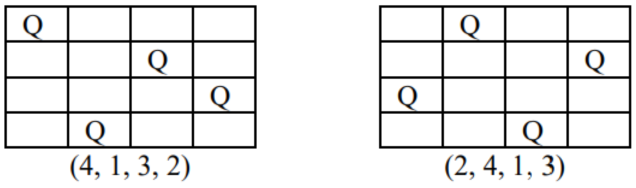


Figure 2.2: N-tuple notation examples when $N = 4$, [8].

[1]Sometimes we will refer to one of these tuples as "the genes" of the individual that the tuple in consideration is representing.

It is important to take into account this condition of valid individuals for when performing **crossover** and **mutation** procedures. We have to assure that at any time during the algorithm, all individuals are valid and thus potential solutions to the problem. Under these restrictions about the individuals, we have that a necessary and sufficient condition to an individual be a solution of the problem is that any pair of queens threaten one another in a diagonal.

2.2 Fitness function

Traditionally and according to the biological inspiration of genetic algorithms, one usually chooses a **fitness** function such that better individuals have higher fitness values and then tries to maximize it over the population. For convenience we will do the other way round, our fitness function will take smaller values for better individuals and our goal will be to minimize it over the population.

Given an individual $I = (q_1, .., q_N)$, let F_I be the total number of different pairs of queens that meet one another (in a diagonal), i.e., the cardinality of the set $\{(Q_i, Q_j) \mid i < j \ , \ Q_i \text{ threatens } Q_j \text{ in some diagonal}\}$, and N the number of queens. It is easy to see that the maximum value that F_I can take corresponds to the situation when all queens are located in the same diagonal i.e., either if $I = (1, 2, ..., N)$ or $I = (N, N-1, ..., 1)$, so that each queen threatens any other queen. In these cases:

$$F_I = \left|\{(Q_i, Q_j) \mid i < j\}\right| = \binom{N}{2} = \frac{N(N+1)}{2}$$

In addition, an individual I is solution of the N-queens problem if and only if there are not pairs of queens threatening one another, that is $F_I = 0$.

Therefore we have that $0 \leq F_I \leq \frac{N(N+1)}{2} =: F_{\max}$ for any individual I.

In order to know if a queen $Q_i \equiv (i, q_i)$ meets another queen $Q_j \equiv (j, q_j)$ in a diagonal, it is enough to check that the straight line formed by joining the points (i, q_i) and (j, q_j) has unitary slope, i.e.,

$$1 = \left| \frac{\Delta\texttt{rows}}{\Delta\texttt{columns}} \right| = \left| \frac{q_j - q_i}{j - i} \right| \Longleftrightarrow |j - i| = |q_j - q_i|$$

Thus, the algorithm to compute F_I for a given individual $I = (q_1, ..., q_N)$ may look like:

Algorithm 1 Fitness of an individual

```
1: procedure fitness( $I \equiv (q_1, ..., q_N), N$ )
2:     $F_I \leftarrow 0$                                   ▷ Initialise fitness to 0
3:     for ( $i; 1 \leq i \leq N$ ) do
4:         for ( $j; i < j \leq N$ ) do
5:             if $|q_j - q_i| = |j - i|$ then $F_I \leftarrow F_I + 1$
6:     return $F_I$                                   ▷ Return the fitness of individual$I$
   end procedure
```

Note that this algorithm to compute the fitness of an individual is quadratic with the number of queens, $\mathcal{O}(N^2)$. In [8] it is proposed an alternative fitness function where the computational cost is linear, $\mathcal{O}(2N-1)$.

3 Genetic algorithm approach

The body of the main algorithm that we consider to solve the N-queens problem by means of a G.A. is shown below (Algorithm 2). We can briefly describe it as follows:

1) First we declare and allocate memory for the necessary variables.
2) Next we initialise the individuals of the current population (i.e., their genes $(q_1, ..., q_N)$ and their id), using the **initiate** function which is described in Algorithm 3.
3) Then, we update the fitness of each individual in the population using a **evaluate** function (see Algorithm 4) which makes use of the **fitness** function described in Algorithm 1.
4) After this initialisation stage, we can run a **genetic_algorithm_strategy** which at the end returns one of the fittest individuals found (which could be a solution or not). This G.A. procedure is described in Algorithm 5.

Algorithm 2 Main algorithm

```
procedure Main( input arguments )
    id ← 1                                                  ▷ First individual id
    population_size ← n_pop                                 ▷ Population size
    Individual *best, *population, *nextpopulation          ▷ Initialise arrays and pointers
    Individual parent1, parent2, child, survivor            ▷ Initialise single individuals
    Save memory space for population and nextpopulation arrays

    best ← ■                                                ▷ Initialise best individual
    initiate(population, id, extra_arguments)               ▷ Initialise the population
    evaluate(population, extra_arguments)                   ▷ Evaluate the population (fitness)
    deaths ← n_deaths                                       ▷ # deaths per generation
    survivors ← population_size - deaths                    ▷ # survivors per generation

    best ← genetic_algorithm_strategy( population, nextpopulation, best,
                  parent1, parent2, child, survivor, extra_arguments)    ▷ G.A.

    show best                                               ▷ Show the “best” individual found
end procedure
```

Algorithm 3 Initiate the properties of each individual in a given population: its genes, its id and its fitness

```
procedure initiate( population (empty), id (last id used), extra_arguments)
    $F_{\max} \leftarrow \frac{N(N+1)}{2}$
    for $I \in$ population do
        $F_I \leftarrow F_{\max}$
        $(q_1, ..., q_N)_I \leftarrow$ random_permutation( (1,2,...,N))
        $id_I \leftarrow$ id; id ← id +1
    end for
end procedure
```

Algorithm 4 Evaluation of the fitnesses of each individual in a given population

```
procedure evaluate( population, $N$ )
    for $I \in$ population do $F_I \leftarrow$ fitness(I, N)
    end for
end procedure
```

The general genetic algorithm strategy that we consider assumes a constant population size (population_size) over generations; a maximum number of generations (max_generations); a fixed number of deaths (individuals from the current population that are replaced by children in the next population), and thus a fixed number of individuals from the current population that survive and form part of the next generation population (survivors). The corresponding Algorithm 5 is quite easy to understand.

Algorithm 5 Genetic algorithm strategy

```
procedure genetic_algorithm_strategy (population, nextpopulation, best, parent1, parent2,
child, survivor, extra_arguments)
    generation $\leftarrow$ 1                                         ▷ Initialise generations counter

    while generation $\leq$ max_generations do

        for $1 \leq i \leq$ deaths do
            parent1 $\leftarrow$ selection(population, selection_arguments)      ▷ Choose first parent
            parent2 $\leftarrow$ selection(population, selection_arguments)  ▷ Choose second parent
            child $\leftarrow$ crossover(parent1, parent2, crossover_arguments)      ▷ Obtain a child
            child $\leftarrow$ mutation(child, mutation_arguments)                  ▷ Mutate the child
            nextpopulation[i] $\leftarrow$ child                   ▷ Add the child to the next population
        end for

        for deaths $< i \leq$ population_size do
            survivor $\leftarrow$ selection(population, selection_arguments)     ▷ Choose a survivor
            nextpopulation[i] $\leftarrow$ survivor          ▷ Add the survivor to the next population
        end for

        swap_roles (population, nextpopulation) ▷ population contains the next population
        evaluate(population, extra_arguments)        ▷ Evaluate the new population (fitness)
        best $\leftarrow$ find_best(population, extra_arguments)            ▷ Update best individual
        generation $\leftarrow$ generation + 1                     ▷ Update the population generation
    end while

    return best                  ▷ Return a best individual from the last generation of individuals
end procedure
```

Until the maximum number of generations is not reached do

1) Repeat death times: two parents are selected via a **selection** operator. Next, a **crossover** operator is applied to these two parents to obtain a child. Then a **mutation** operator is applied to the child. The resulting child obtained at the end is added to the next generation

population. A total of `death` children have been obtained.

2) Select `population_size - deaths` survivor individuals via a **`selection`** operator (it could be different from the one used in the first point) and add them to the next generation population.

3) Swap the roles of the arrays `population` and `population` (a double buffer procedure).

4) Update the fitness of each individual in the new population via the **`evaluate`** operator.

5) Update the `best` individual by looking at the new population and increase by one unit the `generation` counter.

When `generation` has reached the maximum number of generations allowed (`max_generations`), the best individual (one of the fittest, since usually it will not be unique) is returned.

3.1 Selection methods

Parent selection is very crucial to the convergence rate of the GA as good parents drive individuals to better and fitter solutions. However, we should be careful in order to prevent one extremely fit solution from taking over the entire population in a few generations, as this leads to the solutions being close to one another in the solution space thereby leading to a loss of diversity, [9]. This taking up of the entire population by one extremely fit solution is known as premature convergence and is an undesirable condition in a GA. That means that one obtains convergence to a local optimal solution, and not the global optimal solution which is the one of interest.

In this section we discuss the **`selection`** operators that we have implemented in order to perform different G.A. strategies: first the so called **`roulette_selection`** operator and finally the **`tournament_selection`** operator.

3.1.1 Roulette wheel selection

The roulette wheel selection or, simply, **roulette selection** is particular type of **fitness proportionate selection** procedure, where the probability of selecting an individual is directly proportional to its fitness (if we are maximizing) and inversely proportional to its fitness (if we are minimizing). Therefore, such a selection strategy applies a selection pressure to fitter individuals in the population.

The classical approach (assuming that we are interested in maximizing the fitness of the population) is the following:

Consider a population with individuals $I_1, ..., I_P$ and whose fitness is F_{I_i} for each $i = 1, ..., P$. Then the total fitness F of the population will be given by $F = \sum_{i=1}^{P} F_{I_i}$. We can then divide the interval $[0, F)$ in P subintervals $[F_{I_1}, F_{I_1} + F_{I_2})$, ..., $[F - F_{I_P}, F)$, so that the subinterval j has length equal to the fitness of the j-th individual (see Figure 3.1).

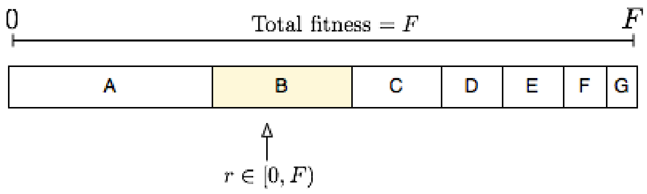


Figure 3.1: Example of the selection of a single individual, [4].

Now if one generates a random number $r \in [0, F)$, then the selected individual will be the one such tat r belongs to its associated subinterval. In other words, the probability p_j of choosing the j-th individual will be

$$p_j = \frac{F_{I_j}}{F} \tag{3.1}$$

Our approach considers that the better individuals are the ones with smallest fitness (we are interested in minimizing the fitness). Thus, for us, the probability of selecting an individual must be inversely proportional to its fitness.

Consider the following function of the fitness F_I of a given individual I:

$$f(F_I) = \frac{1}{(F_I + W)^E} \tag{3.2}$$

where $W, E > 0$ are constants that one can vary.

Now, following the notation introduced in the first illustrative approach that we have described above define

$$F = \sum_{i=1}^{P} f(F_{I_i}), \tag{3.3}$$

and consider a partition of the interval $[0, F)$ in P subintervals C_{I_1},..., C_{I_P} with lengths $f(F_{I_1})$, ..., $f(F_{I_P})$. Similarly as before, if we generate a random number $r \in [0, F)$, the selected individual will be I_k if and only if $r \in C_{I_k} = \left[\sum_{i=1}^{k-1} f(F_{I_i}), \sum_{i=1}^{k} f(F_{I_i})\right)$. Therefore, the probability of selecting the k-th individual will be

$$p_k = \frac{f(F_{I_k})}{F} \tag{3.4}$$

where F is now given by equation (3.3). The case $W = 0, E = 1$ would be the analogous case to classical approach introduced at the beginning of this section. In section 3.5 we will discuss the advantage of having these two coefficients.

The algorithm for the **`roulette_selection`** operator may look like Algorithm 6.

3.1.2 Tournament selection

The so-called K-Way tournament selection consists in selecting K individuals (with or without replacement) from the population at random and select the best out of these to become a parent. Compared to the roulette selection operator which has two constant parameters to vary, the tournament selection has only the parameter K. As we will detail in section 3.5 with only this parameter we can decide to be more or less exploratory (or more or less exploitatory).

In Figure 3.2 it is illustrated a tournament selection without replacement for $K = 3$.

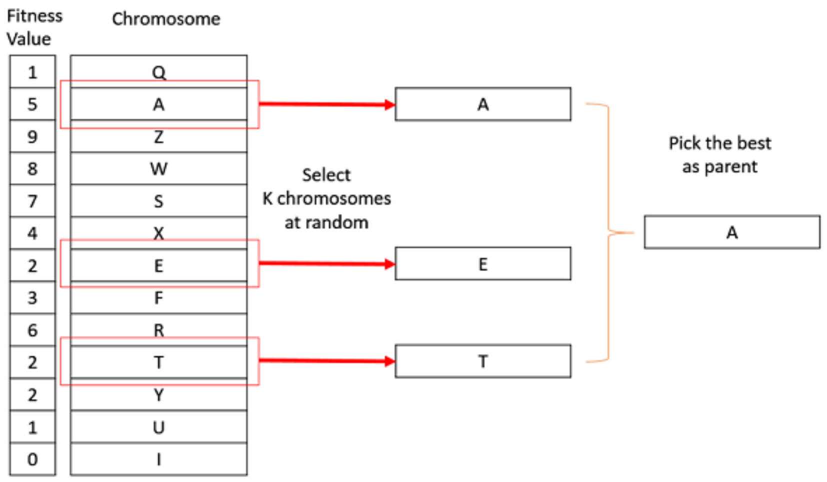


Figure 3.2: Illustrative example of our tournament selection approach, [9]. For us the chromosomes are the N-tuples $(q_1, ..., q_N)$ that define the individuals.

The algorithm that we have implemented for the **`tournament_selection`** operator looks like the one in Algorithm 7.

3.2 Crossover methods

Crossover involves mixing and matching parts of two parents to form a children. In problems such as the N-queens problem where the individuals have to satisfy certain restrictions the child obtained by the crossover procedures must also respect those restrictions.

Imagine that two parents I, I' defined by the tuples $(q_1, q_2, ..., q_N)$ and $(q'_1, q'_2, ..., q'_N)$ respectively are going to form a child by means of a crossover operator. If the resulting child I is defined by the tuple $(q''_1, q''_2, ..., q''_N)$ where dor all $i = 1, ..., N$, $q''_i \in \{q_1, q'_1, q'_2, ..., q'_N, q_1, q_2, ..., q_N\}$ it is necessary that $q''_i \neq q''_j$ for all $i \neq j$.

In this section we discuss the **`crossover`** operator that we have implemented in order to perform different G.A. strategies. This is the so called **`ordered_crossover`** operator. Finally, we will also mention the **`positionbased_crossover`** operator which we also found interesting but still have not implemented it in our program. Both crossover operators lead to children that satisfy the individual restrictions of the N-queens problem.

3.2.1 Ordered crossover

The ordered crossover procedure is performed as follows [7] (see Figure 3.3 for clarification):

Let $(q_1, q_2, ..., q_N)$ and $(q'_1, q'_2, ..., q'_N)$ be the tuples of the two parents I and I' respectively. And $(q''_1, q''_2, ..., q''_N)$ the tuple of the child initialised to the null tuple $(0, 0, ..., 0)$.

1. A parent is chosen at random, lets say I, and a subtuple $(q_i, ..., q_j)$ $(i \leq j)$ is chosen at random (i.e., positions i and j such that $i \leq j$ are chosen at random).
2. Put the subtuple $(q_i, ..., q_j)$ into the child's tuple so that now it becomes $(0, ..., 0, q_i, ..., q_j, 0, ...0)$.
3. Delete the elements from the tuple of the second parent (I') that are already in the subtuple $(q_i, ..., q_j)$. The resultant tuple may look like $(q'_{i_1}, ..., q'_{i_m})$ where $m = N - (j - i + 1)$, $i_1 < \cdots < i_m$ and each q'_{i_k} does not appear in the subtuple $(q_i, ..., q_j)$, for all $k \in \{1, ..., m\}$.
4. Finally, put the values q'_{i_k} in ascendent order into the positions of the child's tuple where there is a 0. The final child's tuple should look like $(q'_{i_1}, q'_{i_2}, ..., q'_{i_l}, q_i, ..., q_j, q'_{i_{l+1}}, ..., q'_{i_m})$.

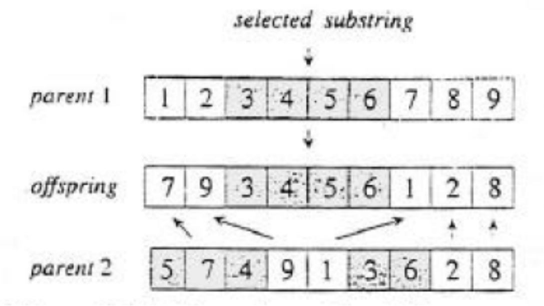


Figure 3.3: Illustrative example of the Ordered crossover procedure when $N = 9$, [7] .

The algorithm for the **ordered_crossover** operator should then be deduced from the above explanation and would depend on the implementation of the genetic algorithm.

3.2.2 Position based crossover

A crossover operator that generalises the **ordered_crossover** operator is the **positionbased_crossover**. Now instead of selecting a subtuple from one of the parents, we select at random a set of positions from a parent (selected at random) and perform the same procedure as for the ordered crossover but now with the genes that are in the set of chosen positions. In Figure 3.4 it is illustrated the position based crossover for the case $N = 8$.

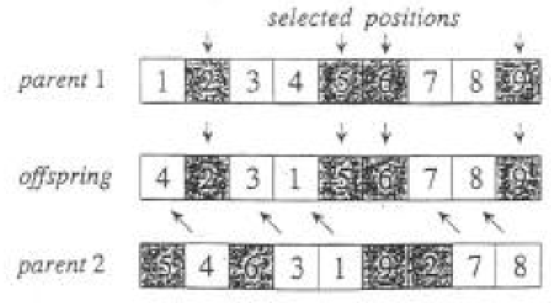


Figure 3.4: Illustrative example of the position based crossover procedure when $N = 8$, [7] .

3.3 Mutation methods

In simple words, mutation may be defined as a small random tweak in the genes of an individual, to get a variation of the original individual. It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability p_m. If the probability is very high, the G.A. gets reduced to a random search, [10].

Mutation is the part of the G.A. which is related to the "exploration" of the search space. It has been observed that mutation is essential to the convergence of the G.A. while crossover is not, [10].

In this section we discuss the **`mutation`** operators that we have implemented in order to perform different G.A. strategies: first the so called **`heuristic_mutation`** operator and finally the **`swap_mutation`** operator.

3.3.1 Heuristic mutation

The so-called heuristic mutation depend basically on two parameters: a probability of mutation p_m and a selection parameter $\lambda \in \{1, ..., N\}$. Given an individual candidate to mutate, lets call it `mutant`, a probability p chosen at random and a value for the parameter λ, if $p < p_m$ then the individual `mutant` will mutate as follows:

1. Pick up λ genes from `mutant` at random.
2. Generate all the possible permutations of these λ genes.
3. Apply all permutations, one by one, to the genes of `mutant`.
4. Evaluate all the resultant individuals and select the best set of genes which is the one that is going to be applied to `mutant`.

In Figure 3.5 we illustrate the heuristic mutation procedure for a particular case when $N = 9$ and $\lambda = 3$.

