Máster en técnicas estadísticas

Proyecto fin de máster

Aplicación de algoritmos heurísticos para optimizar el coste de doblaje de películas

Autor: Alberto Caldas Lima

Directores: Silvia María Lorenzo Freire

María Luisa Carpente Rodríguez

A Coruña, a 4 de septiembre de 2014

Información general

Título del proyecto:	Aplicación de algoritmos heurísticos para optimizar el coste de doblaje de películas
Clase del proyecto:	Proyecto de investigación
Nombre del autor:	Alberto Caldas Lima
Directores del proyecto:	María Luisa Carpente Rodríguez Silvia Lorenzo Freire
Miembros del tribunal:	
Fecha de lectura:	
Calificación:	

Autorización de entrega

D.ª María Luisa Carpente Rodríguez y D.ª Silvia Lorenzo Freire

CERTIFICAN

Que el proyecto titulado **"Aplicación de algoritmos heurísticos para optimizar el coste de doblaje de películas"** ha sido realizado por Alberto Caldas Lima, con D.N.I. 77.411.080-A, bajo la dirección de D.ª María Luisa Carpente Rodríguez y D.ª Silvia Lorenzo Freire. Esta memoria constituye la documentación que, con nuestra autorización, entrega dicho alumno para optar a la titulación de Máster en técnicas estadísticas.

Firmado.

D.ª María Luisa Carpente Rodríguez

D.ª Silvia Lorenzo Freire

A Coruña, a 4 de septiembre de 2014

Resumen

Las películas que van a ser dobladas se dividen en fragmentos cortos (llamados takes) donde uno o más actores graban un conjunto pequeño de frases. El número de takes que se pueden grabar en un día es limitado, con lo que una película se dobla normalmente en más de un día de trabajo. La organización del trabajo para el doblaje de la película corre a cargo del director responsable de la película en el estudio, y este debe tener en cuenta a la hora de realizar la programación que cada actor cobra una cantidad por take y otra por día de trabajo en el estudio. Las programaciones para las películas son hechas de forma manual intentando minimizar el coste de las mismas.

Con el objetivo de mejorar el proceso de planificación, tanto en coste para el estudio como en tiempo de elaboración de la planificación, se aplicarán diversos algoritmos heurísticos y se compararán los resultados de los mismos. Adicionalmente, se plantearán escenarios alternativos que permitan realizar la planificación ajustándose a ciertas restricciones que se puedan dar en un caso real.

La ejecución de los algoritmos se realizará usando una aplicación de escritorio que se construirá con el objetivo de facilitar el trabajo del director. En la interfaz de la aplicación se podrán configurar de forma sencilla los parámetros de la película así como los del algoritmo. Además, se respetarán los formatos de entrada y salida usados por el estudio.

Palabras clave

- Programación lineal
- Algoritmos heurísticos
- Problema de optimización
- Recocido simulado
- Búsqueda tabú
- Algoritmos genéticos

Índice general

Índice de figuras

Algoritmos

1

INTRODUCCIÓN

En este apartado se explicará la motivación que ha llevado a realizar este trabajo y lo que se pretende conseguir en el mismo. Lo planteado en este apartado deberá ser comprobado al finalizar el trabajo para revisar que se han alcanzado todos los objetivos que se habían marcado.

1.1. Motivación

Este trabajo surge a partir del estudio de un caso real que se ha planteado, donde se busca optimizar la programación del doblaje de películas con el fin de que el estudio de grabación minimice los costes de producción. El problemas ya ha sido tratado en [11], donde se plantea el modelo y se propone un algoritmo de recocido simulado para buscar la programación óptima.

Usando [11] como referencia, se pretende comparar el algoritmo implementado con otras filosofías heurísticas desarrollando nuevos algoritmos para resolver el problema. Además, se quiere crear una aplicación de escritorio que permita a los trabajadores del estudio utilizar los algoritmos implementados en este trabajo para conseguir programaciones de manera rápida y cómoda. De esta forma, se busca obtener una aplicación que permita dar solución al problema real al que se enfrenta el estudio de doblaje a la hora de hacer la programación de sus películas.

1.2. Alcance y objetivos

El alcance de este trabajo se centrará en la utilización de varias técnicas heurísticas para buscar la mejor solución y compararlas entre ellas, así como en desarrollar una aplicación que permita manejar los algoritmos desarrollados de manera cómoda. Realizando estas dos tareas se conseguirá una aplicación capaz de dar una solución al problema que pueda ser utilizada en el estudio de doblaje.

Debe tenerse en cuenta que en este trabajo se tratará el problema tal y como se propone en [11], sobre el que se aplicarán los algoritmos heurísticos planteados. Posteriormente se plantearán variaciones sobre el problema que se ha tratado, las cuales podrían adecuarse más al caso real, aunque el estudio de grabación actualmente contempla únicamente el problema inicial. No entra en el alcance de este trabajo el estudiar dichos modelos planteados ni adaptar los algoritmos o la interfaz a los mismos, aunque sí se dejarán implementadas las bases para aplicar algunas variaciones del modelo o nuevas técnicas heurísticas si se decidiera continuar con este trabajo.

Como se ha explicado en el alcance del problema, el objetivo principal de este trabajo es dar solución al problema de programación al que se enfrenta el estudio de doblaje, con lo que habrá que centrarse en dos puntos clave: estudio de las técnicas para obtener la solución al problema y construcción de una aplicación que permita aplicar las técnicas anteriores de manera sencilla. Teniendo esto en cuenta, se plantean los siguientes objetivos detallados:

- Basándose en [11], se planteará el modelo matemático del problema y se implementará el algoritmo recocido simulado para el mismo. Este algoritmo se tomará como base para el problema.
- Construir un algoritmo de búsqueda tabú que se ajuste al problema.
- Elaborar un algoritmo de tipo genético para resolver el problema.
- Realizar pruebas sobre los algoritmos para confirmar su correcto funcionamiento y determinar la adecuación de cada filosofía heurística al problema.
- Construir una interfaz de usuario que permita a los trabajadores del estudio obtener programaciones para las películas de forma cómoda empleando el trabajo,

anteriormente realizado. Para no modificar en exceso el método actual de trabajo se intentará mantener el formato de entrada y de salida con el que trabaja actualmente el estudio.

- Se plantearán posibles variaciones del problema que se podrán tener en cuenta en futuros trabajos, con el objetivo de optimizar el trabajo de programación que realiza actualmente el estudio.

1.3. Estructura de la memoria

La memoria de este trabajo se estructurará en seis bloques diferentes y cuyo orden pretende ir introduciendo al lector en el problema y los conceptos de programación lineal necesarios para posteriormente detallar el trabajo realizado y los resultados obtenidos. Los bloques detallados son los siguientes:

- El primer bloque (en el que nos encontramos) constituye la introducción al problema que se trata.
- En el segundo bloque se explican los conceptos previos que deben ser conocidos para entender por qué se ha planteado este trabajo.
- El tercer bloque contiene el problema concreto y el modelo de programación lineal que será tratado en este trabajo.
- El cuarto bloque detalla los algoritmos que se han desarrollado y todo lo referente a la configuración y parametrización de los mismos, así como la adecuación de cada uno al problema.
- En el quinto bloque se explica cómo se ha llevado a cabo la implementación y se exponen algunos puntos a tener en cuenta sobre la misma. Además, se detallan los resultados obtenidos con la aplicación.
- En el sexto y último bloque se detallan las conclusiones generales que se pueden obtener de este trabajo y se exponen posibles líneas de trabajo empleando este como base.

Además de los bloques explicativos del trabajo, se incluye un anexo con el manual de uso de la aplicación de escritorio implementada.

2

CONCEPTOS PREVIOS

2.1. Programación lineal

2.1.1. Introducción

Los problemas de programación matemática son modelos creados para resolver problemas de decisión, donde normalmente se busca optimizar un objetivo. Para definir un problema de este tipo se deben especificar:

- Las variables del problema, que son los valores que podemos modificar para obtener la solución que se busca.

- Las restricciones, que definen condiciones que las variables deben cumplir para que la solución sea factible. Se pueden presentar tres tipos de restricciones: restricciones de igualdad, restricciones de tipo menor o igual y restricciones de tipo mayor o igual.

- La función objetivo define la bondad de la solución que se obtiene, es decir, la solución será considerada como mejor o peor según el valor proporcionado por la función objetivo, que habrá que maximizar o minimizar.

Estos problemas pueden ser deterministas o estocásticos, dependiendo de si se conoce con certeza el valor de todos los parámetros del problema o si depende de la aleatoriedad. Los problemas de tipo determinista pueden ser a su vez de dos tipos:

- Problemas de programación no lineal (PPNL): este tipo de problemas incluyen alguna restricción no lineal o la función objetivo no es una función lineal.

- Problemas de programación lineal (PPL): todas las restricciones y la función objetivo del problema son lineales. Si todas las variables son de tipo entero se denominan problemas de programación entera. Este tipo de problemas será tratado en las secciones posteriores.

2.1.2. Formulación de un problema de programación lineal

Un problema de programación surge al intentar asignar una serie de recursos de forma óptima, es decir, maximizando las ganancias o minimizando los costes asociados.

Un problema de programación se dice lineal si cumple las siguientes características:

- La función lineal que determina el objetivo (función objetivo) se puede expresar como una ecuación lineal sobre las variables del problema que se tendrá que maximizar o minimizar, es decir, la función objetivo siempre puede ser expresada de la siguiente forma:

$$Z = \sum_{i=0}^{n} c_i x_i$$

donde:

- c_i simboliza el coste de la variable i del problema en el objetivo y es un dato del problema.

- x_i son las variables de decisión del problema y representan si una acción se ha de realizar o no en el problema. Son los únicos valores desconocidos en el modelo.

- Las restricciones del problema se pueden escribir como un conjunto de ecuaciones o inecuaciones lineales usando las variables del problema. Pueden ser de tres tipos:

$$\sum_{j=0}^{n} a_{ij}x_i = b_i$$

$$\sum_{j=0}^{n} a_{ij}x_i \leq b_i$$

$$\sum_{j=0}^{n} a_{ij}x_i \geq b_i$$

2.1.3. Solución de un problema de programación lineal

Se definen como soluciones factibles de un problema de programación lineal todas aquellas soluciones que verifiquen las restricciones del problema. El conjunto de soluciones factibles constituye la región de factibilidad del problema.

La solución óptima de un problema de programación lineal es aquella que es factible y verifica que su función objetivo es mejor o igual que la de cualquier otra solución factible.

Sea F la región de factibilidad de un problema de programación lineal con función objetivo $f(x)$, que se tratará de minimizar. Una solución $s \in F$ es solución óptima al problema si $\forall x \in F,\ f(x) \geq f(s)$.

Un problema de programación matemática puede que no tenga ninguna solución factible, con lo que la región de factibilidad es vacía, y por tanto no existe solución óptima. En caso de que esta región no esté vacía, el problema puede no tener solución óptima finita. Por esto, los problemas de programación matemática se puede clasificar en:

$$Prob.\,de\,prog. \begin{cases} Sin\,solución\,factible \\ Con\,solución\,factible \begin{cases} Con\,solución\,óptima \begin{cases} Única \\ Infinitas \end{cases} \\ Solución\,no\,acotada \end{cases} \end{cases}$$

El conjunto de restricciones define la región de factibilidad de un problema de programación lineal. La elección de las restricciones es clave en un PPL, ya que éstas pueden restringir tanto la región que pueden convertir el problema en no factible.

2.2. Métodos de solución de problemas de programación lineal

Existen múltiples técnicas de resolución de problemas de programación lineal que se agrupan en métodos de resolución exacta y técnicas heurísticas. En este apartado se describirán brevemente algunas de las técnicas más conocidas que se podrían haber empleado para resolver el problema que se plantea en este trabajo.

2.2.1. Métodos de solución exacta

Los métodos de solución exacta siempre obtienen la mejor solución para un problema de optimización. La gran desventaja de estas técnicas es el consumo, tanto de recursos como de tiempo, para alcanzar dicha solución, ya que su filosofía de búsqueda se basa en analizar todo el espacio de soluciones. En los siguientes apartados se describirán brevemente algunas de las técnicas de resolución exacta que se podrían aplicar al problema en cuestión. Una descripción más detallada de estos métodos se puede encontrar en Winston, W. L. [13], Taha, H. A. [12] y Hillier, F. and Lieberman, G. [9].

2.2.1.1. Ramificación y acotación (Branch and bound)

El algoritmo de ramificación y acotación es usado para resolver problemas de programación entera. El algoritmo es capaz de explorar todo el espacio de búsqueda sin recorrer de forma explícita todos los estados válidos.

El objetivo de este algoritmo es minimizar el coste de una función $f(x)$ en un espacio de búsqueda S. El funcionamiento del algoritmo se basa en dos fases:

- Ramificación: el espacio de búsqueda S se divide en varios subconjuntos S_1, $S_2, \ldots$, con lo que el mínimo en S será el mínimo de los mínimos en cada subconjunto S_i. Los diferentes subconjuntos S_i creados constituirán una rama del árbol de búsqueda.
- Acotación: si el coste en la menor rama de un árbol A es mayor que el coste de la rama padre de un árbol B, entonces se puede descartar con seguridad la rama del árbol A.

El problema finaliza cuando el árbol de búsqueda se completa sin poder ramificar más, y la solución óptima es la mejor solución entera obtenida.

2.2.1.2. Planos de corte de Gomory

El método de los planos de corte de Gomory funciona iterando sobre una solución calculada para el problema, que en un primer paso se obtiene resolviendo el problema de forma tradicional. Si la solución obtenida, $x*$, es entera, esa será la solución óptima al problema original. En caso de no ser entera, se construye un plano de corte, un hiperplano $\alpha^t x = \beta$, que divide el conjunto de posibles soluciones, X, en dos subconjuntos: uno que contiene la solución no entera $x*$ y el otro que contiene el conjunto de todas las soluciones enteras al problema. Este proceso se repite generando planos de corte hasta encontrar una solución $x*$ entera, si existe.

Para lograr que se reduzca el conjunto de soluciones se introduce en cada paso una restricción del tipo $\alpha^t x \leq \beta$, que verifican todas las soluciones enteras del problema y elimina del conjunto de soluciones posibles algunas de las no enteras.

2.2.1.3. Generación de columnas

El método de generación de columnas se usa en problemas de optimización en los que el número de variables del problema es muy elevado. Cuando el número de variables es muy grande, muchas de estas variables pueden no tenerse en cuenta, ya que no influirán en la solución final. En este hecho se basa el funcionamiento del método, que trata

de reducir el problema original a un problema más sencillo reduciendo el número de variables.

Para saber qué variables son las que se debe tener en cuenta, se usa un problema auxiliar de optimización que descubra las variables a mínimo coste. El problema auxiliar se realiza hasta que no sean descubiertas más variables candidatas a entrar en la base, con lo que ya está el nuevo problema reducido creado.

2.2.2. Métodos basados en heurísticas

Un método heurístico permite obtener soluciones de alta calidad con un coste computacional razonable, aunque no se garantice la optimalidad de la solución o su factibilidad, y no se pueda determinar lo cerca que está dicha solución del óptimo.

El uso de los métodos heurísticos, en contraposición con los exactos, viene dado por la diferencia de coste computacional requerido, ya que los algoritmos exactos requieren de un coste mucho más elevado, que muchas veces es inviable. Para estos problemas el uso de heurísticas es la única alternativa.

A continuación se explicarán las filosofías de los tres algoritmos heurísticos que serán empleados a lo largo de este trabajo.

2.2.2.1. Recocido simulado

Los algoritmos de recocido simulado se basan en el proceso de enfriamiento que sufren los metales. Cuando la temperatura es alta, los átomos del metal se mueven con más libertad, y a medida que el metal se va enfriando, sus átomos van perdiendo energía hasta asentarse en el estado sólido de mínima energía. Análogamente a este proceso, los algoritmos de recocido simulado aceptarán estados que empeoren el estado actual con una cierta probabilidad que será mayor cuanto más alta sea la temperatura. Las leyes de la termodinámica dicen que la probabilidad de que esto ocurra viene dada por $P(\Delta E) = e^{-\frac{\Delta E}{kT}}$, donde k es la constante de Boltzmann. Para el caso del algoritmo heurístico, podemos omitir esta constante, ya que el incremento de energía dependerá de la función objetivo del algoritmo, y solo se busca construir un método que funcione de forma similar.

El pseudocódigo del Algoritmo 2.1 muestra el funcionamiento básico de una iteración de este algoritmo.

Algoritmo 2.1 Pseudocódigo de iteración del algoritmo de recocido simulado

```
T = TemperaturaInicial;
SolucionActual = SolucionInicial;
while (T > TemperaturaMinima)
{
   NuevaSolucion = GenerarVecino(SolucionActual);
   IncrementoEnergia = Energia(NuevaSolucion) - Energia(SolucionActual);
   if (Energia(NuevaSolucion) < Energia(MejorSolucion))
      MejorSolucion = NuevaSolucion;
   if (Uniforme(0,1) < exp(-IncrementoEnergia / T))
      SolucionActual = NuevaSolucion;
   T = T * FactorEnfriamiento;
}
return MejorSolucion;
```

Este tipo de algoritmos permiten aceptar soluciones peores a la actual cuando la temperatura es alta, minimizando este tipo de cambios a medida que se enfría la temperatura. Esto hace que se explore más región del espacio que con otros algoritmos heurísticos, aunque también se pueden estancar en una parte del espacio cuando este es grande. Por ello, estos algoritmos pueden incluir también mecanismos de escape cuando la solución actual se repita muchas veces, evitando así posibles mínimos locales.

2.2.2.2. Búsqueda tabú

La búsqueda tabú es una técnica de búsqueda local que se ha depurado, incluyendo estructuras de memoria para evitar repeticiones en las soluciones. Este método usa una lista tabú que contiene aquellos movimientos en la solución previamente ejecutados, y su objetivo es evitar que se repitan estos movimientos, ya que es probable que lleven a soluciones previamente alcanzadas. El tamaño de dicha lista es constante con lo que, tras un número de iteraciones, un movimiento dejará de ser tabú.

La lista tabú del algoritmo contiene aquellos movimientos que se han ejecutado recientemente para poder evitar que sean repetidos, y con ello evitar el ciclado del algoritmo. Esta lista sigue una disciplina FIFO (First In First Out) con un tamaño fijo, de modo que un movimiento ejecutado previamente puede dejar de ser tabú.

El algoritmo de búsqueda tabú se ejecuta durante un número de iteraciones sobre una solución inicial dada, y la mejor solución encontrada durante esas iteraciones será

la solución devuelta. En cada iteración se tiene una solución actual $s \epsilon S$, donde S es el conjunto de soluciones al problema. Sea $N(s)$ el conjunto de todos los vecinos de s, obtenidos al ejecutar todos los posibles movimientos permitidos sobre s. En cada iteración se selecciona un conjunto de vecinos $V(s)$ tal que $V(s) \subseteq N(s)$. A cada nueva solución vecina seleccionada se le aplica la función objetivo $F(x)$, que es la encargada de medir la bondad de la solución. Se elige s' tal que $F(s') \geq F(t), \forall t \epsilon V(s)$ como solución candidata a solución actual. Para que una solución candidata s' sea tomada como nueva solución actual debe suceder, que el movimiento que ha generado la solución s' no es un movimiento tabú, o que es un movimiento tabú pero satisface el criterio de aspiración del algoritmo, donde el criterio de aspiración es una función que determina si un movimiento tabú es lo suficientemente bueno para ser aceptado. En caso de que no se acepte la solución s' se buscará una solución $s'' \epsilon V(s)$ tal que $F(s'') \geq F(t), \forall t \epsilon V(s) \setminus \{s\}$. Este proceso se repetirá hasta que se acepte una solución, o hasta que todas las soluciones seleccionadas como vecinos no sean válidas, en cuyo caso se tomará como solución actual s'.

Este algoritmo puede estar muy limitado a una región del espacio, con lo que se hace muy dependiente de la solución inicial tomada. Para evitar esta dependencia se puede ejecutar sobre diversas soluciones iniciales, o se puede desarrollar un mecanismo de escape que permita trasladar la región de búsqueda a otro lugar del espacio de búsqueda total.

2.2.2.3. Algoritmos genéticos

Los algoritmos genéticos se basan en la teoría de la evolución propuesta por Darwin, según la cual solo los individuos más aptos de la población sobreviven, al ser capaces de adaptarse mejor a los cambios que se producen en el entorno; esto se consigue mediante la transmisión a la descendencia de los genes responsables de estos cambios. La aplicación más común de los algoritmos genéticos es la solución de problemas de optimización.

Para construir un algoritmo genético es necesario definir la función de aptitud y la codificación de las respuestas al problema. La función de aptitud, que es la función objetivo del problema a optimizar, busca penalizar las malas soluciones y premiar las buenas, de manera que sean estas últimas las que se propaguen con más rapidez.

El funcionamiento de un algoritmo genético simple se muestra en el Algoritmo 2.2.

Algoritmo 2.2 Pseudocódigo del algoritmo genético

1. Se genera de forma aleatoria una población inicial constituida por un conjunto de cromosomas que representan las soluciones posibles del problema.

2. A cada uno de dichos cromosomas se le aplica la función de aptitud para conocer lo buena que es la solución.

3. Se escogen los mejores cromosomas, en función del valor obtenido, que serán cruzados para generar una nueva población. Existen diferentes métodos de selección de cromosomas, siendo los más comunes los métodos de ruleta y de torneo.

4. Una vez hecha la selección, se cruzan los individuos seleccionados, obteniendo los descendientes que formarán la población de la siguiente generación. Para realizar este cruce existen diversos métodos, pero los más usados son los de uno y dos puntos de cruce. Normalmente, el cruce se asocia a un porcentaje que indica la frecuencia con la que se realizará, y cuyo valor suele ser mayor de un 60 %. De este modo, no todas las parejas de cromosomas se cruzarán, y habrá alguna que se transmita de forma íntegra a la siguiente generación. Además de la selección y el cruce, también se usa la mutación, que consiste en un cambio en uno de los genes de un cromosoma elegido aleatoriamente y que consigue así introducir nuevo material cromosómico en la población. Al igual que el cruce, se asocia con un porcentaje de frecuencia, aunque a diferencia de esta es un porcentaje bajo, siendo normalmente inferior a un 5 %.

5. Para detener el algoritmo, suelen usarse dos criterios: ejecutar el algoritmo un número máximo de iteraciones, o detenerlo cuando se estabilice la población (es decir, la mayoría de los individuos tengan la misma función de aptitud).

Las ventajas que tienen los algoritmos genéticos respecto a otras técnicas de búsqueda son que operan con operadores probabilísticos (al contrario que otras técnicas que usan operadores deterministas), no necesitan conocimientos específicos acerca del problema a resolver, operan simultáneamente con varias soluciones y presentan menos problemas con los máximos locales que las técnicas tradicionales. Por el contrario, tienen el inconveniente de que pueden tanto converger prematuramente, como tardar mucho en hacerlo (llegando en algunos casos a no converger) dependiendo de los parámetros que se utilicen.

3

EL CASO DE ESTUDIO

En este capítulo se presentará y contextualizará el problema que se va a tratar en este trabajo. Se definirá la motivación del mismo y el caso real en el que está basado. Posteriormente, se definirá el modelo matemático que se tomará como base para este trabajo.

3.1. Contexto del caso de estudio

El caso de estudio que se va a tratar en este trabajo se basa en el trabajo realizado en un estudio de doblaje con el fin de convocar a los actores para doblar una película minimizando los costes que esto conlleva.

El proceso de doblaje parte de un guión escrito y dividido en unidades de grabación, denominadas take o toma. Cada unos de estos takes está compuesto por varias frases cortas intercambiadas entre los personajes de las películas, aunque pueden existir takes sin diálogos, ya que toda la película tiene que estar estructurada en takes y existen partes en las que los personajes no intervienen. En este caso de estudio sólo estamos interesados en los takes donde los actores intervengan, por tanto, no se tendrán en cuenta los takes sin intervenciones.

