Problema de las N Reinas:

Maximización de la solución con K Reinas fijas

La primer versión del problema fue planteada en 1848 por el jugador de Ajedrez *Max Brezzel* el cual consistía en colocar 8 reinas en una tablero de ajedrez de dimensiones $8x8$ sin que se exista alguna amenaza por parte de estas piezas. Las distintas formas de colocar las 8 reinas en un tablero de $(8x8)$ son 12 y 80 más a partir de diversas transformaciones de rotación y reflexión. Este problema se puede generalizar al intentar colocar n reinas en un tablero de nxn.

Durante el acercamiento dado en este proyecto, se planteó una variante al problema. Sea un tablero de dimensiones nxn y dadas las posiciones de un número k de reinas fijas en el tablero, ¿cuál es el número máximo de reinas que se pueden colocar posteriormente?

Metodología

Edsger Dijkstra, uno de los personajes mas representativos en el área de las Ciencias de la Computación usó este problema para presentar una solución a través del algoritmo de *backtracking*. La idea esencial de este algoritmo radica en construir y/o buscar la mejor solución generada de resolver problemas más pequeños o similares generados por las combinaciones de las posibilidades de toma de decisiones sobre cómo se podría generar una solución.

En el juego de ajedrez la reina debe cumplir con las siguientes restricciones:

- Se puede mover en las ocho direcciones cuantas casillas se deseen.
- Una pieza está bajo ataque si se encuentra en algún camino posible de otra pieza.

Por tanto para colocar varías reinas en el tablero se reduce a:

- Dos o más reinas no pueden estar en la misma fila o columna.
- Dos o más reinas no pueden estar en la misma diagonal.

La solución mediante backtracking se basa en las restricciones anteriores. Como a lo más debe existir una reina por columna, el problema se reduce generar las distintas combinaciones de colocar una reina por cada columna y verificar cual de ellas es válida.

```
/*
    Búsqueda de soluciones mediante backtraking
    para colocar n reinas
*/
n_queens <- Número de reinas
board[n] <- Vector cuya (posición, indice) representa (fila, columna) de
una reina

function find_solutions(current_col)
```

```
    // Se ha encontrado una solución
    // al poder colocar las n reinas
    if (current_col == n_queens)
        print(board)
        return
    // Generamos las posibles combinaciones
    // de colocar una reina en una columna
    for row in [0:n_queens]
        board[current_col] = row
        // Si la posición (row, current_col) es válida
        // Colocamos otra reina en la siguiente columna
        if(is_valid_config(current_col))
            find_solutions(current_col+1)

find_solutions(0)
```

La solución presentada en el pseudocódigo anterior resuelve una serie de subproblemas (generar las posibles combinaciones de colocar una reina en una columna) recursivamente, y una vez que se logra resolverlos una nueva solución es obtenida.

Cada vez que se coloca una reina en una columna, es necesario verificar que la configuración generada hasta el momento en el tablero es válida. Para ello basta con comparar que la última pieza colocada no esté en la misma fila o en diagonal con otra pieza. Notemos que si dos reinas están en la misma diagonal podemos formar un triángulo rectángulo cuyos catetos tienen la misma dimensión:

```
function is_valid_config(current_col):
    // Verificamos las posiciones de todas las reinas colocadas
    // anteriormente con la última reina colocada
    for i in [0:current_col];
        // Si están en la misma columna
        if board[i] == board[current_col]
            return false
        // Si  están en la misma diagonal
        if curren_col - i == abs(board[i] - board[current_col])
            return false
    return true
```

Para resolver el problema de maximizar el número de reinas posibles dadas una serie de reinas fijas tenemos que realizar una serie de modificaciones al pseudocódigo anterior:

- Las soluciones son generadas colocando reinas de izquierda a derecha a través de las columnas. Si una reina no puede ser colocada en columnas previas dado que las reinas fijas lo impiden, el decidir no colocar una reina en una columna debe formar parte de la solución.
- Se debe contar el número de reinas colocadas en cada solución y quedarse con la que la maximice.

```
/*
    Búsqueda de soluciones mediante backtraking
    para maximizar el número de reinas con k reinas fijas
*/
n_queens <- Número de reinas
```

```
board[n, -1] <- Vector tablero, inicializado en -1 representa la ausencia de reina
fixed_cols = set() <- Columnas fijas en donde se han colocado las reinas

max_queens_placed = 0 <- Número máximo de reinas colocadas
best_solution[n] <- Mejor solución

function find_solutions(current_col, queens_placed)
    if (current_col == n_queens)
        // Si tenemos una mejor solución la guardamos
        if (max_queens_placed < queens_placed)
            max_queens_placed = queens_placed
            best_solution = board
        return
    // Si está columna es fija saltar
    if (fixed_cols.find(current_col)):
        find_solutions(current_col+1, queens_placed+1)
        return
    // Generamos las posibles combinaciones
    for row in [0:n_queens]
        board[current_col] = row
        if(is_valid_config(current_col))
            find_solutions(current_col+1, queens_placed+1)
    // No colocamos ninguna reina y generamos las próximas soluciones
    board[current_col] = -1
    find_solutions(current_col+1, queens_placed)

find_solutions(0, 0)
```