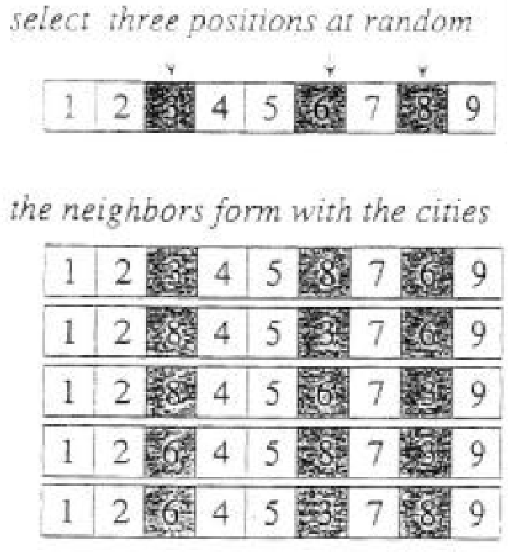


Figure 3.5: Illustrative example of the swap heuristic mutation procedure when $N = 9$ and $\lambda = 3$, [7].

The algorithm for the **heuristic_mutation** operator that we have implemented looks like the one in Algorithm 8

3.3.2 Swap mutation

In swap mutation, we select two positions on the sequence of genes at random, and interchange the values. This is illustrated in Figure 3.6.

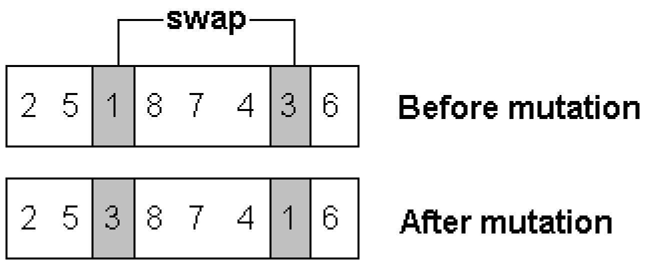


Figure 3.6: Illustrative example of the swap mutation procedure when $N = 8$, [11].

The algorithm for **swap_mutation** is described in Algorithm 9.

3.4 Proposed evolution strategies

In this section we describe the different evolution strategies that we have implemented. Basically, an strategy consists of a variation of Algorithm 5 where we use different combinations of the **selection**, **crossover** and **mutation** operators that we have described in sections 3.1, 3.2 and 3.3, respectively.

If we remember Algorithm 5 we can divide it in two main parts:

- **Stage 1**: selection of parents, crossover between parents to obtain a child, and mutation of the obtained child.
- **Stage 2**: selection of survival individuals.

The concrete algorithms for each of the strategies that we propose can be found in Appendix A.4.

Strategy 1

The first strategy uses the following combination of operators in each stage:

- **Stage 1**: **roulette_selection** , **ordered_crossover** and **swap_mutation**.
- **Stage 2**: **roulette_selection**.

Strategy 2

The second strategy uses the following combination of operators in each stage:

- **Stage 1**: **roulette_selection** , **ordered_crossover** and **heuristic_mutation**.

- **Stage 2**: **`roulette_selection`**.

Strategy 2 is Strategy 1 but using **`heuristic_mutation`**.

Strategy 3

The third strategy uses the following combination of operators in each stage:

- **Stage 1**: K-way **`tournament_selection`** with replacement, **`ordered_crossover`** and **`heuristic_mutation`**.
- **Stage 2**: K-way **`tournament_selection`** with replacement.

Strategy 4

The fourth strategy is identical to Strategy 3 but now the tournament selections are **without replacement**. Thus, it uses the following combination of operators in each stage:

- **Stage 1**: K-way **`tournament_selection`** without replacement, **`ordered_crossover`** and **`heuristic_mutation`**.
- **Stage 2**: K-way **`tournament_selection`** without replacement.

Strategy 5

The fifth strategy uses the following combination of operators in each stage:

- **Stage 1**: **`roulette_selection`**, **`ordered_crossover`** and **`heuristic_mutation`**.
- **Stage 2**: K-way **`tournament_selection`** without replacement.

Strategy 6

The sixth strategy is identical to Strategy 3 but now we change the mutation operator and we use the **`swap_mutation`** instead. Thus, it uses the following combination of operators in each stage:

- **Stage 1**: K-way **`tournament_selection`** with replacement, **`ordered_crossover`** and **`swap_mutation`**.
- **Stage 2**: K-way **`tournament_selection`** with replacement.

Strategy 7

The seventh strategy that we propose but that we have not implemented yet, is identical to Strategy 6 but now we would use **`positionbased_crossover`** instead of the already implemented **`ordered_crossover`** operator. Thus, it uses the following combination of operators in each stage:

- **Stage 1**: K-way **`tournament_selection`** with replacement, **`positionbased_crossover`** and **`swap_mutation`**.
- **Stage 2**: K-way **`tournament_selection`** with replacement.

3.5 Exploration versus exploitation

In this section we explain how the parameters of our program can modify the behaviour of the G.A. in terms of exploration and exploitation. The first term stands for travelling over all the problem's space. In our case, it refers to look at the most different variety of queen-arrangements as possible. This is interesting, if we want to find different solutions of the problem.

However, if our goal is to find a single solution for the N-queens problem, we need to be more exploitatory. This means, that we must focus on going towards a solution. We will also need to perform some exploration so as not to get stucked in a local minima. This could happen, for instance, if we find a queens rearrangement with a low scorer but which is still far away in the problem space from any of the possible solutions. Only an exploratory algorithm will be able to get rid of this situation. This is why exploration is also so important.

In our G.A. implementation there are some parameters that determine the behaviour of the algorithm in terms of exploration and exploitation. They are summarized in Table 3.1:

Table 3.1: Effect of the parameters over exploration and exploitation

Operators	**Parameters**	**Behaviour description**
General	`population_size`	The higher, (at first) the more exploratory.
General	`death_ratio`	—
General	`p_mutation`	The higher, the more exploratory.
`roulette_selection`	W	The higher, the more exploratory.
`roulette_selection`	E	The higher, the more exploitatory.
`tournament_selection`	K	The higher, the more exploitatory.
`heuristic_mutation`	λ	The higher, the more exploitatory.

First of all, it is important to start our G.A. from an enriched population. This means that our initial individuals should be uniformly and randomly distributed along all the problem space. In this way, its scope will be wide enough to look for the solutions at the vast majority of places. Then, `population_size` is related with exploration at first.

The role of `death_ratio` is more complex. It defines the number of deaths per generation. So, it is clear that the higher this parameter is, the more number of crossover and mutation operations are performed. Then, its effect over the exploration/exploitation depends on these other parameters.

Moreover, `p_mutation` (p_m) stands for the probability of a newborn to experience a mutation. Thus, the higher it is, the more exploratory we are being. However, it is worth to mention that its final behaviour will depend, of course, on the mutation operator that we are applying. For instance, the **`heuristic_mutation`** operator is more exploitatory than the **`swap_mutation`** one.

On its side, **`roulette_selection`** depends on parameters W and E. If we take a look at equation 3.2, we can see how these parameters modify the wideness of each subinterval of the roulette. On the one hand, as W increases, the size of each subinterval becomes closer to the size of the others regardless of their fitness scores. On the other hand, the power W enlarges the difference on the subinterval sizes according to their fitness score. This is why the first parameter increases the exploration and the last one increases the exploitation.

The **`tournament_selection`** operator relies on the parameter K. It stands for the number of selections to be performed at the beginning of the tournament. In this way, the higher this parameter is, the more individuals are competing in the tournament and the more exploitative we are being. This is because we are only selecting the best one among all of them.

Finally, the last parameter that can tune the exploration/exploitation behaviour is λ. It is used by the **`heuristic_mutation`** to manage the total amount of genes that are going to be permuted. As we are keeping the best individual among all the permutations, the higher the lambda is, the more exploratory the algorithm is.

4 G.A. implementation in C

All the constant variables that can be modified in our program are summarized in Table B.1 from Appendix B. They let us modify the its behaviour and can be given through command-line flags after calling the executable or can be placed in an external input file. More information can be found in the `README.md` file of the main folder of the project [1]. Here, there are some examples:

To pass some parameters as flags through command-line arguments, we can do:

```
./queens_GA.out -q 300 -p 100 -i -y 3 -m 0.5
```

Here, we are asking to find a solution with 300 queens (`-q 300`) by using a population of 100 individuals (`-p 100`), infinite generations until reaching a solution (`-i`), strategy number 3 (`-y 3`) and a mutation probability of 0.5 (`-m 0.5`).

We can also pass some parameters through an external input file as the following one.

```
N_QUEENS 200
N_POPULATION 100
P_MUTATION 0.6
INFINITE_GENERATIONS true
DENOM_POWER 10
STRATEGY 1
```

Then, execute the program with the following command:

```
./queens_GA.out -f input_file.in
```

In this case, we are asking for a solution with 200 queens (`N_QUEENS 200`) by using a population of 100 individuals (`N_POPULATION 100`), a mutation probability of 0.6 (`P_MUTATION 0.6`), infinite generations until finding a solution (`INFINITE_GENERATIONS true`), a E parameter equal to 10 (`DENOM_POWER 10`) and strategy number 1 (`STRATEGY 1`).

5 Results

In this section we will analyse the fitness convergence results of the different proposed strategies and for the cases $N = 15$ and $N = 100$. To study which strategies are better or worse, we have considered to fix the following values for some of the parameters of the problem (see Table B.1):

- `population_size` $= 20$,
- `max_generations` $= 200$ (201 including the initial population) when $N = 15$ and `max_generations` $= 15.000$ (15.001 including the initial population) when $N = 100$,
- `deaths` (ratio) $= 0.3$,
- $p_m = 0.2$.

Then we are going to consider three sets of values for (λ, K, W, E). The first one, lets call it `reference parameter values`, $(\lambda, K, W, E) = (3, 3, 0.1, 2)$ will serve us as a kind of "origin" to compare results with a more exploitatory or more exploratory set of these parameters. As `exploratory parameter values` set, we choose $(\lambda, K, W, E) = (2, 2, 20, 1)$. As `exploitatory parameter values` set we choose $(\lambda, K, W, E) = (5, 5, 0.1, 4)$. These choices are made taking into account the information that was summarized in Table 3.1.

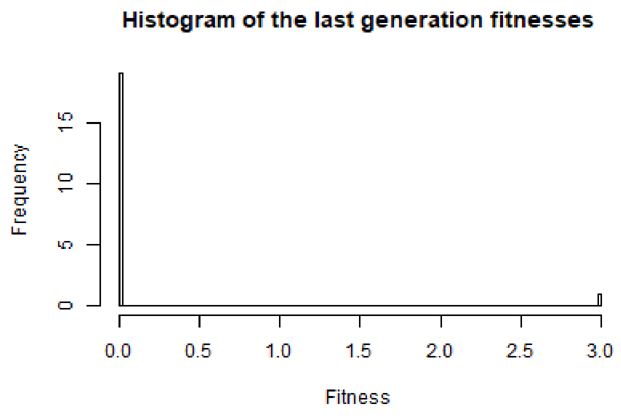
5.1 N-Queens problem, N = 15

5.1.1 Reference parameter values

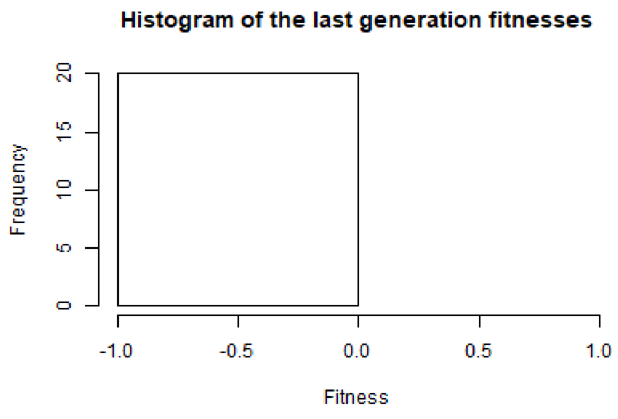
A population of individuals is said to be converged if all the genes of all individuals have converged to the same values.

A necessary condition that has to be satisfied so as to have a convergent population to the optimal solution is that all individuals of the population have the same fitness, and this fitness is the optimal one. Thus, one have to check that the histogram of fitnesses of the last population accumulates in the optimum fitness (0 for the N-queens problem). We illustrate this situation only now, for the reference parameter values when $N = 15$, but one must check this for every set of parameter values and, in addition, one should check that all the genes of all individuals have converged to the same values (since it could happen that the population converge to more than one optimal solution, but if the G.A. is well implemented it should not happen).

Lets look at the fitnesses histograms from Figure 5.1b. We can see that in the last generation, most of the individuals have optimal fitnesses but there still remain few of them that have not converged to the optimal fitness (thus in this case we say that the population have not converged with 20 generations). On the contrary, for strategies 2,..,6, all individuals share the optimal fitness and, moreover, we have checked (by locking at the genes of all individuals) that all the genes of all individuals are the same so all individuals have converged to the same optimal solution. In these cases we say that the population have converged.



(a)



(b)

Figure 5.1: Last population fitnesses histograms for the `reference parameter values`$setandN = 15$. (a) Strategy 1 (b) Strategies 2,3,4,5 and 6.

We insist on the convergence concept when working with Genetic Algorithms:

If the GA has been correctly implemented, the population will evolve over successive generations so that the fitness of the best and the average individual in each generation increases towards the global optimum [6].

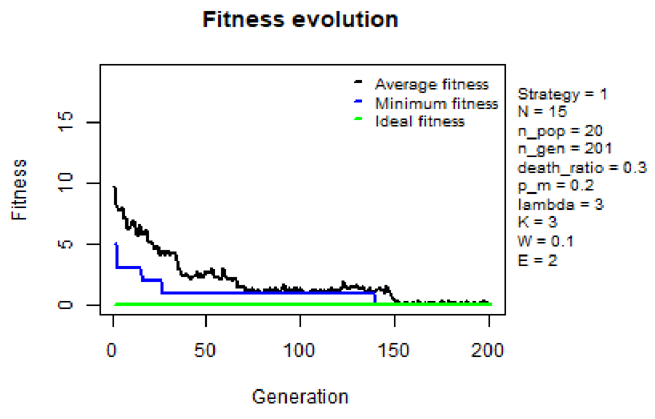
Convergence is the progression towards increasing uniformity. A gene is said to have converged when 95% *of the population share the same value. The population is said to have converged when all of the genes have converged* [6].

To check if those or similar conditions are satisfied, we must look at the distribution of genes of all the individuals of the population. We have checked in most of the cases that when a optimal individual (solution) is found by any of the algorithm strategies we propose, all the individuals of the population tend to that individual (tend to have the same genes) over generations. We leave these analyses as future work but we can find the necessary files to do so in [1]: all population genes for all generations in the simulations that we are going to discuss during the next sections are stored in `Genes*.csv` files, where * means some combination of characters that identify the simulation run corresponding to that file.

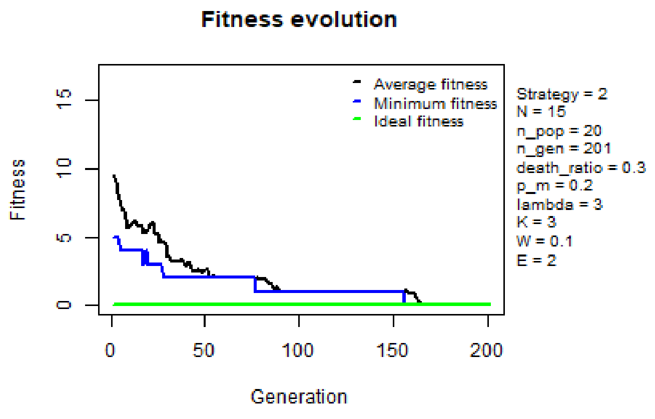
Lets now look at Figure 5.2. There we have plotted the average fitness, the optimal fitness and the optimal fitness (which we know that for the N-queens problem it is 0) as a function of the population generation. We can summarize the results as follows:

- The G.A. with Strategies 1,2 and 3 converges slower to the optimal solution. It requires over 100 generations to converge to the optimal fitness.
- The G.A. with Strategies 4,5, and 6 converge to the optimal fitness with less than 50 generations.

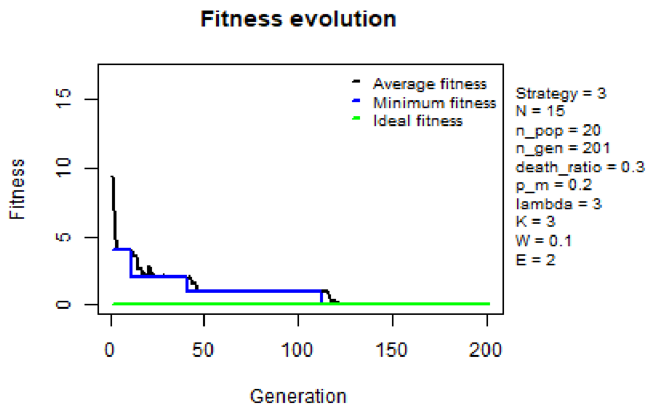
In short, from these results we could say that the best Strategies working with the `reference parameter values` set are the 4th, 5th and 6ht. A good first result is that both the best and the average fitnesses of the population tends towards the optimum over generations, regardless of the strategy.