Con el guión dividido en takes, el estudio es el encargado de realizar la programación

de las convocatorias de los actores. Cada convocatoria se corresponde con un día de grabación del estudio, y en cada convocatoria se puede grabar un número limitado de takes que se decide de antemano según la disponibilidad del estudio de grabación. En cada convocatoria se puede llamar a uno o más actores para que participen en el rodaje, y un take puede ser grabado en más de una convocatoria, es decir, una parte de los actores que participan en el take pueden ir en una convocatoria y otra parte en otra. Sin embargo, el proceso de fusión de un take conlleva un coste, con lo que un take no puede ser dividido en más partes de un máximo, que se fija al comienzo de la programación y suele ser de 3 o 4 divisiones por take. Además de esto, el estudio dispone de un número máximo de días para la grabación de la película, y por tanto, en ningún caso se puede superar este límite.

Siguiendo estas pautas, y teniendo en cuenta que cada actor cobra un fijo por cada convocatoria a la que es llamado y una cantidad por cada take que graba, el estudio trata de construir una programación que minimice el coste total del doblaje de la película. Como el número de takes que realiza cada actor es fijo, el estudio tratará de minimizar el número de convocatorias totales (la suma de todas las convocatorias de los actores) como prioridad. A igual número de convocatorias se tienen en cuenta otros aspectos que conlleven costes para el estudio, y en este caso tenemos el número de divisiones de take. Una vez definido el aspecto económico, se introduce un tercer parámetro de bondad de ajuste de una solución, que es la diferencia de takes existente entre días de doblaje. Igualando el número de takes que se realiza en cada convocatoria se minimiza el riesgo de que las sesiones se alarguen demasiado debido a que unos días se graba mucho más que otros, y que se puedan llegar a producir cambios en las programaciones por los retrasos.

3.2. El modelo

- Sea A el conjunto de actores, con índice $a \epsilon A$.
- Sea S el conjunto de sesiones de grabación, con índice $s \epsilon S$.
- Sea T el conjunto de takes, con índice $t \epsilon T$.
- Sea n_s el número máximo de takes permitidos en la sesión de grabación s, con $n_s \epsilon \mathbb{N}$. Para el modelo original tomaremos que todas las convocatorias tienen el mismo número de takes máximo.

- Sea M es el número máximo de sesiones de grabación en las que un take puede ser dividido, con $M \epsilon \mathbb{N}$.
- Para todo $a \epsilon A$ y $t \epsilon T$, $\delta_{at} = \begin{cases} 1, \textit{ si el actor a interviene en el take t} \\ 0, \textit{ en otro caso} \end{cases}$

Las variables de decisión son las siguientes variables binarias:

- Para todo $a \epsilon A$ y $s \epsilon S$, $x_{as} = \begin{cases} 1, \textit{ si el actor a está en la sesión s} \\ 0, \textit{ en otro caso} \end{cases}$
- Para todo $t \epsilon T$ y $s \epsilon S$, $z_{ts} = \begin{cases} 1, \textit{ si el take t está en la sesión s} \\ 0, \textit{ en otro caso} \end{cases}$
- Para todo $a \epsilon A$, $t \epsilon T$ y $s \epsilon S$, $y_{ats} = \begin{cases} 1, & \textit{si el actor a interviene} \\ & \textit{en el take t en la sesión s} \\ 0, & \textit{en otro caso} \end{cases}$

Definidas estas variables, el problema de programación lineal sería el siguiente:

$$min\, z = \sum_{a \epsilon A} \sum_{s \epsilon S} x_{as}$$

Sujeto a:

- $\sum_{s \epsilon S} y_{ats} = \delta_{at} \quad \forall a \epsilon A, \forall t \epsilon T$
- $\sum_{t \epsilon T} z_{ts} \leq n_s \quad \forall s \epsilon S$
- $\sum_{s \epsilon S} z_{ts} \leq M \quad \forall t \epsilon T$
- $y_{ats} \leq z_{ts} \quad \forall a \epsilon A, \forall t \epsilon T, \forall s \epsilon S$
- $y_{ats} \leq x_{as} \quad \forall a \epsilon A, \forall t \epsilon T, \forall s \epsilon S$

3.2.1. Aclaraciones sobre el modelo

Existen algunos aspectos que deben ser aclarados en la formulación del problema:

- La función objetivo del modelo matemático únicamente tiene en cuenta el número total de convocatorias de los actores, omitiendo de esta forma el número de divisiones máxima de take y la diferencia de número de takes entre sesiones (que se especificaban en la descripción del problema). Esto se debe a que el objetivo fundamental del problema es minimizar el número de convocatorias, pero se incluyen los otros factores en los algoritmos heurísticos para depurar las programaciones con el mismo número de convocatorias. De esta forma, se define una función objetivo lexicográfica siguiendo el orden:

$$\text{Convocatorias} > \text{Divisiones máximas} > \text{Diferencia de takes.}$$

- En el modelo se introduce la variable M como el número máximo de sesiones en las que puede aparecer un take. Esta variable limita el número máximo de divisiones de un take en el problema, y al tratarse de uno de los parámetros que se introducen en la función objetivo, se ha decidido que este parámetro se definirá siempre como el número máximo de días en los que se puede crear la programación.

4

ALGORITMOS

En este capítulo se explicará todo lo referente a los algoritmos que se han desarrollado para dar solución al problema planteado.

En primer lugar se definirán ciertos conceptos generales necesarios para los algoritmos heurísticos y se explicará qué opciones de estos algoritmos se han implementado en este trabajo para cada uno de ellos.

Posteriormente se pasará a detallar los algoritmos que se han implementado en este trabajo para intentar dar solución al modelo planteado. Para cada uno de los algoritmos se comenzará explicando la motivación que ha llevado a aplicarlo al problema, para luego exponer los componentes y posibles configuraciones del mismo. A continuación, se incluirá el pseudocódigo del algoritmo para ayudar a la comprensión de su funcionamiento, y para acabar se discutirá la adecuación del algoritmo al problema, y su correspondiente modelo, planteado según los resultados obtenidos.

4.1. Conceptos generales

Los algoritmos desarrollados utilizan conceptos comunes que pueden ser implementados de diferentes formas. A continuación se definen estos conceptos y las implementaciones que se han realizado sobre ellos.

4.1.1. Solución inicial

Los algoritmos de optimización parten de una solución o soluciones iniciales y a partir de la misma iteran para buscar la solución óptima. El punto de inicio es determinante a la hora de encontrar la solución inicial, ya que un punto inicial próximo a la misma hará que el algoritmo converja más rápido, y un punto alejado de la misma puede llevar al mismo algoritmo a no converger.

Para este problema en concreto se han elaborado dos métodos de obtención de solución inicial que permiten obtener soluciones factibles, lo que implica que si existe la solución inicial el problema tiene solución. En caso de que uno de estos métodos no encuentre una solución inicial factible, el problema no es factible con los parámetros establecidos.

4.1.1.1. Solución por actores

Se ordenan los actores que intervienen en la película por el número de takes en los que participan. Siguiendo la ordenación anterior, para cada actor se añaden sus takes al día de trabajo actual hasta alcanzar el número máximo de takes permitidos para una jornada. En el momento que esto ocurra, se continuará el proceso empezando en un nuevo día de trabajo. Cada vez que se incorpora un nuevo take a un día de trabajo se incluye en dicha convocatoria a todos los actores que participan en el mismo.

La ordenación inicial se puede realizar de forma ascendente o descendente, lo que resulta en dos formas diferentes de obtención de solución inicial.

Este método de obtención de soluciones iniciales es determinista, ya que siempre obtendrá la misma solución para el mismo problema.

4.1.1.2. Solución aleatoria

Los actores que participan en la película se van escogiendo de forma aleatoria y añadiendo a una convocatoria, incluyendo todos los takes en los que participa en dicha convocatoria. Todos los actores que participen en cada take que se vaya añadiendo a una convocatoria se añaden también a dicha convocatoria. Se repite este proceso hasta completar el número máximo de takes por jornada de trabajo, en cuyo caso se añadirán los takes siguientes a un nuevo día de grabación.

Este método de obtención de soluciones iniciales no es determinista, es decir, ante el mismo problema de entrada puede obtener soluciones diferentes.

4.1.2. Movimientos

Los movimientos son acciones que se realizan sobre una solución previamente calculada con el objetivo de modificarla en busca de una solución mejor o un nuevo punto donde empezar a buscar mejores soluciones. Existen diferentes tipos de movimientos que se pueden realizar sobre la misma solución, y se pueden utilizar con diferentes objetivos.

A continuación se definen los tipos de movimientos que serán usados en los algoritmos desarrollados.

4.1.2.1. Movimiento de actor

Este movimiento tratará de unificar dos convocatorias distintas para el mismo actor en una única sesión, y, en caso de no ser factible fusionar ningún par de convocatorias, mover una convocatoria del actor a una sesión en la que no se encuentre convocado dicho actor. Para que este movimiento se pueda ejecutar, la sesión de destino tiene que contener todos los takes en los que interviene el actor en la sesión de origen, o tener hueco suficiente para incluir todos los takes que no están presentes.

El movimiento se realiza eligiendo un actor al azar entre todos los presentes en la película y se determinan las sesiones en las que el actor está convocado y en las que no. En primer lugar, se seleccionarán de forma aleatoria dos sesiones en las que el actor está presente, y se tratarán de unificar. En caso de no ser posible repetimos el proceso para todos los pares posibles de sesiones. La Figura 4.1 muestra este proceso del movimiento.

Si no se ha conseguido realizar el movimiento para ningún par, se crean nuevos pares de sesiones, esta vez con una en la que sí esté presente y una en la que no, buscando mover todos los takes en los que el actor interviene a la nueva sesión. En caso de no ser posible, repetimos el proceso para todos los pares. En la Figura 4.2 se representa dicho funcionamiento.

Si el movimiento no se ha podido realizar para ningún par, se elige un nuevo actor y se repite el proceso. En caso de que no se pueda realizar para ningún actor, el movimiento

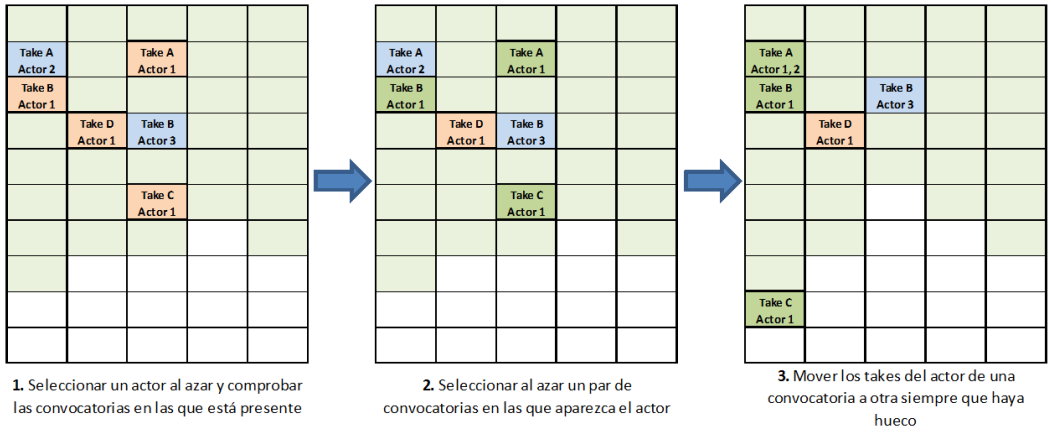

Figura 4.1: Movimiento de actor fusionando convocatorias

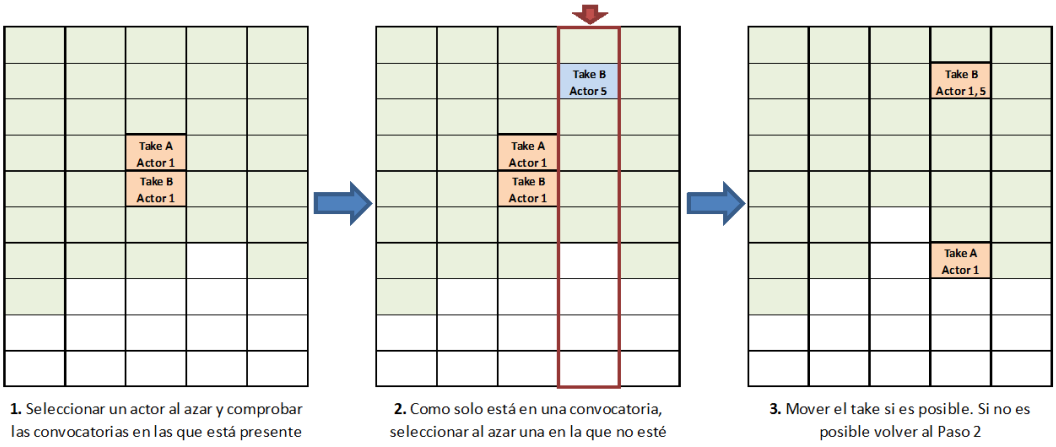

Figura 4.2: Movimiento de actor a nueva convocatoria

no es posible para la programación dada.

El Algoritmo 4.1 muestra el pseudocódigo de este movimiento. Nótese que al añadir un actor, este se añade únicamente si no estaba ya convocado en dicha sesión. Con los takes ocurre lo mismo, solo se añaden los que no estuvieran presentes en dicha sesión.

Se plantea también una variación de este movimiento en el que se trata de ponderar la influencia de una actor a la hora de ser escogido. En el movimiento original todos los actores tienen la misma probabilidad de ser escogidos para realizar el movimiento, sin embargo, los actores que están convocados más días tiene más probabilidad de poder unificar sus convocatorias. Por ello, como alternativa a la selección totalmente al azar

Algoritmo 4.1 Movimiento de actor

```
while (!listaActores.Vacio())
{
   actor = ElegirAzar(listaActores);
   listaEsta = sesiones.Esta(actor);
   listaNoEsta = sesiones.NoEsta(actor);

   //Pares de sesiones de la misma lista
   foreach(Par in listaEsta)
   {
      if (Par[1].CuentaTakes + CuentaNuevosTakes < MAX_TAKES)
      {
         Par[0].EliminarActor(actor);
         Par[1].AñadirActor(actor);
         return;
      }
   }

   //Pares con una sesión de cada lista
   foreach(Par in (listaEsta, listaNoEsta))
   {
      if (Par[1].CuentaTakes + CuentaNuevosTakes < MAX_TAKES)
      {
         Par[0].EliminarActor(actor);
         Par[1].AñadirActor(actor);
         return;
      }
   }
   listaActores.Quitar(actor);
}
```

se implementa un método de elección en donde cada actor se contará tantas veces como días sea convocado para el total.

De este modo, si una película se dobla en tres sesiones donde un actor trabaja los tres días, otro dos, y un tercero solo uno, los actores tendrían $\frac{1}{2}$, $\frac{1}{3}$ y $\frac{1}{6}$ probabilidades de ser elegidos, respectivamente.

4.1.2.2. Movimiento de take

El objetivo de este movimiento será unificar un take que se encuentre dividido en varias sesiones y, en caso de no llegar a una solución factible, buscará recolocar dicho take en una sesión en la que no tenga presencia. Para que este movimiento pueda ser ejecutado, los actores que intervienen en el take seleccionado para una sesión dada, deben estar presentes en la sesión de destino, lo que evita que el objetivo principal empeore.

Para realizar este movimiento se elige un take al azar entre todos los takes en los que intervenga al menos un actor, y se determina en qué jornadas de trabajo tiene presencia y en cuales no. Se eligen pares de sesiones al azar en las que el take esté presente y se tratan de unificar. La Figura 4.3 muestra una representación de este movimiento.

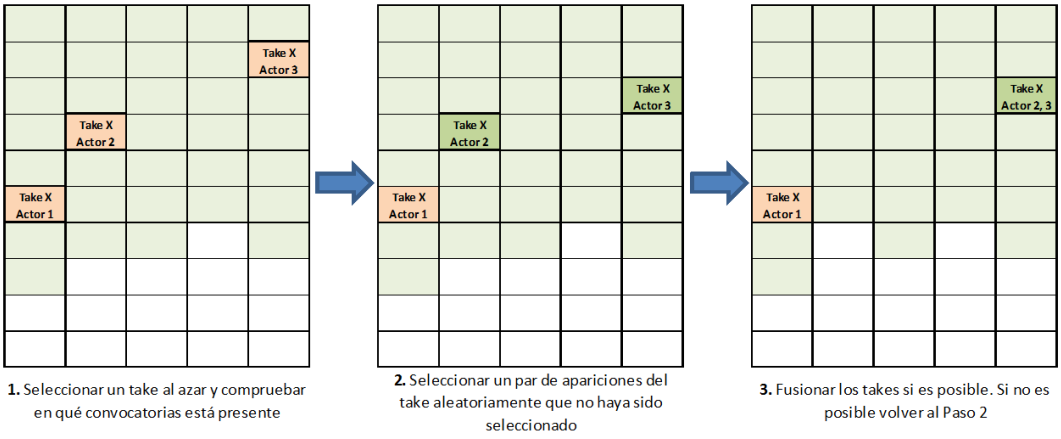

Figura 4.3: Movimiento de take dividido

En caso de que esta unificación no sea posible para ningún caso, se pasará a probar con las sesiones donde el take no está presente. Para ello se elegirá una convocatoria donde esté presente y una donde no esté presente, al azar, y se probará si el traslado es posible, tanto porque los actores estén presentes, como porque la sesión tenga espacio para el nuevo take. En caso de que no lo sea, se probará con un nuevo par. En la Figura 4.4 se explica este comportamiento.

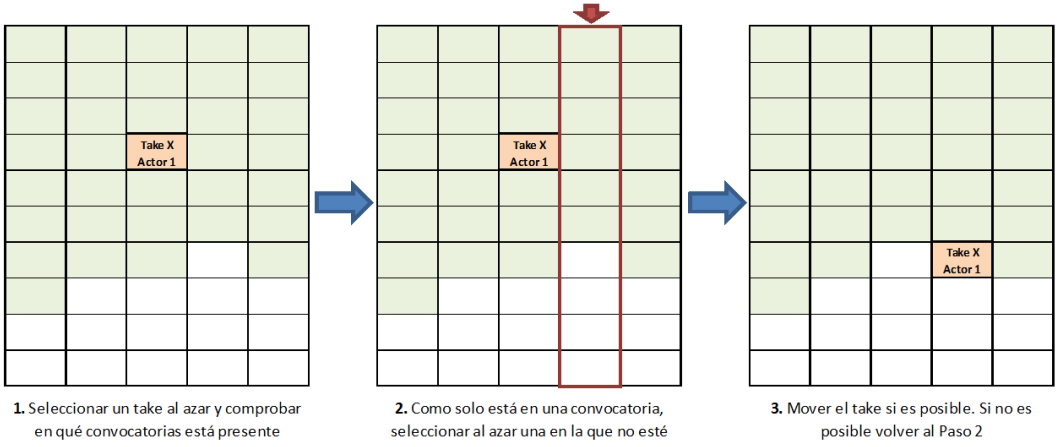

Figura 4.4: Movimiento de take sin dividir

Si se han probado todos los pares de sesiones y no se ha obtenido una solución factible, se elegirá un nuevo take y se repetirá el proceso. En caso de que se haya probado con todos los takes sin resultado, el movimiento no puede ser ejecutado.

El Algoritmo 4.2 muestra el pseudocódigo del movimiento. Nótese que al añadir un take solo se añade si no estaba en la sesión de destino, y se incluyen todos los actores que participan en el take en la sesión de origen.

Algoritmo 4.2 Movimiento de take

```
while (!listaTakes.Vacio())
{
   take = ElegirAzar(listaTakes);
   listaEsta = sesiones.Esta(take);
   listaNoEsta = sesiones.NoEsta(take);

   //Pares de sesiones de la misma lista
   foreach(Par in listaEsta)
   {
      if (Par[1].Contiene(Par[0].Actores))
         {
            Par[0].EliminarTake(take);
            Par[1].AñadirTake(take);
            return;
         }
   }

   //Pares con una sesión de cada lista
   foreach(Par in (listaEsta, listaNoEsta))
   {
      if ((Par[1].Contiene(Par[0].Actores)) AND
         (Par[1].CuentaTakes + CuentaNuevosTakes < MAX_TAKES))
         {
            Par[0].EliminarTake(take);
            Par[1].AñadirTake(take);
            return;
         }
   }
   listaTakes.Quitar(take);
}
```

Al igual que ocurría con el movimiento de actor de la sección anterior, se plantea una variación del movimiento en el que se tiene en cuenta el número de divisiones de un mismo take en la solución actual a la hora de elegir los takes para hacer el movimiento. Cada take será contado para el total un número de veces igual al número de convocatorias en las que está presente, de forma que takes más divididos tengan más probabilidades de ejecutar el movimiento y, por tanto, unificarse.

4.1.2.3. Intercambio de takes

Este movimiento, como su nombre indica, realiza el intercambio de dos takes que se encuentren planificados en dos sesiones de grabación diferentes. En la Figura 4.5 se puede ver una representación de este movimiento.

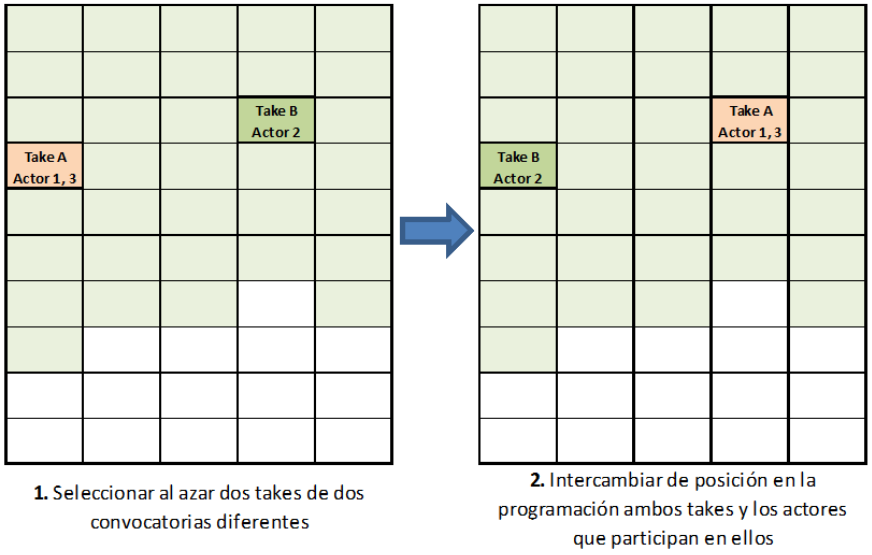

Figura 4.5: Movimiento de intercambio de takes

Al eliminar un take de una convocatoria se comprobará para cada actor que participaba en él si no está convocado para dicha sesión por participar en otro take. En caso de que esto ocurra, el actor deja de estar presente en dicha convocatoria.

Por otro lado, al añadir un take a una convocatoria diferente se debe comprobar si los actores que participan en dicho take ya estaban convocados para la sesión de grabación y, en caso de no estarlo, deben ser convocados.

Ya que este movimiento no varía el número de takes por sesión de trabajo, si la solución de partida era factible, la solución resultado de aplicar el movimiento siempre será factible también.

El Algoritmo 4.3 muestra el pseudocódigo del movimiento. Nótese que no se contemplan todas las posibilidades de elección del mismo take para el intercambio para no complicar el pseudocódigo innecesariamente.

Algoritmo 4.3 Intercambio de takes

```
sesion1 = ElegirAzar(sesiones);
sesion2 = ElegirAzar(sesiones.Quitar(sesion1))

take1 = ElegirAzar(sesion1.Takes);
take2 = ElegirAzar(sesion2.Takes);
while (take1 = take2)
{
   take2 = ElegirAzar(sesion2.Takes.Quitar(take2));
}

sesion1.EliminarTake(take1);
sesion2.EliminarTake(take2);
sesion1.AñadirTake(take2);
sesion2.AñadirTake(take1);
```

4.1.2.4. Salto n-dimensional

El salto n-dimensional es la ejecución de n movimientos de forma consecutiva. Únicamente se comprobarán las soluciones intermedias para compararlas con la mejor solución global alcanzada hasta el momento, pero en ningún caso detendrá el ciclo de n ejecuciones hasta completarlas. Los n movimientos que se realizarán en cada salto serán elegidos de entre los posibles movimientos que permite el algoritmo de forma aleatoria.

