Capítulo 7

|  |  |  |
| --- | --- | --- |
| Jonathan Zinzan Salisbury Vega  *Universitat de les Illes Balears* Palma, España jonathan.salisbury1@estudiant.uib.cat | Joan Sansó Pericás  *Universitat de les Illes Balears* Palma, España joan.sanso4@estudiant.uib.cat | Joan Vilella Candia  *Universitat de les Illes Balears* Palma, España joan.vilella1@estudiant.uib.cat |

*La ramificación y poda es una técnica de programación muy útil para algoritmos con solución basada en árboles. En este documento se propone una solución al problema del puzle N-1 basado en dicha técnica utilizando un conjunto de distintas heurísticas y comparando el rendimiento de estas.*

Keywords—MVC, rendimiento, algoritmo, ramificación y poda, puzzle NxN.

# Introducción

Los algoritmos probabilísticos se han posicionado como una de las opciones más viables ante la solución de distintos problemas dado su funcionamiento. Son especialmente útiles cuando la complejidad de un problema tiene un coste asintótico muy elevado. Además, al ser “menos pesados” este tipo de algoritmos, permiten que puedan ser ejecutados en una mayor cantidad de máquinas, por lo que permite no depender de ordenadores especializados para la solución de problemas. Esto ha permitido que su uso en multitud de dispositivos móviles.

Ahora bien, estos algoritmos no son perfectos, ya que sino serían los únicos que se usarían. Los algoritmos probabilísticos fallan, lo cual es algo a priori extraño ya que no es un comportamiento que se haya considerado admisible hasta ahora. Pero si se logra controlar estos fallos, o al menos, acotarlos dentro de un percentil de error, se obtienen unos tiempos de ejecución considerablemente más cortos. Permitiendo así, múltiples ejecuciones hasta encontrar la solución.

Más adelante se ahondará en profundidad en los principios teóricos de este algoritmo, por el momento es importante destacar que se utilizará esta técnica nueva (desde el punto de vista del estudiante) para la identificación de banderas.

Para ello se han usado técnicas de colorimetría (ciencia que estudia el color) juntamente con los algoritmos probabilísticos. Con esto se busca clasificar cada uno de los píxeles de la imagen dentro de un rango de colores definidos. Una vez obtenidos estos datos, se comparará con la base de datos propia de las banderas.

Todo este proyecto se enmarca en el patrón de diseño MVC (Modelo Vista Controlador), permitiendo así aprovechar muchas de sus ventajas: mejor detección de errores, mayor escalabilidad del proyecto, código mejor estructurado, entre otros. Con el pequeño contra de que esto suponga una mayor complejidad a la hora de programar la solución.

# Contexto y entorno de estudio

Uno de los requisitos a la hora de realizar la práctica es el uso del lenguaje de programación Java. Además, se ha dado la opción de elegir entre varios IDE’s (*Integrated Development Environment*).

* NetBeans
* IntelliJ
* VS Code

En este caso se ha escogido el IDE de NetBeans por familiaridad de uso. Además, se utiliza una herramienta de control de versiones (Git). Más específicamente su versión de escritorio *Github Desktop* por su facilidad de uso mediante interfaz gráfica.

# Descripción del problema

# Solución Propuesta

En este apartado se comentará a modo de carácter general el planteamiento seguido en la solución implementada. Los detalles técnicos serán comentados en apartados posteriores, especialmente en la implementación del modelo.

# patrón mvc

El Modelo-Vista-Controlador (MVC) [1] es un patrón de diseño de software[[1]](#footnote-2) en el que se divide la lógica del programa de su representación gráfica, además se hace uso de un controlador para los eventos y comunicaciones entre las distintas partes.

Este patrón de arquitectura se basa en las ideas de reutilización de código y la separación de conceptos distintos, estas características pretenden facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento. Para ello se proponen la construcción de tres componentes:

* Modelo:

Es donde se almacena la información necesaria para la ejecución de la aplicación. Gestiona el acceso a dicha información mediante peticiones a través del controlador. También contiene los procedimientos lógicos que hagan uso de esa información.

* Vista:

Contiene el código que muestra la aplicación, es decir, que va a producir la visualización tanto de la interfaz del usuario como de los resultados

* Controlador:

Responde a eventos usualmente generados por el usuario y los convierte en comandos para la vista o modelo. Se podría decir que hace la función de intermediario entre la vista y el modelo encargándose de la lógica del programa.