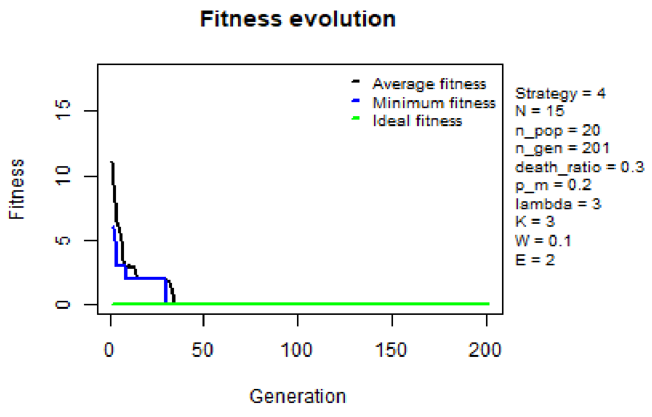
(a)



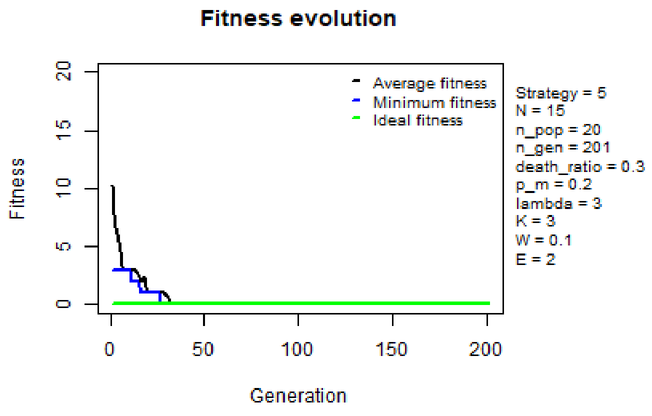
(b)



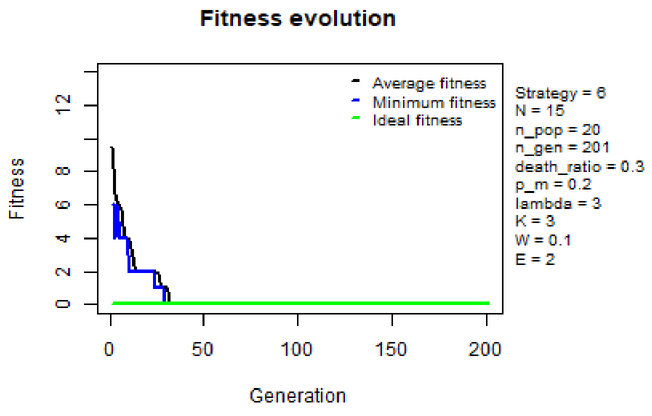
(c)



(d)



(e)



(f)

Figure 5.2: Reference parameter values simulations for $N = 15$. Strategy (a) 1 (b) 2 (c) 3 (d) 4 (e) 5 (f) 6

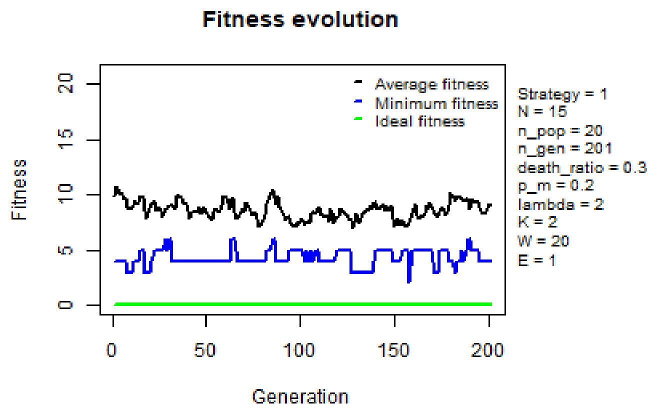
5.1.2 More exploratory parameter values

If we repeat the same analysis as did for the `reference parameter values` set but now with a more `exploratory parameter values` set we obtain the following results (see Figure 5.3):

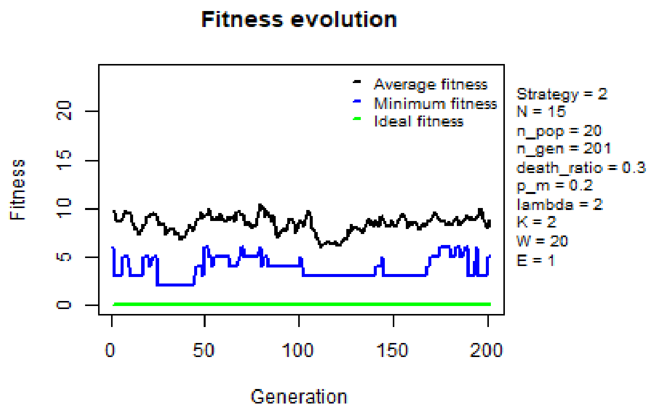
- The G.A. with Strategies 1 and 2 are now too exploratory, so it is like a random search algorithm and both the average and the best fitnesses of the population oscillate around some average values and no optimal solution is found.
- The G.A with Strategies 3, 4 and 5 is both exploratory and exploitatory, but both the average and best fitnesses of the population tend towards a local minima so that at the end the population have not converged to the optimal (ideal) solution. Thus, we can say that at the end, these strategies behave too exploitatory, although we have considered a set of more exploratory parameters (when compared to the reference parameter values from the last section).
- Finally, the G.A. with the 6th strategy is able to make the population converge to the optimal fitness in about 100 generations.

Taking into account the results discussed before for the `reference parameter values` set, and the results discussed in this section for a more `exploratory parameter values` set we can classify the proposed strategies in terms of quality as follows:

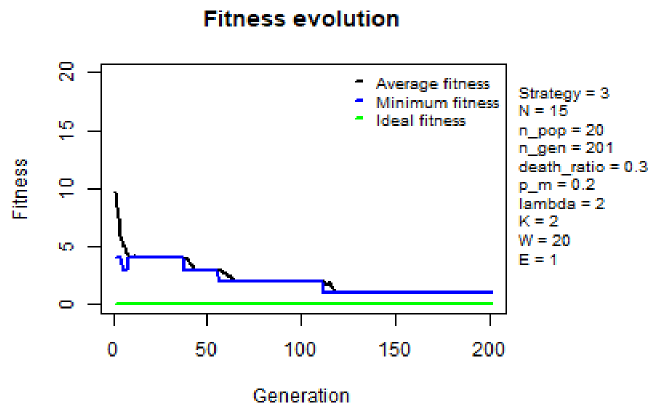
- Strategies 1 and 2 are too much exploratory, and we could say that are the worst ones.
- Strategies 3, 4 and 5 are better than the two first strategies but use to converge to a local fitness minima quite far away from the (ideal) optimal fitness that corresponds to a solution of the 15-queens problem.
- Finally, Strategy 6 is observed to be the most balanced one in terms of exploration and exploitation, so that it usually makes the population to converge to the optimal fitness, and thus to a solution of the 15-queens problem after about 100 generations.



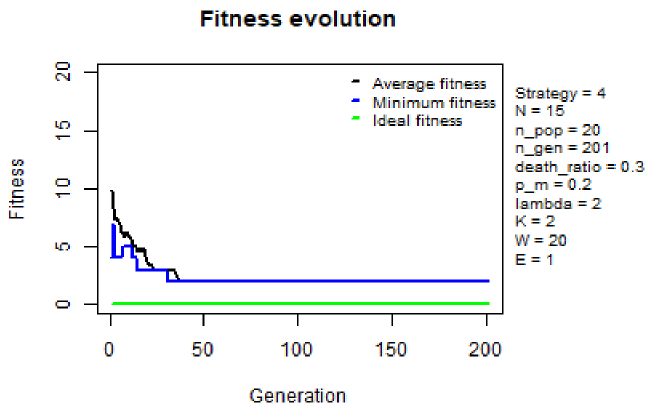
(a)



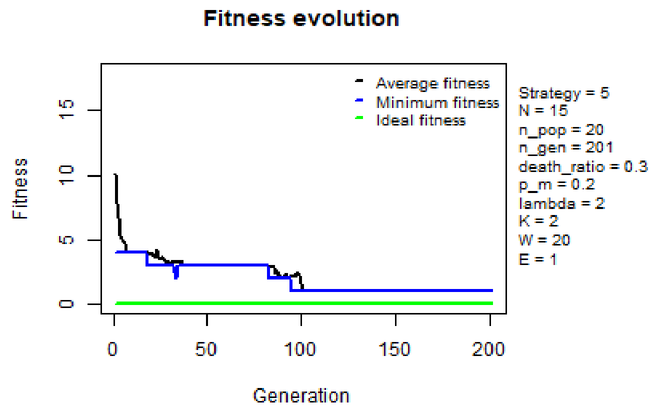
(b)



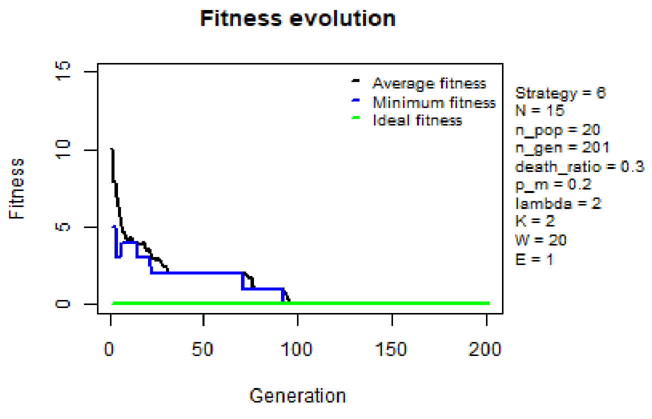
(c)



(d)



(e)



(f)

Figure 5.3: More exploratory combination of parameters. Strategy (a) 1 (b) 2 (c) 3 (d) 4 (e) 5 (f) 6

5.1.3 More exploitatory parameter values

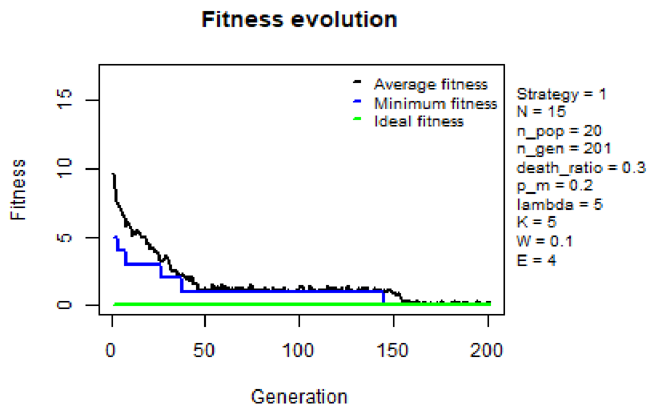
Now we repeat the same analyses as did for the `reference parameter values` and the `exploratory parameters value` sets but now with a more `exploitatory parameter values` set we obtain the following results (see Figure 5.4):

- The G.A with Strategy 1 seems to behave both exploratory and exploitatory, but both the average and best fitnesses of the population usually tend towards a local minima so that at the end the population have not converged to the optimal (ideal) solution. In the run that corresponds to Figure 5.4a, the population fitness tends to converge to the optimal fitness but behaves to exploratory so the average population fitness still oscillates a bit around the optimal (we should consider more generations).
- Finally, the G.A. with Strategies 2 to 6 are able to make the population converge to the optimal fitness in less than about 25 generations (less than 50 for the 6th, in the program run for which we are showing the results).

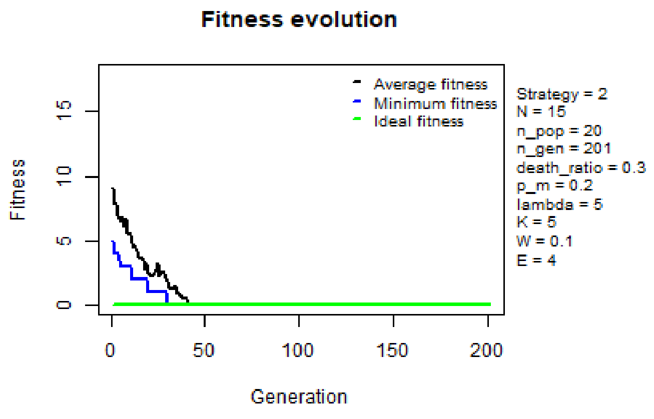
Taking into account the results discussed before for the `reference parameter values` and the more `exploratory parameter values` sets, and the results discussed in this section for a more `exploitatory parameter values` set, we can re-classify the proposed strategies in terms of quality as follows:

- Strategy 1 seems to be the most (too) exploratory one, although we have considered a more exploitatory set of values.
- The other strategies (2 to 6) seem to behave very well when the parameters are more exploitatory, and for the case $N = 15$ seem to behave equally.

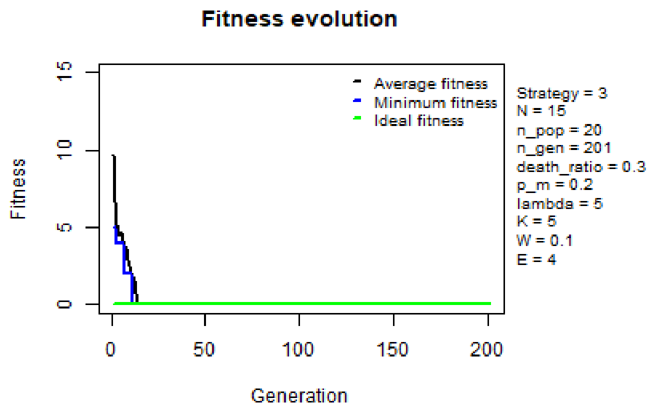
In order to see which of these strategies (2 to 6) are the best ones, we should consider a larger problem size (i.e. a larger value of N) where the balance between being exploratory and exploitatory is crucial for the G.A. to make the population converge to an optimal solution of the N-queens problem. We analyse results for the 100-queens problem in the next section.



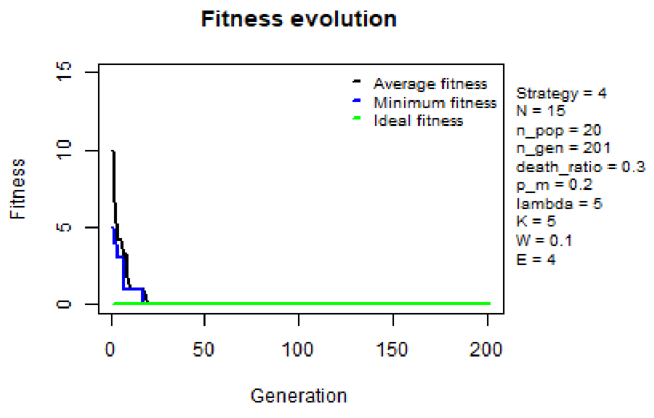
(a)



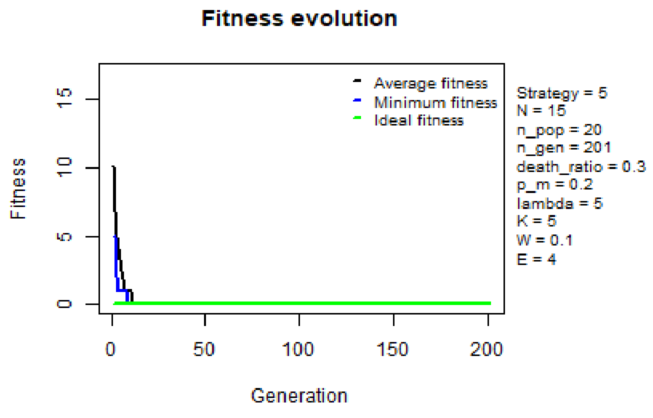
(b)



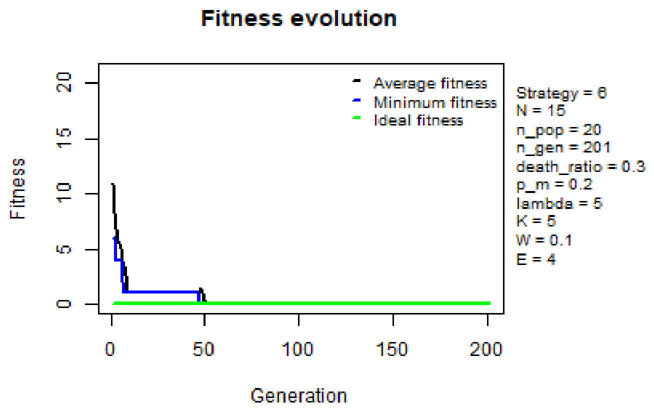
(c)



(d)



(e)



(f)

Figure 5.4: More exploitatory combination of parameters. Strategy (a) 1 (b) 2 (c) 3 (d) 4 (e) 5 (f) 6

5.2 N-Queens problem, N = 100

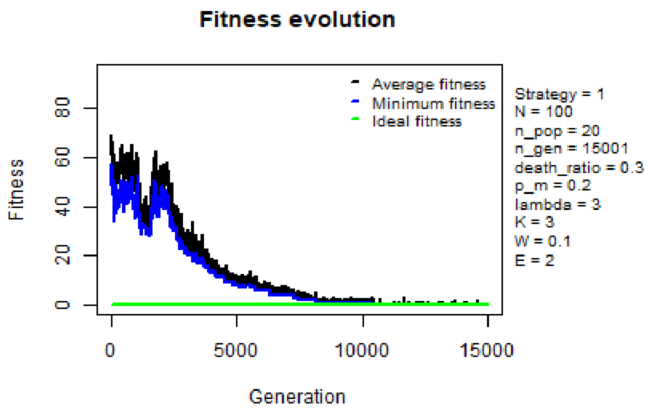
In this section we study the effectiveness of the proposed strategies for a larger problem size of the N-queens problem; $N = 100$.

We work with the same sets of reference, exploratory and exploitatory parameter values sets as we used for the case $N = 15$ but now considering a larger population size and a larger number of maximum generations.

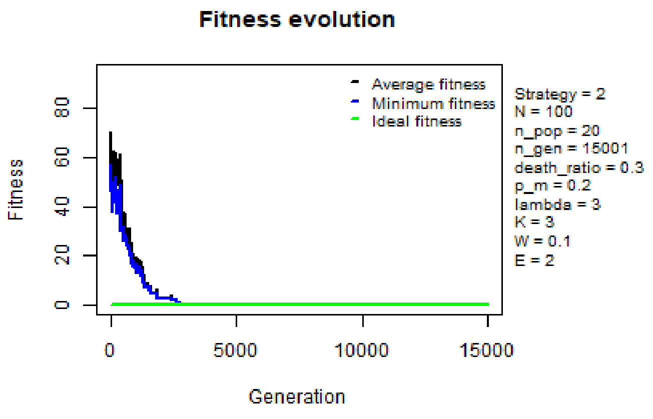
5.2.1 Reference parameter values

The results for $N = 100$ and using the `reference parameter values` set are shown in Figure 5.5. The qualitative results for all the strategies coincides exactly with those from the case $N = 15$ discussed in section 5.1.1 (see Figure 5.2).

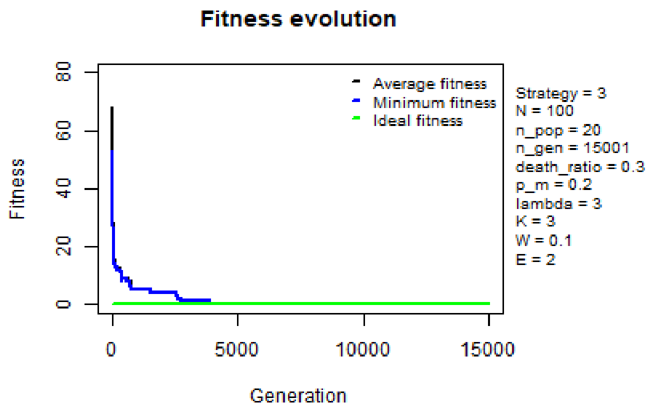
Strategy 1 seems to be the worst one and the others behave pretty equal, and in particular make the population fitness of the population to converge to the optimal solution in less than about 1000 generations.



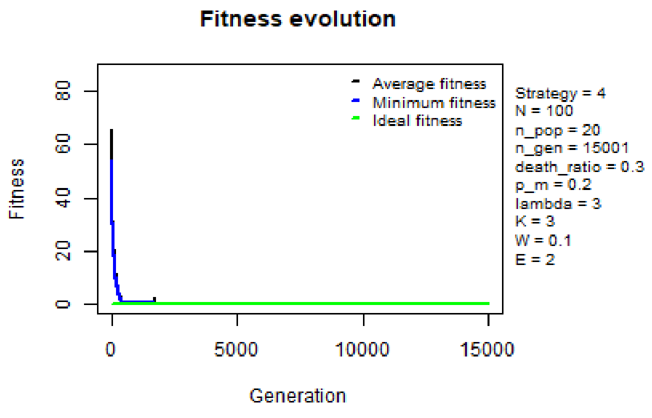
(a)



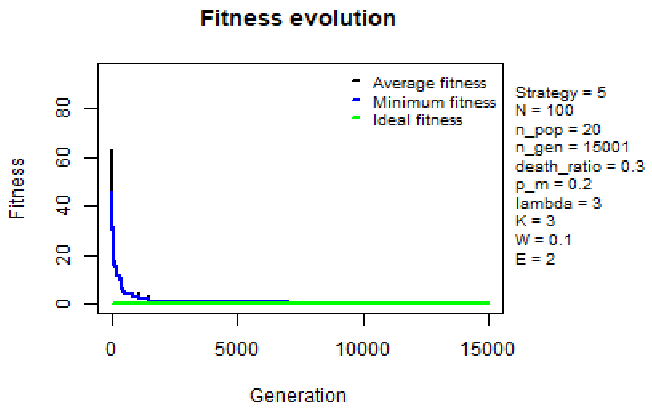
(b)



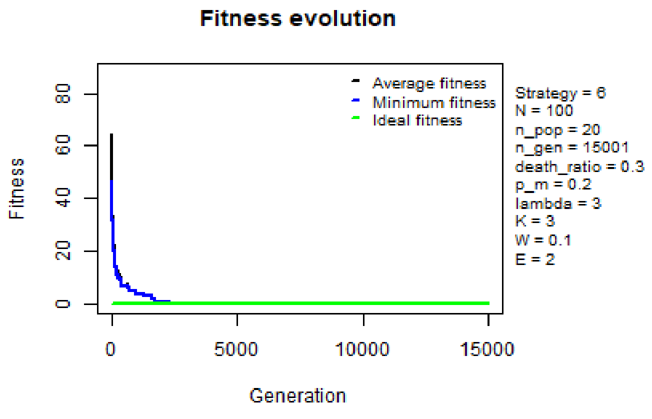
(c)



(d)



(e)



(f)

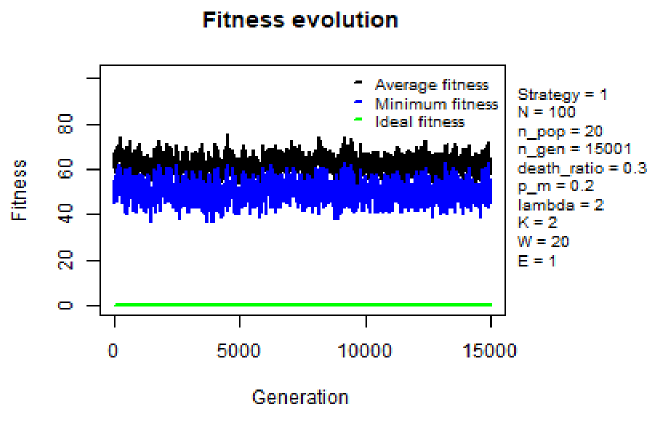
Figure 5.5: Reference parameter values simulations for $N = 100$. Strategy (a) 1 (b) 2 (c) 3 (d) 4 (e) 5 (f) 6

5.2.2 More exploratory parameter values

The results for $N = 100$ and using the more `exploratory parameter values` set are shown in Figure 5.6. Again, the qualitative results for all the strategies coincides exactly with those from the case $N = 15$ discussed in section 5.1.2 (see Figure 5.3).