El objetivo de este movimiento no es buscar una solución mejor directamente, ya que se basa en la aleatoriedad (aunque puede alcanzar mejores soluciones). La filosofía de este movimiento es modificar el espacio de búsqueda actual, ya que se puede considerar completamente explorado, y de esta forma poder alcanzar soluciones diferentes.

El Algoritmo 4.4 muestra el pseudocódigo de este movimiento.

Algoritmo 4.4 Salto n-dimensional

```
for(i=1 to n)
{
   movimiento = ElegirAzar(movimientoAlgoritmo);
   programacionActual = movimiento.Ejecutar(programacionActual);
   if (programacionActual.Mejor(mejorProgramacion))
      mejorProgramacion = programacionActual;
}
```

4.1.3. Cruces

Los cruces son un tipo de movimiento especial que se realizan a partir de dos soluciones en lugar de una. Este tipo de movimiento simula el proceso de reproducción,

donde cada progenitor aporta parte del material genético a los hijos. Cada una de las soluciones empleadas como entrada para el movimiento representan a un progenitor, y el resultado de aplicar el cruce se correspondería con el hijo.

Se plantean dos tipos de cruce distintos que serán empleados en los algoritmos de tipo genético de este trabajo.

4.1.3.1. Cruce Tipo1

En este tipo de cruce, el hijo conservará una parte inicial de la información del padre y completará el resto de su programación haciendo uso de la programación de la madre.

Se parte eligiendo una convocatoria al azar en la programación padre. Sobre dicha convocatoria se realiza una elección aleatoria de un take y, dentro del mismo, un actor que intervenga en él. Ahora se copia toda la información del padre hasta ese punto seleccionado y, a partir de dicho punto, se debe completar la programación haciendo uso de la información de la madre. Se recorrerán todos los takes de la madre y se comprobará si ya se han incluido en la programación hija y, en caso de no estarlo, se incluirán en orden hasta completar el tamaño máximo de takes en la sesión de grabación. Una vez se han recorrido todos los pares take-actor de la madre, toda la información debe estar ya incluida en el hijo, con lo que se da por finalizado el cruce. En la Figura 4.6 se muestra una representación de este comportamiento de forma simplificada, ya que se considera que cada take no está dividido y siempre está interpretado por un único actor.

El Algoritmo 4.5 muestra el pseudocódigo de este cruce, y sobre él deben considerarse los siguientes aspectos:

- La función *añadirHasta* del hijo copiará toda la información del padre hasta el punto marcado por esa convocatoria, take y actor.
- La función *añadir* del hijo incluirá el par take-actor en su programación en el siguiente hueco que tenga dentro de una convocatoria. Una vez la convocatoria está completa, se creará una nueva.

Otro factor importante que se debe tener en cuenta para este tipo de cruce es el orden de las convocatorias, los takes y los actores. El orden no es importante en el

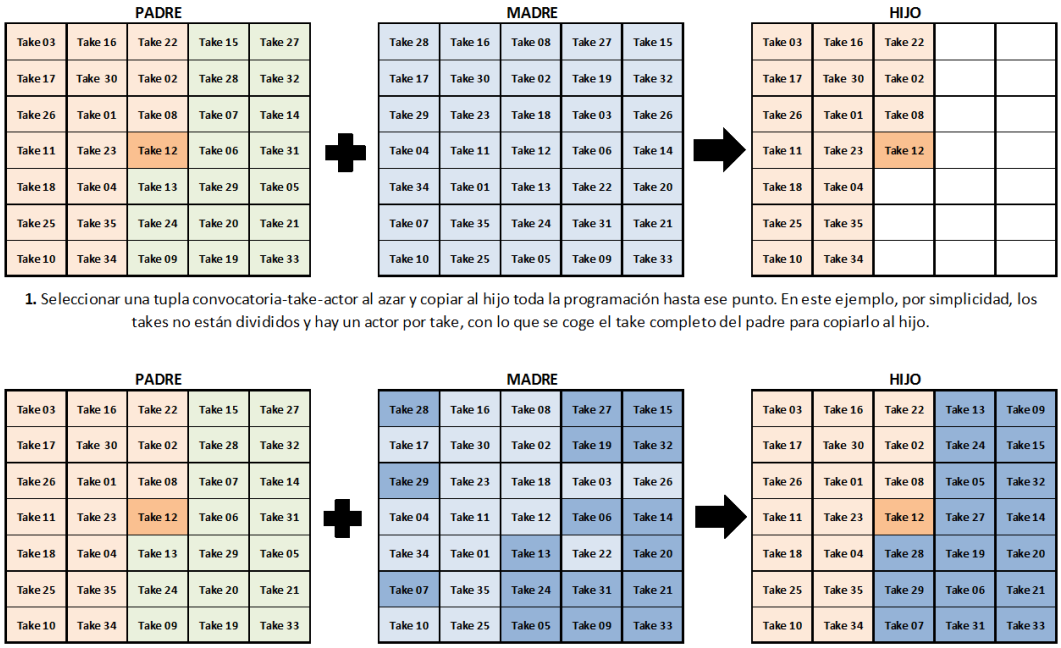

Figura 4.6: Cruce de Tipo 1

modelo planteado, ya que las convocatorias se definen como días separados, pero no influye el orden de estos. Con los takes dentro de una convocatoria ocurre lo mismo, ya que lo que importa es qué takes se incluyen en qué convocatoria y no el orden dentro de esta. Para poder aplicar este cruce sí se tendrá el cuenta el orden tanto de convocatorias, takes dentro de las convocatorias y actores dentro de takes.

4.1.3.2. Cruce Tipo2

Este tipo de cruce hará prevalecer toda la información común a las programaciones de los dos progenitores, y completará el resto de la programación para dar una solución factible. Se plantean dos formas diferentes de completar la programación del hijo: elegir la información de uno de los progenitores aleatoriamente en cada punto de la programación, o completar aleatoriamente los huecos que han quedado sin cubrir. La Figura 4.7 muestra un diagrama de este cruce simplificado, ya que se considera que cada take no está dividido y siempre está interpretado por un único actor.

El Algoritmo 4.6 muestra el pseudocódigo de este cruce, sobre el que se deben aclarar

Algoritmo 4.5 Pseudocódigo de Cruce Tipo1

```
convocatoriaAzar = Aleatorio(padre.convocatorias);
takeAzar = Aleatorio(convocatoriaAzar.takes);
actorAzar = Aleatorio(takeAzar.actores);

hijo = CrearProgramacionVacia();
hijo.añadirHasta(padre, convocatoriaAzar, takeAzar, actorAzar);

for (convocatoria in madre.convocatorias)
{
   for (take in convocatoria.takes)
   {
      for (actor in take.actores)
      {
         if (hijo.noContiene(take, actor))
         {
            hijo.añadir(take,actor);
         }
      }
   }
}
return hijo;
```

los siguientes aspectos:

- El método *ObtenerComunes* busca los pares take-actor que se repiten en las dos convocatorias que se le pasan, sin tener en cuenta el orden de los mismos dentro de las convocatorias.
- El método *ObtenerTakeActorNoIncluidos* devuelve la lista con los pares take-actor que están presentes en la primera programación y no están presentes en la segunda, ordenados según aparezcan en la primera programación.
- En este cruce se definen dos maneras de seleccionar los takes para completar la información del hijo una vez se han programado las partes comunes: al azar, o usando la información de uno de los progenitores. El método *Seleccionar* realizaría esta selección, de forma que si la elección es al azar usaría una de las listas de la que iría sacando pares take-actor aleatoriamente y, si se usa la información de los progenitores, se elegiría una de las dos listas al azar y se obtendría el primer elemento de ellas, comprobando que no esté ya añadido en el hijo.

Al igual que ocurría con el Cruce Tipo 1, el orden es importante para realizar este cruce. Al obtener los pares take-actor comunes no se ha tenido en cuenta el orden dentro de las convocatorias, pero sí sería importante tenerlo en cuenta para modelos en

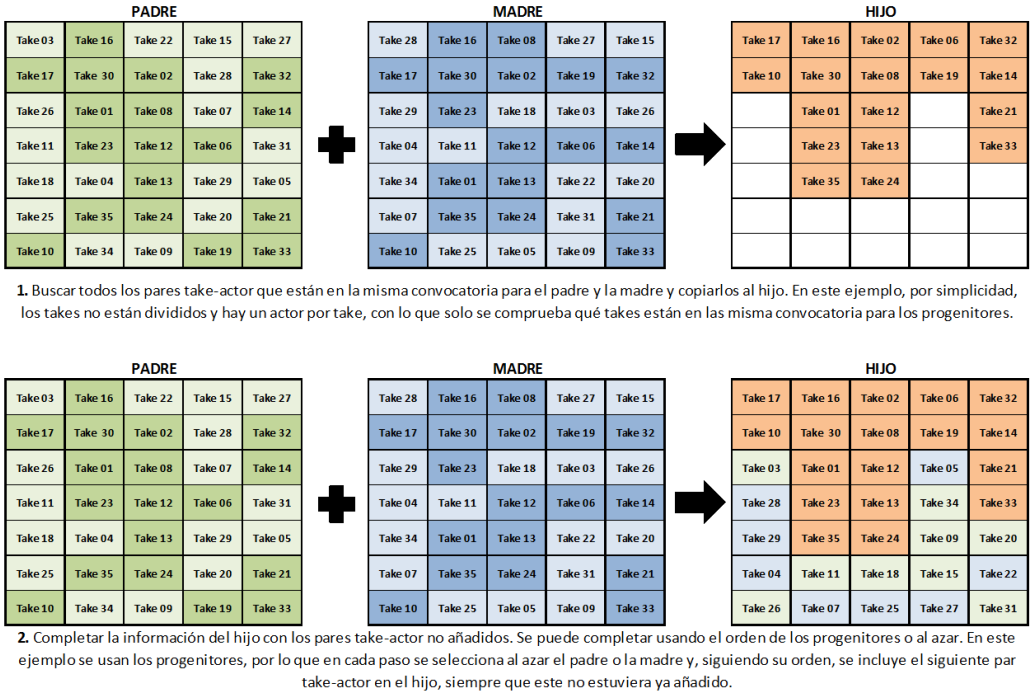

Figura 4.7: Cruce de Tipo 2

los que se tenga en cuenta el orden, como los que se plantearán en la Sección (6.2) más adelante en este trabajo. Además, si se tuviera en cuenta el orden a la hora de obtener las partes comunes, en el caso de completar con la información de los progenitores, los pares take-actor se podrían elegir aleatoriamente justo para las posiciones a completar.

4.2. Recocido simulado

El primer algoritmo que se ha implementado para resolver el problema es el algoritmo de recocido simulado, cuya base y motivación se ha detallado previamente en la Sección 2.2.2.1. El algoritmo partirá de una solución sobre la que irá iterando conforme desciende la temperatura, hasta llegar a una temperatura mínima donde se detendrá y volverá a empezar con la temperatura máxima en caso de estar configurado para ello. En temperaturas elevadas se permitirán cambios que empeoren la solución actual y, conforme se vaya reduciendo la temperatura, las nuevas soluciones deberán ir yendo a mejor.

Algoritmo 4.6 Pseudocódigo del Cruce Tipo2

```
hijo = CrearProgramacionVacia();
for (i== to NUM_DIAS)
{
   listaTakeActor = ObtenerComunes(padre.convocatorias[i], madre.convocatorias[i
       ]);
   convocatoriaNueva = CrearConvocatoriaVacia();
   convocatoriaNueva.añadir(listaTakeActor);
   hijo.añadir(convocatoriaNueva);
}

listaTakeActorPadre = ObtenerTakeActorNoIncluidos(padre, hijo);
listaTakeActorMadre = ObtenerTakeActorNoIncluidos(madre, hijo);

while (listaTakeActorPadre.noVacio() && listaTakeActorMadre.noVacio())
{
   actorTake = Seleccionar(listaTakeActorPadre, listaTakeActorMadre, hijo);
   hijo.añadir(actorTake);
}
return hijo;
```

4.2.1. Motivación

El caso de estudio que se trata en este trabajo se ha basado en [11]. En este último se plantea el problema de doblaje de películas y se implementa un algoritmo de recocido simulado que lo resuelve. Los buenos resultados obtenidos por el algoritmo muestran que se puede tomar como referencia para compararlo con otras técnicas heurísticas, por lo que se decide implementarlo.

El algoritmo implementado se basa en el trabajo de referencia, pero no es exactamente igual, ya que se ha modificado para permitir diversas variaciones en la configuración del mismo. Estas variaciones se han incluido con el objetivo de comprobar si pueden servir para mejorar tanto el algoritmo de referencia como otras aplicaciones de técnicas heurísticas para el problema.

4.2.2. Componentes

Este algoritmo posee dos tipos de componentes que deben ser definidos: los movimientos que se ejecutan en cada iteración y el mecanismo de escape. En los siguientes apartados de definen en detalle los componentes del algoritmo.

4.2.2.1. Movimientos

Cada paso del algoritmo ejecuta un movimiento que varía la solución inicial. De todos los movimientos explicados en la Sección 4.1.2 únicamente se permiten los movimientos por actor (Sección 4.1.2.1) y por take (Sección 4.1.2.2), ya que se ha considerado que son los movimientos que mejor encajan con la ejecución del algoritmo. Como se ha explicado en las secciones de definición de estos dos movimientos, existe una posible variante que pondera la aparición de los actores o takes en varias convocatorias, haciendo que cuanto más se repitan más probable sea que se ejecute el movimiento sobre ellos. A la hora de realizar la configuración inicial del algoritmo se deberá elegir entre los movimientos originales o estas variaciones.

Las variaciones expuestas anteriormente no se han tenido en cuenta en el trabajo tomado como referencia [11] para elaborar este algoritmo, con lo que se podrá comprobar si dicha variación afecta al rendimiento del mismo.

El movimiento que se ejecuta en cada iteración del algoritmo se elige aleatoriamente entre los dos posibles dependiendo de la probabilidad establecida para estos. En la configuración inicial del algoritmo se establecerá una probabilidad de selección para cada movimiento, que determinará las opciones de que este sea elegido en cada iteración, siendo siempre la suma de las dos probabilidades el total, es decir, uno.

4.2.2.2. Mecanismo de escape

El mecanismo de escape permitirá al algoritmo moverse en el espacio de búsqueda hacia nuevas zonas de exploración en caso de que se considere que se ha estancado. Como mecanismo de escape deben ser usados movimientos que varían lo suficiente la solución para evitar que vuelva a caer rápidamente en la solución de la que se quiere huir.

Los movimientos permitidos como mecanismo de escape para este algoritmo serán el Intercambio de takes (Sección 4.1.2.3) y el Salto n-dimensional (Sección 4.1.2.4). Para este último movimiento se elegirán los saltos entre los movimientos permitidos para el algoritmo (movimiento de actor o de take) y el número de saltos realizados en cada ejecución del movimiento podrá ser configurado en la configuración inicial del algoritmo.

El movimiento Salto n-dimensional no se ha planteado en el trabajo de referencia

[11], donde únicamente se utiliza el intercambio de takes, con lo que se estudiará en futuras secciones si el uso de esta variación afecta al rendimiento del algoritmo.

4.2.3. Parámetros

El algoritmo posee parámetros y configuraciones de componentes que deben ser establecidos antes de iniciar la ejecución del mismo. En la Sección 4.2.2 se han detallado los componentes que pueden ser utilizados en el algoritmo y sus posibles configuraciones. Antes de comenzar la ejecución del algoritmo se deben elegir los componentes y configuraciones que van a ser empleadas.

Además de los componentes y sus configuraciones, existen otros parámetros que deben ser establecidos antes de la ejecución del algoritmo y que pueden variar el comportamiento del mismo. A continuación se detallan estos valores y su significado.

- **Número de iteraciones:** número de veces que se va a ejecutar el algoritmo completo, es decir, desde la temperatura inicial hasta la temperatura mínima. Tiene que ser mayor que 0.
- **Temperatura inicial:** temperatura máxima que alcanza el algoritmo. Es la temperatura de la que parte cada iteración del algoritmo.Tiene que ser mayor que 0.
- **Temperatura mínima:** temperatura en la que se considera que el algoritmo ha alcanzado su temperatura mínima. Al alcanzar esta temperatura se considera finalizada la ejecución de la iteración del algoritmo. Tiene que ser menor que la temperatura inicial y siempre mayor que 0.
- **Factor de enfriamiento:** factor que determina el decrecimiento de la temperatura actual en cada paso del algoritmo. El valor debe estar comprendido entre 0 y 1 (excluyendo ambos extremos). Cuanto más próximo a 0 esté el factor más rápido decrecerá la temperatura, y cuando más próximo a 1 el proceso será más lento.
- **Máximo de repeticiones:** número máximo de repeticiones consecutivas de la misma solución para llevar a cabo el mecanismo de escape del algoritmo. Se considera que dos soluciones son iguales cuando su número de convocatorias y de

divisiones de take es el mismo, aunque varíe la diferencia de takes. Se toma esta decisión para intentar buscar soluciones que realmente otorguen ahorro económico al estudio. El valor de este parámetro debe ser siempre mayor que 1.

4.2.4. Algoritmo

El Algoritmo 4.7 muestra el pseudocódigo del algoritmo que se ha implementado.

Las funciones *ElegirMovimientoPrincipal* y *ElegirMovimientoSecundario* sirven para elegir el tipo de movimiento que se va a ejecutar entre los movimientos permitidos. El objetivo de estas funciones es determinar el orden de ejecución de los movimientos, para que en caso de que no se pueda ejecutar un movimiento de orden superior, se ejecute el siguiente movimiento. Únicamente se plantean dos funciones de elección de movimientos porque solo se permiten dos tipos de movimientos, y la función *ElegirMovimientoSecundario* devolverá siempre el movimiento que no se ha elegido como primera opción.

En caso de que no se pueda ejecutar ninguno de los movimientos del algoritmo, se considerará que no se puede variar la solución y el algoritmo terminará la ejecución devolviendo la mejor solución encontrada hasta el momento.

4.2.5. Adecuación al problema

Como ya hemos mencionado anteriormente, en [11] se demuestra que su adecuación al problema es buena.

Tras implementar nuevamente el algoritmo y añadir modificaciones en algunos de sus componentes, las pruebas realizadas sobre el algoritmo confirman su buen funcionamiento y adecuación al problema.

4.3. Búsqueda tabú

El segundo algoritmo implementado es un algoritmo que sigue la filosofía de búsqueda tabú explicada en la Sección 2.2.2.2. En cada iteración del algoritmo se generará una lista de movimientos que crearán un conjunto de soluciones vecinas a la actual. De dicha lista de movimientos se elegirá aquel que genere la mejor programación vecina,

omitiendo de dicha selección aquellos movimientos que se encuentren en la lista tabú, a no ser que cumplan la función de aspiración del algoritmo.

La lista tabú irá incorporando todos los movimientos ejecutados y funcionará de forma cíclica debido a que su tamaño es limitado. De esta forma, cuando no se permitan más elementos en la lista y se desee introducir otro, se eliminará de la misma el que lleve más tiempo en ella.

4.3.1. Motivación

Los algoritmos de búsqueda tabú han demostrado ser buenas opciones al resolver problemas de planificación como el que se plantea en este trabajo, con lo que su aplicación a este problema podría dar buenos resultados.

Se plantea el uso de un algoritmo de búsqueda tabú como primera alternativa al de recocido simulado implementado previamente ya que ambos algoritmos tienen bastantes conceptos análogos como son: iteración sobre una única solución actual, uso de movimientos para generar nuevas soluciones y uso de mecanismos de escape para variar el espacio de búsqueda. Implementando un algoritmo con los mismos componentes permitirá determinar la bondad de aplicación de una u otra técnica a la resolución del problema.

4.3.2. Componentes

El algoritmo de búsqueda tabú, al igual que el algoritmo de recocido simulado, consta de dos componentes: movimientos que se ejecutan en cada paso del algoritmo, y movimiento ejecutado como mecanismo de escape. En los siguientes apartados se definen estos componentes y sus posibles valores.

4.3.2.1. Movimientos

Los movimientos se ejecutarán en cada paso del algoritmo con el objetivo de generar nuevas soluciones vecinas a la solución actual. Las nuevas soluciones deben incorporar pequeños cambios que permitan determinar cuál es el mejor camino a seguir por el algoritmo. Debido a esto, se utilizarán como posibles movimientos para el algoritmo el Movimiento de actor (Sección 4.1.2.1) y el Movimiento de take (Sección 4.1.2.2). Se

permitirá también elegir las variantes de estos movimientos que ponderan la aparición de los actores y takes, respectivamente, en las convocatorias.

En cada paso de ejecución del algoritmo se generará una lista de vecinos empleando alguno de los movimientos posibles. Para determinar qué tipo de movimiento se emplea para generar cada vecino se definirá de antemano una probabilidad de elección de cada tipo, de forma que cuanta más probabilidad tenga un tipo de movimiento, más vecinos serán generados usándolo.

4.3.2.2. Mecanismo de escape

La búsqueda llevada a cabo por el algoritmo tabú puede estancarse en un mínimo local de forma que no se consiga mejorar la solución. Con el objetivo de intentar evitarlos se establece un movimiento como mecanismo de escape que permita cambiar la región de búsqueda actual. Se podrá emplear el Movimiento de intercambio de take (Sección 4.1.2.3) o el Salto n-dimensional (Sección 4.1.2.4) como mecanismo de escape. Para este último habrá que definir el número de saltos que se realizarán de antemano, y los movimientos se elegirán entre la lista de movimientos permitidos para cada paso del algoritmo, usando las probabilidades de selección definidas para los mismos.

4.3.3. Parámetros

Además de los diferentes componentes que pueden ser configurados para el algoritmo detallados en la Sección 4.3.2, se debe definir un conjunto de parámetros antes de comenzar su ejecución. A continuación se definen estos parámetros y sus posibles valores.

- **Número de iteraciones:** número de veces que se ejecutará el proceso de selección del mejor vecino por el algoritmo antes de dar por concluida la ejecución. Valor mayor que 0.
- **Máximo de repeticiones:** número máximo de veces que una solución puede aparecer repetida como solución actual antes de ejecutar el mecanismo de escape del algoritmo. Se considera que una solución es igual a otra para este caso cuando tienen el mismo número de convocatorias y divisiones de take. No se tiene en cuenta la diferencia de takes entre convocatoria ya que con este proceso se buscan soluciones que mejoren el coste de producción de la película. Valor mayor que 1.

- **Tamaño de lista tabú:** número máximo de elementos que se permiten en la lista tabú. La lista comienza vacía al ejecutar el algoritmo y se irán metiendo en ella los movimientos conforme se vayan ejecutando. En caso de alcanzar el tamaño máximo, se eliminará el movimiento que lleve más tiempo para dar entrada a uno nuevo.

- **Número de vecinos:** cantidad de movimientos que deben ser ejecutados para construir un conjunto con el mismo número de soluciones. Valor mayor que 0.

4.3.3.1. Función de aspiración

La función de aspiración determina si un movimiento de la lista tabú es lo suficientemente bueno para obviar que ya se ha ejecutado recientemente y volver a ejecutarlo. Esta función define la diferencia que debe haber entre el ajuste de la solución actual y el ajuste de una solución generada por un movimiento tabú para permitir que este sea ejecutado.

Los parámetros de la función de aspiración podrán ser definidos en la configuración previa a la ejecución del algoritmo, y serán todos los parámetros que se tienen en cuenta para determinar la bondad de una solución: número de convocatorias, divisiones de take y diferencia de takes. Debe tenerse en cuenta que se usará el mismo mecanismo para evaluar si una solución es mejor que otra a la hora de evaluar la función de aspiración, de forma que si se ha definido como una mejora en división de take o diferencia de take y se produce una mejora en el número de convocatorias, la función se cumplirá.

4.3.4. Algoritmo

El Algoritmo 4.8 muestra el pseudocódigo del algoritmo de búsqueda tabú implementado.

4.3.5. Adecuación al problema

El algoritmo de búsqueda tabú implementado utiliza prácticamente los mismos componentes que el algoritmo de recocido simulado, y se plantea para comparar ambas filosofías de búsqueda.

Los resultados obtenidos en las pruebas indican que este algoritmo se ajusta correctamente al problema planteado, iterando y mejorando las programaciones.