Como se ha mencionado anteriormente se ha decidido utilizar este patrón debido a la facilidad que aporta al programar la separación de conceptos. Además, se obtiene una capa más de abstracción ya que es posible utilizar diferentes vistas con un mismo modelo.

Las ventajas principales del MVC son: Escalabilidad, facilidad de tratamiento de errores y reutilización de componentes. Existen otras ventajas, pero esta arquitectura se aprovecha en mayor medida en aplicaciones web, el cual no es este caso.

La principal desventaja que existe del patrón MVC es la complejidad que añade a la programación. Ya que, para el mismo problema, hay que modificar el acercamiento para que quepa dentro de esta arquitectura. Lo que implica una mayor complejidad.

Hoy en día, es muy común el uso de la programación

orientada a objetos (POO) como paradigma principal, por ello cada componente del patrón MVC suele implementarse como una clase independiente. A continuación, se explica que proceso se ha seguido y como se ha implementado cada parte.

## Implementación del Modelo

## Implementación de la Vista

Para la implementación de la vista se ha creado la clase *View*.

La clase *View* es la ventana principal del programa que contiene tanto los componentes de la interfaz de usuario como toda la lógica que hay detrás de estos componentes.

La ventana principal contiene los siguientes elementos:

* **JButtons**:
  + “Random Flag”: pone en la vista una bandera aleatoria con su nombre, para luego poder adivinarla.
  + “Guess Country”: hace las correspondientes llamadas al modelo para adivinar la bandera, actualiza los porcentajes de colores, y pone el país de la solución propuesta con su bandera.
* **JPanel**: dos JPanel para mostrar la bandera actual y la bandera de la solución propuesta
* **JLabel**: Dos JLabel para los nombres del país actual y el propuesto, y luego unos JLabels para los colores.
* **JProgressBar**: una para cada color, para mostrar el porcentaje de ese color en esa bandera.

Hay definidos varios métodos, pero solo explicaremos con mayor detalle los más interesantes y que aportan valor a la documentación:

### setFlagImage

Este método se encarga de poner la imagen y nombre del país pasados por parámetro en la ventana.

### setGuessImage

Este método se encarga de poner la imagen y nombre del país propuesto, así como actualizar las barras de porcentajes de los colores. Si la solución propuesta es correcta, el nombre del país se pondrá en verde. En caso contrario, se pondrá en rojo.

Este devuelve un entero, 1 si es la solución correcta y 0 si es incorrecto. Este valor no se usa para nada en el programa como tal, solo se usa para hacer los *tests* de rendimiento.

## Implementación del Controlador

Para la implementación se ha creado la clase *Controller* que contiene todos los métodos necesarios para poder comunicar ambas partes del patrón MVC.

En primer lugar, se han definido dos atributos de la clase que contienen las instancias del modelo y la vista, estos se reciben directamente como parámetros a través

del constructor. A continuación se ha definido el método start() que es al que debe llamar la aplicación principal para comenzar la ejecución de la aplicación.

Este método únicamente añade los listeners correspondientes al modelo y a la vista, posteriormente

hace la vista visible al usuario.

Finalmente hay definidos dos métodos que son los que se han proporcionado como listeners. Estos métodos se

ejecutarán cuando el modelo o la vista lancen el evento

pertinente. A continuación, se explican en mayor

detalle:

* viewActionPerformed():  
  Dependiendo del evento que se reciba, se hará lo siguiente:
  + - “Random Flag”: se genera una bandera aleatoria y se carga en el modelo (para los cálculos posteriores a la hora de adivinar). Tambíen se carga en la vista.
    - “Guess Country”: cargamos la base de datos de las banderas (si no se ha hecho ya antes), se calculan los porcentajes de la bandera con el número de pixeles de muestreo, se encuentra la bandera, y se actualiza la solución propuesta en la vista.

### Programación Concurrente

La programación concurrente es una forma de cómputo en la que el trabajo se divide en varios hilos de ejecución distintos que trabajan simultáneamente. Esto suele mejorar el rendimiento de una aplicación al poder realizar cálculos largos en un hilo aparte. Para este proyecto se ha decidido utilizar este concepto sobre el modelo para realizar los cálculos en segundo plano.