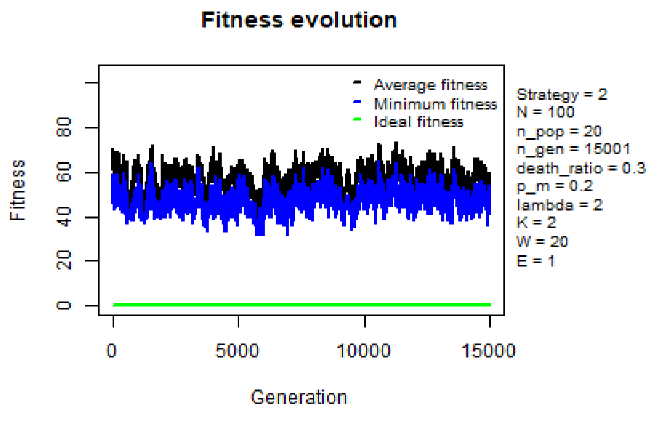
Strategies 1 and 2 are too exploratory, so we can say that working with the `exploratory parameter values` set these strategies are the worst ones.

On the contrary, strategies 3 to 6 are able to make the population fitness to converge in at most 15000 generations. Note that now there is a quite significant difference between these four strategies: Strategy 6 uses make the population to converge with less generations. Thus, if we had to choose one of these four strategies, we would consider Strategy 6.



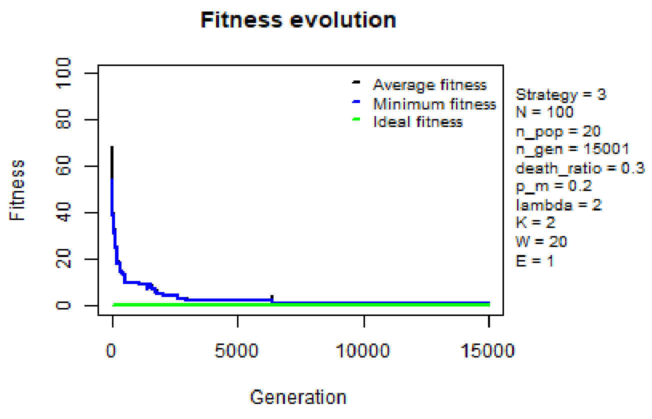
Strategy = 1
N = 100
n_pop = 20
n_gen = 15001
death_ratio = 0.3
p_m = 0.2
lambda = 2
K = 2
W = 20
E = 1

(a)



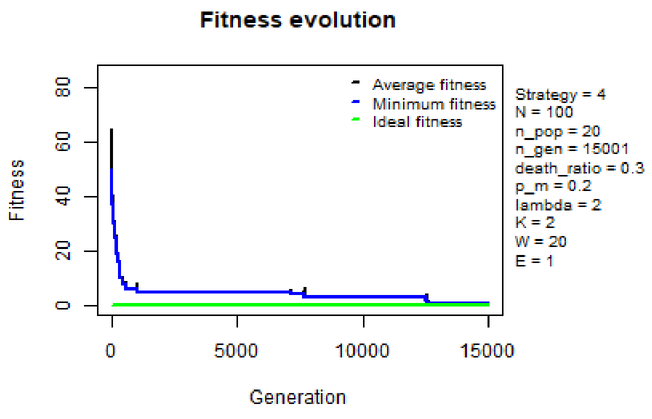
Strategy = 2
N = 100
n_pop = 20
n_gen = 15001
death_ratio = 0.3
p_m = 0.2
lambda = 2
K = 2
W = 20
E = 1

(b)



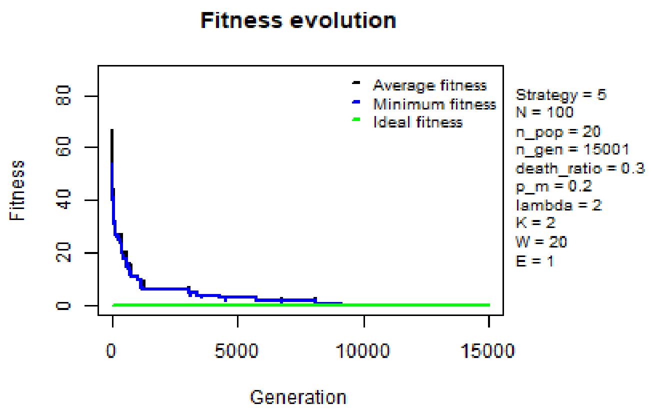
Strategy = 3
N = 100
n_pop = 20
n_gen = 15001
death_ratio = 0.3
p_m = 0.2
lambda = 2
K = 2
W = 20
E = 1

(c)



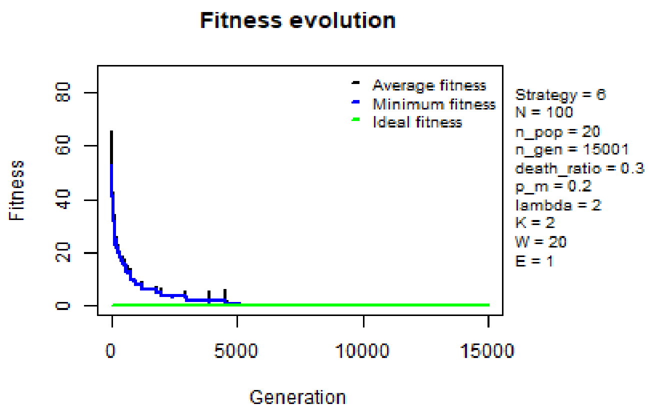
Strategy = 4
N = 100
n_pop = 20
n_gen = 15001
death_ratio = 0.3
p_m = 0.2
lambda = 2
K = 2
W = 20
E = 1

(d)



Strategy = 5
N = 100
n_pop = 20
n_gen = 15001
death_ratio = 0.3
p_m = 0.2
lambda = 2
K = 2
W = 20
E = 1

(e)



Strategy = 6
N = 100
n_pop = 20
n_gen = 15001
death_ratio = 0.3
p_m = 0.2
lambda = 2
K = 2
W = 20
E = 1

(f)

Figure 5.6: More exploratory parameter values simulations for $N = 100$. Strategy (a) 1 (b) 2 (c) 3 (d) 4 (e) 5 (f) 6

5.2.3 More exploitatory parameter values

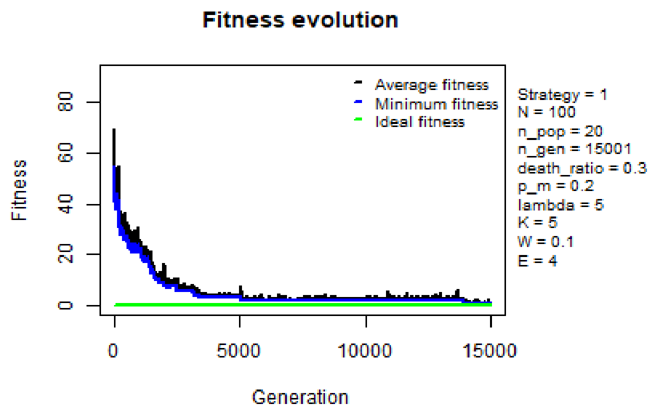
The results for $N = 100$ and using the more `exploitatory parameter values` set are shown in Figure 5.7. Again, the qualitative results for all the strategies coincides exactly with those from the case $N = 15$ discussed in section 5.1.3 (see Figure 5.4).

Strategy 1 is quite too exploratory, so we can say that working with a more `exploitatory parameter values` set this strategy is the worst one. It is not very bad since by considering a larger number of maximum generations the population will probably converge.

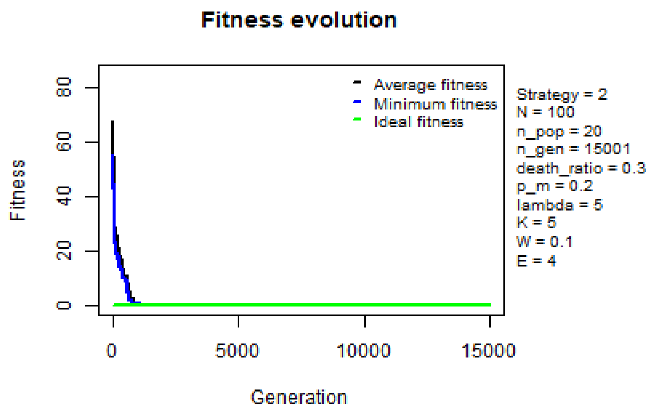
On the contrary, strategies 2 to 6 are able to make the population fitness converge in less than 1500 generations. Although it is quite hard to distinguish the differences, one could see that that now there is a quite significant difference between these five strategies: strategies 5 and 6 converge faster than strategies 2, 3 and 4.

To sum up, from all the results that we have discussed for the cases $N = 15$ and $N = 100$, we can conclude that strategies 1 and 2 are quite bad since are worse adaptive when the parameters of the problem vary. The adaptability to the variation of parameter values of Strategies 3 and 4 is quite good but in some cases the convergence rate is lower compared to that from strategies 5 and 6.

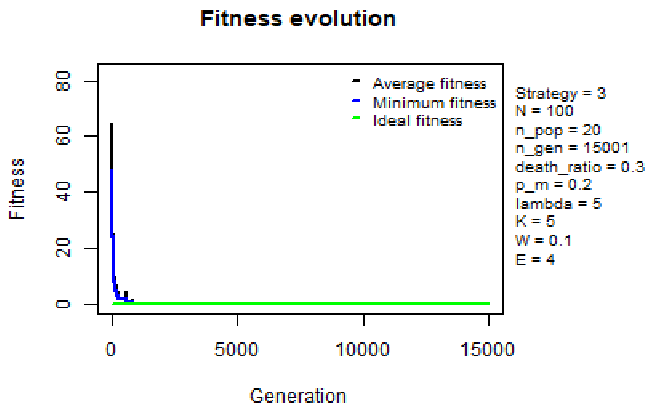
In particular, the strategy that has seen to behave better regardless of the parameter values used and the problem size (N), is Strategy 6.



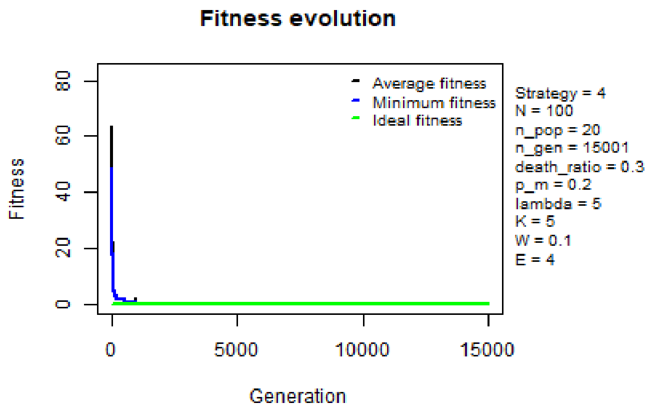
(a)



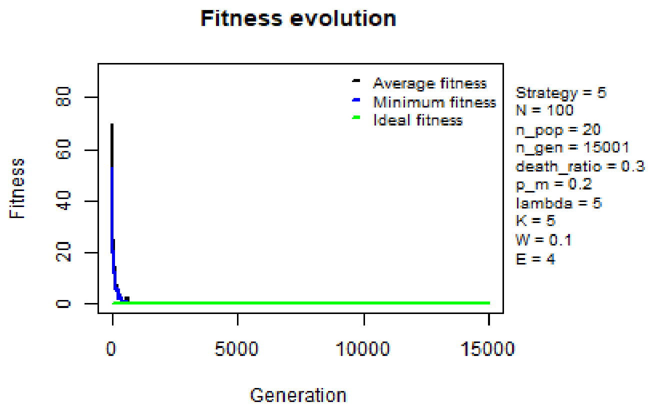
(b)



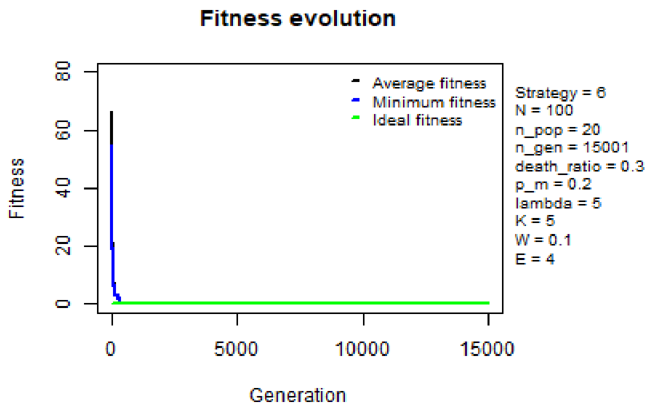
(c)



(d)



(e)



(f)

Figure 5.7: More exploitatory parameter values simulations for $N = 100$. Strategy (a) 1 (b) 2 (c) 3 (d) 4 (e) 5 (f) 6

5.3 Results in terms of the genetic operators used

We can relate the results commented in the two sections above to the quality and behaviour of the different genetic operators of selection, crossover and mutation of the strategies that we have proposed. The key point to take into account is the balance between exploration and exploitation, and the combination of operators used by a concrete G.A. strategy, which we have discussed in section 3.5.

Lets focus on our best chosen strategy which we concluded that it was Strategy 6. It makes sense that Strategy 6 is the one that has a better balance between exploration and exploitation over generations, since it uses a quite exploitative but at the same time explorative selection method as it is the **`tournament_selection`** operator, and at the same time focus on a purely explorative mutation operator as it is **`swap_mutation`**.

Although some of the other strategies we proposed are still good alternatives, they are less adaptive to the variation of the operator parameters.

5.4 Other results

In this section we include some extra work that we have done in order to expand the study of the N queens problem.

Taking our program to the limit In order to test the power of our implementation, we tried to solve the N-queens problems with a huge size. With Strategy 3 and the settings from (see [1]):

`input_examples/tournament_settings2.in`

a solution was achieved for a 2000 queens system. It demonstrates that our implementation is really powerful and can handle large systems. It needed a total of 209 minutes to find it. This 2000 queens solution is included in the Appendix C of this report.

Finding all the solutions We have implemented a program that attempts to find all the different solutions from a general N queens problem. Our implementation was proved to be able to find all the solutions for problem sizes up to 11 queens.

The name of this program is `allqueens_GA.out` and can be executed by using the following command:

```
./allqueens_GA.out -f ./input_examples/allqueens_settings.in
```

Where the flag `-f` requests parameters from the external input file called `allqueens_settings.in`. It contains a nice configuration of parameters able to easily find all the solutions.

To get these results we have used the previous Strategy 1 with a mutation probability (`p_mutation`) equal to 1 to be as much exploratory as possible. We used the same main program but with a modification on how we punctuate the individuals. Those that are solutions which were already obtained in previous steps are penalized with a high score. In this way, we get rid of all the repeated solutions and we end up obtaining all of them after several generations.

For this implementation, we have developed a variable called SIEVE that represents the amount of repeated individuals found in a row that the program tolerates before ending the loop and assuming that all the solutions have been found.

Table 5.1 summarizes the total running time that the program needed in order to find all the solutions for each system.

Table 5.1: Effect of the parameters over exploration and exploitation

Queens	**Solutions**	**Time (s)**
4	2	2
5	10	1
6	4	26
7	40	13
8	92	34
9	352	86
10	724	318
11	2680	1380

The complete list of solutions for the 8 queens problem can be found in Appendix D.

Random walk to find the solution We have also implemented an algorithm that tries to find a solution for the N queens problem by performing a random search along all the possible distributions. However, it does not look at those distributions that contain two or more queens in the same row or column. In this way, the number of possibilities is strongly lowered.

The name of this program is queens_random.out and can be executed by using the following command:

```
./queens_random.out -q 8
```

With the flag -q we can choose the number of queens for our system.

The conclusions that can be achieved with this implementation are that for small problem sizes (number of queens below 20) the random walk works as good as the G.A. This is because the problem space is not big enough to prove the goods of the G.A. Nevertheless, for bigger problem sizes the brute force of the random algorithm, which is $\mathcal{O}(N!)$, is not sufficient to find a solution in a acceptable time.

6 Conclusions

In this work we have developed a genetic algorithm able to solve the 8 queens problem. However, when we realized the real power of the genetic algorithms we wanted to apply it to solve more demanding cases. In this way, rather than applying it to solve the 8 queens problem, which a simple random walk algorithm can solve, we chose to expand the problem and solve it for a general N-queens case.

We have been able to solve the problem with many more queens than 8; we have obtained a solution for a 2000-queens problem. This demonstrates that our implementation in conjunction with the general idea of genetic algorithms are powerful tools to optimize complex systems in a very intuitive way.

We have studied different strategies to perform the genetic algorithm and we have compared the results obtained with each one of them. We know that there is work left to do regarding the optimization of the strategies, the parameters and the algorithms that are performed. It is the weakness of the genetic algorithms: there is not a unique way to achieve the results, and the parameters and operators that can work well in a system, can be completely useless in some others. This is why it is so important to analyse the fitness and the genes evolution along the generations of the program. We can use the conclusions of this data to optimize even more the parameters of the genetic algorithm.

Moreover, we decided not to get just a single solution but all the possible solutions for the N-queens problem by using the same genetic algorithm approach. However, we had to develop a more exploratory genetic algorithm with more constraints which proved to be reliable. We tested it with systems up to 11 queens and in all cases, waiting enough time, we obtained all the solutions.

This last implementation was a nice alternative study since we were forced to work with a more exploratory version of our algorithm. In this way, we had to work with a more exploitatory version of the program to solve large systems made up to 2000 queens and with a more exploratory version to find out all the possible solution of smaller systems. Besides this, we studied the effect on the fitness evolution along the generations of the program and we saw how the program changes its behaviour to either become a pure random walk or to directly go to the best solution.

This freedom and this capability to be applied to study such these heterogeneous and complex problems are one of the most relevant attributes of genetic algorithms.