4.4. Genético

El siguiente algoritmo construido para resolver este problema se basa en la filosofía de los algoritmos genéticos explicados en la Sección 2.2.2.3. El algoritmo partirá de una población inicial a partir de la que se creará una nueva generación de programaciones. Sobre la nueva generación se repetirá el proceso hasta alcanzar un número fijado de generaciones de la población.

Las programaciones de la nueva generación se crearán realizando cruces entre programaciones de la población actual. Se escogerán parejas de programaciones de la población actual para ir generando la nueva generación.

4.4.1. Motivación

Los algoritmos genéticos se han empleado en diversos problemas de optimización de programaciones como el que corresponde a este problema, con lo que su aplicación podría obtener buenos resultados sobre el mismo.

Este tipo de algoritmos posee un funcionamiento completamente diferente a los dos planteados anteriormente, y su modo de generar las nuevas soluciones no se corresponde con el sistema de movimientos de los dos algoritmos iterativos implementados anteriormente. Por ello se plantea este tipo de algoritmo para resolver el problema de forma que se puedan contrastar los resultados de algoritmos completamente diferentes.

4.4.2. Componentes

En esta sección se explicarán los componentes que forman el algoritmo genético implementado. Algunos de estos componentes son fijos en el algoritmo y no pueden ser modificados en la configuración, mientras que otros sí pueden ser elegidos. En los siguientes apartados se explicarán dichos componentes.

4.4.2.1. Población inicial

La población inicial del algoritmo determinará sustancialmente el resultado del mismo, ya que en ella se basarán las futuras generaciones. Las programaciones incluidas en la población inicial se crearán empleando los métodos de generación de solución inicial de la Sección (4.1.1).

El tamaño de todas las poblaciones con las que trabaja el algoritmo es fijo con valor 40 y no puede modificarse en los parámetros de configuración del algoritmo. Dos de las 40 programaciones iniciales se generarán empleando las dos ordenaciones diferentes de la solución inicial por actores, y las 38 programaciones restantes se generarán de forma aleatoria siguiendo lo indicado en la Sección (4.1.1.2).

Al generar la parte de la población inicial de forma aleatoria, aunque se configuren los mismos parámetros de ejecución, no se puede repetir fielmente una ejecución del algoritmo, ya que en gran medida depende de las soluciones generadas.

4.4.2.2. Selección de progenitores

Las programaciones de cada nueva generación se creará a partir de parejas de programaciones de la población actual. Los cruces deben realizarse con las programaciones consideradas mejores, de modo que las estructuras de estas se propaguen hacia las siguientes generaciones. Por ello, se crearán grupos de 4 individuos dentro de la población actual, y como esta tiene 40 individuos, se crearán 10 grupos. De cada uno de los grupos se realizará un torneo sobre el que se elegirán las dos programaciones con los mejores ajustes, que serán progenitores de programaciones de la siguiente generación. La programación con el mejor ajuste tomará el rol de padre en el cruce, mientras que la segunda mejor será la madre.

4.4.2.3. Cruces

El algoritmo de tipo genético consta de dos tipos de componente que se definirán a continuación: los cruces y la mutación.

Los cruces generan nuevas programaciones usando dos programaciones ya existentes. Usando los cruces el algoritmo generará nuevas generaciones de la población en cada paso del algoritmo.

Para este problema se emplearán los cruces Tipo1 (Sección 4.1.3.1) y Tipo2 (Sección 4.1.3.2) definidos anteriormente. El tipo de cruce usado para generar una programación nueva a partir de dos existentes se elegirá según una probabilidad previamente fijada antes de arrancar el algoritmo. Además de fijar las probabilidades de ejecución de cada cruce, se debe fijar el tipo de relleno que realizará el cruce Tipo2, ya sea usando la información de los padres o de forma aleatoria.

4.4.2.4. Mutación

El proceso de mutación en este tipo de algoritmos se corresponde con el mismo proceso biológico, donde los descendientes pueden sufrir cambios en su estructura genética. La mutación se podrá producir en cada uno de los descendientes en la nueva generación con una cierta probabilidad que debe ser fijada antes de ejecutar el algoritmo.

Las mutaciones que realiza el algoritmo serán siempre del mismo tipo en la misma ejecución, y se corresponderá con uno de los siguientes movimientos: Movimiento de actor (Sección 4.1.2.1), Movimiento de take (Sección 4.1.2.2) o Intercambio de takes (Sección 4.1.2.3). Antes de ejecutar el algoritmo se debe fijar el movimiento que se va a usar para generar las mutaciones.

4.4.3. Parámetros

Solamente existen dos parámetros configurables para este tipo de algoritmos y son los siguientes:

- Número de generaciones: determina el número de generaciones de población que va a crear el algoritmo.
- Tipo de selección de descendencia: cada pareja de progenitores seleccionada genera 4 programaciones que formarán parte de la generación siguiente. Esas cuatro programaciones pueden ser generadas nuevas a partir de los progenitores si se selecciona la opción *Sólo hijos*, o pueden generarse dos programaciones nuevas e incluir a los progenitores si se selecciona *Hijos y padres*.

4.4.4. Algoritmo

El Algoritmo 4.9 muestra el pseudocódigo del algoritmo tipo genético implementado.

4.4.5. Adecuación al problema

Se ha planteado el uso de este tipo de algoritmos como alternativa bastante diferente a las filosofías anteriores.

El comportamiento del algoritmo en las pruebas ha demostrado que no se adecua correctamente al problema tratado en este trabajo. Se han identificado varios motivos por los que este algoritmo no funciona bien para este problema que se explican a continuación.

Número de takes variables por convocatoria El parámetro que regula el número de takes que puede ir en cada convocatoria es el máximo de takes por convocatoria, que debe ser definido antes de la ejecución del algoritmo. Este parámetro no fija el valor para cada convocatoria, sino que únicamente lo limita, con lo que las soluciones pueden tener cualquier número de takes menor que ese valor. El algoritmo genético completa siempre las convocatorias hasta cumplir el máximo de takes permitidos en cada convocatoria, con lo que los takes siempre tienden a aglomerarse en las primeras sesiones de grabación. Las mutaciones permiten que se altere el número de takes en las convocatorias, pero al crear la siguiente generación los takes vuelven a estar agrupados al comienzo.

Orden de convocatorias y takes Cuando se han definido los cruces anteriormente, tanto de Tipo 1 (Sección 4.1.3.1) como de Tipo 2 (Sección 4.1.3.2), se explica que se toman ciertas consideraciones respecto al orden de las convocatorias y de los takes dentro de estas para poder aplicar los cruces correctamente para este problema. Pese a estas modificaciones, al no ser el orden un aspecto importante en la obtención de las soluciones para el problema, los cruces no funcionan correctamente, ya que aunque se obtenga de forma correcta la información del padre en el cruce Tipo 1 y la información común en el cruce Tipo 2, no se completa la programación hija siguiendo un orden útil, sino que se puede considerar un proceso pseudoaleatorio.

Contrariedad entre la función objetivo y el funcionamiento del algoritmo La función objetivo evalúa como uno de sus factores la máxima diferencia entre el número de takes de dos convocatorias. Dicho factor choca con la filosofía del algoritmo, donde es importante completar las convocatorias antes de pasar a las siguientes. Se podría pensar

en cambiar el funcionamiento del algoritmo para completar de manera equilibrada las diferentes convocatorias, pero esto provocaría que no se pudieran introducir conceptos de orden en la solución.

Población inicial incierta La mayor parte de la población inicial se genera de forma pseudoaleatoria usando el mecanismo definido en la Sección 4.1.1.2, con lo que los resultados del algoritmo pueden depender de las programaciones incluidas en esta población inicial. En las pruebas se ha comprobado que en la mayor parte de los casos la mejor solución encontrada por el algoritmo era una de las programaciones que formaban parte de la población inicial.

A pesar de que el algoritmo no ha resultado adecuarse bien al problema original, al final de este trabajo, en la Sección 6.2 se propondrán variaciones del mismo en donde se incluyan restricciones de orden y cambios en la función objetivo. Algunas de estas variaciones corregirían estos factores que hacen inútil aplicar este algoritmo al problema.

4.5. Recocido simulado - Genético

Este algoritmo surge de la combinación de los algoritmos de recocido simulado y genético definidos en secciones anteriores. Su funcionamiento base es el mismo que el del algoritmo genético de la sección 4.4 pero se modifica la forma de generar la población inicial, siendo esta construida ahora con soluciones obtenidas al aplicar el algoritmo de recocido.

4.5.1. Motivación

Uno de los factores que se han identificado en la Sección 4.4.5 que hacen que el algoritmo genético no sea aplicable al problema es la aleatoriedad de la población inicial. Con el objetivo de corregir este problema se propone este nuevo algoritmo que obtendrá las programaciones que compondrán la población inicial empleando el algoritmo de recocido simulado. Aunque estas programaciones no serán constantes para todas las ejecuciones del algoritmo, las pruebas sobre el algoritmo de recocido simulado han demostrado que sus ajustes son generalmente buenos.

Otro de los problemas de la no adecuación del algoritmo genético es la variabilidad

del número de takes por convocatoria. Las soluciones generadas usando el algoritmo de recocido simulado, por lo general, se compensan rápidamente, con lo que se podrían usar los valores calculados en la población inicial para definir el máximo de takes por convocatoria con el que trabajará el genético. De esta forma, se establecerá el número máximo de takes por convocatoria al número máximo de takes que se encuentra en una convocatoria de la población inicial del algoritmo.

4.5.2. Componentes

Los componentes de este algoritmo son la unión de todos los componentes que forman parte del algoritmo de recocido simulado definidos en la Sección 4.2.2, más los componentes definidos para el algoritmo genético en la Sección 4.4.2.

4.5.3. Parámetros

Al igual que ocurre con los componentes, los parámetros de este algoritmo serán la unión de los parámetros del algoritmo de recocido simulado de la Sección 4.2.3, y los parámetros del algoritmo genético definidos en la Sección 4.4.3.

Debe tenerse en cuenta que las ejecuciones del recocido simulado, para obtener las soluciones iniciales, no deberían ser muy costosas, ya que el objetivo de aplicar este algoritmo es obtener una base para que el algoritmo genético mejore.

4.5.4. Algoritmo

Este algoritmo obtiene las programaciones de la población inicial usando el recocido simulado y luego ejecuta el algoritmo genético sobre dicha población, con lo que apoyándonos en los pseudocódigos de estos algoritmos definidos en sus respectivas secciones, podemos definir el pseudocódigo de este algoritmo de la forma indicada en el Algoritmo 4.10.

4.5.5. Adecuación al problema

Este algoritmo se ha planteado para intentar corregir la no adecuación del algoritmo genético al problema. A pesar de que se corrige el problema de la población inicial, parece

que los demás motivos identificados sobre el algoritmo genético siguen presentes en este, incluso la variabilidad del número de takes.

Al introducir la población inicial calculada empleando el algoritmo de recocido simulado se pretendía obtener un nuevo valor para el número máximo de takes por sesión con el que trabaja el algoritmo genético. Sin embargo, en las pruebas del algoritmo se ha visto que muchas veces alguna de las programaciones de la población inicial tiene una o más convocatorias con un número de takes igual al máximo permitido, con lo que el problema persiste. Si se aumenta el número de iteraciones o se varía la temperatura de forma que las soluciones construidas por el algoritmo de recocido estén más cerca del óptimo se minimiza el problema, pero se aumenta lo suficiente el tiempo de ejecución como para que la aplicación del genético sobre las programaciones encontradas no merezca la pena.

Algoritmo 4.7 Pseudocódigo del algoritmo de Recocido simulado

```
actual = ObtenerProgramacionInicial();
mejor = programacion;
repeticiones = 0;

for (i=0 to ITERACIONES)
{
   T = TEMPERATURA_INICIAL;
   while (T > TEMPERATURA_MINIMA)
   {
      movimiento1 = ElegirMovimientoPrincipal();
      movimiento2 = ElegirMovimientoSecundario();

      nueva = EjecutarMovimientos(movimiento1, movimiento2);
      if (nueva == null)
         return mejor;

      if (nueva.esMejor(mejor))
         mejor = nueva;

      if (nueva.convocatorias < actual.convocatorias)
      {
         repeticiones = 0;
         actual = nueva;
      }
      else if (nueva.convocatorias == actual.convocatorias)
         if (nueva.divisionesTake < actual.divisionesTake)
         {
            repeticiones = 0;
            actual = nueva;
         }
         else if (nueva.divisionesTake == actual.divisionesTake)
         {
            repeticiones = repeticiones +1;
            if (repeticiones == MAXIMO_REPETICIONES)
               actual = EjecutarMecanismoEscape();
               repeticiones = 0;
               if (actual.esMejor(mejor))
                  mejor = actual;
            else
               actual = nueva;
         }
         else
         {
            repeticiones = 0;
            if (AleatorioUniforme() <
               exp((actual.divisiones - nueva.divisiones) / T)
               actual = nueva;
         }
      else
      {
         repeticiones = 0;
         if (AleatorioUniforme() <
            exp((actual.convocatorias - nueva.convocatorias) / T)
            actual = nueva;
      }
      T = T * FACTOR_ENFRIAMIENTO;
   }
}
return mejor;
```

Algoritmo 4.8 Pseudocódigo del algoritmo de Búsqueda tabú

```
actual = ObtenerProgramacionInicial();
mejor = programacion;
repeticiones = 0;
listaTabu = CrearListaVacia();

for (i=0 to ITERACIONES)
{
   vecinos = GenerarVecinos(NUMERO_VECINOS);
   mejorMovimiento = nulo;
   while (mejorMovimiento == nulo AND vecinos.noEsVacio())
   {
      mejorMovimiento = ElegirMejor(vecinos);
      if (listaTabu.noContiene(mejorMovimiento)
         OR CumpleAspiracion(mejorMovimiento))
      {
         listaTabu.añadirMovimiento(mejorMovimiento);
      }
      else
      {
         vecinos.quitar(mejorMovimiento);
         mejorMovimiento = nulo;
      }

      if (mejorMovimiento <> nulo)
      {
         nueva = mejorMovimiento.ejecutar();
         if (SonIguales(nueva, actual))
         {
            repeticiones = repeticiones + 1;
            if (repeticiones == MAXIMO_REPETICIONES)
               actual = EjecutarMecanismoEscape();
               repeticiones = 0;
            else
               actual = nueva;
         }
         else
         {
            actual = nueva;
            repeticiones = 0;
         }

         if (actual.esMejor(mejor))
            mejor = actual;
      }
   }
}
return mejor;
```

Algoritmo 4.9 Pseudocódigo del algoritmo Genético

```
poblacion = ObtenerPoblacionInicial(TAMANO_POBLACION);
mejor = BuscarMejor(poblacion);

for (i=0 to NUMERO_GENERACIONES)
{
   siguienteGeneracion = ListaVacia();
   while (poblacio.noEstaVacio())
   {
      torneo = ListaVacia();
      for (j=0 to NUMERO_PARTICIPANTES)
      {
         participante = ElegirAleatorio(poblacion);
         poblacion.quitar(participante);
         torneo.añadir(participante);
      }
      padre = ElegirMejor(torneo);
      torneo.quitar(padre);
      madre = ElegirMejor(torneo);

      if (SOLO_HIJOS)
         numHijos = NUMERO_PARTICIPANTES;
      else
      {
         numHijos = NUMERO_PARTICIPANTES - 2;
         siguienteGeneracion.añadir(madre);
         siguienteGeneracion.añadir(padre);
      }

      for (k=0 to numHijos)
      {
         hijo = Cruzar(padre, madre);
         siguienteGeneracion.añadir(hijo);
         if (hijo.esMejor(mejor))
            mejor = hijo;
      }
   }

   for (l=0 to TAMANO_POBLACION)
   {
      if (HayMutacion())
      {
         poblacion.obtener(l).mutar();
         if (poblacion.obtener(l).esMejor(mejor))
            mejor = poblacion.obtener(l)
      }
   }
}
return mejor;
```

Algoritmo 4.10 Pseudocódigo del algoritmo Recocido simulado - Genético

```
poblacion = ListaVacia();
for (i=0 to TAMANO_POBLACION)
{
   poblacion.añadir(ObtenerSolucionRecocido());
}

maximoTakesConvocatoria = CalcularMaximoTakesConvocatoria(poblacion);

mejor = EjecutarGenetico(poblacion);
return mejor;
```

5

IMPLEMENTACIÓN Y RESULTADOS

En esta sección se explicará en primer lugar cómo se ha llevado a cabo la implementación de la aplicación y algunas consideraciones que se han tenido en cuenta a la hora de diseñarla. Posteriormente se expondrán los resultados obtenidos utilizando dicha aplicación.

5.1. Consideraciones de la implementación

La aplicación se ha implementado como una aplicación de escritorio de Windows empleando el framework 4.0 de .NET y el lenguaje de programación C#. La elección de estas tecnologías para el desarrollo de la aplicación se ha apoyado en las siguientes consideraciones:

- El estudio de doblaje trabaja sobre ordenadores con el sistema operativo Windows, con lo que no es necesario crear una aplicación exportable a otros sistemas operativos.
- Los datos de entrada se encuentran en archivos de Excel, y las programaciones de salida se realizan usando también este tipo de archivos. El framework de .NET da soporte directamente a este tipo de archivos, con lo que su tratamiento es directo.
- Se ha decidido incluir una base de datos portable en la aplicación, para conseguir

una instalación fácil de la misma, donde se puedan almacenar todos los datos de la películas registradas. Se puede configurar la aplicación para trabajar también contra bases de datos externas para centralizar todo el conjunto de datos que podrán ser tratados por diferentes usuarios de la aplicación. Esta dualidad se puede conseguir fácilmente usando MS SQL Server y su versión compacta para realizar la instalación portable.

- El desarrollador de la aplicación posee más conocimientos en este framework de trabajo que en otras posibles alternativas como podrían ser Java y MySQL.

A continuación, se exponen algunas consideraciones sobre la implementación que deberían ser tenidas en cuenta.

5.1.1. Formatos de entrada y salida

Tal y como se ha comentado, los formatos de entrada y de salida con los que trabaja el estudio de doblaje son archivos de Microsoft Excel. Dichos archivos contienen una matriz en donde las filas se corresponden con personajes de la película, y las columnas con los números de take en los que está dividida la misma. Cada celda de la matriz se completará con un valor que indica si el personaje interviene en el take o no. En el caso de las programaciones de salida el número de cada celda indicará la sesión en la que el personaje (indicado por la fila) grabará el take en cuestión (indicado por la columna). Si no se indica valor para una celda, o su valor es 0, se considera que el personaje no interviene nunca en dicho take.

Se han modificado ligeramente los ficheros con el objetivo de que la aplicación sea capaz de tratar variaciones sobre este mismo problema. Los cambios introducidos son:

- Se incluye el nombre de la película en el propio fichero de forma que todos los datos de la misma se carguen directamente del archivo en cuestión, y se puedan identificar más fácilmente las programaciones.
- Se crea una nueva columna, anterior al nombre de los personajes que intervienen, donde se define el actor que realizará el doblaje. Este cambio permite que se tome a cada actor por separado sin tener en cuenta el personaje sobre el que está realizando el doblaje. Gracias a esta modificación, un actor podrá doblar a varios

personajes de una misma película y se podrán planificar varias películas en las que intervengan actores comunes. En los ficheros de salida se elimina la columna de personaje, ya que lo importante es saber a qué convocatoria va cada actor.

- Se incluye una nueva fila de duración de take que se debería completar con la duración prevista de grabado del take, lo que permite planificar de manera más precisa las soluciones. Este valor no se tendrá en cuenta en el modelo tratado en este trabajo, pero sí se podría usar en alguna de las variaciones propuestas en la Sección 6.2.

En la Figura 5.1 se resaltan estos cambios sobre los ficheros de entrada.

	A	B	C	D	E	F	G	H
1	Título	Arma letal 4						
2								
3		Duración	20	20	20	20	20	
4	Actor		1	2	3	4	5	
5	Actor001	RADIO		7				
6	Actor002	MURTAUGH		1	1	1	1	
7	Actor003	RIGGS		1	1	1	1	
8	Actor004	LEO						
9	Actor005	RADIO 2						
10	Actor006	PROODY						
11	Actor007	BENNY						
12	Actor008	SUBTÍTULOS						

Figura 5.1: Cambios en el formato de entrada

Pese a que se dispone de datos reales de películas que han sido dobladas por el estudio, los ficheros de entrada existentes deben ser modificados para incluir las modificaciones que se han expuesto anteriormente. La duración de los takes no se tendrá en cuenta, con lo que puede completarse con cualquier valor, mientras que el nombre de los actores sí será usado en la carga de los datos de entrada. Dos actores con el mismo nombre serán considerados el mismo actor, por lo que para mantener el problema intacto respecto al trabajo realizado por el estudio, todos los personajes de todas las películas deberían ser interpretados por actores diferentes.

Si se quisiera resolver el problema real del doblaje de una película, el estudio debería completar los ficheros con los datos correctos de los actores que han interpretado a cada personaje. Estos datos pueden ser modificados también por los usuarios de la aplicación con el objetivo de encontrar una combinación de actores-personajes que minimice el número de convocatorias de los actores.

5.1.2. Programación de múltiples películas

Las modificaciones descritas en la Sección 5.1.1 permiten que un actor intervenga como varios personajes de una o más películas diferentes. En la aplicación desarrollada se ha dado la opción de incluir los datos de entrada de varias películas para obtener una única programación, lo que permitiría al estudio optimizar sus recursos.

Este cambio no se había contemplado anteriormente, y las soluciones dependen en gran medida de los actores que se definan para cada personaje. Por ello, aunque la aplicación permite este tipo de soluciones, no se analizarán los resultados obtenidos para varias películas simultáneas en este trabajo.

5.1.3. Modularidad de la aplicación

La aplicación se ha construido de forma que pueda ser modificada para adaptarla a diferentes cambios que se puedan plantear de forma rápida y sin tener que remodelar toda la estructura de la misma. Los algoritmos se han creado de manera independiente, así como la parte de la interfaz de configuración de cada uno, con lo que añadir nuevos algoritmos no debería influir en los algoritmos actuales.

5.2. Resultados obtenidos

En este apartado se detallarán los resultados obtenidos utilizando la aplicación para cada uno de los algoritmos implementados. No se realizará un estudio completo de parametrización de algoritmos, si no que éstos serán fijados antes de las pruebas a ciertos valores que empíricamente han demostrado dar buenos resultados.

En primer lugar, se detallarán los datos de entrada disponibles para realizar pruebas sobre la aplicación, para continuar exponiendo los resultados de cada algoritmo individualmente y finalizar con una comparativa de todos los resultados obtenidos.

5.2.1. Datos de prueba

Se dispone de datos de prueba de 10 películas que serán usados para comprobar el correcto funcionamiento de los algoritmos. La Tabla 5.1 muestra los detalles de cada una de estas películas. Se debe tener en cuenta que, por lo explicado en la Sección 5.1.1,

el número de personajes es igual al número de actores que intervienen en la película, ya que hay una correspondencia directa entre personaje y actor.

Película	**Número de takes**	**Personajes**
Arma letal 4	255	67
Catwoman	195	64
Charlie y la fábrica de chocolate	213	43
Donkey Xote	181	37
Mogambo	231	18
Poseidón	207	40
Premonition	176	27
Rare birds	186	33
Saddle the wind	179	28
The wedding planner	243	52

Tabla 5.1: Resumen de películas de prueba

Para cada una de las películas de prueba se dispone tanto de los datos de la programación realizada por el estudio, como de la programación óptima calculada con una herramienta de resolución exacta de problemas de programación lineal. Tanto los datos de las programaciones del estudio como de las programaciones óptimas han sido obtenidos del documento de referencia [11], y se muestran en las tablas 5.2 y 5.3 respectivamente.