Con esto en mente, sabemos que todo lo que tiene que ver con la vista se ejecuta en un hilo aparte, usando el SwingUtilities.invokeLater().

# Estudio de Rendimiento

En este apartado se mostrarán los resultados de unas pruebas de rendimiento que se han realizado.

La variable principal en esta práctica era el número de muestras (píxeles) que se tomaban de las imágenes. Así que se ha probado de realizar ejecuciones con distintos números de muestras, para ver qué porcentaje de acierto tenían.que

El número de píxeles (N\_PIXELS) ha variado desde 10 hasta 50.000. Tenemos que recordar, que cuántos más píxeles se muestrean, más tarda el algoritmo, pero también más certero es.

E aquí los resultados:

Como se puede observar, sobrepasamos el 90% de acierto en las 2000 muestras. Teniendo en cuenta que las imágenes tienen entre 150.000 y 200.000 píxeles, eso es apenas un 0,1% de la cantidad total.

# Guía de usuario

A continuación, se explicará como ejecutar el programa mostrando el flujo del programa de cara al usuario y todas las funcionalidades de las que dispone.

# conclusiones

Esta práctica nos ha servido para entender lo útiles que son los algoritmos probabilísticos.

Teniendo una buena base de conocimientos sobre un problema, reducirlo a muestras sobre las que poder aplicar unos cálculos probabilísticos, puede reducir el coste del problema significativamente sin empeorar ni alterar los resultados del programa.

Como hemos podido ver en el estudio de rendimiento, con apenas un 0,1% de los píxeles de la imagen podemos acertar 9 de cada 10 banderas, ahorrándonos un 99,9% del procesamiento.

# Bibliografía

*15 Puzzle -- from wolfram MathWorld*. (s. f.). Wolfram Mathword. <https://mathworld.wolfram.com/15Puzzle.html>

GeeksforGeeks. (2020, 10 junio). Difference between Backtracking and Branch-N-Bound technique. <https://www.geeksforgeeks.org/difference-between-backtracking-and-branch-n-bound-technique/>

GeeksforGeeks. (2022). Branch and Bound Algorithm. <https://www.geeksforgeeks.org/branch-and-bound-algorithm/>

Gupta, S., & Bairagi, S. (s. f.). Evaluating Search Algorithms for Solving n-Puzzle. Sumitg. <http://sumitg.com/assets/n-puzzle.pdf>

Javatpoint. (s. f.). Branch and Bound. www.javatpoint.com. <https://www.javatpoint.com/branch-and-bound>

Latombe, J. (2011). Heuristic (Informed) Search. Standford University <http://ai.stanford.edu/~latombe/cs121/2011/slides/D-heuristic-search.pdf>

Marshall, J. (2005). Heuristic Search. Sarah Lawrence College <http://science.slc.edu/%7Ejmarshall/courses/2005/fall/cs151/lectures/heuristic-search/>

R. Kunkle, D. (2001). Solving the 8 Puzzle in a Minimum Number of Moves: An Application of the A\* Algorithm. Massachusetts Institute of Technology. <https://web.mit.edu/6.034/wwwbob/EightPuzzle.pdf>

Solving the 8-Puzzle using A\* Heuristic Search. (2009). IIT Kanpur <https://cse.iitk.ac.in/users/cs365/2009/ppt/13jan_Aman.pdf>

# AUTORES

Jonathan Salisbury Vega, nació en Costa Rica en el 2000. A los 17 años se graduó de bachillerato en el instituto de educación secundaria IES Pau Casesnoves. Actualmente está cursando el Grado de Ingeniería Informática en la Escuela Politécnica Superior de la Universidad de las Islas Baleares.

Joan Sansó Pericás, nació en Manacor en 2001. A los 18 años se graduó de bachillerato en el instituto de educación secundaria IES Manacor. Actualmente está cursando el Grado de Ingeniería Informática en la Escuela Politécnica Superior de la Universidad de las Islas Baleares.

Joan Vilella Candia, nació en Elche en el 2000. A los 18 años se graduó de bachillerato en el instituto de educación secundaria IES Inca. Actualmente está cursando el Grado de Ingeniería Informática en la Escuela Politécnica Superior de la Universidad de las Islas Baleares.

1. Un patrón de diseño es un conjunto de técnicas utilizadas para resolver problemas comunes en el desarrollo de software. [↑](#footnote-ref-2)