7 Future work

In this section we would like to summarize some future analyses and studies that would improve the work we have done so far:

- Analyse and check the quality of the randomness of our random number generators.
- Study if the initialization of individuals performs a real uniform distribution of genes, for example by means of some histogram representations.
- Study and analyse the convergence of the population in terms of the genes that characterize every individual. Our program already prints in a file the genes of each individual in the population and for each simulated generation. The files `Genes*.csv` that can be found on our GitHub repository contain that information.
- Implement the **`positionbased_crossover`** and study possible strategies using this crossover operator.
- Study the algorithmic complexity of the implemented functions and procedures, and try to optimize them and look for computationally costless alternatives.

References

[1] Genetic-Algorithms GitHub repository, Martí Municoy & Daniel Salgado, https://github.com/martimunicoy/GeneticAlgorithms.

[2] Natural selection. (2017, December 3). In Wikipedia, The Free Encyclopedia. Retrieved 17:49, December 8, 2017, from https://en.wikipedia.org/w/index.php?title=Natural_selection&oldid=813495959.

[3] 8 queens problem, general information. https://en.wikipedia.org/wiki/Eight_queens_puzzle.

[4] Fitness proportionate selection. (2017, September 20). In Wikipedia, The Free Encyclopedia. Retrieved 00:23, December 9, 2017, from https://en.wikipedia.org/w/index.php?title=Fitness_proportionate_selection&oldid=801637463.

[5] Essentials of Metaheuristics (chapter 3), A Set of Undergraduate Lecture Notes, Sean Luke, Department of Computer Science, George Mason University.

[6] An Overview of Genetic Algorithms: Part 1, Fundamentals, David Beasley, David R. Bull, Ralph R. Martin.

[7] A list of genetic operations (crossover and mutation) http://mat.uab.cat/~alseda/MasterOpt/GeneticOperations.pdf.

[8] Solving n-Queen problem using global parallel genetic algorithm (extended abstract), Marko Božikovic, Marin Golub, Leo Budin. EUROCON 2003 Ljubljana, Slovenia.

[9] Parent selection methods, https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm

[10] Mutation methods, https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation.htm.

[11] Swap mutation Figure, http://www.wardsystems.com/manuals/genehunter/mutation_of_enumerated_chromosomes.htm.

[12] The number for different solutions of the N-queens problem, https://math.stackexchange.com/questions/1872444/how-many-solutions-are-there-to-an-n-by-n-queens-problem, http://oeis.org/A000170.

A Algorithms

A.1 Selection operators

A.1.1 Roulette wheel selection

Algorithm 6 Roulette wheel selection procedure

```
1:  procedure roulette_selection( population, extra_arguments)
2:      $P \leftarrow$ population_size
3:      $F \leftarrow \sum_{i=1}^{P} f(\mathbf{fitness}(I_i))$
4:      $r \leftarrow$ random_number([0, F))
5:      for $I \in$ population do
6:          $C_I \leftarrow \left[\sum_{i=1}^{k-1} f(\mathbf{fitness}(I_i)), \sum_{i=1}^{k} f(\mathbf{fitness}(I_i))\right)$
7:          if $r \in C_I$ then return I
8:          end if
9:      end for
10: end procedure
```

A.1.2 K-way tournament selection

Algorithm 7 K-Tournament selection procedure

```
1: procedure tournament_selection( population, K, replacement, extra_arguments)
2:     $P \leftarrow$ population_size
3:     random_index $\leftarrow \square$                                  ▷ Declare a integer index
4:     selected_individuals $\leftarrow \square$                  ▷ Initialise a empty array of individuals
5:     best_selected_individual $\leftarrow \square$; individual $\leftarrow \square$         ▷ Initialise a empty individuals
6:
7:     for ($k; 1 \leq k \leq K$) do                  ▷ Iterate until K individuals have been selected
8:         random_index $\leftarrow$ random_number($\{1, ..., P\}$)
9:         individual $\leftarrow$ population[random_index]
10:        if ( replacement = false ) then                    ▷ If the selections are with replacement
11:            if (individual not already chosen ) then
12:                individual chosen $\leftarrow$ true
13:                selected_individuals[$k$] $\leftarrow$ individual
14:                $k \leftarrow k + 1$
15:            end if
16:        else                                          ▷ If the selections are without replacement
17:            selected_individuals[$k$] $\leftarrow$ individual
18:            $k \leftarrow k + 1$
19:        end if
20:    end for
21:
22:    best_selected_individual $\leftarrow$ find_best(selected_individuals)
23:    return best_selected_individual
24: end procedure
```

A.2 Crossover operators

To be continued.

A.3 Mutation operators

A.3.1 Heuristic mutation

Algorithm 8 Heuristic mutation procedure

```
procedure heuristic_mutation( mutant, $p_m$, $\lambda$, extra_arguments )
    $p \leftarrow$ random_number((0,1))
    n_perms $\leftarrow$ factorial($\lambda$)                ▷ Initialise number of permutations
    selected_positions $\leftarrow$ ■
    genes_to_mutate $\leftarrow$ ■

    if ( $p < p_m$ ) then

        for ($i$; $1 \leq i \leq \lambda$) do
            selected_positions[i] $\leftarrow$ random_number($\{1, ..., N\}$)
            genes_to_mutate[i] $\leftarrow$ mutant.genes[selected_positions[i]]
            if (genes_to_mutate[i] = genes_to_mutate[j] for some $i \neq j$) then
                $i \leftarrow i - 1$
            end if
        end for

        permutations $\leftarrow$ □
        permutations $\leftarrow$ all_permutations(genes_to_mutate)
        neighbours $\leftarrow$ ■

        for (i; $1 \leq i \leq$ n_perms) do
            neighbours[i] $\leftarrow$ generate_neighbour(mutant, permutations[i])
        end for

        evaluate(neighbours, $\lambda$)
        mutant $\leftarrow$ find_best(neighbours)
    end if
    return mutant
end procedure
```

A.3.2 Swap mutation

Algorithm 9 Swap mutation procedure

```
1: procedure swap_mutation( mutant, $p_m$, extra_arguments )
2:     $p \leftarrow$ random_number((0,1))
3:     if ( $p < p_m$ ) then
4:         index1 $\leftarrow$ random_number($\{1, ..., N\}$)
5:         index2 $\leftarrow$ index1
6:         while ( index1= index2 ) do
7:             index2 $\leftarrow$ random_number($\{1, ..., N\}$)
8:         end while
9:
10:        tmp $\leftarrow$ mutant.genes[index1]
11:        mutant.genes[index1] $\leftarrow$ mutant.genes[index2]
12:        mutant.genes[index2] $\leftarrow$ tmp
13:    end if
14:    return mutant
15: end procedure
```

A.4 Strategies

Algorithm 10 G.A. Strategy 1

```
1: procedure genetic_algorithm_strategy1 (population, nextpopulation, best, parent1, parent2, child, survivor, extra_arguments)
2:     generation ← 1
3:     while generation ≤ max_generations do
4:         for $1 \leq i \leq$ deaths do
5:             parent1 ← roulette_selection(population, roulette_selection_arguments)
6:             parent2 ← roulette_selection(population, roulette_selection_arguments)
7:             child ← ordered_crossover(parent1, parent2, ordered_crossover_arguments)
8:             child ← swap_mutation(child, $p_m$, swap_mutation_arguments)
9:             nextpopulation[i] ← child
10:        end for
11:
12:        for deaths $< i \leq$ population_size do
13:            survivor ← roulette_selection(population, roulette_selection_arguments)
14:            nextpopulation[i] ← survivor
15:        end for
16:
17:        swap_roles (population, nextpopulation)
18:        evaluate(population, extra_arguments)
19:        best ← find_best(population, extra_arguments)
20:        generation ← generation + 1
21:    end while
22:    return best                ▷ Return a best individual from the last generation of individuals
23: end procedure
```

Algorithm 11 G.A. Strategy 2

```
1: procedure genetic_algorithm_strategy2 (population, nextpopulation, best, parent1, parent2, child, survivor, extra_arguments)
2:     generation ← 1
3:     while generation ≤ max_generations  do
4:
5:         for  $1 \le i \le$ deaths do
6:             parent1 ← roulette_selection(population, roulette_selection_arguments)
7:             parent2 ← roulette_selection(population, roulette_selection_arguments)
8:             child ← ordered_crossover(parent1, parent2, ordered_crossover_arguments)
9:             child ← heuristic_mutation(child, $p_m$, $\lambda$, heuristic_mutation_arguments)
10:            nextpopulation[i] ← child
11:        end for
12:
13:        for  deaths $< i \le$ population_size do
14:            survivor ← roulette_selection(population, roulette_selection_arguments)
15:            nextpopulation[i] ← survivor
16:        end for
17:
18:        swap_roles (population, nextpopulation)
19:        evaluate(population, extra_arguments)
20:        best ← find_best(population, extra_arguments)
21:        generation ← generation + 1
22:    end while
23:
24:    return best                ▷ Return a best individual from the last generation of individuals
25: end procedure
```

Algorithm 12 G.A. Strategy 3

```
1: procedure genetic_algorithm_strategy3 (population, nextpopulation, best, parent1, parent2, child, survivor, extra_arguments)
2:     generation ← 1
3:     while generation ≤ max_generations do
4:
5:         for 1 ≤ i ≤ deaths do
6:             parent1 ← tournament_selection(population, K, replacement = true, tournament_selection_arguments)
7:             parent2 ← tournament_selection(population, K, replacement = true, tournament_selection_arguments)
8:             child ← ordered_crossover(parent1, parent2, ordered_crossover_arguments)
9:             child ← heuristic_mutation(child, $p_m$, $\lambda$, heuristic_mutation_arguments)
10:            nextpopulation[i] ← child
11:        end for
12:
13:        for deaths < i ≤ population_size do
14:            survivor ← tournament_selection(population, K, replacement = true, tournament_selection_arguments)
15:            nextpopulation[i] ← survivor
16:        end for
17:
18:        swap_roles (population, nextpopulation)
19:        evaluate(population, extra_arguments)
20:        best ← find_best(population, extra_arguments)
21:        generation ← generation + 1
22:    end while
23:
24:    return best                ▷ Return a best individual from the last generation of individuals
25: end procedure
```

Algorithm 13 G.A. Strategy 4

```
1: procedure genetic_algorithm_strategy4 (population, nextpopulation, best, parent1, parent2, child, survivor, extra_arguments)
2:     generation ← 1
3:     while generation ≤ max_generations do
4:
5:         Identical to Algorithm 12 by setting replacement = false in the tournament_selection.
6:     end while
7:
8:     return best                    ▷ Return a best individual from the last generation of individuals
9: end procedure
```

Algorithm 14 G.A. Strategy 5

```
 1: procedure genetic_algorithm_strategy5 (population, nextpopulation, best, parent1, parent2, child, survivor, extra_arguments)
 2:     generation ← 1
 3:     while generation ≤ max_generations  do
 4:
 5:         for  $1 \leq i \leq$ deaths do
 6:             parent1 ← roulette_selection(population, roulette_selection_arguments)
 7:             parent2 ← roulette_selection(population, roulette_selection_arguments)
 8:             child ← ordered_crossover(parent1, parent2, ordered_crossover_arguments)
 9:             child ← heuristic_mutation(child, $p_m$, $\lambda$, heuristic_mutation_arguments)
10:             nextpopulation[i] ← child
11:         end for
12:
13:         for  deaths $< i \leq$ population_size do
14:             survivor ← tournament_selection(population, K, replacement = false, tournament_selection_arguments)
15:             nextpopulation[i] ← survivor
16:         end for
17:
18:         swap_roles (population, nextpopulation)
19:         evaluate(population, extra_arguments)
20:         best ← find_best(population, extra_arguments)
21:         generation ← generation + 1
22:     end while
23:
24:     return best                ▷ Return a best individual from the last generation of individuals
25: end procedure
```

Algorithm 15 G.A. Strategy 6

```
procedure genetic_algorithm_strategy6 (population, nextpopulation, best, parent1, parent2, child, survivor, extra_arguments)
    generation ← 1
    while generation ≤ max_generations do

        for 1 ≤ i ≤ deaths do
            parent1 ← tournament_selection(population, K, replacement = true, tournament_selection_arguments)
            parent2 ← tournament_selection(population, K, replacement = true, tournament_selection_arguments)
            child ← ordered_crossover(parent1, parent2, ordered_crossover_arguments)
            child ← swap_mutation(child, $p_m$, swap_mutation_arguments)
            nextpopulation[i] ← child
        end for

        for deaths < i ≤ population_size do
            survivor ← tournament_selection(population, K, replacement = true, tournament_selection_arguments)
            nextpopulation[i] ← survivor
        end for

        swap_roles (population, nextpopulation)
        evaluate(population, extra_arguments)
        best ← find_best(population, extra_arguments)
        generation ← generation + 1
    end while

    return best                    ▷ Return a best individual from the last generation of individuals
end procedure
```

Algorithm 16 G.A. Strategy 7

```
procedure genetic_algorithm_strategy7 (population, nextpopulation, best, parent1, parent2, child, survivor, extra_arguments)
    generation ← 1
    while generation ≤ max_generations do

        for 1 ≤ i ≤ deaths do
            parent1 ← tournament_selection(population, K, replacement = true, tournament_selection_arguments)
            parent2 ← tournament_selection(population, K, replacement = true, tournament_selection_arguments)
            child ← positionbased_crossover(parent1, parent2, positionbased_crossover_arguments)
            child ← swap_mutation(child, $p_m$, swap_mutation_arguments)
            nextpopulation[i] ← child
        end for

        for deaths < i ≤ population_size do
            survivor ← tournament_selection(population, K, replacement = true, tournament_selection_arguments)
            nextpopulation[i] ← survivor
        end for

        swap_roles (population, nextpopulation)
        evaluate(population, extra_arguments)
        best ← find_best(population, extra_arguments)
        generation ← generation + 1
    end while

    return best                ▷ Return a best individual from the last generation of individuals
end procedure
```

B Table of parameter constants of our G.A. program

Table B.1

Constant variable	**Nomenclature**	**Description**	**Flag**	**Range**
N_QUEENS	N	Number of queens	-q	$\mathbb{N}\backslash\{2,3\}$
N_POPULATION	n_pop	Population size	-p	$\mathbb{N}\backslash\{0\}$
N_GENERATIONS	max_generations	Maximum number of generations	-g	$\mathbb{N}\backslash\{0\}$
DEATH_RATIO	$\frac{\texttt{deaths}}{\texttt{population\_size}}$	Fraction of deaths	-d	$(0,1)$
P_MUTATION	p_m	Probability of mutation	-m	$(0,1)$
LAMBDA	λ	heuristic mutation parameter	-l	$\mathbb{N}, \leq$ n_pop
FORCE_TO_CONTINUE	—	Iterate until max_generations	-c	boolean
INFINITE_GENERATIONS	—	Iterate until finding solution	-i	boolean
WRITE_FITNESS	—	Write fitnesses to file	-r	boolean
MAX_FITNESS_POINTS	—	Maximum fitness points to write	-x	$\mathbb{N}$
WRITE_GENES	—	Write genes to file	-n	boolean
MAX_GENES_POINTS	—	Maximum genes points to write	-a	$\mathbb{N}$
SUMMARIZE_FREQ	—	Frequency to show GA summary	-s	$\mathbb{N}$
TOURNAMENT_SELECTIONS	K	K-way tournament selections	-t	$1 \leq K \leq$ n_pop
FRAC_WEIGTH	W	See equation (3.2)	-w	$\mathbb{R}^+$
DENOM_POWER	E	See equation (3.2)	-e	$\mathbb{R}^+$
STRATEGY	$i \in \{1,2,...,6\}$	G.A. Strategy	-y	$\{1,2,...,6\}$
SIEVE	S	Sieve to stop allqueens_GA.out	-v	$\mathbb{N}$
FITNESS_DIR	—	Fitness file directory	-b	—
GENES_DIR	—	Genes file directory	-k	—

C Some solutions for the N-queens problem

C.1 N= 8

```
8 . . . . X . . .
7 . X . . . . . .
6 . . . X . . . .
5 . . . . . X . .
4 . . . . . . . X
3 . . X . . . . .
2 X . . . . . . .
1 . . . . . . X .
  A B C D E F G H
```

```
================================================================================

 - Solution:

 (    2,     7,     3,     6,     8,     5,     1,     4)

================================================================================
```

C.2 N= 10

```
10 . . . . . . . X . .
 9 . . . . X . . . . .
 8 . X . . . . . . . .
 7 . . . . . . . . X .
 6 . . . . . . X . . .
 5 . . . X . . . . . .
 4 X . . . . . . . . .
 3 . . X . . . . . . .
 2 . . . . . X . . . .
 1 . . . . . . . . . X
   A B C D E F G H I J
```

```
================================================================================

 - Solution:

 (    4,     8,     3,     5,     9,     2,     6,    10,     7,     1)

================================================================================
```

C.3 N= 50

```
50 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . .
49 . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . .
48 . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
47 . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
46 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X .
45 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . .
44 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . .
43 . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . .
42 . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
41 . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
40 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . .
39 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . .
38 X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
37 . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
36 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . .
35 . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
34 . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . .
33 . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
32 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . .
31 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . .
30 . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
29 . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
28 . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
27 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X
26 . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
25 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . .
24 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . .
23 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . .
22 . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . .
21 . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
20 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . .
19 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . .
18 . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
17 . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
16 . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . .
15 . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
14 . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . .
13 . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
12 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . .
11 . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
10 . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . .
 9 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . .
 8 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . .
```

```
7 . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
6 . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . .
4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . .
3 . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . .
2 . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . . . . . . .
  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z A B C D E F G H I J K L M N O P Q R S T U V W X
  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```
==============================================================================

 - Solution:

 (  38,   17,   41,    7,   13,   26,    6,   29,   42,   15,   47,   18,   37,
    33,   28,   30,   35,    2,   21,   11,   48,   14,   49,    3,   34,   22,
    43,   10,   16,    4,   25,   39,   31,    1,   45,    5,   12,   23,   50,
    44,   24,   20,   36,    9,   40,   32,    8,   19,   46,   27)

==============================================================================
```

C.4 N= 100

```
==============================================================================

 - Solution:

 (  44,   50,   61,   88,   92,   71,   34,   99,   48,   84,   72,   41,   60,
    96,   27,   68,   59,   38,   36,   15,   19,    8,   55,   77,   46,   23,
    82,    6,   86,   56,   85,   67,   93,   28,    3,   70,   33,   83,   43,
    62,   81,   51,   57,   95,   29,   90,   65,   17,   80,   13,   24,   16,
    94,    4,   14,   75,    9,   91,   58,   97,   53,   10,   42,   21,   30,
    74,   12,   40,   49,   26,   20,   73,   78,   89,  100,   66,   79,   63,
     2,   47,    7,   64,   37,    5,   18,   25,   11,   35,    1,   45,   52,
    32,   39,   31,   98,   54,   22,   87,   69,   76)

==============================================================================
```

C.5 N= 200

```
==============================================================================

 - Solution:

 ( 124,   44,   97,   82,  122,  146,   42,  182,   93,   50,   60,  165,  128,
   75,  119,  166,    6,  198,   38,   34,  194,   19,   16,  172,  196,    3,
  178,  125,   53,  109,  167,  108,  153,   32,   39,   94,   31,   47,  155,
   71,   91,   23,   88,  175,  126,   63,   20,  117,   79,  157,   98,  120,
  136,  189,  111,  195,  181,   21,  132,   99,  127,   58,  197,  149,   77,
   37,   11,  143,   36,  179,   96,   64,   18,   30,  101,   92,  164,  176,
  193,  100,  118,   33,  104,   59,  188,  115,  159,    9,   29,    8,  148,
   70,   17,  168,   86,  151,    4,   14,   56,   84,   69,  138,   35,   54,
   24,    2,   68,  170,   12,  187,   81,    1,  180,  177,  156,  154,   43,
  162,   85,  110,   48,  173,   25,  103,   26,  139,  134,  131,   15,   78,
  158,   13,  102,   83,  123,   28,   95,  190,   22,   57,   61,    5,   40,
   80,  105,  141,  169,  150,   74,  161,   27,   66,  186,  160,  163,    7,
   52,  130,  133,  183,  107,  144,  191,  140,  145,   76,   67,  200,  121,
  185,   55,  114,  112,   46,   10,  106,  142,   51,   73,   89,  199,   62,
  137,   72,  116,   45,  192,  171,  174,  113,   90,  129,  152,  147,   41,
   49,   87,  184,   65,  135)

================================================================================
```