Película	**Días**	**Conv.**	**Máx. Div.**	**Dif. Takes**
Arma letal 4	7	60	5	26
Catwoman	3	62	3	9
Charlie y la fábrica de chocolate	4	40	3	52
Donkey Xote	6	43	6	82
Mogambo	5	22	4	22
Poseidón	4	41	4	5
Premonition	3	28	2	29
Rare birds	4	35	3	23
Saddle the wind	5	29	4	78
The wedding planner	4	56	3	4

Tabla 5.2: Resultados del estudio de grabación de películas de prueba

5.2.1.1. Consideraciones sobre los datos de prueba

A pesar de que se dispone tanto de los ficheros de entrada como de las soluciones encontradas por el estudio de grabación para cada película, no se dispone de los pa-

Película	Días	Conv.	Máx. Div.	Dif. Takes	Tiempo
Arma letal 4	7	57	4	0	2829,80 s
Catwoman	3	62	3	0	1009,80 s
Charlie y ...	4	39	3	0	289296,60 s
Donkey Xote	6	35	5	74	14,94 s
Mogambo	5	21	4	78	289440,60 s
Poseidón	4	41	4	0	1218,50 s
Premonition	3	28	2	2	67,44 s
Rare birds	4	35	3	25	62928,53 s
Saddle the wind	5	29	3	28	2520,45 s
The wedding planner	4	55	2	0	361225,12 s

Tabla 5.3: Resultados óptimos de películas de prueba

rámetros que han tenido en cuenta a la hora de calcular la programación en cuestión. Observando los resultados que han obtenido para las 10 películas de prueba, se observa un rango muy grande de takes máximos por día, ya que existe una película con 108 takes programados para un día y otra con 67 takes programados como máximo por día. Debido a este problema de parametrización, es posible que se encuentren soluciones mejores (o peores) que las soluciones óptimas, ya que se han usado parámetros diferentes. Los datos de las soluciones se utilizarán para configurar los parámetros de los algoritmos, de forma que podamos obtener soluciones similares a las expuestas.

5.2.2. Características del equipo de pruebas

El equipo donde se han realizado las pruebas de la aplicación posee un procesador Intel® Core™ i7-2600 (3.40GHz, 8MB L3 Cache, 4C) y una memoria RAM de 8GB. El sistema operativo sobre el que se ha ejecutado la aplicación es Windows® 7 Home Premium de 64 bits.

5.2.3. Configuración por defecto de los algoritmos

Los algoritmos implementados llevan unos parámetros por defecto configurados que se han probado en este trabajo. En este apartado se definirá la configuración por defecto de cada uno de los cuatro algoritmos planteados. Un conjunto de parámetros se definirán directamente según consideraciones realizadas basadas en las pruebas realizadas sobre los algoritmos, mientras que otros serán probados en las diferentes secciones de este apartado para determinar qué opción da mejores resultados.

Al tratarse de algoritmos heurísticos, todos ellos tienen un parámetro de configuración que determina cuantas veces se ha de repetir el proceso antes de darlo por finalizado (número de iteraciones en el recocido simulado y búsqueda tabú, y número de generaciones en el algoritmo genético). Dicho parámetro puede ser modificado para realizar búsquedas más o menos exhaustivas, empleando más o menos tiempo de ejecución. En los parámetros por defecto se han configurado valores lo suficientemente grandes para considerar la búsqueda completa pero sin penalizar el tiempo. En la mayoría de los casos la solución mínima se encuentra mucho antes de completar la ejecución.

5.2.3.1. Configuración por defecto del algoritmo de recocido simulado

La configuración por defecto del algoritmo de recocido simulado se basará tanto en las pruebas empíricas realizadas como en las indicaciones del trabajo de referencia [11]. Existen componentes cuyas configuraciones pueden hacer variar el funcionamiento del algoritmo, y para ellos se estudiará el impacto de cada una de sus configuraciones para decidir el valor por defecto del mismo.

Solución inicial

En la Sección 4.1.1 se planteaban dos tipos de soluciones iniciales, pero la solución inicial aleatoria se descarta porque puede variar entre ejecuciones y no obtiene siempre valores buenos. Dentro de la solución inicial por actores existen dos variaciones según se ordenen estos por el número de takes en los que intervienen: ascendentemente o descendentemente. Para comparar ambas ordenaciones, se calculan todas las soluciones iniciales para las películas del juego de pruebas. En la Tabla 5.4 se muestran las soluciones empleando la ordenación ascendente, y en la Tabla 5.5 la ordenación descendente.

El número máximo de divisiones de take siempre es 1 para todas las soluciones, ya que nunca se dividen los takes en este método de cálculo de la solución. La diferencia máxima de takes entre sesiones y el número de días de grabación es el mismo para las dos ordenaciones ya que se sigue el mismo mecanismo de completar las convocatorias hasta el máximo por orden.

Al fijarse en el número de convocatorias, se observa claramente que la ordenación ascendente obtiene mejores resultados, con lo que se usará dicha solución inicial para el algoritmo.

Película	Días	Conv.	Máx. Div.	Dif. Takes
Arma letal 4	3	63	1	30
Catwoman	2	65	1	8
Charlie y la fábrica de chocolate	3	49	1	84
Donkey Xote	2	47	1	11
Mogambo	3	25	1	24
Poseidón	3	47	1	92
Premonition	2	32	1	14
Rare birds	3	36	1	84
Saddle the wind	2	33	1	11
The wedding planner	3	61	1	42

Tabla 5.4: Tabla de soluciones iniciales por actor ascendente para el juego de pruebas

Película	Días	Conv.	Máx. Div.	Dif. Takes
Arma letal 4	3	79	1	30
Catwoman	2	78	1	8
Charlie y la fábrica de chocolate	3	60	1	84
Donkey Xote	2	55	1	11
Mogambo	3	37	1	24
Poseidón	3	50	1	92
Premonition	2	42	1	14
Rare birds	3	48	1	84
Saddle the wind	2	44	1	11
The wedding planner	3	79	1	42

Tabla 5.5: Tabla de soluciones iniciales por actor descendente para el juego de pruebas

Parámetros del algoritmo

Los parámetros propios del algoritmo dependerán del tipo de prueba que se quiera llevar a cabo, pero tras las pruebas realizadas se ha decidido optar por los siguientes valores por defecto:

- *Número de iteraciones:* 200
- *Temperatura inicial:* 100
- *Temperatura mínima*: 0.000001
- *Factor de enfriamiento:* 0.95
- *Máximo de repeticiones:* 10

Movimientos

Existen dos tipos de movimientos cuyas probabilidades de elección pueden ser variables. Basándose en los resultados de las pruebas y en lo expuesto en el trabajo [11], se decide optar por una probabilidad de 0.9 para el movimiento de actor y un 0.1 para el movimiento de take.

Tal y como se ha explicado en las Secciones 4.1.2.1 y 4.1.2.2, existe una variación de los movimientos originales donde se pondera el peso de cada actor o take en la película a la hora de ser elegidos. Para comprobar que la ponderación en los movimientos es efectiva se van a realizar pruebas sobre tres de las películas de prueba. En dichas pruebas se reducirá el número de iteraciones del algoritmo, ya que lo que se quiere comprobar es si el cambio ayuda a que el algoritmo converja más rápido hacia la solución óptima. Por ello, se definirán 50 iteraciones del algoritmo en cada ejecución, y cada una de las tres películas de prueba se ejecutará 50 veces para comprobar la convergencia de la solución. Todas las pruebas realizadas se han hecho aplicando la ponderación a los dos tipos de movimiento o sin aplicarla a ninguno.

A continuación, se detallan cada una de las películas elegidas para las pruebas, sus parámetros de configuración y los resultados obtenidos:

Premonition Los resultados se muestran en la Figura 5.2, donde se ve claramente que la solución ponderada ha obtenido mejores resultados.

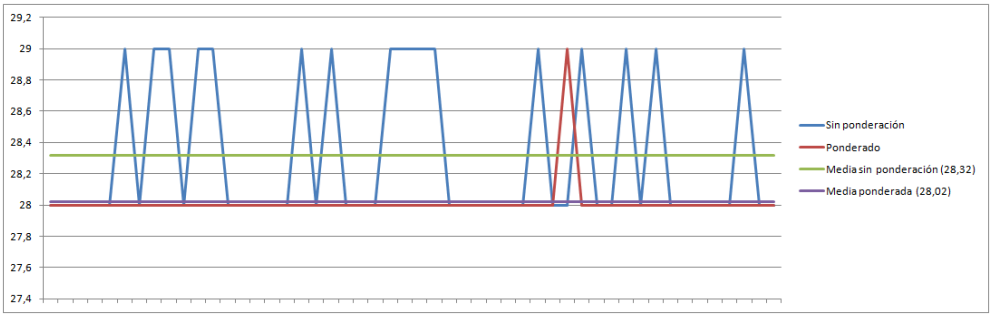

Figura 5.2: Ponderación en Premonition

Arma letal 4 La Figura 5.3 muestra los resultados obtenidos, donde se observa que todas las ejecuciones ponderadas han obtenido 56 convocatorias, mientras que la no ponderada ha llegado a obtener 58 en varias ocasiones.

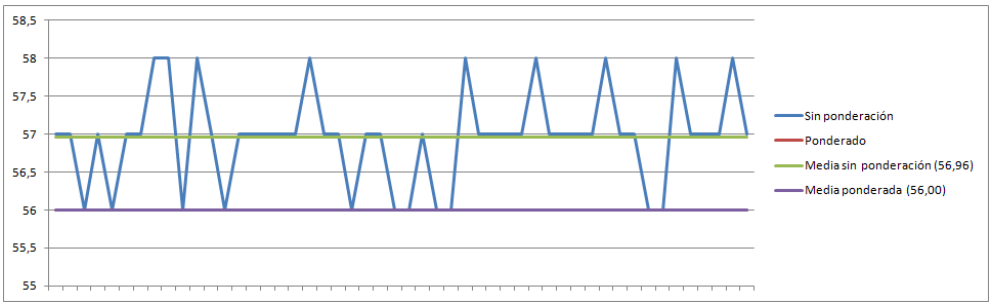

Figura 5.3: Ponderación en Arma letal 4

Mogambo En la Figura 5.4 se observan bastantes variaciones en ambos casos, con valores oscilando entre 22 y 23 convocatorias aunque se encuentra una programación de 21 para la solución ponderada. Pese a las oscilaciones, la media de convocatorias para las 50 ejecuciones sigue siendo menor para la solución ponderada.

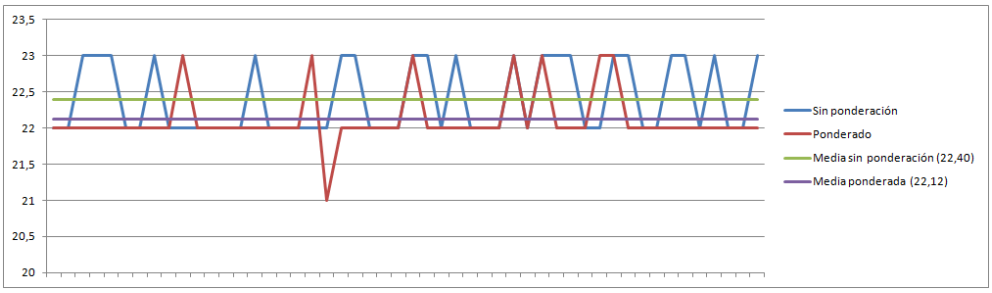

Figura 5.4: Ponderación en Mogambo

A la vista de los resultados obtenidos, se fijarán por defecto las opciones de ponderación en ambos movimientos, ya que, en este caso, converge más rápido hacia la solución óptima.

Movimiento de meneo

El movimiento de meneo permite cambiar el espacio actual de búsqueda para evitar caer en espacios con mínimos locales. Originalmente el problema se plantea con el movimiento de intercambio de takes (Sección 4.1.2.3) como único mecanismo de escape, pero se plantea como alternativa el salto n-dimensional (Sección 4.1.2.4). La ventaja del uso del salto n-dimensional es que se puede calibrar el cambio que se va a realizar en el espacio de búsqueda, pudiendo provocar cambios grandes en el mismo.

Con el objetivo de probar la utilidad del salto n-dimensional frente al movimiento de intercambio de takes, se definirá un número de saltos grande en cada ejecución del movimiento, de forma que se comprobará si saltos mayores mejoran la convergencia de la solución. Dicho número de saltos se fijará a 15.

Tomando como referencia los datos obtenidos anteriormente para movimientos ponderados, se compararán los resultados con los obtenidos al cambiar el movimiento de meneo por el salto 15-dimensional para las tres películas de prueba.

Premonition En la Figura 5.5 únicamente se muestran la división máxima y la diferencia de takes ya que siempre se ha obtenido una solución con 28 convocatorias para las 50 ejecuciones del algoritmo. Los datos muestran que las soluciones encontradas usando el salto 15-dimensional tienen una división máxima de take de 2 y una diferencia de takes entre sesiones de máximo 1, lo que es sensiblemente mejor que los resultados obtenidos con el movimiento de intercambio.

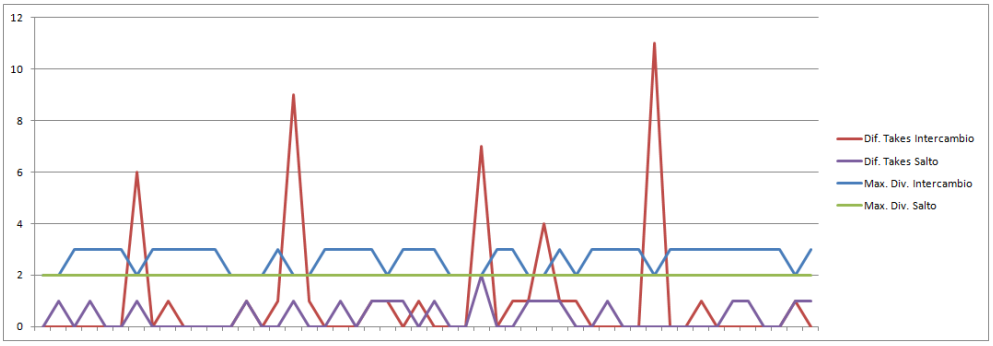

Figura 5.5: Comparativa de movimientos de meneo en el algoritmo de recocido simulado para Premonition

Arma letal 4 La Figura 5.6 muestra los resultados obtenidos, donde se omite el número de convocatorias porque siempre se ha obtenido 56. En esta gráfica no se observa una mejora considerable ni en el número máximo de divisiones de take, ni en la diferencia máxima entre sesiones, aunque sí se ven unos picos en este último valor en ciertos puntos. Los picos en la diferencia de takes se producen cuando el número máximo de divisiones es 3, en lugar de 4 como en el resto de valores, por lo que en realidad esos valores de la solución son los mejores.

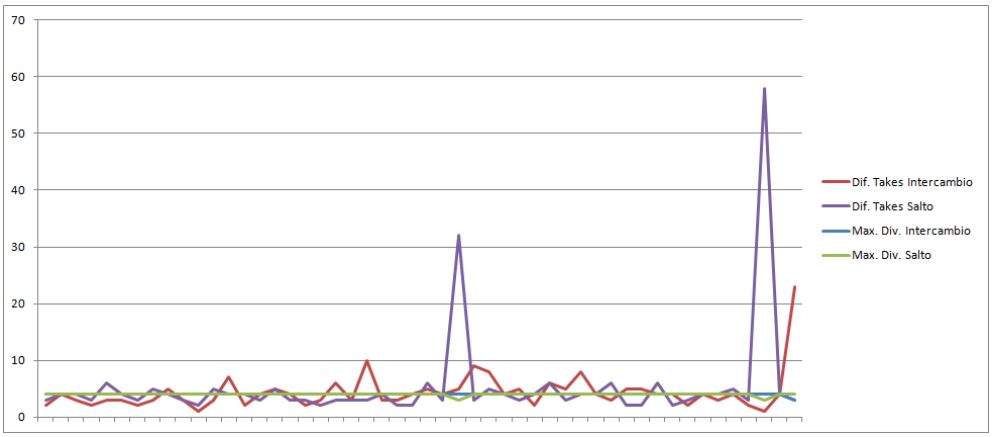

Figura 5.6: Comparativa de movimientos de meneo en el algoritmo de recocido simulado para Arma letal 4

Mogambo En la Figura 5.7 se muestra la gráfica con los resultados obtenidos para la película. En este caso se muestran las convocatorias además de la división máxima de take y la diferencia de takes entre sesiones ya que se aprecia una reducción con el cambio. Al rebajarse el número de convocatorias las programaciones son mejores, con lo que el cambio de movimiento mejora considerablemente el funcionamiento del algoritmo.

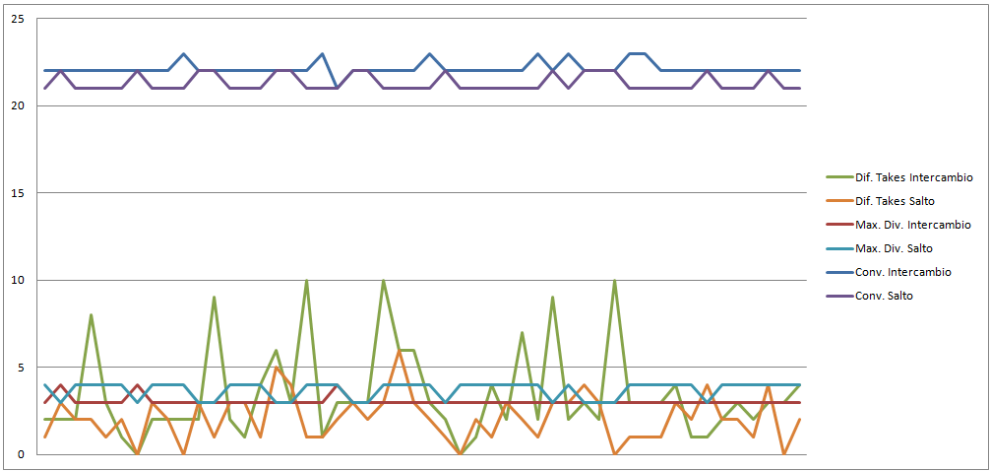

Figura 5.7: Comparativa de movimientos de meneo en el algoritmo de recocido simulado para Mogambo

Después de analizar los resultados obtenidos de las pruebas sobre las tres películas se fijará por defecto el salto n-dimensional como movimiento de meneo. El número de saltos del movimientos se dejará establecido a 15, ya que ha demostrado obtener

buenas soluciones en las pruebas, pero este valor podría depurarse más con el objetivo de acelerar la convergencia del algoritmo.

5.2.3.2. Configuración del algoritmo de búsqueda tabú

La configuración por defecto del algoritmo de búsqueda tabú se ha basado en los resultados obtenidos por el algoritmo de recocido simulado, donde se ha demostrado que ciertas configuraciones de los componentes funcionan mejor que otras.

Solución inicial

El algoritmo de búsqueda tabú es un algoritmo iterativo que parte de una única solución inicial sobre la que itera, al igual que ocurre con el algoritmo de recocido simulado. Por ello, la solución inicial empleada en este algoritmo será la misma que en el de recocido simulado, que es la solución por actores con orden ascendente, cuya justificación se puede revisar en la Sección 5.2.3.1.

Parámetros del algoritmo

Después de realizar pruebas sobre el algoritmo, se ha decidido definir los parámetros propios del mismo como sigue:

- *Número de iteraciones:* 2.000
- *Máximo de repeticiones:* 10
- *Tamaño lista tabú:* 60
- *Número de vecinos:* 60

Debe tenerse en cuenta que el aumentar el número de vecinos beneficia claramente la convergencia del algoritmo con las iteraciones, pero también penaliza mucho el rendimiento del mismo, ya que en cada iteración se deben generar muchos movimientos. Se ha elegido ese valor ya que se considera un valor equilibrado entre los dos factores.

Función de aspiración
La función de aspiración determina cuándo un movimiento mejora lo suficiente la solución actual como para ejecutarlo pese a estar en la lista tabú. Se ha decidido definir esta función de forma que para satisfacerla se deba producir algún ahorro económico para el estudio, esto es, debe mejorar al menos en uno el número máximo de divisiones de take. Si se mejora el número total de convocatorias también se cumplirá la función, ya que el ahorro es mayor. Por ello, se define con los siguientes valores:

- *Convocatorias:* 0
- *Divisiones take:* 1
- *Diferencia takes:* 0

Movimientos
Los movimientos tanto de take como de actor empleados por el algoritmo utilizarán la variante ponderada de los mismos explicada en las Secciones 4.1.2.2 y 4.1.2.1 respectivamente. Se ha decidido emplear directamente esta variante y no comprobar los resultados, ya que se ha demostrado en las pruebas de configuración del recocido simulado de la Sección 5.2.3.1 que los movimientos ponderados hacen converger más rápido hacia la solución óptima.

Además, se conservarán las probabilidades de los dos movimientos del algoritmo de recocido simulado, siendo la probabilidad de movimiento de actor 0,9 y la probabilidad de movimiento de take 0,1.

Meneos
En las pruebas de configuración del algoritmo de recocido simulado se ha demostrado que el movimiento de salto n-dimensional con un salto considerablemente grande ayuda a encontrar mejores soluciones. Sin embargo, estos resultados no son extrapolables directamente a este algoritmo ya que el funcionamiento del mismo es diferente. Por ello se ha decidido repetir las pruebas sobre las tres películas del juego de datos.

Premonition En la Figura 5.8 se muestran los resultados comparativos entre emplear el movimiento de salto 15-dimensional como mecanismo de meneo del algoritmo,

frente al movimiento de intercambio de takes. En la comparativa del número de convocatorias se ve claramente que el movimiento de salto 15-dimensional funciona mejor que el movimiento de intercambio de takes.

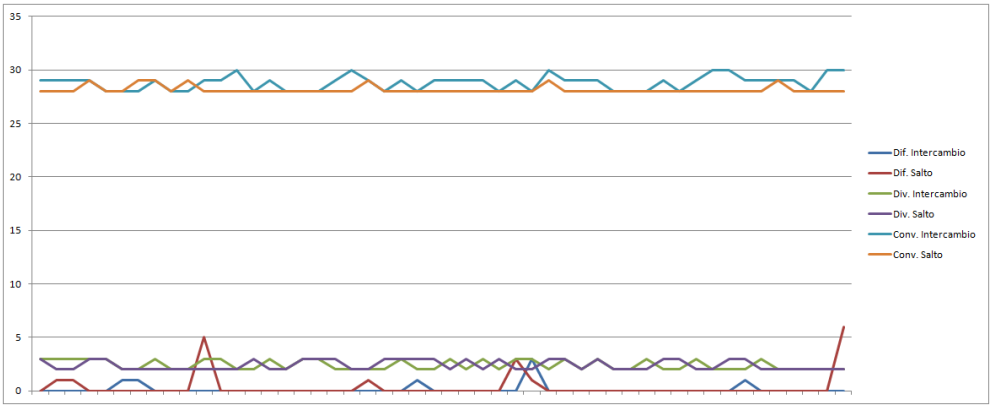

Figura 5.8: Comparativa de movimientos de meneo en el algoritmo de búsqueda tabú para Premonition

Arma letal 4 La Figura 5.9 contiene la gráfica comparativa entre el uso de los dos movimientos de meneo. Al igual que en el caso anterior, la comparativa sobre el número de convocatorias deja claro que el movimiento de salto 15-dimensional funciona mejor para este algoritmo.

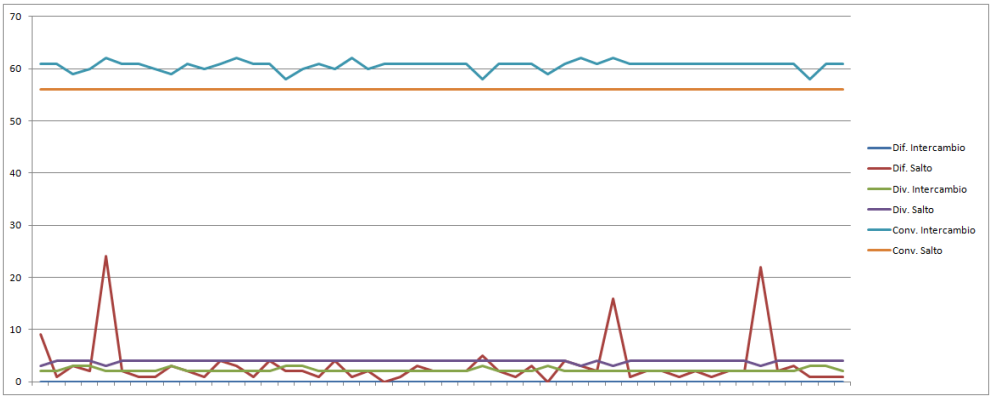

Figura 5.9: Comparativa de movimientos de meneo en el algoritmo de búsqueda tabú para Arma letal 4

Mogambo En la Figura 5.10 se muestra la gráfica que compara los dos tipos de movimiento de meneo. Tal como ocurría en las dos películas probadas anteriormente,

el número de convocatorias se reduce sensiblemente al usar el movimiento de salto 15-dimensional.