La función de verificación de la configuración del tablero ahora debe buscar sobre todo este para verificar con las reinas fijas y no solo hasta la columna actual.

```
function is_valid_config(current_col):
    // Esta vez se recorre todo el tablero
    for i in [0:n_queens];
        // No hay ninguna reina colocada o es la misma columna
        if board[i] == -1 || i == current_col
            continue
        if board[i] == board[current_col]
            return false
        if curren_col - i == abs(board[i] - board[current_col])
            return false
    return true
```

Implementación y resultados

Uno de los objetivos de este proyecto es realizar la implementación del algoritmo a través de la metodología de Programación Orientada a Objetos. Así, se definieron las siguientes clases

- Chess: Abstracción de la metodología de solución sobre un tablero de ajedrez de nxn.
- CCanvas: Abstracción de un tablero de ajedrez gráfico que implementa las funcionalidades necesarias para dibujar en pantalla una de las soluciones obtenidas usando las bibliotecas de *GTK* y *Cairo*.

El siguiente diagrama de clases en *UML* muestra las relaciones entre clases implementadas:

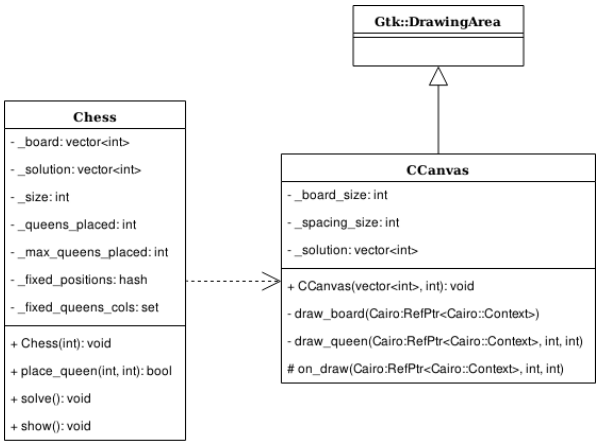

Fig 1. Diagrama de clases UML

Como ilustración se muestra la solución para un tablero de $8x8$ celdas con dos reinas fijas con la configuración siguiente:

```
Board size: 8
Number of fixed queens: 2
Coordenates for queen 1: 0 0
Coordenates for queen 2: 3 5

 Q  -  -  -  -  -  -  -
 -  -  -  -  -  -  Q  -
 -  -  -  Q  -  -  -  -
 -  -  -  -  -  Q  -  -
 -  -  -  -  -  -  -  Q
 -  Q  -  -  -  -  -  -
 -  -  -  -  Q  -  -  -
 -  -  Q  -  -  -  -  -

Maximun number of queens placed: 8
```

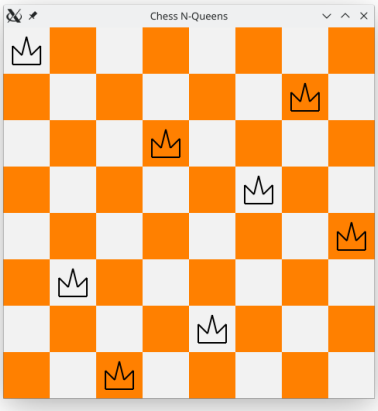

Fig 2. Solución para un tablero de 8x8 con 2 reinas fijas

Conclusiones

Dado que buscamos todas las soluciones al generar la combinación de las posibles casillas que puede ocupar una reina en cada columna, la explosión combinatoria del número de soluciones es grande (aunque debido a las restricciones del problema no crece tan rápidamente) haciendo de este un algoritmo lento y costoso mientras más grande es el tablero; por ejemplo para un tablero de $16x16$ el número de soluciones es $14,772,512$.

Una posible alternativa es utilizar algunas heurísticas junto con algoritmos evolutivos/genéticos para generar soluciones plausibles y poder reducir así la complejidad de este método de búsqueda exhaustiva como lo es el backtraking.