C.6 N= 500

```
================================================================================

 - Solution:

 ( 215,  327,  198,  175,  298,  129,  261,  199,  406,   83,   60,  260,  247,
   155,   20,  457,  323,  140,  459,  279,   64,  208,  385,  373,  177,  430,
   127,  331,  317,  306,  184,  313,  391,  241,   16,   94,  321,  427,  303,
   174,    6,  370,  156,  286,  439,  289,   67,  193,  137,  409,  132,  435,
   486,  224,  166,  223,  477,  148,  341,  333,  225,  245,  257,  381,   53,
    35,   79,   34,  280,  249,  338,  473,  352,  396,  121,   74,   36,   32,
   420,  181,   38,   66,   12,  342,  332,  478,  410,  464,  160,  358,  498,
   483,  250,  218,  310,  281,   10,  100,  390,   43,  367,  111,  403,  107,
    80,  117,  221,  311,   28,  400,  240,  278,   18,   13,  119,  259,  135,
     3,  394,    8,  297,  285,  359,  355,  371,   99,  496,  349,  465,  112,
   114,  253,  330,  272,  412,  227,   68,  178,  130,   50,  397,  322,  369,
   475,  497,  438,   71,  411,  203,  445,  125,  401,  147,  234,  169,  233,
   106,   15,  399,  194,  343,  212,  201,   23,  354,  228,  219,  466,  329,
   432,  470,  189,  153,  402,  487,  481,  455,  293,  216,  440,  346,  350,
   444,  273,  316,  243,  405,  300,   65,   61,  256,  404,   29,  162,  368,
    88,  499,  113,  452,  283,  236,   63,  170,   51,  229,  447,   89,   25,
    84,  326,  450,  288,    4,  485,  145,  364,  345,   55,  172,  265,  377,
     7,  433,  462,  448,   48,  110,  207,  252,  299,   59,  449,  165,  294,
   251,  134,  356,  270,  109,   73,  492,  324,  347,  469,  222,  366,   54,
   269,  191,  118,  102,  456,  131,  182,  463,   49,    9,   22,  168,  195,
    24,   72,  453,  419,  188,  372,  339,   87,  206,   69,  458,  159,   45,
   275,  393,  460,  407,  142,  480,   70,   42,  301,  418,  276,  491,  360,
   335,  421,  185,  398,  287,  318,  468,    1,  157,   82,  471,  123,  305,
   180,  213,  442,  302,  266,   97,   81,  386,   75,  376,  413,   46,  348,
   467,  255,  357,  384,   14,  490,   57,  328,   44,  122,  284,  408,   76,
   262,   17,  246,  192,  128,  237,  379,  374,  484,  474,  434,  105,  500,
   414,   26,  149,   86,   19,  235,  437,  103,  190,  443,   62,  116,  493,
   451,   90,  271,  362,  200,  183,  380,   40,  171,  351,   92,  146,  489,
   337,  139,  248,  108,  320,  395,  186,  120,  428,  290,  295,  242,  244,
   152,   37,  431,  425,  353,   58,  494,   77,   52,   85,  291,  202,  136,
   378,   41,   30,  387,  454,  423,  231,  429,  495,  314,  150,  144,   11,
   417,   56,  210,  422,  282,  479,  344,  267,  309,  154,  325,  476,   78,
   220,  264,  312,  365,  104,   47,  101,  211,  214,  115,    5,    2,   31,
   173,  461,  141,  383,  263,  126,  336,  258,  340,   96,  196,  158,  315,
   482,  296,  163,  217,  124,  232,  307,  274,  204,  197,  361,  179,  382,
   238,  292,  151,  239,  488,  446,  334,  308,  176,  161,  392,  187,  389,
   319,  416,  143,  205,  363,  268,   27,  424,   39,  209,   93,  230,  388,
    21,  472,  436,  167,  304,   33,  426,  254,   95,  133,  138,  415,  164,
    91,  277,  441,  375,   98,  226)

=====================================================================
```

C.7 N= 1000

```
=====================================================================

 - Solution:

 ( 506,  175,  767,  440,  364,   90,  276,  366,  815,  512,  844,   39,   33,
   194,  671,  511,   88,   16,  523,  256,  873,  861,  235,  978,  410,  251,
   442,  461,  683,  258,  911,  754,  540,  982,  310,  343,  698,  290,  404,
   586,  390,  151,  241,  135,  963,  968,  481,  445,  651,  869,  954,  678,
   571,  518,  249,  499,  924,  237,  332,  465,  771,  482,   55,  450,  709,
   580,  850,  747,  628,  618,  979,  271,  383,  556,  750,  427,  627,  931,
   595,  240,  795,  550,  346,  494,  872,  402,  631,  700,  381,  503,  224,
   157,  730,  722,  242,  351,  280,  833,  524,   63,  362,   17,  803,  504,
   864,  581,  696,  162,  184,  912,  328,  661,  285,   52,  436,   34,   42,
   657,  884,  199,  715,  216,  490,  938,  997,  460,  521,  599,  246,  462,
   444,  113,   67,  876,  243,  231,  205,  302,   75,  471,  365,  629,  314,
    98,  261,  219,   64,  927,   36,  408,  862,  553,  193,  792,  980,  555,
   342,   58,  737,  191,  144,  321,  642,  655,  306,  719,  263,  733,  495,
   160,  569,  649,  847,  394,   30,  526,  764,  326,  917,  611,  701,  812,
   790,  800,  226,  621,  137,  248,  732,  532,  959,  546,  337,  238,  405,
   681,  284,  347,  895,  209,  325,  549,  575,  418,  697,  252,  341,  102,
   164,  107,  353,  147,  103,  932,  838,  594,  610,  529,  171,  742,  587,
   430,  724,  576,  995,  535,  958,   99,  636,  590,   15,   43,  201,  824,
   438,  805,  222,  542,  904,  602,  173,  630,  823,  573,  667,  265,  100,
   981,  468,  875,  854,   87,  936,  296,  327,  951,   40,  396,  947,  773,
   295,  770,  138,  502,  426,  315,  476,  431,   97,  564,  525,  355,   25,
   658,   78,  900,  464,  682,  619,   77,  469,   47,  294,  929,  233,  451,
   974,  691,  855,  953,  134,  559,  269,  133,  202,  517,  785,   56,  317,
    93,  492,  520,  633,  695,  802,  185,  429,  606,  910,   72,  971,  208,
   646,  182,  152,  921,  832,  496,  433,  843,  913,  605,   95,  992,  514,
   146,  967,  612,  784,  786,  666,  498,   66,   24,   28,  313,  835,  615,
   116,  293,  354,   29,  174,  154,  787,  728,  163,  272,  781,  637,  813,
   852,  920,  899,  673,  336,  480,  300,  119,  885,  307,  665,  839,  507,
   388,  139,  996,  857,  926,  554,  497,   23,   27,  689,  349,  804,  908,
   467,  158,  204,  942,  475,  380,  828,  848,   89,  943,  731,  247,  670,
   452,  762,  741,  220,  373,  259,  845,  772,  229,   84,  983,  217,  609,
   930,   57,  558,  391,  788,  986,  922,   83, 1000,  297,  907,  334,  752,
   806,  883,  530,  739,  842,  859,  305,  322,  132,  676,   94,  886,  841,
    20,  244,  544,  188,   51,  215,  363,  820,    8,   48,  125,  447,  562,
   881,  161,  969,  975,    7,  598,  392,  168,  454,  987,  527,  990,   74,
   148,  966,  680,  324,  181,   85,   70,  250,  890,  955,  299,  811,  538,
   213,  572,  759,  799,  583,  425,  836,  718,   32,  989,  705,  275,  121,
   108,  638,  470,  110,  902,  372,  906,  448,  863,  868,   10,  662,  415,
   115,  264,  279,  794,    9,  557,  915,  858,  654,   12,  985,  740,  169,
   352,  360,  143,  359,  378,  652,  386,  211,   53,  239,  493,  756,   68,
   407,    1,  829,   37,  707,  603,  262,  874,  745,  519,  721,  780,  853,
   228,  536,    3,  101,  387,  273,  865,  153,  547,  560,  118,  304,  515,
   379,    4,  877,  601,  312,  298,   41,  274,  130,  150,  172,  288,  122,
   675,  331,  180,  834,  644,  270,  937,  946,  867,  801,  617,  640,  896,
   311,  776,  212,  489,  774,  807,  227,  510,  491,  356,  704,  660,  851,
   218,  409,  870,  267,  935,  357,  539,  600,   46,   50,  766,  457,  939,
   513,  339,  977,   91,    6,  537,  128,  882,  190,  393,  335,  167,  484,
   710,  260,  622,   81,  195,  669,   38,  420,  827,  645,  370,  708,  623,
   692,  292,  165,  505,  735,   14,  178,  892,  905,  333,  888,  257,  625,
   320,  437,  127,  382,  176,  817,  702,  289,   69,  592,  345,  928,  753,
   866,  624,  690,  140,  543,  455,  653,  825,  976,  192,  206,  819,  375,
   223,  369,  385,  472,  821,  189,  970,  301,  428,  916,  860,  765,  789,
   340,  925,   65,  486,  528,   26,  914,  933,  477,  878,  779,  713,  286,
   960,   73,  596,  998,  488,  891,  419,   61,   22,  659,  893,  487,  717,
   769,  522,  614,  197,  278,  903,  145,  749,  186,   60,  593,  395,   54,
   949,  579,  177,  919,  111,  348,  131,  725,  424,  456,  361,  478,  984,
   791,  934,  871,  635,  763,    5,  664,  406,  972,  106,  453,  141,  720,
   798,  687,  561,  416,   13,  901,  993,  277,  961,  124,  516,  999,  236,
   563,  358,  443,  230,  809,  413,  814,  421,  117,  500,  639,  412,   71,
   991,  268,  485,   11,  196,  760,  856,   86,  441,  693,  734,  613,  616,
   626,  225,  318,  179,  401,  129,   79,   45,  585,  783,  909,  808,  822,
   303,  183,  679,  234,  458,  837,  156,  368,  793,  344,  316,  778,  952,
   384,  736,  126,  712,   96,  650,  397,  684,  283,  782,  198,  400,  944,
   643,  112,  957,  255,   80,  796,  685,  964,  149,  508,  761,  620,  641,
   578,  214,  210,  994,   76,  758,  253,  142,  940,  768,  962,  308,  120,
    18,  672,  830,  291,  706,  604,  988,  187,  459,  744,  797,  748,  411,
   565,  727,  831,  726,  439,  474,  567,  591,  203,  584,  155,  309,  723,
   777,   35,  551,  897,  574,  109,  207,  846,  548,  738,  254,  918,  374,
   879,  688,  887,  663,  136,  694,  435,  399,  656,  898,  632,  281,  607,
   647,   62,  338,  403,   21,  826,  114,  367,   49,  674,  319,  350,  566,
   422,  945,  941,  463,  816,  432,  746,  880,  329,  956,  634,  449,  200,
   509,  716,  389,  479,  287,  894,  757,  266,   19,  775,  159,  232,  668,
   221,  531,  570,  166,   59,  434,  743,   92,  501,  589,   82,  755,  376,
     2,  608,   31,  703,  577,  965,  473,  123,  398,  282,  104,  541,  568,
   597,  414,  648,  534,  330,  677,  371,  751,  582,  818,  545,  923,  533,
   446,   44,  973,  840,  849,  105,  711,  466,  948,  686,  170,  552,  588,
   729,  714,  699,  423,  950,  889,  245,  810,  323,  377,  483,  417)

================================================================================
```