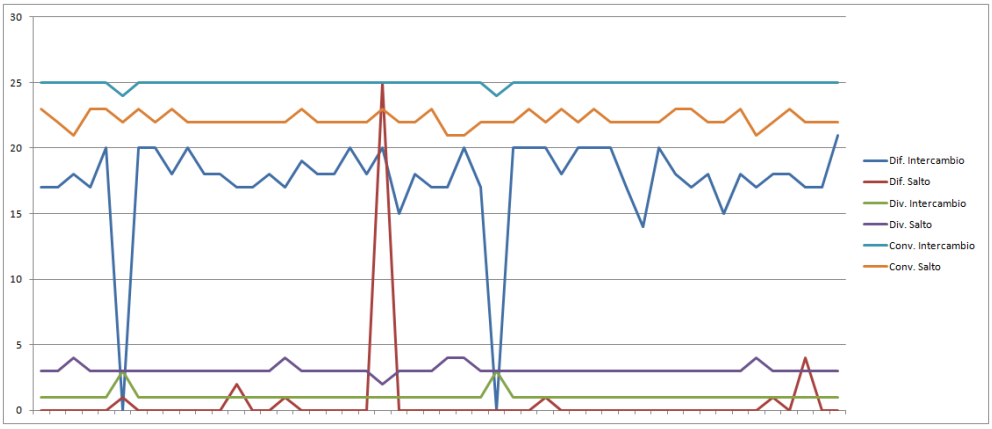

Figura 5.10: Comparativa de movimientos de meneo en el algoritmo de búsqueda tabú para Mogambo

Tras analizar los resultados de las tres películas de prueba, se concluye que el uso del movimiento de salto n-dimensional, con un valor de salto considerable, mejora la convergencia de la solución hacia el óptimo, ya que en todos los casos se ha visto mejorado el número de convocatorias. A la vista de los buenos resultados obtenidos, se decide conservar el valor de 15 como valor de salto del movimiento.

5.2.3.3. Configuración del algoritmo genético

Después de lo explicado en la Sección 4.4.5, se ha concluido que el algoritmo no se adecua al problema que se está tratando. En las pruebas que se han realizado sobre el algoritmo se ha probado a variar todos los parámetros sin conseguir unos resultados aceptables para el mismo, con lo que cualquier configuración por defecto del mismo sería igual de válida.

Los valores por defecto que se han definido se han establecido de forma un tanto teórica, por si se aplicara el algoritmo sobre alguna de las variaciones del problema que se propondrán en la Sección 6.2. Estos valores se definen a continuación.

Parámetros del algoritmo

El número de generaciones se ha establecido pensando en los tiempos de ejecución

obtenidos para los otros algoritmos, de forma que fueran próximos. Para el tipo de descendencia se considera que funcionaría mejor mantener los padres en las futuras generaciones, de forma que los individuos más aptos prevalezcan.

- *Número de generaciones:* 150
    - *Hijos y padres*

Cruces

A la hora de establecer las probabilidades de cada uno de los dos cruces, se ha pensado que el cruce Tipo 2 va a funcionar mejor que el de Tipo 1, ya que hará que prevalezca la información común a los individuos más aptos de cada población. Un abuso del cruce de Tipo 1 podría provocar que la parte inicial de algunas programaciones se propagara a toda la descendencia, reduciendo así la variabilidad.

El tipo de relleno aleatorio del cruce Tipo 2 podría añadir un factor de variación a la generación de soluciones, ya que las filosofías de ambos cruces se basan en la preservación, lo que haría al algoritmo depender en gran medida de las mutaciones.

Por ello, los valores por defecto se configurarán del siguiente modo:

- *Probabilidad cruce Tipo 1:* 0,15
- *Probabilidad cruce Tipo 2:* 0,85
    - *Relleno aleatorio*

Mutación

La mutación es importante en este tipo de algoritmos basados en la conservación de la información, ya que conservar demasiada información puede reducir mucho el espacio de búsqueda. Estas mutaciones introducirán cambios imprevistos en las programaciones con lo que se podrán alcanzar nuevas soluciones.

Por otro lado, abusar de la mutación llevaría a depender mucho de los movimientos ejecutados de esta forma, lo que incluiría un valor alto de aleatoriedad que choca con el

funcionamiento de este tipo de algoritmos. Por ello, se fijará la probabilidad de mutación a un valor bajo, de forma que se generen únicamente unas pocas mutaciones de media por generación.

Se dan tres opciones de movimiento para ejecutarse en el caso de mutación: movimiento de actor (Sección 4.1.2.1), movimiento de take (Sección 4.1.2.2) y movimiento de intercambio de takes (Sección 4.1.2.3). No se da la opción de seleccionar el movimiento de salto n-dimensional (Sección 4.1.2.4), ya que se entiende que produciría un cambio demasiado grande para considerarse una mutación. Entre los tres movimientos disponibles se cree que va a funcionar mejor el movimiento de actor, ya que así lo ha demostrado en los algoritmos de recocido simulado y búsqueda tabú. Los movimientos de actor y take empleados en las mutaciones utilizarán siempre los factores de ponderación, ya que han demostrado dar mejores resultados.

Así, los valores de mutación quedarán configurados como sigue:

- *Probabilidad de mutación:* 0,05
    - *Movimiento de actor*

5.2.3.4. Configuración del algoritmo recocido simulado - genético

Al igual que ocurre con el algoritmo genético, y como ya se ha explicado en la Sección 4.5.5, el algoritmo mezcla de recocido simulado y genético no se adecua correctamente al problema.

Se han realizado múltiples pruebas cambiando los parámetros del algoritmo para intentar aprovechar los cambios introducidos respecto al algoritmo genético original, pero no se han conseguido resultados satisfactorios para ninguna configuración.

La configuración por defecto de este algoritmo se basa en las configuraciones por defecto del algoritmo de recocido simulado, expuesta en la Sección 5.2.3.1, y del algoritmo genético, definida en la Sección 5.2.3.3. Sobre estas configuraciones únicamente se modificarán algunos parámetros para conseguir que la ejecución total del algoritmo no sea muy costosa, ya que habría que sumar el tiempo de cálculo de todas las programaciones de la población inicial empleando el algoritmo de recocido simulado, más el tiempo del algoritmo genético.

Por defecto este algoritmo irá configurado de la siguiente forma:

Parámetros recocido simulado

- **Solución inicial**
  - *Solución por actores ascendente*
- **Parámetros del algoritmo**
  - *Nº de iteraciones:* 10
  - *Temperatura inicial:* 100
  - *Temperatura mínima:* 0,0001
  - *Factor de enfriamiento:* 0,95
  - *Máximo de repeticiones:* 10
- **Movimientos**
  - *Prob. movimiento take:* 0,1
    - *Considerar takes una vez:* NO
  - *Prob. movimiento actor:* 0,9
    - *Considerar actores una vez:* NO
- **Meneos**
  - *Realizar salto*
    - *Pasos de salto:* 15

Parámetros genético

- **Parámetros del algoritmo**

- *Número de generaciones:* 100
  - *Hijos y padres*

- **Cruces**
  - *Probabilidad cruce Tipo 1:* 0,15
  - *Probabilidad cruce Tipo 2:* 0,85
    - *Relleno aleatorio*

- **Mutación**
  - *Probabilidad de mutación:* 0,05
    - *Movimiento de actor*

5.2.4. Resultados individuales de los algoritmos

En este apartado se detallarán las pruebas realizadas individualmente sobre cada uno de los cuatro algoritmos implementados.

Las pruebas se realizarán usando los datos de configuración por defecto definidos en la Sección 5.2.3 y se ejecutarán los algoritmos sobre cada una de las películas del juego de pruebas definido en la Sección 5.2.1.

Con el objetivo de usar un conjunto de parámetros estandarizado para todas las películas, y basándose en los datos de las programaciones obtenidas por el estudio y las que se manejan en el trabajo de referencia [11], se define un número máximo de takes por sesión de 95.

El número de días de grabación disponible para cada película se tomará usando los datos de las ejecuciones del algoritmo original, escogiendo el valor con el que se haya obtenido el menor número de convocatorias para 95 takes por día. En caso de empate se elegirá el número de días menor.

Existen dos casos especiales en esta configuración:

- En la película *Rare birds,* al establecer 95 takes máximos por día y 3 días, uno de estos queda vacío. Por ello, se ha reducido el número de takes por sesión a 90, y se ha mantenido el número de días.

- Para la película *Donkey Xote* se ha detectado un problema de concordancia de datos, ya que la Tabla 5.1 muestra que hay 37 personajes y la Tabla 5.3 alcanza una programación de 35 convocatorias, lo que no es posible. Por ello, no se han tenido en cuenta los resultados de dicha película a la hora de establecer el número de sesiones de grabación y se ha fijado en 4.

La Tabla 5.6 define los parámetros que se fijarán para cada una de las películas del juego de pruebas en todas las ejecuciones de los algoritmos.

Película	**Takes por sesión**	**Sesiones**
Arma letal 4	95	6
Catwoman	95	3
Charlie y la fábrica de chocolate	95	4
Donkey Xote	95	4
Mogambo	95	5
Poseidón	95	4
Premonition	95	3
Rare birds	90	3
Saddle the wind	95	4
The wedding planner	95	4

Tabla 5.6: Parámetros comunes de las películas de prueba

Las ejecuciones de cada uno de los algoritmos para cada película se repetirán 20 veces con el fin de poder evaluar correctamente los resultados. Sobre los resultados para cada película, se identificará el ajuste mínimo encontrado por el algoritmo y la media de los mejores ajustes encontrados. Al calcular la media de los ajustes no se tendrá en cuenta el valor de la diferencia máxima de takes entre convocatorias, ya que puede ser bastante variable según el número de divisiones máximo de takes del ajuste encontrado. Además, se mostrará el tiempo medio de ejecución del algoritmo para alcanzar el mínimo, calculado como la media de los tiempos de ejecución del algoritmo hasta encontrar la solución mínima. Estos tiempos no tienen que coincidir con los tiempos de ejecución totales del algoritmo para los parámetros de configuración por defecto.

5.2.4.1. Resultados del algoritmo de recocido simulado

La Tabla 5.7 muestra las soluciones mínimas encontradas para cada película empleando el algoritmo de recocido simulado.

Película	Días	Conv.	Máx. Div.	Dif. Takes
Arma letal 4	6	56	3	25
Catwoman	3	62	2	2
Charlie y la fábrica de chocolate	4	39	3	0
Donkey Xote	4	37	3	0
Mogambo	5	20	3	5
Poseidón	4	40	4	0
Premonition	3	28	2	0
Rare birds	3	35	2	0
Saddle the wind	4	28	3	10
The wedding planner	4	55	3	0

Tabla 5.7: Resultados mínimos del algoritmo de recocido simulado

En la Tabla 5.8 se muestra la media de las mejores programaciones encontradas para cada ejecución. El número de convocatorias medio de todas las películas está siempre cercano al mínimo encontrado, lo que indica que el algoritmo funciona correctamente porque sus resultados son estables.

Película	Tiempo	Días	Conv.	Máx. Div.
Arma letal 4	50,386 s	6	56	3,9
Catwoman	30,518 s	3	62,2	2,85
Charlie y la fábrica de chocolate	20,075 s	4	39	3
Donkey Xote	72,112 s	4	37,1	3,25
Mogambo	33,099 s	5	20,25	3,6
Poseidón	23,560 s	4	40,75	3,75
Premonition	14,651 s	4	28	2
Rare birds	15,346 s	3	35	2
Saddle the wind	79,784 s	4	28,05	3
The wedding planner	29,822 s	4	55,2	3

Tabla 5.8: Resultados medios del algoritmo de recocido simulado

5.2.4.2. Resultados del algoritmo de búsqueda tabú

Empleando los parámetros de configuración por defecto se obtienen los valores de la Tabla 5.9 como programaciones mínimas para cada película del juego de pruebas. El número medio y mínimo de convocatorias está muy próximo, con lo que el algoritmo es estable.

La Tabla 5.10 contiene los datos medios calculados a partir de todos las ejecuciones

Película	Días	Conv.	Máx. Div.	Dif. Takes
Arma letal 4	6	56	3	3
Catwoman	3	62	3	0
Charlie y la fábrica de chocolate	4	39	3	0
Donkey Xote	4	37	4	0
Mogambo	5	20	3	0
Poseidón	4	41	4	0
Premonition	3	28	2	0
Rare birds	3	35	2	0
Saddle the wind	4	28	4	7
The wedding planner	4	55	3	0

Tabla 5.9: Resultados mínimos del algoritmo de búsqueda tabú

del algoritmo de búsqueda tabú sobre el juego de pruebas.

Película	Tiempo	Días	Conv.	Máx. Div.
Arma letal 4	21,593 s	6	56	3,6
Catwoman	10,231 s	3	63,25	2,75
Charlie y la fábrica de chocolate	30,574 s	4	39,05	3,2
Donkey Xote	21,418 s	4	37,35	3,65
Mogambo	39,794 s	5	20,45	3,15
Poseidón	29,465 s	4	41,6	3,55
Premonition	23,190 s	3	28,1	2,2
Rare birds	0,449 s	3	35	2
Saddle the wind	29,745 s	4	29	3,05
The wedding planner	38,543 s	4	55,7	2,95

Tabla 5.10: Resultados medios del algoritmo de búsqueda tabú

5.2.4.3. Resultados del algoritmo genético

Tal y como se explicó en la Sección 4.4.5, las programaciones calculadas usando el algoritmo tienden a concentrar los takes al comienzo de la misma, dejando en muchas ocasiones sesiones enteras vacías. La Figura 5.11 muestra en la parte izquierda la representación de una solución correctamente formada y equilibrada obtenida con el algoritmo de recocido simulado, y en la parte derecha una representación de las soluciones conseguidas usando el algoritmo genético.

Al no adecuarse el algoritmo correctamente al problema, los resultados obtenidos con el mismo no aportan ninguna información relevante, por lo que no se han documentado

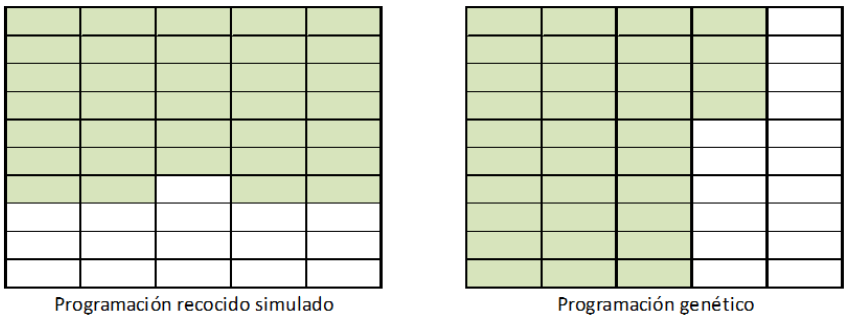

Figura 5.11: Comparativa solución recocido simulado y genético

resultados de pruebas como se ha hecho anteriormente en los algoritmos de recocido simulado y búsqueda tabú.

5.2.4.4. Resultados del algoritmo recocido simulado - genético

Este algoritmo no se adecua correctamente al problema, al igual que ocurre con el algoritmo genético, y por ello no se han realizado pruebas de funcionamiento como se hizo con el algoritmo de recocido simulado y el de búsqueda tabú.

Pese a las modificaciones respecto al algoritmo genético original, no se ha conseguido corregir el comportamiento para conseguir hacerlo adecuado para este problema. La población inicial generada con el algoritmo de recocido suele estar equilibrada, y hay ocasiones en las que se reduce el máximo de takes por sesión de grabación que podrían ayudar al algoritmo genético. Sin embargo, se sigue produciendo el problema de aglomeración en las sesiones iniciales comentado en la Sección 5.2.4.3. En la mayor parte de las pruebas realizadas, salvo en algún caso poco probable de mutación en la primera generación, la mejor programación era una de las que formaban parte de la población inicial.

La Figura 5.12 muestra una posible población inicial obtenida usando el algoritmo de recocido simulado, y cómo el algoritmo genético compacta las programaciones de las siguientes generaciones.

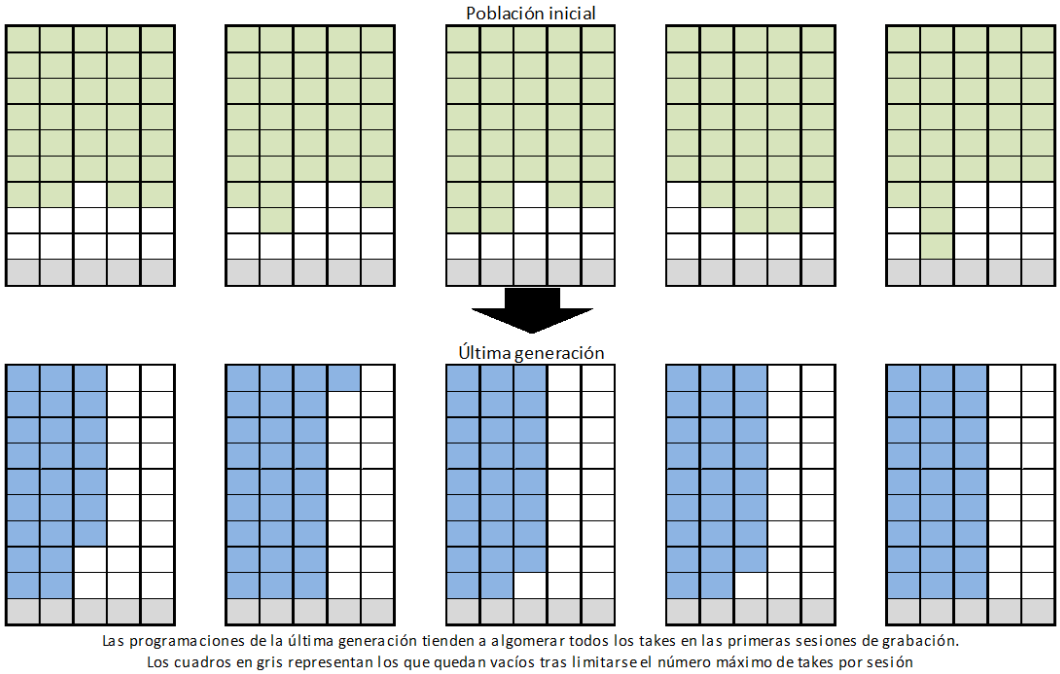

Figura 5.12: Funcionamiento del algoritmo de recocido simulado - genético

5.2.5. Análisis conjunto de resultados

En este apartado se analizarán todos los resultados que se han obtenido en las pruebas de los algoritmos. Para realizar un análisis completo se seguirán dos fases: comparar los resultados individuales para cada uno de los algoritmos implementados, y contrastar los resultados obtenidos con los reflejados en el trabajo de referencia[11].

Aunque se han implementado cuatro algoritmos diferentes, dos de ellos han resultado no ser adecuados para el problema, con lo que no se han podido obtener resultados para ellos, y por tanto, no serán tenidos en cuenta en todo este análisis.

5.2.5.1. Comparación de algoritmos implementados

Al no incluir los algoritmos genético y de recocido simulado - genético en este análisis, únicamente se compararán los resultados de la aplicación del algoritmo de recocido simulado y de búsqueda tabú. Se compararán los resultados tanto medios como mínimos para todas las películas del juego de pruebas.

En primer lugar se comparará el número mínimo de convocatorias obtenido por cada uno de los algoritmos para determinar cual obtiene mejores soluciones. La Figura 5.13 muestra una gráfica comparativa del número mínimo de convocatorias para todas las películas. En todas salvo en una, *Poseidón*, el número de convocatorias mínimo es el mismo.

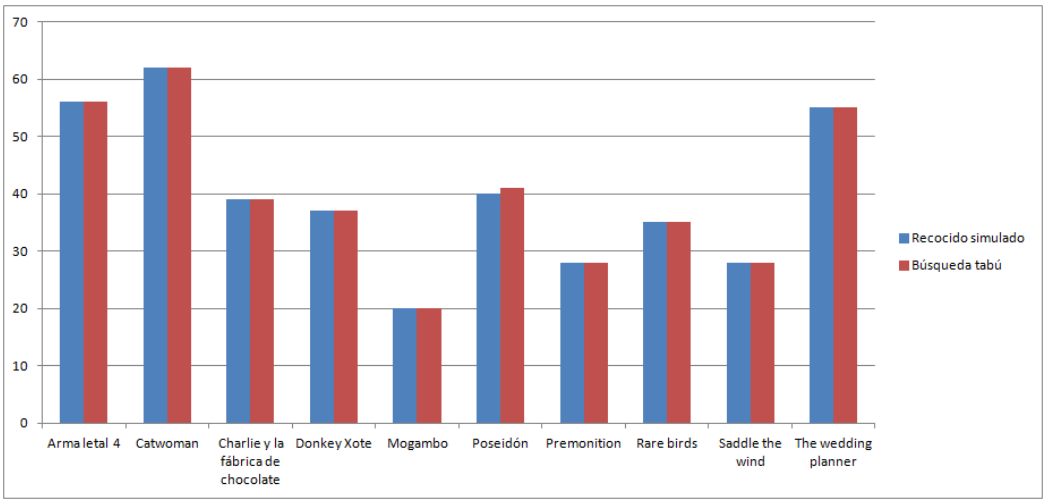

Figura 5.13: Comparativa de convocatorias entre algoritmos

Una diferencia mínima en uno de los casos de prueba no indica que un algoritmo funcione mejor que otro, con lo que para determinar qué algoritmo obtiene mejores soluciones, se compara el número máximo de divisiones de take para todas las películas menos *Poseidón*, construyendo la gráfica de la Figura 5.14. En dicha gráfica se aprecia que el número maximo de divisiones de take es el mismo para todas las películas, con lo que se podría concluir que ambos algoritmos funcionan de forma similar obteniendo buenas soluciones.

Habiendo comparado los ajustes de las programaciones mínimas sin obtener una resolución clara sobre cuál funciona mejor, se compararán los ajustes medios. En la Figura 5.15 se muestra la comparativa de convocatorias medias obtenidas utilizando cada uno de los algoritmos en las pruebas realizadas. Se puede apreciar en esta gráfica que las soluciones medias obtenidas usando el algoritmo de recocido simulado son iguales en 3 casos, y mejores en otros 7, con lo que parece que este algoritmo, en media, funciona mejor que el de búsqueda tabú.

Otro factor a tener en cuenta a la hora de evaluar los algoritmos es el tiempo de

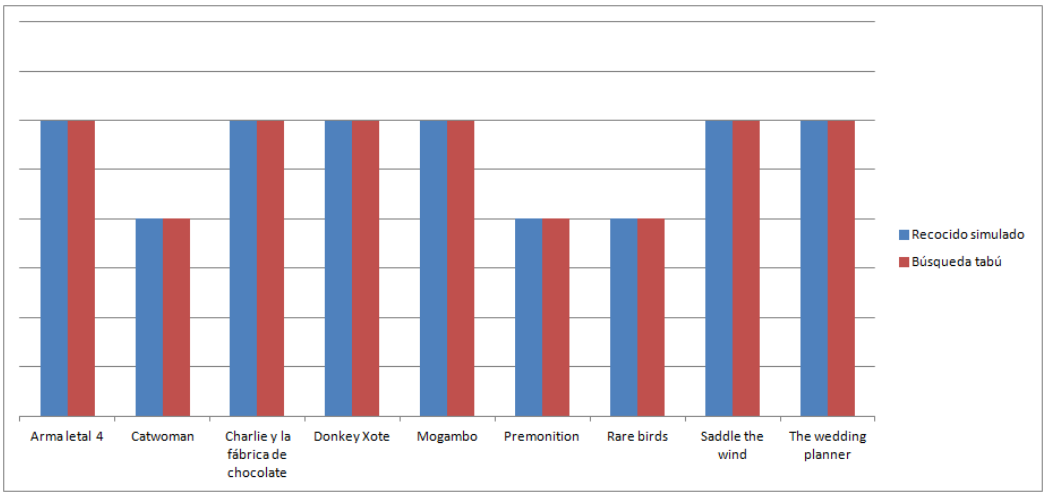

Figura 5.14: Comparativa de número máximo de divisiones de take entre algoritmos

convergencia, ya que puede influir en la elección de uno u otro. La Figura 5.16 contiene los datos de los tiempos medios en los que se ha alcanzado la mejor solución para el algoritmo. Cabe mencionar que dichos tiempos se corresponden con la solución mínima en esa ejecución, que puede no coincidir con la solución mínima global.

La línea de tiempos del algoritmo de búsqueda tabú está por debajo de la línea del algoritmo de recocido simulado. Sin embargo, anteriormente se comprobó que el algoritmo de recocido simulado obtenía mejores soluciones en media, lo que podría explicar estos resultados, ya que dichas soluciones pueden ser obtenidas tras realizar bastantes iteraciones sobre el algoritmo. Para el algoritmo de búsqueda tabú, el no encontrar las soluciones mínimas tan habitualmente como el de recocido simulado, unido a un menor tiempo en hallazgo de las soluciones mínimas podría indicar que el algoritmo no realiza una búsqueda completa y se estanca en mínimos locales.

Para concluir, una vez analizados los diferentes aspectos de los algoritmos implementados, aunque el algoritmo de búsqueda tabú ha demostrado obtener buenas soluciones, se elige el algoritmo de recocido simulado como el que mejor resuelve el problema planteado.

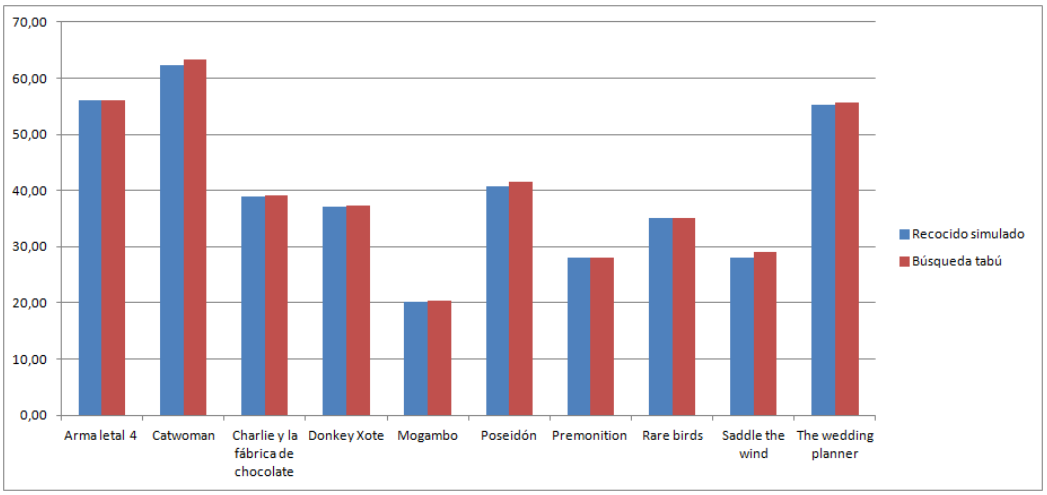

Figura 5.15: Comparativa de convocatorias medias entre algoritmos

5.2.5.2. Comparación con el trabajo de referencia

Una vez se ha elegido el algoritmo de recocido simulado como el mejor entre los implementados, se compararán sus resultados con los que aparecen en el trabajo de referencia. Esta comparación únicamente servirá para demostrar que el algoritmo ha sido bien implementado, y los posibles cambios que se produzcan en las mejores soluciones pueden depender de la aleatoriedad presente en el algoritmo o de los parámetros de configuración.

La Tabla 5.11 muestra los resultados reflejados en el trabajado de referencia para el conjunto de datos de prueba y los parámetros establecidos en la Tabla 5.6. De esta tabla se omite la película *Donkey Xote* por las discrepancias encontradas en el trabajo.

El número de convocatorias es el mismo que el mínimo obtenido con el algoritmo implementado en este trabajo (véase Tabla 5.7) para todas las películas excepto una, *Poseidón*. Esta discrepancia puede deberse al número de repeticiones de ejecución del algoritmo, ya que si se observa el número medio de convocatorias de la Tabla 5.8 para esta película, se comprueba que es mayor que 40, y por tanto, que no se ha obtenido en todas las repeticiones.

La Figura 5.17 muestra la comparación entre el número máximo de divisiones de take para todas las películas con el mismo número de convocatorias en el trabajo de referencia

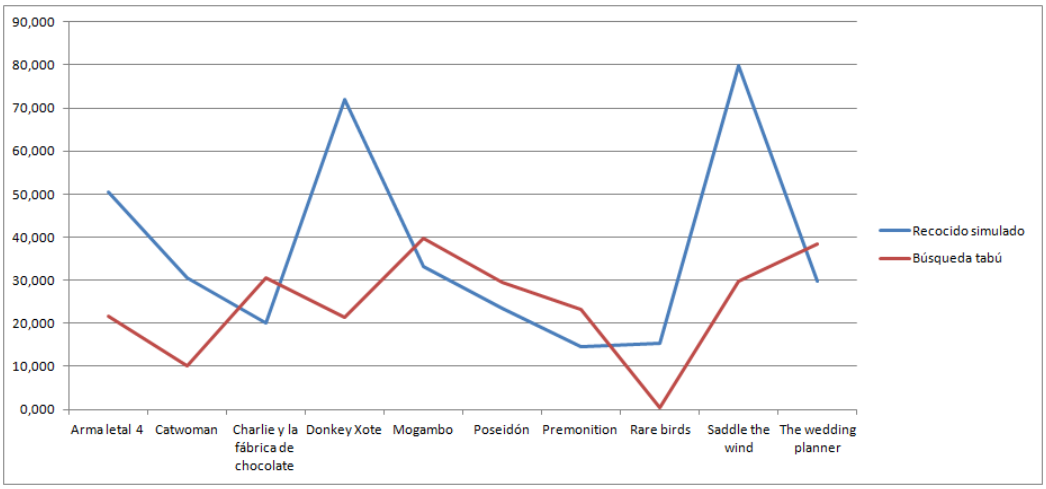

Figura 5.16: Comparativa de tiempos medios entre algoritmos

Película	**Días**	**Conv.**	**Máx. Div.**	**Dif. Takes**	**Tiempo**
Arma letal 4	6	56	4	28	42,76 s
Catwoman	3	62	3	0	25,01 s
Charlie y ...	4	39	3	0	32,62 s
Mogambo	5	20	4	2	24,89 s
Poseidón	4	41	3	1	23,64 s
Premonition	3	28	2	0	18,75 s
Rare birds	3	35	2	0	18,56 s
Saddle the wind	4	28	3	27	22,84 s
The wedding planner	4	55	3	0	42,81 s

Tabla 5.11: Resultados del algoritmo de recocido simulado en el trabajo de referencia

y en la programación mínima del algoritmo de recocido. En la gráfica se aprecia que el algoritmo implementado obtiene valores al menos tan buenos como el de referencia, lo que indica que el algoritmo funciona correctamente. Las mejoras obtenidas con el nuevo algoritmo, al igual que ocurría con el número de convocatorias, puede deberse a la repetición de ejecución del algoritmo para calcular la solución mínima.

Una vez se ha determinado que el algoritmo implementado funciona correctamente, se compararán los tiempos de obtención de solución. La Figura 5.18 muestra la comparación entre los tiempos medios del algoritmo de recocido simulado implementado, y los tiempos de referencia para las soluciones. Los tiempos, aunque diferentes, pueden considerarse buenos, y sobre ellos hay que tener las siguientes consideraciones:

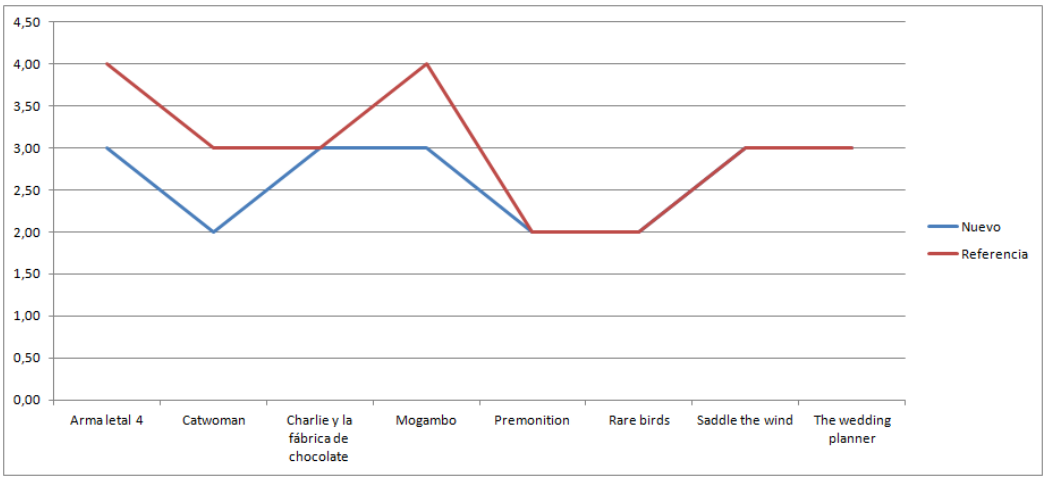

Figura 5.17: Comparativa de división máxima de take con el trabajo de referencia

- Las máquinas de ejecución de los algoritmos no son iguales.
- El nuevo algoritmo corre sobre una interfaz, lo que penaliza el tiempo de ejecución.
- Los tiempos del trabajo de referencia se supone que son de una ejecución concreta, mientras que para el nuevo algoritmo se han tomado tiempos medios.

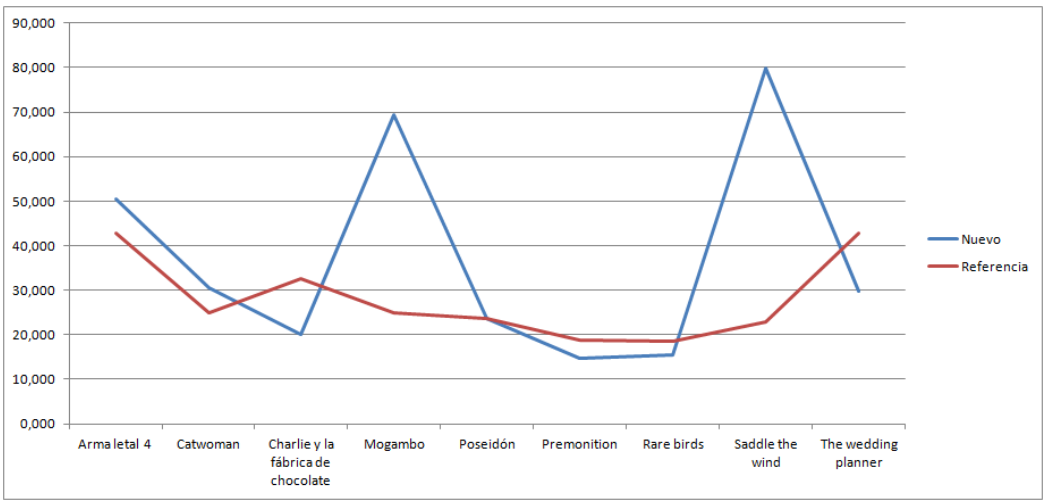

Figura 5.18: Comparativa de tiempos con el trabajo de referencia

6

CONCLUSIONES Y TRABAJO FUTURO

Una vez concluido el trabajo se deben extraer las conclusiones que se pueden obtener del mismo analizando los resultados. Se debe también proponer líneas de trabajo sobre este problema que permitan tanto mejorar los métodos de resolución como adaptar el problema a posibles variaciones. Por último, se debe validar que se han cumplido todos los objetivos que se marcaron para este trabajo.

6.1. Conclusiones

En este trabajo se ha creado una aplicación para dar solución al problema de gestión de programación al que se enfrenta un estudio de doblaje. La aplicación emplea algoritmos heurísticos para obtener programaciones que minimicen el coste, y se ha creado con el objetivo de facilitar la labor de los trabajadores encargados de gestionar el doblaje en el estudio.

El estudio realiza actualmente las programaciones de forma manual, donde un trabajador debe cuadrar todas las convocatorias de forma que el coste de doblaje sea el menor posible. Con esta aplicación se permite realizar el trabajo de forma casi automática, ya que únicamente se han de seleccionar los parámetros de la propia programación deseada, que son el número de días y el número de takes máximos por día. Los demás parámetros de los algoritmos están configurados a unos valores que se han probado y

han dado buenos resultados, aunque es posible modificarlos, si así se desea, desde la propia aplicación.

El formato de los ficheros de entrada y salida se ha mantenido en un formato casi igual al original con el que trabaja el estudio, de forma que el impacto del uso de la aplicación sea el menor posible. No se han dejado los ficheros tal cual estaban los originales para poder permitir que un actor interprete varios personajes y programar varias películas simultáneamente. Además, se ha incluido la duración de los takes por si se tienen en cuenta en alguna variación del modelo.

Pese a que se han creado tres nuevos algoritmos, ya que el algoritmo de recocido simulado ya se había aplicado a este problema, dos de ellos no han resultado adecuados para el problema actual, con lo que sus soluciones no son útiles. Aún así, se han dejado implementadas las bases de los mismos por si se modifica el problema de forma que se adecuen mejor a él.

Respecto a los otros dos algoritmos, el de recocido simulado y el de búsqueda tabú, se han obtenido buenos resultados en ambos casos, con lo que cualquiera de ellos se podría usar en la aplicación para obtener buenas programaciones. El tiempo de convergencia es variable según la configuración y la película en cuestión, pero los parámetros establecidos deberían ser suficientes para encontrar una buena solución. De todas maneras, se incluye un botón para parar la búsqueda cuando se considere suficientemente buena la solución, y quedarse así con la mejor programación encontrada.

Una vez resuelto el problema tal y como lo presentó el estudio y se modela en [11], se pueden plantear nuevas variaciones y/o ampliaciones del mismo de forma que se pueda ajustar a nuevas situaciones o problemas usando este trabajo como base.

6.2. Líneas de trabajo futuras

El problema que se trata en este trabajo, así como su modelo de problema de programación lineal, es el mismo que en el trabajo de referencia [11]. Los objetivos de este proyecto buscaban crear algoritmos alternativos al planteado en dicho trabajo, así como crear una aplicación de escritorio para los mismos, con lo que no se ha considerado tratar problemas diferentes. Sin embargo, sí se planteaba al inicio de este trabajo el proponer posibles líneas de trabajo que tomen este como referencia, ya que se ha preparado la

implementación para ser extensible.

En este apartado se definirán algunas de las líneas por las que se puede extender este trabajo para adaptarlo a otras situaciones o buscar mejoras del mismo. Algunas de estas líneas de trabajo sugieren establecer relaciones de orden dentro de las programaciones, lo que podría conllevar una adecuación de los algoritmos basados en la filosofía genética.

6.2.1. Cambios en la función objetivo

A lo largo de la elaboración de este trabajo se han detectado situaciones que podrían carecer de sentido práctico a la hora de evaluar las programaciones obtenidas, ya que programaciones más largas pueden considerarse mejores que programaciones más cortas. Por otro lado, también se han identificado factores que no se tienen en cuenta a la hora de evaluar la bondad de una programación y pensando en el coste de producción de cada película se considera que sí deberían tenerse en cuenta. A continuación se definen estos cambios y sus motivaciones.

6.2.1.1. Considerar el número total de takes

Uno de los factores que influye en la bondad de una solución es la diferencia máxima entre el número de takes de dos sesiones de grabación. Dicha diferencia se tiene en cuenta con el objetivo de unificar el tiempo de grabación entre sesiones para evitar riesgos ante sesiones de grabación muy cargadas de trabajo. Sin embargo, la búsqueda de soluciones equilibradas puede llevar a dividir en exceso takes para lograrlo.

Utilizando la función objetivo actual, una programación de 3 días con 80 takes cada días es considerada mejor que una solución con 70 takes los primeros dos días y 71 el tercero. Esto choca cuando el número total en el primer caso sería de 240 takes grabados y en el segundo 211. En una situación como esta el número extra de takes de la primera solución se debería a una excesiva división de takes, pero que no se tiene en cuenta siempre que no se divida el mismo take en exceso.

Se propone elaborar una función objetivo en la que se tenga en cuenta el número total de takes antes que la máxima diferencia de takes entre dos sesiones, ya que se considera que grabar más takes implica un coste económico mayor para el estudio. De este modo, la función objetivo del problema evaluaría los parámetros en el siguiente

orden:

Total de convocatorias > *División máxima de take* > *Número total de takes* > *Diferencia de takes entre sesiones*

Empleando esta nueva función objetivo es probable que el movimiento de take cobrara más importancia en el problema, ya que trataría de fusionar takes divididos.

6.2.1.2. Considerar el número total de divisiones de take ponderando el número de divisiones de take

A la hora de evaluar una solución se tiene en cuenta como segundo factor de bondad de ajuste el número máximo de divisiones de un take dentro de la programación. Se evalúa este factor ya que los takes grabados separados deben ser unidos posteriormente por técnicos del estudio, lo que implica un coste, y cuantas más divisiones tenga un take, más aumenta la dificultad de esta unión. Además, dividir un take implica rodar el take las mismas veces que el número de divisiones del mismo, lo que también implicaría un coste.

El factor de división de un take es importante y debe ser tenido en cuenta a la hora de evaluar una solución, pero utilizando únicamente el valor máximo no se tiene en cuenta el número total de divisiones. En un caso extremo, una programación sobre una película con n takes, donde todos se dividen en 2 partes, es considerada mejor que otra programación en donde un único take se divide en 3 partes. Cuanto mayor es n, más se acentuaría este problema.

Para corregir este comportamiento se propone tener en cuenta el número de divisiones de take totales en la película, pero para tener en cuenta que cuantas más divisiones haya en un take más complicada es la unión, se debería ponderar el número de divisiones de cada take. Así, se propone evaluar las divisiones de take totales de la película del siguiente modo:

$$dif = \sum_{t \epsilon T} d_t^k, \quad \forall t \epsilon T$$

d_t es el número de divisiones del take t y k es una constante de ponderación del número de divisiones de cada take que debe ser mayor que 1.

6.2.1.3. Reducción del tiempo de grabación

La función objetivo actual se orienta hacia tener un número equilibrado de takes grabados entre los días disponibles en el estudio. Este enfoque puede no ser el adecuado si el tiempo en el estudio implica un coste, o se busca doblar el mayor número de películas en el menor tiempo posible, ya que siempre se tenderá a expandir el número de takes grabados cada día para igualarlos.

Con este cambio en la función objetivo se pretende comprimir lo más posible en el tiempo la solución, con lo que primará completar convocatorias completas, y reducir el número total de takes grabados, es decir, reducir al mínimo las divisiones totales de take. Pese a aplicar este nuevo enfoque, el factor principal a la hora de ponderar el ajuste de una solución sigue siendo el número de convocatorias totales, que es el factor determinante en el coste de producción de la película.

Para construir esta función objetivo se hará uso de la ponderación del número de divisiones de take explicado anteriormente en la Sección (6.2.1.2). De este modo, se evaluará primero el número total de convocatorias, a continuación la ponderación de las divisiones de take, y por último la diferencia de takes con el máximo en todas las convocatorias menos en la última. Para evaluar este último factor se deben elegir todas las convocatorias menos una, que será la que menos número de takes tenga asignados. Se calcula la diferencia entre el máximo de takes por sesión y el número de takes de cada convocatoria y se va acumulando el resultado. El valor final será el que se evaluará en esta última consideración de la función objetivo.

Total de convocatorias > *Ponderación de divisiones* > *Huecos en sesiones intermedias*

Aplicar esta función objetivo puede hacer que los algoritmos genéticos usados en este trabajo funcionen de manera correcta, ya que estos tienden a aglomerar toda la planificación en las primeras sesiones de grabación.

6.2.1.4. Proximidad de grabación

El factor principal que evalúa las programaciones es el número total de convocatorias, ya que cada actor cobra una cantidad cada vez que es convocado para grabar. En el modelo del problema no se tiene en cuenta el orden de grabación de los takes, ni el tiempo que cada actor debe permanecer en el estudio de grabación durante un día para doblar todo lo que le corresponda. Pensando en un ahorro de costes, podría tenerse en cuenta el tiempo que cada actor debe permanecer en el estudio para completar el doblaje estableciendo una tarificación por horas, de modo que cuanto más próximos estén todos los takes que debe doblar un actor, menos tiempo debe permanecer este en el estudio.

Analizando el problema de programación lineal actual, no se puede tener en cuenta este factor en la función objetivo del problema, ya que no se tiene en cuenta el orden de grabación de cada take dentro de una sesión. En variaciones que se propondrán posteriormente sí se tendrá en cuenta dicho orden, con lo que este cambio sería aplicable al mismo.

Si se desea tener en cuenta el tiempo de rodaje de cada actor por sesión, lo mejor sería establecer un coste en la función objetivo, sustituyendo a la función actual que evalúa los parámetros en determinado orden. La función objetivo se definiría de la siguiente forma:

- Sea cc_a el coste de convocar al actor a a una sesión.
- Sea ch_a el coste por hora de trabajo del actor a.
- Sea nh_{as} el número de horas de trabajo del actor a en la sesión s.
- Sea ct_n el coste de unir un take dividido en n partes.

$$min\, z = \sum_{a \epsilon A} \sum_{s \epsilon S} x_{as}(cc_a + nh_{as} ch_a) + \sum_{t \epsilon T} ct_{\sum_{s \epsilon S} z_{ts}}$$

Esta función objetivo no tendría en cuenta la diferencia máxima de takes entre sesiones ya que no supone un ahorro en cuanto a costes para el estudio. Al tener en cuenta el coste de grabación y unión de los takes, esta función puede aunar las dos variaciones de la función objetivo propuestas anteriormente.

6.2.2. Variantes del problema

El modelo del problema de programación lineal de este trabajo se ha planteado siguiendo las indicaciones en el trabajo de referencia [11] que se basa en el problema real planteado por el estudio. Dicho problema puede considerarse una versión reducida ya que se omiten algunos factores que podrían ser tenidos en cuenta, como la duración de grabación de los takes, la disponibilidad del estudio de grabación... A continuación se plantean algunas de las variaciones de este problema que podrían ser tratadas utilizando este trabajo como base.

6.2.2.1. Duración de takes

Todos los takes de cada una de las películas son considerados igual, y por tanto, con la misma duración. Cabe pensar que takes con mayor número de frases y/o de actores que intervengan en el mismo conllevarán más tiempo de grabación que otros takes más simples. Por ello, se propone añadir una duración de grabación estimada a cada uno de los takes de cada película. Se podrían construir reglas de estimación de dicho tiempo a partir del número de frases del mismo y los actores que intervienen, o podría ser responsabilidad del director de doblaje de la película en cuestión.

La duración de cada take se podría indicar en el fichero de entrada de los datos de la película. El fichero de entrada usado en este trabajo ha sido preparado para introducir dicho valor para cada take, como se explica en la Sección 5.1.1. Aunque un take pueda dividirse en varias convocatorias, lo que podría reducir el tiempo de grabación al reducir el número de convocatorias, por simplicidad, debería tomarse siempre la misma duración para el take en cada división del mismo.

Al establecer una duración por cada take de grabación y considerar los takes de forma individualizada, la restricción del número de takes por sesión de grabación debería ser sustituida por un control del tiempo total de grabación. La suma de las duraciones de todos los takes de una sesión no podría nunca exceder este tope.

Teniendo en cuenta todo esto, el modelo de esta variación quedaría como sigue:

- Sea A el conjunto de actores, con índice $a \epsilon A$.
- Sea S el conjunto de sesiones de grabación, con índice $s \epsilon S$.