C.8 N= 2000

```
================================================================================

 - Solution:

 ( 459,  328, 1450, 1117,  692, 1850, 1945,   91,  615,  897, 1223, 1543, 1187,
  1219,  432, 1149,  230,  355,  877,  917,  445, 1113,  945,  716, 1383, 1390,
  1562,  831,  376,  868, 1245, 1039, 1708, 1026,   19, 1079,  691, 1265, 1018,
  1239, 1750, 1599,  872,  288,  949, 1092, 1287, 1501,  901, 1064, 1827, 1563,
  1378, 1915,  627,  537, 1713, 1523,  827,  873,  817,  633,   96, 1958, 1698,
   703, 1803,  743,    7,  728, 1834,  955, 1428,  384,  806,  641, 1106, 1013,
  1392, 1066,  931,  244,  666,  837, 1933,  420, 1730,  892, 1985,  815, 1019,
  1300,  522, 1783, 1592, 1431,  209,   72,  589,  804, 1954, 1003, 1344, 1218,
   290,  215, 1717, 1023,  792, 1203,  909,  874,  372, 1083,   67, 1833,  654,
   891, 1101,  886,  566,  154, 1386, 1030, 1481, 1774,  839, 1831,  635, 1097,
   563, 1365,  910,  513,  664, 1261,  638,  345, 1349,  809,  390, 1103,  919,
  1011,   16,  499, 1781,  139,  787,  596, 1047, 1420, 1303,  412, 1901,  383,
   327, 1762, 1475, 1509,  906,  305,  960,  434,  359, 1478,  348,  354, 1057,
  1304, 1870,  914,  990, 1847,  783, 1124, 1141,  496,  388,  521, 1246,  925,
   620,  488, 1482,  198, 1197, 1248,  313,  487,   11,  972,    1, 1701,  156,
  1363, 1813, 1306, 1469, 1369,   98,  870, 1835, 1231, 1906,  678, 1627, 1270,
   856, 1385, 1145, 1690,  495, 1050,  278,  807, 1583, 1040,  991, 1710,  668,
   231,  284,  344,  120, 1508, 1471,  416, 1867,  494, 1157,  318, 1314, 1455,
  1639, 1396, 1336,  172, 1820,  835,  803,  997, 1902,  550, 1135, 1350, 1158,
   251,  607, 1002, 1555, 1500,  888,  688,  266, 1212, 1914, 1241, 1900, 1763,
  1513, 1598, 1943, 1564,  159, 1977, 1111,  397, 1274, 1960,   76,  332,  466,
  1772,    4, 1100,  631,  303,  720,  476,  907,  212,   22, 1707, 1751,  594,
   208, 1859, 1857, 1807, 1055, 1872, 1464,  570,  900,  826, 1776,  503,  449,
   320,  263, 1063, 1990, 1694,  529,  470,  605,  143, 1668,  282, 1044, 1178,
  1949, 1493, 1621,  996,  203, 1352,   50, 1206,  711,  428,  861, 1321,  193,
  1457,  590,  405,  922,  447, 1913,  310,  614, 1854,  465,  780, 1494,  677,
  1259,  216,  411,  954,   36,  694, 1825,  779,  983, 1000,  889, 1689,  333,
  1896, 1611, 1855,  381,  275, 1749,  904, 1169,  944,  847, 1448,  113,  821,
   885,  933, 1934, 1890, 1534, 1930, 1012,  491, 1367, 1155,  908, 1052,  214,
   670,  441, 1734, 1312, 1356, 1015, 1506,  399,  757, 1646, 1467, 1094, 1243,
  1830,    9,  775,  714, 1724, 1208,  687, 1791,  583, 1466, 1996, 1253, 1076,
  1277, 1699,  723,  624, 1876,  296, 1677, 1620,  564, 1938, 1968, 1861, 1727,
  1320,  554,  325, 1926,  509,  644,   63, 1070, 1497, 1275, 1318,  935, 1086,
  1148,  699,  302,  771, 1586,  236,  796,  269, 1167,   93, 1421,  442,   61,
   131,   33, 1340,  192, 1884, 1017, 1845,  632, 1589,   64,  180,  414,  311,
  1416,  729,  616, 1764,  601,  608,  530, 1561,  277, 1142,  297, 1530,  317,
  1700,  240, 1179,  871,  578,  876, 1840,   62, 1394,  858, 1987,  184,  750,
  1059,  881,  650,  369, 1887, 1978, 1527, 1170,  289,  969, 1495,  322, 1842,
   157,  291, 1556, 1726, 1972,  485, 1479,  961, 1293, 1952, 1389,  671, 1935,
    18, 1671, 1615, 1153,  507,  902,  213,   81, 1285,  777,   42, 1795,  879,
  1549,  118,  573, 1358, 1546,  585,  682,  995, 1576, 1152,  286,  686,  339,
  1309,  443, 1782, 1731,  981, 1491,  337,  913,  965, 1473,  548, 1458, 1296,
  1237, 1009, 1168, 1605,   43,  964,  936,  453, 1879, 1470,  480,  602, 1946,
  1907, 1696, 1104, 1916, 1037,  232,  795,  393, 1558, 1969, 1359,  862, 1970,
   768, 1793, 1099,  623,   13,  767, 1993,  875, 1134, 1838, 1769,  684,  356,
    31, 1354, 1991, 1091, 1337,  828, 1539, 1395, 1324, 1022,  347,  386, 1778,
  1216,  160, 1742,  468,  237,  773,  853, 1140, 1412,  121, 1183,  324,  662,
   101, 1108, 1235,  458, 1329, 1460, 1364,  656,  406, 1591, 1144,  210,  658,
   781, 1963, 1923,  829, 1653,  883, 1027, 1912,  455,  752,  567,  759, 1073,
  1156, 1311, 1204, 1452, 1196, 1244,  527,   15, 1786,  350, 1043,   71,  565,
   145,  128,  312, 1756,  304,  173,   55, 1873,  790,  617,  739, 1936, 1669,
    29,  425,   57, 1332,  918, 1670, 1683, 1374,  152, 1805,  848,  855, 1595,
   334, 1766,  321, 1295, 1351, 1331,  659,   20,   60, 1379, 1641,  903,   32,
  1920,  952, 1909,  774,  127,  822, 1849,  640,  797, 1722, 1163,  475, 1682,
   168,  112, 1941, 1672,  280,  273, 1443,  621, 1894, 1120, 1492, 1574, 1339,
  1207, 1743,  379,  226, 1541, 1779,  222, 1036, 1664,  516,  309,  798, 1585,
   199,  700, 1016, 1182,  474, 1931,  454,  887, 1519, 1372,  257, 1904, 1650,
  1198, 1637,  202,  599,  471, 1409,  812, 1651, 1049,  663,  669, 1804, 1601,
  1728, 1638, 1999, 1121, 1754,  740, 1950,  619,   75, 1308,   37, 1087, 1965,
   187,  546, 1190,  943,  575, 1166,  419, 1647,  770, 1526, 1008, 1130, 1816,
  1540,  115, 1335, 1511,  713,  893,  394,  854,  976,  264,   97,   84, 1921,
  1979,  657, 1688, 1377,  756, 1398,  362, 1956,  630, 1522, 1955,  179, 1711,
   966, 1260, 1323, 1966, 1533, 1430, 1067, 1061,  665,  846,   41, 1147, 1839,
   467, 1660, 1818,  151,  108, 1090, 1744, 1877,  710, 1215, 1729,  508, 1472,
   814,  736, 1579,  167,  429, 1853, 1415,  916,  979,  377,   83, 1765, 1316,
   898,   40, 1371, 1995, 1266, 1102, 1434, 1681,  227, 1864,  636,  122, 1345,
  1718,  748, 1626,  978, 1975, 1740, 1221, 1360,  558, 1110,  676,  366, 1282,
   510,    6,  880,  543,  137, 1286, 1279, 1384,  200,  741, 1983,  176,   35,
  1435, 1459, 1898, 1697,  946,  150,   12,  436,  396,   79, 1623, 1927, 1422,
  1721,  766, 1636, 1184, 1504,  523,  402, 1444, 1676,  446,  643, 1290,  857,
  1705, 1069,  655,  387,   49,  755,  844, 1937, 1737, 1294,  433, 1897, 1310,
   959, 1373, 1944,  294,  223,  105,  219, 1041, 1757,  426, 1753,  183,  712,
  1715,  391,  440,  319, 1298, 1631, 1257, 1548,  618,   25, 1547, 1706, 1612,
  1892,  582,  534, 1186,  950,  371, 1997, 1474,  148,  930,  849,  924, 1205,
   360,  398, 1796,  646, 1463,   77,  170,  734,  973, 1211, 1252,  982,  153,
  1441,  301, 1112,  764, 1939,  998,  460, 1692, 1600, 1024,  307, 1535, 1704,
   532,  581, 1401,  542,  926, 1425,  588,  830,  178, 1608, 1784,  652,  937,
  1947,  746, 1957, 1832, 1514,  363, 1122, 1594, 1502, 1490,   89, 1819,   39,
   190,  763, 1732,  217, 1962,   38, 1759,  484,  352,  512,  610,  841, 1210,
   241,  220, 1085,  867,  587, 1817,  125,  415,  252, 1899, 1476, 1554, 1667,
     8,  865, 1375,   88, 1220, 1829,  947, 1391, 1413,  323, 1536,  538, 1596,
  1327, 1480, 1882,  517,  672, 1005,  224, 1403,   47,  382, 1799,   68,  107,
  1723,  482, 1388,   34, 1038, 1088,  629, 1725,  186, 1317,  500, 1046,  380,
  1691, 1609,  409, 1642, 1755,  634, 1622, 1180,  315,   78,  726,  195,   86,
   444, 1291, 1883,  158,  177,  229, 1809,  769, 1695,  285, 1062,  247,  142,
   569,  598,  343, 1528, 1175,  451, 1488,  490,  452,  721, 1582,  544,  531,
  1192, 1806,  697,  794,  824, 1992, 1405,  308, 1747,  144,  427, 1633,  365,
  1254,  559,  479, 1785, 1932,  818, 1984, 1959, 1580, 1603, 1570, 1654,  675,
   653, 1123,  784, 1202, 1758, 1662,  206,  561, 1553,  358, 1733,  571,  731,
  1988,  181,   56,   69, 1234,  912, 1404, 1498,  603, 1194, 1154, 1238,   58,
  1575, 1800,  890, 1143, 1411, 1746,   46,  207, 1678,  340,  437,  704,  463,
  1529, 1624, 1193, 1129,  201,  431,  493,  175,  256, 1675, 1355,  595,  988,
  1368,  808,  109, 1524,  341,  984, 1862, 1042, 1771,   54, 1770,   87, 1114,
   604, 1666,   17,   52, 1189, 1449, 1414, 1578,  225, 1581,  374, 1663, 1267,
   126, 1262, 1307, 1247, 1702,  497,  504, 1442,  637,  958,  987, 1075, 1510,
   295,  852,  185,  882,  611, 1797, 1656,  725, 1693,  299,  477,  147,  336,
  1280, 1811,  259,  593,  923,  576, 1880,  102, 1632, 1881,  970,  942, 1568,
   249,  242,  836,  843, 1188,  574, 1686,  738,  368,  401, 1346, 1917,  702,
   695, 1940, 1035, 1496,  974,  331, 1334,  140,  957, 1021, 1789, 1865,  895,
  1072,  338, 1229,  859, 1878, 1081,  116, 1618,  751,  141, 1176,   53,   90,
  1029,   70,  353,    3,  520, 1427, 1989, 1447,  135, 1891, 1604, 1228,  413,
  1630,  722, 1313, 1652, 1748, 1980,  204,  533,  525,   44, 1487, 1893, 1250,
    80, 1531,  800, 1679,  560,  161, 1846, 1674,  626, 1439,   23,  261,  464,
   422,  866,  905, 1918,  985,  552, 1001, 1866, 1703,  351,  408,  928,  776,
    51, 1150,  450, 1948, 1889, 1249, 1856, 1644,  255, 1326, 1376,   45, 1661,
  1084, 1230, 1982, 1736, 1928,  823, 1823,  100,  953, 1328, 1305, 1886, 1848,
   625,  762, 1908,  745,  580, 1325,  667, 1643,  461, 1484,  250, 1380, 1794,
  1714,  174, 1593, 1720, 1588, 1802, 1768,  609,  524,  850, 2000, 1357,   99,
   268, 1559, 1174,    5,  501,  724,  182, 1322, 1801,  591,  962,  956,  696,
    59, 1172,  189,  233, 1301, 1964, 1417, 1951,  205,  165,  514, 1629, 1851,
  1418,  196,  424,  462,  117,   82, 1810, 1315, 1283, 1438, 1128, 1199, 1537,
  1217,  526, 1868, 1105, 1273, 1841,  785, 1998, 1798, 1542, 1177,  572, 1161,
   761,  717,    2,  246,  281, 1640,  986,  732, 1361,  674, 1200, 1773, 1269,
  1836, 1165,  819, 1719, 1242, 1685,  361,   10,   24, 1885, 1503,  300, 1538,
  1054,  940,  367, 1863,  706, 1437,  747,  314,  298, 1226, 1034,  228, 1468,
  1240, 1489, 1004, 1518,  941,  690,  647,  592,  579,  577,  478, 1976,  403,
  1617,  492,  103, 1716, 1828, 1658, 1843,  114, 1347, 1048, 1745, 1125, 1606,
  1387,  410, 1382,  132, 1191, 1440,  920, 1133,  329,  163, 1645,  545,  628,
  1151, 1330, 1348,  326,  894, 1454,  166, 1397,  932,  481,  860,  162, 1456,
  1659,  385,  993,  188,  789, 1426, 1680,  927, 1507, 1319, 1095, 1860,  164,
  1446, 1292, 1408,  539,  878,  528, 1164, 1429,  730, 1613,  505,  735,  825,
    28, 1107, 1814, 1822,  389,  404,  110, 1053,  788, 1407, 1159, 1929,  622,
  1602,  262,  701,  218,  899, 1752,  342,  392, 1278,  813,  547,  155, 1910,
   742, 1792,  130,  270, 1423,  486, 1974,  801,  651, 1227, 1545,  378, 1381,
  1025,  980,  557,   21, 1515,  283,  239,  370, 1419, 1572,  456,  869,  613,
   435,  679,  373,  584,   73,  191,  553,   30,  502, 1060,  124, 1924,  134,
   287, 1342,  884,  810, 1550,  689, 1424,  929,  707, 1032,  939,  778,  541,
   645,  737, 1138, 1288,  967,  718, 1119, 1499, 1077,  438, 1014, 1994, 1485,
    66,  511,  104,  820,  816, 1973, 1616,   92,  600,  119, 1338, 1432, 1986,
   357,  253, 1812,  146,  430, 1971, 1684,  221, 1911,  267,  260, 1826, 1209,
   977,  506, 1131, 1673,  802,  845,  498,  489, 1080,  473, 1573,  197, 1567,
  1400, 1557,  793, 1393,  708, 1837,  749,  349, 1402,  989, 1132, 1967, 1343,
  1648,  758,  680, 1566,  681, 1201,  335,  975, 1271, 1981,  833,  234,  834,
   515, 1171, 1093, 1875,   14,  423,  276,  417,  395, 1089,  705, 1486, 1462,
  1126, 1139, 1195, 1185,  519,  133, 1058,  235, 1451, 1844,  948, 1655,  685,
  1074, 1824,  693,  551, 1532, 1281,  540, 1268, 1096, 1276, 1625,  999, 1255,
    26,  106, 1056, 1761, 1739,  911,  805, 1922, 1552,  400, 1815, 1871,   65,
   211, 1657,  448, 1233,  306, 1289,  129, 1869,  258,  271, 1251,  864,  760,
   248, 1007, 1821, 1160,   94, 1071,  951,  719, 1852,  238, 1433,  765, 1284,
   407, 1410,  136, 1136, 1560, 1068,  123, 1919, 1116,  364,  330, 1302,  472,
  1263,  279, 1445,  274,  612,  683, 1483,  786,  727,  606, 1236, 1109,  673,
   535,  194,  568,  994, 1353, 1571,   85, 1565, 1370, 1264, 1942,  963, 1362,
  1006,  254,  971,  272, 1115, 1777,  782,  483,  938,  421, 1775,  744,  851,
  1213, 1787, 1033, 1903,  549,  346,  811, 1628, 1162,  439, 1687, 1272, 1010,
  1082, 1584, 1767,  292, 1961, 1256, 1925, 1577,  457, 1634, 1512, 1735, 1020,
   642, 1790, 1597,  375,  469, 1874,  791,  245, 1520,  915,  562,  169,  715,
  1031, 1905, 1224,  772, 1551, 1808, 1028, 1045, 1406,  639,  518, 1173,  138,
  1619,  709,  556, 1232,  799,  754, 1610,  896, 1225,   74,  842,  863,  586,
   555, 1146, 1525,  698, 1517, 1118, 1607, 1741,  968, 1366,  921,  536, 1590,
  1297,  418,  733, 1222,  597, 1953,  648, 1299, 1436,  840, 1569,  171, 1665,
  1635,   48, 1712, 1051, 1738,  934, 1098, 1895,  293,  992, 1788, 1587, 1544,
   838, 1858,  265, 1214, 1888,  111, 1465, 1709, 1453,  316,  149, 1078, 1521,
  1341, 1649, 1614, 1516,  832, 1137, 1127, 1505,  243, 1461, 1065,  753,   95,
    27, 1333, 1477,  649, 1760, 1399, 1258, 1181,  661, 1780,  660)

================================================================================
```

D The 92 different solutions of the 8-Queens problem

```
  8 . . . . . . . X
  7 . . X . . . . .
  6 X . . . . . . .
  5 . . . . . X . .
  4 . X . . . . . .
  3 . . . . X . . .
  2 . . . . . . X .
  1 . . . X . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   6,    4,    7,    1,    3,    5,    2,    8)

================================================================================

  8 . . . . X . . .
  7 . X . . . . . .
  6 . . . X . . . .
  5 . . . . . X . .
  4 . . . . . . . X
  3 . . X . . . . .
  2 X . . . . . . .
  1 . . . . . . X .
    A B C D E F G H

================================================================================

 - Solution:

 (   2,    7,    3,    6,    8,    5,    1,    4)

================================================================================

  8 X . . . . . . .
  7 . . . . . . X .
  6 . . . . X . . .
  5 . . . . . . . X
  4 . X . . . . . .
  3 . . . X . . . .
  2 . . . . . X . .
  1 . . X . . . . .
    A B C D E F G H
```

```
================================================================================

 - Solution:

 (   8,    4,    1,    3,    6,    2,    7,    5)

================================================================================

  8 . . . . . . X .
  7 X . . . . . . .
  6 . . X . . . . .
  5 . . . . . . . X
  4 . . . . . X . .
  3 . . . X . . . .
  2 . X . . . . . .
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   7,    2,    6,    3,    1,    4,    8,    5)

================================================================================

  8 . . X . . . . .
  7 . . . . . . X .
  6 . X . . . . . .
  5 . . . . . . . X
  4 . . . . . X . .
  3 . . . X . . . .
  2 X . . . . . . .
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   2,    6,    8,    3,    1,    4,    7,    5)

================================================================================

  8 . . . . . X . .
  7 . . X . . . . .
  6 . . . . . . X .
  5 . X . . . . . .
  4 . . . . . . . X
  3 . . . . X . . .
  2 X . . . . . . .
  1 . . . X . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   2,    5,    7,    1,    3,    8,    6,    4)

================================================================================

  8 . . . . . . . X
  7 . X . . . . . .
  6 . . . X . . . .
  5 X . . . . . . .
  4 . . . . . . X .
  3 . . . . X . . .
  2 . . X . . . . .
  1 . . . . . X . .
    A B C D E F G H

================================================================================

 - Solution:

 (   5,    7,    2,    6,    3,    1,    4,    8)

================================================================================

  8 . . X . . . . .
  7 . . . . . X . .
  6 . . . X . . . .
  5 X . . . . . . .
  4 . . . . . . . X
  3 . . . . X . . .
  2 . . . . . . X .
  1 . X . . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   5,    1,    8,    6,    3,    7,    2,    4)

================================================================================
  8 . . . . X . . .
  7 . . . . . . X .
  6 X . . . . . . .
  5 . . X . . . . .
  4 . . . . . . . X
  3 . . . . . X . .
  2 . . . X . . . .
  1 . X . . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   6,    1,    5,    2,    8,    3,    7,    4)

================================================================================

  8 . . . X . . . .
  7 . . . . . X . .
  6 . . . . . . . X
  5 . . X . . . . .
  4 X . . . . . . .
  3 . . . . . . X .
  2 . . . . X . . .
  1 . X . . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    1,    5,    8,    2,    7,    3,    6)

================================================================================

  8 . . . . X . . .
  7 . . . . . . X .
  6 . X . . . . . .
  5 . . . X . . . .
  4 . . . . . . . X
  3 X . . . . . . .
  2 . . X . . . . .
  1 . . . . . X . .
    A B C D E F G H

================================================================================

 - Solution:
```

```
 (   3,    6,    2,    5,    8,    1,    7,    4)

================================================================================

  8 . . . X . . . .
  7 . . . . . . . X
  6 . . . . X . . .
  5 . . X . . . . .
  4 X . . . . . . .
  3 . . . . . . X .
  2 . X . . . . . .
  1 . . . . . X . .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    2,    5,    8,    6,    1,    3,    7)

================================================================================

  8 . . . . X . . .
  7 . X . . . . . .
  6 . . . . . X . .
  5 X . . . . . . .
  4 . . . . . . X .
  3 . . . X . . . .
  2 . . . . . . . X
  1 . . X . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   5,    7,    1,    3,    8,    6,    4,    2)

================================================================================

  8 . . . . . X . .
  7 . . . X . . . .
  6 . . . . . . X .
  5 X . . . . . . .
  4 . . X . . . . .
  3 . . . . X . . .
  2 . X . . . . . .
  1 . . . . . . . X
```

```
    A B C D E F G H

================================================================================

 - Solution:

 (   5,    2,    4,    7,    3,    8,    6,    1)

================================================================================

  8 . . . . . X . .
  7 . X . . . . . .
  6 . . . . . . X .
  5 X . . . . . . .
  4 . . . X . . . .
  3 . . . . . . . X
  2 . . . . X . . .
  1 . . X . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   5,    7,    1,    4,    2,    8,    6,    3)

================================================================================

  8 . . . . . X . .
  7 . X . . . . . .
  6 . . . . . . X .
  5 X . . . . . . .
  4 . . X . . . . .
  3 . . . . X . . .
  2 . . . . . . . X
  1 . . . X . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   5,    7,    4,    1,    3,    8,    6,    2)

================================================================================

  8 X . . . . . . .
  7 . . . . . X . .
  6 . . . . . . . X
  5 . . X . . . . .
  4 . . . . . . X .
  3 . . . X . . . .
  2 . X . . . . . .
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   8,    2,    5,    3,    1,    7,    4,    6)

================================================================================

  8 . . . X . . . .
  7 . . . . . . X .
  6 . . X . . . . .
  5 . . . . . . . X
  4 . X . . . . . .
  3 . . . . X . . .
  2 X . . . . . . .
  1 . . . . . X . .
    A B C D E F G H

================================================================================

 - Solution:

 (   2,    4,    6,    8,    3,    1,    7,    5)

================================================================================

  8 . . X . . . . .
  7 . . . . X . . .
  6 . . . . . . . X
  5 . . . X . . . .
  4 X . . . . . . .
  3 . . . . . . X .
  2 . X . . . . . .
  1 . . . . . X . .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    2,    8,    5,    7,    1,    3,    6)
```

```
================================================================================

  8 X . . . . . . .
  7 . . . . . . X .
  6 . . . X . . . .
  5 . . . . . X . .
  4 . . . . . . . X
  3 . X . . . . . .
  2 . . . . X . . .
  1 . . X . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   8,    3,    1,    6,    2,    5,    7,    4)

================================================================================

  8 . . . . . . X .
  7 . . X . . . . .
  6 . . . . . . . X
  5 . X . . . . . .
  4 . . . . X . . .
  3 X . . . . . . .
  2 . . . . . X . .
  1 . . . X . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   3,    5,    7,    1,    4,    2,    8,    6)

================================================================================

  8 . . . . . . X .
  7 . X . . . . . .
  6 . . . X . . . .
  5 X . . . . . . .
  4 . . . . . . . X
  3 . . . . X . . .
  2 . . X . . . . .
  1 . . . . . X . .
    A B C D E F G H

================================================================================
 - Solution:

 (   5,    7,    2,    6,    3,    1,    8,    4)

================================================================================

  8 . . X . . . . .
  7 . . . . . X . .
  6 . . . . . . . X
  5 X . . . . . . .
  4 . . . X . . . .
  3 . . . . . . X .
  2 . . . . X . . .
  1 . X . . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   5,    1,    8,    4,    2,    7,    3,    6)

================================================================================

  8 . . . . . . . X
  7 . X . . . . . .
  6 . . . . X . . .
  5 . . X . . . . .
  4 X . . . . . . .
  3 . . . . . . X .
  2 . . . X . . . .
  1 . . . . . X . .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    7,    5,    2,    6,    1,    3,    8)

================================================================================

  8 . X . . . . . .
  7 . . . . . . . X
  6 . . . . . X . .
  5 X . . . . . . .
  4 . . X . . . . .
  3 . . . . X . . .
  2 . . . . . . X .
  1 . . . X . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   5,    8,    4,    1,    3,    6,    2,    7)

================================================================================

  8 . . . . X . . .
  7 X . . . . . . .
  6 . . . . . . . X
  5 . . . . . X . .
  4 . . X . . . . .
  3 . . . . . . X .
  2 . X . . . . . .
  1 . . . X . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   7,    2,    4,    1,    8,    5,    3,    6)

================================================================================

  8 . . X . . . . .
  7 . . . . . . . X
  6 . . . X . . . .
  5 . . . . . . X .
  4 X . . . . . . .
  3 . . . . . X . .
  2 . X . . . . . .
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    2,    8,    6,    1,    3,    5,    7)

================================================================================

  8 . . . . X . . .
  7 X . . . . . . .
  6 . . . X . . . .
  5 . . . . . X . .
  4 . . . . . . . X
  3 . X . . . . . .
  2 . . . . . . X .
  1 . . X . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   7,    3,    1,    6,    8,    5,    2,    4)

================================================================================

  8 . . X . . . . .
  7 . . . . . X . .
  6 . . . . . . . X
  5 X . . . . . . .
  4 . . . . X . . .
  3 . . . . . . X .
  2 . X . . . . . .
  1 . . . X . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   5,    2,    8,    1,    4,    7,    3,    6)

================================================================================

  8 . . X . . . . .
  7 . . . . X . . .
  6 . . . . . . X .
  5 X . . . . . . .
  4 . . . X . . . .
  3 . X . . . . . .
  2 . . . . . . . X
  1 . . . . . X . .
    A B C D E F G H

================================================================================

 - Solution:
```

```
 (   5,    3,    8,    4,    7,    1,    6,    2)

================================================================================

  8 . . . . . X . .
  7 . . X . . . . .
  6 . . . . . . X .
  5 . X . . . . . .
  4 . . . X . . . .
  3 . . . . . . . X
  2 X . . . . . . .
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   2,    5,    7,    4,    1,    8,    6,    3)

================================================================================

  8 . . . . X . . .
  7 . . . . . . . X
  6 . . . X . . . .
  5 X . . . . . . .
  4 . . X . . . . .
  3 . . . . . X . .
  2 . X . . . . . .
  1 . . . . . . X .
    A B C D E F G H

================================================================================

 - Solution:

 (   5,    2,    4,    6,    8,    3,    1,    7)

================================================================================

  8 . X . . . . . .
  7 . . . . . X . .
  6 X . . . . . . .
  5 . . . . . . X .
  4 . . . X . . . .
  3 . . . . . . . X
  2 . . X . . . . .
  1 . . . . X . . .
    A B C D E F G H
```

```
================================================================================

 - Solution:

 (   6,    8,    2,    4,    1,    7,    5,    3)

================================================================================

  8 . . . X . . . .
  7 . . . . . X . .
  6 . . . . . . . X
  5 . X . . . . . .
  4 . . . . . . X .
  3 X . . . . . . .
  2 . . X . . . . .
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   3,    5,    2,    8,    1,    7,    4,    6)

================================================================================

  8 . . . . X . . .
  7 . . . . . . . X
  6 . . . X . . . .
  5 X . . . . . . .
  4 . . . . . . X .
  3 . X . . . . . .
  2 . . . . . X . .
  1 . . X . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   5,    3,    1,    6,    8,    2,    4,    7)

================================================================================

  8 . . . . X . . .
  7 . . X . . . . .
  6 X . . . . . . .
  5 . . . . . . X .
  4 . X . . . . . .
  3 . . . . . . . X
  2 . . . . . X . .
  1 . . . X . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   6,    4,    7,    1,    8,    2,    5,    3)

================================================================================

  8 . . X . . . . .
  7 . . . . . . X .
  6 . X . . . . . .
  5 . . . . . . . X
  4 . . . . X . . .
  3 X . . . . . . .
  2 . . . X . . . .
  1 . . . . . X . .
    A B C D E F G H

================================================================================

 - Solution:

 (   3,    6,    8,    2,    4,    1,    7,    5)

================================================================================

  8 . . . . X . . .
  7 . X . . . . . .
  6 . . . . . . . X
  5 X . . . . . . .
  4 . . . X . . . .
  3 . . . . . . X .
  2 . . X . . . . .
  1 . . . . . X . .
    A B C D E F G H

================================================================================

 - Solution:

 (   5,    7,    2,    4,    8,    1,    3,    6)

================================================================================
  8 . . . . . X . .
  7 . . X . . . . .
  6 X . . . . . . .
  5 . . . . . . X .
  4 . . . . X . . .
  3 . . . . . . . X
  2 . X . . . . . .
  1 . . . X . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   6,    2,    7,    1,    4,    8,    5,    3)

================================================================================

  8 . . . . X . . .
  7 X . . . . . . .
  6 . . . . . . . X
  5 . . . X . . . .
  4 . X . . . . . .
  3 . . . . . . X .
  2 . . X . . . . .
  1 . . . . . X . .
    A B C D E F G H

================================================================================

 - Solution:

 (   7,    4,    2,    5,    8,    1,    3,    6)

================================================================================

  8 . . X . . . . .
  7 X . . . . . . .
  6 . . . . . . X .
  5 . . . . X . . .
  4 . . . . . . . X
  3 . X . . . . . .
  2 . . . X . . . .
  1 . . . . . X . .
    A B C D E F G H

================================================================================
 - Solution:

 (   7,    3,    8,    2,    5,    1,    6,    4)

================================================================================

  8 . . X . . . . .
  7 . . . . X . . .
  6 . X . . . . . .
  5 . . . . . . . X
  4 X . . . . . . .
  3 . . . . . . X .
  2 . . . X . . . .
  1 . . . . . X . .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    6,    8,    2,    7,    1,    3,    5)

================================================================================

  8 . . . X . . . .
  7 . X . . . . . .
  6 . . . . . . X .
  5 . . . . X . . .
  4 X . . . . . . .
  3 . . . . . . . X
  2 . . . . . X . .
  1 . . X . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    7,    1,    8,    5,    2,    6,    3)

================================================================================

  8 . . X . . . . .
  7 . . . . . X . .
  6 . X . . . . . .
  5 . . . . . . X .
  4 . . . . X . . .
  3 X . . . . . . .
  2 . . . . . . . X
```

```
  1 . . . X . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   3,    6,    8,    1,    4,    7,    5,    2)

================================================================================

  8 . . . . . . X .
  7 . . X . . . . .
  6 X . . . . . . .
  5 . . . . . X . .
  4 . . . . . . . X
  3 . . . . X . . .
  2 . X . . . . . .
  1 . . . X . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   6,    2,    7,    1,    3,    5,    8,    4)

================================================================================

  8 . . . . . . X .
  7 . . . . X . . .
  6 . . X . . . . .
  5 X . . . . . . .
  4 . . . . . X . .
  3 . . . . . . . X
  2 . X . . . . . .
  1 . . . X . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   5,    2,    6,    1,    7,    4,    8,    3)

================================================================================

  8 . . . . . X . .
  7 . . . X . . . .
  6 X . . . . . . .
  5 . . . . X . . .
  4 . . . . . . . X
  3 . X . . . . . .
  2 . . . . . . X .
  1 . . X . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   6,    3,    1,    7,    5,    8,    2,    4)

================================================================================

  8 . . . . X . . .
  7 . . X . . . . .
  6 X . . . . . . .
  5 . . . . . X . .
  4 . . . . . . . X
  3 . X . . . . . .
  2 . . . X . . . .
  1 . . . . . . X .
    A B C D E F G H

================================================================================

 - Solution:

 (   6,    3,    7,    2,    8,    5,    1,    4)

================================================================================

  8 . . . X . . . .
  7 . X . . . . . .
  6 . . . . . . . X
  5 . . . . X . . .
  4 . . . . . . X .
  3 X . . . . . . .
  2 . . X . . . . .
  1 . . . . . X . .
    A B C D E F G H

================================================================================

 - Solution:

 (   3,    7,    2,    8,    5,    1,    4,    6)
```

```
================================================================================

  8 . . . X . . . .
  7 . . . . . . . X
  6 X . . . . . . .
  5 . . X . . . . .
  4 . . . . . X . .
  3 . X . . . . . .
  2 . . . . . . X .
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   6,    3,    5,    8,    1,    4,    2,    7)

================================================================================

  8 . X . . . . . .
  7 . . . . . X . .
  6 . . . . . . . X
  5 . . X . . . . .
  4 X . . . . . . .
  3 . . . X . . . .
  2 . . . . . . X .
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    8,    5,    3,    1,    7,    2,    6)

================================================================================

  8 . X . . . . . .
  7 . . . . . . X .
  6 . . X . . . . .
  5 . . . . . X . .
  4 . . . . . . . X
  3 . . . . X . . .
  2 X . . . . . . .
  1 . . . X . . . .
    A B C D E F G H
```

```
================================================================================

 - Solution:

 (   2,    8,    6,    1,    3,    5,    7,    4)

================================================================================

  8 . . . . . X . .
  7 . . X . . . . .
  6 . . . . . . X .
  5 . . . X . . . .
  4 X . . . . . . .
  3 . . . . . . . X
  2 . X . . . . . .
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    2,    7,    5,    1,    8,    6,    3)

================================================================================

  8 . . . . . X . .
  7 . . . X . . . .
  6 . X . . . . . .
  5 . . . . . . . X
  4 . . . . X . . .
  3 . . . . . . X .
  2 X . . . . . . .
  1 . . X . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   2,    6,    1,    7,    4,    8,    3,    5)

================================================================================

  8 . . . X . . . .
  7 X . . . . . . .
  6 . . . . X . . .
  5 . . . . . . . X
  4 . X . . . . . .
  3 . . . . . . X .
  2 . . X . . . . .
  1 . . . . . X . .
    A B C D E F G H

================================================================================

 - Solution:

 (   7,    4,    2,    8,    6,    1,    3,    5)

================================================================================

  8 . . X . . . . .
  7 . . . . . X . .
  6 . X . . . . . .
  5 . . . . X . . .
  4 . . . . . . . X
  3 X . . . . . . .
  2 . . . . . . X .
  1 . . . X . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   3,    6,    8,    1,    5,    7,    2,    4)

================================================================================

  8 . . . X . . . .
  7 . . . . . . X .
  6 . . . . X . . .
  5 . . X . . . . .
  4 X . . . . . . .
  3 . . . . . X . .
  2 . . . . . . . X
  1 . X . . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    1,    5,    8,    6,    3,    7,    2)

================================================================================
  8 . . . . . X . .
  7 . . . . . . . X
  6 . X . . . . . .
  5 . . . X . . . .
  4 X . . . . . . .
  3 . . . . . . X .
  2 . . . . X . . .
  1 . . X . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    6,    1,    5,    2,    8,    3,    7)

================================================================================

  8 . . . . . . X .
  7 . . . X . . . .
  6 . X . . . . . .
  5 . . . . X . . .
  4 . . . . . . . X
  3 X . . . . . . .
  2 . . X . . . . .
  1 . . . . . X . .
    A B C D E F G H

================================================================================

 - Solution:

 (   3,    6,    2,    7,    5,    1,    8,    4)

================================================================================

  8 . . . X . . . .
  7 . X . . . . . .
  6 . . . . . . X .
  5 . . X . . . . .
  4 . . . . . X . .
  3 . . . . . . . X
  2 X . . . . . . .
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:
```

```
 (   2,    7,    5,    8,    1,    4,    6,    3)

================================================================================

  8 . X . . . . . .
  7 . . . . X . . .
  6 . . . . . . X .
  5 . . . X . . . .
  4 X . . . . . . .
  3 . . . . . . . X
  2 . . . . . X . .
  1 . . X . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    8,    1,    5,    7,    2,    6,    3)

================================================================================

  8 . . . . X . . .
  7 . . . . . . X .
  6 . X . . . . . .
  5 . . . . . X . .
  4 . . X . . . . .
  3 X . . . . . . .
  2 . . . . . . . X
  1 . . . X . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   3,    6,    4,    1,    8,    5,    7,    2)

================================================================================

  8 . . . X . . . .
  7 . . . . . . X .
  6 . . . . X . . .
  5 . X . . . . . .
  4 . . . . . X . .
  3 X . . . . . . .
  2 . . X . . . . .
  1 . . . . . . . X
```

```
    A B C D E F G H

================================================================================

 - Solution:

 (   3,    5,    2,    8,    6,    4,    7,    1)

================================================================================

  8 . . . X . . . .
  7 . . . . . X . .
  6 X . . . . . . .
  5 . . . . X . . .
  4 . X . . . . . .
  3 . . . . . . . X
  2 . . X . . . . .
  1 . . . . . . X .
    A B C D E F G H

================================================================================

 - Solution:

 (   6,    4,    2,    8,    5,    7,    1,    3)

================================================================================

  8 . X . . . . . .
  7 . . . X . . . .
  6 . . . . . X . .
  5 . . . . . . . X
  4 . . X . . . . .
  3 X . . . . . . .
  2 . . . . . . X .
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   3,    8,    4,    7,    1,    6,    2,    5)

================================================================================

  8 . . . . . X . .
  7 . . X . . . . .
  6 X . . . . . . .
  5 . . . . . . . X
  4 . . . X . . . .
  3 . X . . . . . .
  2 . . . . . . X .
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   6,    3,    7,    4,    1,    8,    2,    5)

================================================================================

  8 . . X . . . . .
  7 . . . . . X . .
  6 . . . . . . . X
  5 . X . . . . . .
  4 . . . X . . . .
  3 X . . . . . . .
  2 . . . . . . X .
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   3,    5,    8,    4,    1,    7,    2,    6)

================================================================================

  8 . . . . X . . .
  7 . . . . . . X .
  6 . X . . . . . .
  5 . . . . . X . .
  4 . . X . . . . .
  3 X . . . . . . .
  2 . . . X . . . .
  1 . . . . . . . X
    A B C D E F G H

================================================================================

 - Solution:

 (   3,    6,    4,    2,    8,    5,    7,    1)
```

```
================================================================================

  8 . . . X . . . .
  7 . . . . . . X .
  6 X . . . . . . .
  5 . . . . . . . X
  4 . . . . X . . .
  3 . X . . . . . .
  2 . . . . . X . .
  1 . . X . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   6,    3,    1,    8,    4,    2,    7,    5)

================================================================================

  8 X . . . . . . .
  7 . . . . X . . .
  6 . . . . . . . X
  5 . . . . . X . .
  4 . . X . . . . .
  3 . . . . . . X .
  2 . X . . . . . .
  1 . . . X . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   8,    2,    4,    1,    7,    5,    3,    6)

================================================================================

  8 . . . . . X . .
  7 . . X . . . . .
  6 . . . . X . . .
  5 . . . . . . X .
  4 X . . . . . . .
  3 . . . X . . . .
  2 . X . . . . . .
  1 . . . . . . . X
    A B C D E F G H

================================================================================
 - Solution:

 (   4,    2,    7,    3,    6,    8,    5,    1)

================================================================================

  8 . X . . . . . .
  7 . . . . . . X .
  6 . . . . X . . .
  5 . . . . . . . X
  4 X . . . . . . .
  3 . . . X . . . .
  2 . . . . . X . .
  1 . . X . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    8,    1,    3,    6,    2,    7,    5)

================================================================================

  8 . . . . . . X .
  7 . X . . . . . .
  6 . . . . . X . .
  5 . . X . . . . .
  4 X . . . . . . .
  3 . . . X . . . .
  2 . . . . . . . X
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    7,    5,    3,    1,    6,    8,    2)

================================================================================

  8 . . . . . X . .
  7 . . X . . . . .
  6 . . . . X . . .
  5 . . . . . . . X
  4 X . . . . . . .
  3 . . . X . . . .
  2 . X . . . . . .
  1 . . . . . . X .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    2,    7,    3,    6,    8,    1,    5)

================================================================================

  8 . . . . X . . .
  7 . X . . . . . .
  6 . . . X . . . .
  5 . . . . . . X .
  4 . . X . . . . .
  3 . . . . . . . X
  2 . . . . . X . .
  1 X . . . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   1,    7,    4,    6,    8,    2,    5,    3)

================================================================================

  8 . . X . . . . .
  7 . . . . . X . .
  6 . . . X . . . .
  5 . X . . . . . .
  4 . . . . . . . X
  3 . . . . X . . .
  2 . . . . . . X .
  1 X . . . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   1,    5,    8,    6,    3,    7,    2,    4)

================================================================================

  8 . . . X . . . .
  7 X . . . . . . .
  6 . . . . X . . .
  5 . . . . . . . X
  4 . . . . . X . .
  3 . . X . . . . .
  2 . . . . . . X .
  1 . X . . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   7,    1,    3,    8,    6,    4,    2,    5)

================================================================================

  8 . . . . . . . X
  7 . . . X . . . .
  6 X . . . . . . .
  5 . . X . . . . .
  4 . . . . . X . .
  3 . X . . . . . .
  2 . . . . . . X .
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   6,    3,    5,    7,    1,    4,    2,    8)

================================================================================

  8 . . . X . . . .
  7 . X . . . . . .
  6 . . . . X . . .
  5 . . . . . . . X
  4 . . . . . X . .
  3 X . . . . . . .
  2 . . X . . . . .
  1 . . . . . . X .
    A B C D E F G H

================================================================================

 - Solution:
```

```
 (   3,    7,    2,    8,    6,    4,    1,    5)

================================================================================

  8 . . . . . . X .
  7 . . . X . . . .
  6 . X . . . . . .
  5 . . . . . . . X
  4 . . . . . X . .
  3 X . . . . . . .
  2 . . X . . . . .
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   3,    6,    2,    7,    1,    4,    8,    5)

================================================================================

  8 . . . . . X . .
  7 . . X . . . . .
  6 X . . . . . . .
  5 . . . . . . . X
  4 . . . . X . . .
  3 . X . . . . . .
  2 . . . X . . . .
  1 . . . . . . X .
    A B C D E F G H

================================================================================

 - Solution:

 (   6,    3,    7,    2,    4,    8,    1,    5)

================================================================================

  8 . . . . X . . .
  7 . . X . . . . .
  6 . . . . . . . X
  5 . . . X . . . .
  4 . . . . . . X .
  3 X . . . . . . .
  2 . . . . . X . .
  1 . X . . . . . .
    A B C D E F G H
```

```
================================================================================

 - Solution:

 (   3,    1,    7,    5,    8,    2,    4,    6)

================================================================================

  8 . . . X . . . .
  7 . X . . . . . .
  6 . . . . . . X .
  5 . . X . . . . .
  4 . . . . . X . .
  3 . . . . . . . X
  2 . . . . X . . .
  1 X . . . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   1,    7,    5,    8,    2,    4,    6,    3)

================================================================================

  8 . . . . X . . .
  7 . . . . . . X .
  6 X . . . . . . .
  5 . . . X . . . .
  4 . X . . . . . .
  3 . . . . . . . X
  2 . . . . . X . .
  1 . . X . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   6,    4,    1,    5,    8,    2,    7,    3)

================================================================================

  8 . . . X . . . .
  7 . . . . . . . X
  6 X . . . . . . .
  5 . . . . X . . .
  4 . . . . . . X .
  3 . X . . . . . .
  2 . . . . . X . .
  1 . . X . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   6,    3,    1,    8,    5,    2,    4,    7)

================================================================================

  8 . . . . X . . .
  7 . . . . . . X .
  6 . . . X . . . .
  5 X . . . . . . .
  4 . . X . . . . .
  3 . . . . . . . X
  2 . . . . . X . .
  1 . X . . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   5,    1,    4,    6,    8,    2,    7,    3)

================================================================================

  8 . . . . . X . .
  7 X . . . . . . .
  6 . . . . X . . .
  5 . X . . . . . .
  4 . . . . . . . X
  3 . . X . . . . .
  2 . . . . . . X .
  1 . . . X . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   7,    5,    3,    1,    6,    8,    2,    4)

================================================================================
  8 . . . X . . . .
  7 . X . . . . . .
  6 . . . . . . . X
  5 . . . . . X . .
  4 X . . . . . . .
  3 . . X . . . . .
  2 . . . . X . . .
  1 . . . . . . X .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    7,    3,    8,    2,    5,    1,    6)

================================================================================

  8 . . . . . X . .
  7 . . . X . . . .
  6 . . . . . . X .
  5 X . . . . . . .
  4 . . . . . . . X
  3 . X . . . . . .
  2 . . . . X . . .
  1 . . X . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   5,    3,    1,    7,    2,    8,    6,    4)

================================================================================

  8 . X . . . . . .
  7 . . . . X . . .
  6 . . . . . . X .
  5 X . . . . . . .
  4 . . X . . . . .
  3 . . . . . . . X
  2 . . . . . X . .
  1 . . . X . . . .
    A B C D E F G H

================================================================================
 - Solution:

 (   5,    8,    4,    1,    7,    2,    6,    3)

================================================================================

  8 . . X . . . . .
  7 . . . . X . . .
  6 . X . . . . . .
  5 . . . . . . . X
  4 . . . . . X . .
  3 . . . X . . . .
  2 . . . . . . X .
  1 X . . . . . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   1,    6,    8,    3,    7,    4,    2,    5)

================================================================================

  8 . . X . . . . .
  7 . . . . . X . .
  6 . X . . . . . .
  5 . . . . . . X .
  4 X . . . . . . .
  3 . . . X . . . .
  2 . . . . . . . X
  1 . . . . X . . .
    A B C D E F G H

================================================================================

 - Solution:

 (   4,    6,    8,    3,    1,    7,    5,    2)

================================================================================
```