- Sea T el conjunto de takes, con índice $t \epsilon T$.
- Sea u_s el tiempo máximo de grabación de la sesión s, con $u_s \epsilon \mathbb{N}$.
- Sea M es el número máximo de sesiones de grabación en las que un take puede ser dividido, con $M \epsilon \mathbb{N}$.
- Sea d_t la duración de grabación del take t.
- Para todo $a \epsilon A$ y $t \epsilon T$, $\delta_{at} = \begin{cases} 1, \textit{ si el actor a interviene en el take t} \\ 0, \textit{ en otro caso} \end{cases}$

Las variables de decisión son las siguientes variables binarias:

- Para todo $a \epsilon A$ y $s \epsilon S$, $x_{as} = \begin{cases} 1, \textit{ si el actor a está en la sesión s} \\ 0, \textit{ en otro caso} \end{cases}$
- Para todo $t \epsilon T$ y $s \epsilon S$, $z_{ts} = \begin{cases} 1, \textit{ si el take t está en la sesión s} \\ 0, \textit{ en otro caso} \end{cases}$
- Para todo $a \epsilon A$, $t \epsilon T$ y $s \epsilon S$, $y_{ats} = \begin{cases} 1, & \textit{si el actor a interviene} \\ & \textit{en el take t en la sesión s} \\ 0, & \textit{en otro caso} \end{cases}$

Definidas estas variables, el problema de programación lineal sería el siguiente:

$$min\, z = \sum_{a \epsilon A} \sum_{s \epsilon S} x_{as}$$

Sujeto a:

- $\sum_{s \epsilon S} y_{ats} = \delta_{at} \quad \forall a \epsilon A, \forall t \epsilon T$
- $\sum_{t \epsilon T} z_{ts} d_t \leq u_s \quad \forall s \epsilon S$
- $\sum_{s \epsilon S} z_{ts} \leq M \quad \forall t \epsilon T$

- $y_{ats} \leq z_{ts} \quad \forall a \epsilon A, \forall t \epsilon T, \forall s \epsilon S$
- $y_{ats} \leq x_{as} \quad \forall a \epsilon A, \forall t \epsilon T, \forall s \epsilon S$
- $d_t \geq 0 \quad \forall t \epsilon T$

6.2.2.2. Ordenación de takes

En el modelo original no se tiene en cuenta el orden de los takes dentro de una convocatoria, con lo que a la hora de hacer la programación el director de doblaje debería establecer los horarios o tomar uno al azar para cada convocatoria. Por lógica, el director del estudio tratará de ordenar los takes de forma que los actores tengan que permanecer el menor tiempo en el estudio y no tener que convocar a todos durante todo el día.

Aplicar una ordenación sobre los takes de cada convocatoria no varía el modelo matemático del problema, ya que las restricciones del mismo seguirían siendo las mismas y el ahorro producido no se tendría en cuenta bajo los supuestos originales. Si se varía la función objetivo aplicando lo propuesto en la Sección 6.2.1.4, se podría cuantificar el ahorro producido, y por tanto el modelo del problema de programación lineal sería distinto.

Incluir una ordenación sobre los takes de cada convocatoria beneficiaría el funcionamiento de los algoritmos implementados basados en los algoritmos genéticos, ya que la información de los progenitores sería más estable, lo que podría llevar a hijos mejores.

6.2.2.3. Horarios de grabación

Basándose en la idea de ordenación de takes de la Sección 6.2.2.2 se puede definir una estructura temporal para organizar la programación del doblaje. De este modo, cada sesión de grabación dispondrá de un horario de grabación que debe ser cumplido, y cada uno de los takes tendrá una duración determinada, de forma que la suma de las duraciones de los takes dentro de una sesión pueda cumplirse en el horario indicado.

La duración de los takes podría tomarse en primer lugar como constante de forma que se podría construir una rejilla fija de programación sobre la que trabajarían los algoritmos. En una variante más compleja se podría introducir la duración de takes variable explicada en la Sección 6.2.1.4.

El establecimiento de un horario de doblaje establecería unas condiciones temporales que podrían ser útiles a la hora de gestionar aspectos del propio proceso, como el horario del estudio o la hora de convocación de los actores.

A la hora de definir el modelo del problema de programación lineal se tomará que el horario para cada sesión de trabajo es continuado entre dos instantes de tiempo, pero si fuera necesario podría dividirse en segmentos. El modelo quedaría de la siguiente forma:

- Sea A el conjunto de actores, con índice $a \epsilon A$.
- Sea S el conjunto de sesiones de grabación, con índice $s \epsilon S$.
- Sea T el conjunto de takes, con índice $t \epsilon T$.
- Sea M es el número máximo de sesiones de grabación en las que un take puede ser dividido, con $M \epsilon \mathbb{N}$.
- Sea d_t la duración de grabación del take t.
- Sean TI_s y TF_s los instantes de comienzo y final de la grabación de la sesión s.
- Sean ti_{ts} y tf_{ts} los instantes de comienzo y final de grabación del take t en la sesión s.
- Para todo $a \epsilon A$ y $t \epsilon T$, $\delta_{at} = \begin{cases} 1, \textit{ si el actor a interviene en el take t} \\ 0, \textit{ en otro caso} \end{cases}$

Las variables de decisión son las siguientes variables binarias:

- Para todo $a \epsilon A$ y $s \epsilon S$, $x_{as} = \begin{cases} 1, \textit{ si el actor a está en la sesión s} \\ 0, \textit{ en otro caso} \end{cases}$
- Para todo $t \epsilon T$ y $s \epsilon S$, $z_{ts} = \begin{cases} 1, \textit{ si el take t está en la sesión s} \\ 0, \textit{ en otro caso} \end{cases}$
- Para todo $a \epsilon A$, $t \epsilon T$ y $s \epsilon S$, $y_{ats} = \begin{cases} 1, & \textit{si el actor a interviene} \\ & \textit{en el take t en la sesión s} \\ 0, & \textit{en otro caso} \end{cases}$

Definidas estas variables, el problema de programación lineal sería el siguiente:

$$min\, z = \sum_{a \epsilon A} \sum_{s \epsilon S} x_{as}$$

Sujeto a:

- $\sum_{s \epsilon S} y_{ats} = \delta_{at} \quad \forall a \epsilon A, \forall t \epsilon T$
- $\sum_{s \epsilon S} z_{ts} \leq M \quad \forall t \epsilon T$
- $y_{ats} \leq z_{ts} \quad \forall a \epsilon A, \forall t \epsilon T, \forall s \epsilon S$
- $y_{ats} \leq x_{as} \quad \forall a \epsilon A, \forall t \epsilon T, \forall s \epsilon S$
- $d_t \geq 0 \quad \forall t \epsilon T$
- $TF_s \geq TI_s \quad \forall s \epsilon S$
- $tf_{ts} \geq ti_{ts} \quad \forall t \epsilon T, \forall s \epsilon S$
- $Si\, y_{ats} \neq 0, entonces\, tf_{ts} - ti_{ts} = d_t \quad \forall a \epsilon A, \forall t \epsilon T, \forall s \epsilon S$
- $ti_{ts} \geq TI_s \quad \forall t \epsilon T, \forall s \epsilon S$
- $tf_{ts} \leq TF_s \quad \forall t \epsilon T, \forall s \epsilon S$
- $ti_{ts} \geq tf_{t's}\, ó\, tf_{ts} \leq ti_{t's} \quad \forall t \epsilon T, \forall t' \epsilon T, \forall s \epsilon S$

6.2.2.4. Restricciones horarias por actor

Puede ocurrir que uno o varios actores de doblaje de las películas tengan restricciones horarias que impidan que estén disponibles durante toda la jornada, por ejemplo, que sólo pueden estar por las mañanas. Por ello, se propone una variación del problema con horarios de grabación de la Sección 6.2.2.3 donde se añade una restricción sobre los actores que impida que estos tengan programado un take fuera de horario. Los cambios en el modelo se definen como sigue:

- Sean ai_{as} y af_{as} los instantes de comienzo y final del periodo de disponibilidad del actor a para grabar en la sesión s.

- Se añaden las siguientes restricciones:
  - $ai_{as} \geq TI_s \quad \forall a \epsilon A, \forall s \epsilon S$
  - $af_{as} \leq TF_s \quad \forall a \epsilon A, \forall s \epsilon S$
  - $af_{as} \geq ai_{as} \quad \forall a \epsilon A, \forall s \epsilon S$
  - Si $y_{ats} \neq 0$ entonces $\begin{cases} ti_{ts} \geq ai_{as} & \forall a \epsilon A, \forall t \epsilon T, \forall s \epsilon S \\ tf_{ts} \leq af_{as} & \forall a \epsilon A, \forall t \epsilon T, \forall s \epsilon S \end{cases}$

Debe tenerse en cuenta que estas restricciones pueden provocar que en algunos casos el problema no sea factible, por ejemplo si dos actores que intervienen en el mismo take sólo pueden ir a una sesión y en horarios disjuntos. Esta situación se podría solventar permitiendo grabar un take varias veces en la misma sesión de grabación, pero para ello habría que realizar varias modificaciones en el modelo original.

6.2.2.5. Varias salas de doblaje

En un supuesto de un estudio de doblaje grande podría darse el caso de que hubiera más de una sala de grabación, con lo que podría paralelizarse el trabajo. Este beneficio se incrementaría si se planificaran simultáneamente varias películas en donde parte de su elenco sea compartido. Al existir más de una sala, se pueden programar más grabaciones para los actores, lo que probablemente redujera el número de convocatorias de los mismos, y por tanto el coste de doblaje de la película.

El incluir varias salas de doblaje hace que se tenga que guardar en qué sala se graba cada take y, además, como un actor no puede grabar en dos salas al mismo tiempo, se fuerza a que haya una ordenación u horario sobre la programación. Por ello, el modelo se construirá basándose en la variación que tiene en cuenta los horarios de la Sección 6.2.2.3 y quedaría de la siguiente forma:

- Sea A el conjunto de actores, con índice $a \epsilon A$.
- Sea S el conjunto de sesiones de grabación, con índice $s \epsilon S$.
- Sea T el conjunto de takes, con índice $t \epsilon T$.
- Sea E el conjunto de estudios de grabación, con índice $e \epsilon E$.

- Sea M es el número máximo de sesiones de grabación en las que un take puede ser dividido, con $M \epsilon \mathbb{N}$.
- Sea d_t la duración de grabación del take t.
- Sean TI_s y TF_s los instantes de comienzo y final de la grabación de la sesión s.
- Sean ti_{tse} y tf_{tse} los instantes de comienzo y final de grabación del take t en la sesión s usando el estudio e.
- Para todo $a \epsilon A$ y $t \epsilon T$, $\delta_{at} = \begin{cases} 1, \textit{ si el actor a interviene en el take t} \\ 0, \textit{ en otro caso} \end{cases}$

Las variables de decisión son las siguientes variables binarias:

- Para todo $a \epsilon A$ y $s \epsilon S$, $x_{as} = \begin{cases} 1, \textit{ si el actor a está en la sesión s} \\ 0, \textit{ en otro caso} \end{cases}$
- Para todo $t \epsilon T$ y $s \epsilon S$, $z_{ts} = \begin{cases} 1, \textit{ si el take t está en la sesión s} \\ 0, \textit{ en otro caso} \end{cases}$
- Para todo $a \epsilon A$, $t \epsilon T$, $s \epsilon S$ y $e \epsilon E$, $y_{atse} = \begin{cases} 1, & \textit{si el actor a interviene} \\ & \textit{en el take t en la sesión s} \\ & \textit{usando el estudio e} \\ 0, & \textit{en otro caso} \end{cases}$

Definidas estas variables, el problema de programación lineal sería el siguiente:

$$min\, z = \sum_{a \epsilon A} \sum_{s \epsilon S} x_{as}$$

Sujeto a:

- $\sum_{s \epsilon S} \sum_{e \epsilon E} y_{atse} = \delta_{at} \quad \forall a \epsilon A, \forall t \epsilon T$
- $\sum_{s \epsilon S} z_{ts} \leq M \quad \forall t \epsilon T$

- $y_{atse} \leq z_{ts} \quad \forall a \epsilon A, \forall t \epsilon T, \forall s \epsilon S, \forall e \epsilon E$
- $y_{atse} \leq x_{as} \quad \forall a \epsilon A, \forall t \epsilon T, \forall s \epsilon S, \forall e \epsilon E$
- $d_t \geq 0 \quad \forall t \epsilon T$
- $TF_s \geq TI_s \quad \forall s \epsilon S$
- $tf_{tse} \geq ti_{tse} \quad \forall t \epsilon T, \forall s \epsilon S, \forall e \epsilon E$
- $Si\, y_{atse} \neq 0,\, entonces\, tf_{tse} - ti_{tse} = d_t \quad \forall a \epsilon A, \forall t \epsilon T, \forall s \epsilon S, \forall e \epsilon E$
- $ti_{tse} \geq TI_s \quad \forall t \epsilon T, \forall s \epsilon S, \forall e \epsilon E$
- $tf_{tse} \leq TF_s \quad \forall t \epsilon T, \forall s \epsilon S, \forall e \epsilon E$
- $ti_{tse} \geq tf_{t'se}\, ó\, tf_{tse} \leq ti_{t'se} \quad \forall t \epsilon T, \forall t' \epsilon T, \forall s \epsilon S, \forall e \epsilon E$
- $Si\, y_{atse} \neq 0\, y\, y_{at'se'} \neq 0, \quad entonces\, ti_{tse} \geq tf_{t'se'}\, ó\, tf_{tse} \leq ti_{t'se'}$
  $\forall a \epsilon A, \forall t \epsilon T, \forall t' \epsilon T, \forall s \epsilon S, \forall e \epsilon E, \forall e' \epsilon E$

6.2.2.6. Combinaciones y otras variaciones

Con todas las variaciones definidas anteriormente se puede pensar en varias combinaciones que hagan uso de algunas de ellas, e incluso modificar las funciones objetivo usando alguna de las definidas en la Sección 6.2.1. De este modo se podrían crear variantes como añadir las restricciones de horario de los actores a la variante de varios estudios, e incluso modificar la función objetivo para ponderar el coste por horas de los actores, o se podrían incluir nuevas restricciones, como diferente horario de grabación para cada estudio.

El número de variaciones del problema es muy amplio y puede dar lugar a la creación de nuevos movimientos o cruces que puedan dar buenos resultados sobre los mismo, o incluso a la aplicación de nuevas técnicas de búsqueda.

6.3. Objetivos del proyecto

Tras haber finalizado el trabajo, se debe revisar si se han cumplido los objetivos que se habían marcado al comienzo del mismo en la Sección 1.2. A continuación, se detallará el trabajo realizado y se verificará si se ha alcanzado o no cada uno de los objetivos.

Objetivo 1: *Basándose en [11], se planteará el modelo matemático del problema y se implementará el algoritmo recocido simulado para el mismo. Este algoritmo se tomará como base para el problema.*

Se ha planteado el modelo matemático y se ha construido el algoritmo de recocido simulado basado en dicho trabajo. Además, se han propuesto variaciones a dicho algoritmo que se han implementado en el mismo. Sobre estas variaciones se han realizado pruebas y se ha determinado que producen una mejora en la convergencia del algoritmo.

Objetivo 2: *Construir un algoritmo de búsqueda tabú que se ajuste al problema.*

Se ha aplicado la filosofía de los algoritmos de búsqueda tabú para construir un nuevo algoritmo sobre este problema. Se ha implementado el algoritmo utilizando los movimientos del algoritmo de recocido simulado y se han realizado pruebas sobre él.

Objetivo 3: *Elaborar un algoritmo de tipo genético para resolver el problema.*

Se ha construido un algoritmo de tipo genético, para el problema que se ha implementado, y probado posteriormente. Se han utilizado los movimientos de los algoritmos de recocido simulado y búsqueda tabú para las mutaciones del algoritmo y se han creado dos tipos de cruces diferentes que se han creído útiles para este problema en concreto.

Además, se ha creado un nuevo algoritmo mezcla de recocido simulado y genético con el objetivo de aprovechar las virtudes de ambos algoritmos.

Objetivo 4: *Realizar pruebas sobre los algoritmos para confirmar su correcto funcionamiento y determinar la adecuación de cada filosofía heurística al problema.*

Se han realizado pruebas funcionales sobre cada uno de los algoritmos implementados, y una vez se ha verificado que eran correctos se han realizado pruebas sobre todos los datos de ejemplo disponibles. Se han estudiado los resultados de los algoritmos tanto individualmente como en conjunto.

Objetivo 5: *Construir una interfaz de usuario que permita a los trabajadores del estudio obtener programaciones para las películas de forma cómoda empleando el trabajo anteriormente realizado. Para no modificar en exceso el método actual de trabajo, se intentará mantener el formato de entrada y de salida con el que trabaja actualmente el estudio.*

Se ha construido una aplicación de escritorio para trabajar sobre los algoritmos desarrollados. Se han creado dos modos diferentes de trabajo sobre la aplicación, donde uno se orienta a un gestor de la misma en el cual se pueden configurar todos los parámetros y componentes de los algoritmos, y el otro modo, más simple, usaría siempre los últimos valores configurados (o los valores por defecto) de los parámetros de cada algoritmo para obtener la solución.

Aunque en un principio se deseaba mantener el formato de entrada de los datos, este se ha tenido que modificar para permitir ciertas variaciones explicadas en la Sección 5.1.1. Aún así, el cambio realizado es un cambio menor y la adaptación de los ficheros no requiere gran esfuerzo por parte de los usuarios.

Objetivo 6: *Se plantearán posibles variaciones del problema que se podrán tener en cuenta en futuros trabajos, con el objetivo de optimizar el trabajo de programación que realiza actualmente el estudio.*

En la Sección 6.2 se han planteado distintas variaciones de este problema que puede dar nuevos enfoques al mismo. Las variaciones propuestas pueden ser estudiadas en conjunto o separadas tomando este trabajo como base, ya que se ha implementado buscando la modularidad de sus componentes para hacerlo más adaptable a futuros cambios.

A la vista de lo expuesto anteriormente, se concluye que **todos los objetivos planteados al comienzo del trabajo han sido cumplidos**.

Bibliografía

[1] Documentación Framework .NET.

[2] Manual de LaTeX.

[3] R.W. Eglese. Simulated Annealing: A tool for Operational Research. *European Journal of Operational Research*, 46:271–281, 1990.

[4] F. Glover. Tabu Search - Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[5] F. Glover. Tabu Search - Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[6] F. Glover and M. Laguna. *Tabu Search*. Kluwer, Norwell, MA, 1997.

[7] D.E. Goldberg. *Genetic algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[8] D.E. Goldberg. *The design of Innovation: Lessons from and for Competent Genetics Algorithms*. Kluwer Academic Publisher, 2002.

[9] F. Hillier and G. Lieberman. *Investigación de operaciones*. Mc Graw Hill, 7 edition, 2002.

[10] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[11] Nieves R. Brisaboa, Luisa Carpente, Ana Cerdeira-Pena, and Silvia Lorenzo-Freire. Optimization in dubbing scheduling. *Enviado para su posible publicación*, Junio 2014.

[12] H.A. Taha. *Investigación de operaciones*. Prentice Hall, 7 edition, 2004.

[13] W.L. Winston. *Investigación de operaciones: Aplicaciones y algoritmos*. Thompson, 4 edition, 2004.

A

MANUAL DE USUARIO

En este capítulo se mostrará al lector el funcionamiento de la aplicación de escritorio que se ha desarrollado para permitir a los trabajadores del estudio de grabación manejar los algoritmos implementados en este trabajo y así conseguir programaciones de forma rápida y cómoda.

La aplicación permite gestionar las películas sobre las que se quiere hacer un estudio para realizar la programación, configurar los parámetros con los que se desea ejecutar los algoritmos y mostrar de forma visual los resultados obtenidos, así como guardar el resultado conseguido.

A.1. Ventana principal (Vista de usuario)

La ventana principal se muestra al arrancar la aplicación, y desde ella se podrán realizar la mayor parte de las acciones permitidas por la aplicación.

Esta ventana tiene dos modos de presentación: la Vista de usuario (explicada en este apartado) y la Vista de experto (explicada en la Sección A.2). Se puede cambiar de modo desde el menú Vista en la barra de menús (véase Sección A.1.1).

La Figura A.1 muestra el aspecto de esta ventana cuando se arranca la aplicación. Tiene diferentes elementos que permiten realizar ciertas acciones y que serán definidos

a continuación en este apartado.

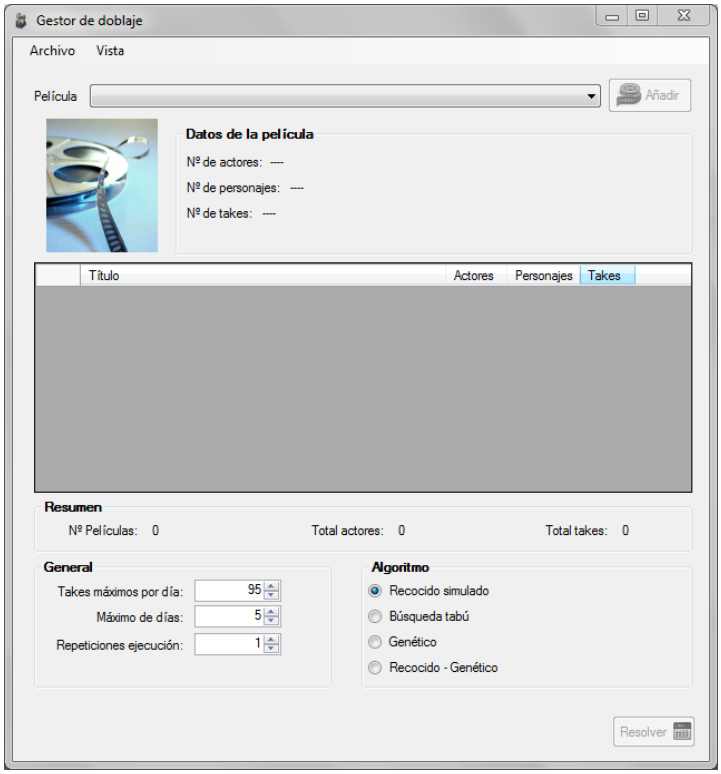

Figura A.1: Ventana principal (Vista de usuario)

A.1.1. Barra de menús

La barra de menús de la aplicación se sitúa en la parte superior de la ventana y tiene dos entradas: *Archivo* y *Vista* (véase Figura A.2).

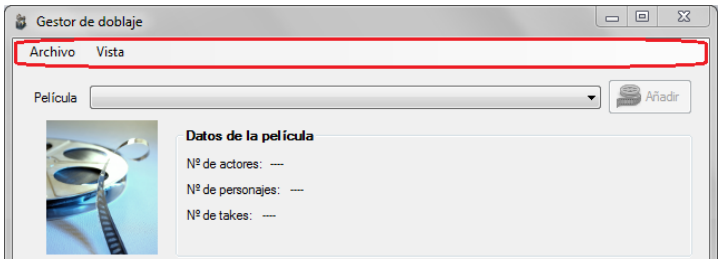

Figura A.2: Ventana Principal - Barra de menús

A.1.1.1. Menú *Archivo*

El menú *Archivo* ofrece únicamente la opción *Gestionar películas*. Esta opción abrirá una nueva ventana desde la cual se podrán gestionar las películas de la base de datos de la aplicación. Para más información sobre esta Ventana y su funcionamiento consultar la Sección A.3.

A.1.1.2. Menú *Vista*

El menú *Vista* permite seleccionar el modo de trabajo con el que se desea utilizar la aplicación. Las opciones son dos: *Usuario* y *Experto*.

El modo *Usuario* es el más simple y evita la configuración avanzada de componentes y parámetros de los algoritmos, usando así los valores por defecto o los últimos configurados en el modo *Experto*. Este modo es el configurado por defecto para la aplicación.

El modo *Experto* permite gestionar todos los parámetros de los algoritmos ejecutados desde la aplicación. Para más información acerca de este modo véase la Sección A.2 de este manual.

A.1.2. Selección de películas

En la parte superior del cuerpo de la Ventana principal (véase FiguraA.3) se puede realizar la selección de las películas que van a ser tratadas por los algoritmos en la aplicación.

La parte superior de esta sección de la ventana permite seleccionar una de las películas disponibles en la base de datos para ser añadida a la programación que se pretende hacer. Al seleccionar una película en el menú desplegable se muestran sus datos más relevantes (número de actores, de personajes y de takes) justo debajo, como se muestra en la Figura A.4.

Una vez seleccionada una película, esta se puede añadir a la lista de películas de la programación. Para ello, se debe pulsar el botón *Añadir*, situado a la derecha del desplegable de las películas, lo que incluirá la película a la lista, la eliminará del desplegable, y por tanto, borrará los datos de la película seleccionada, y actualizará los datos

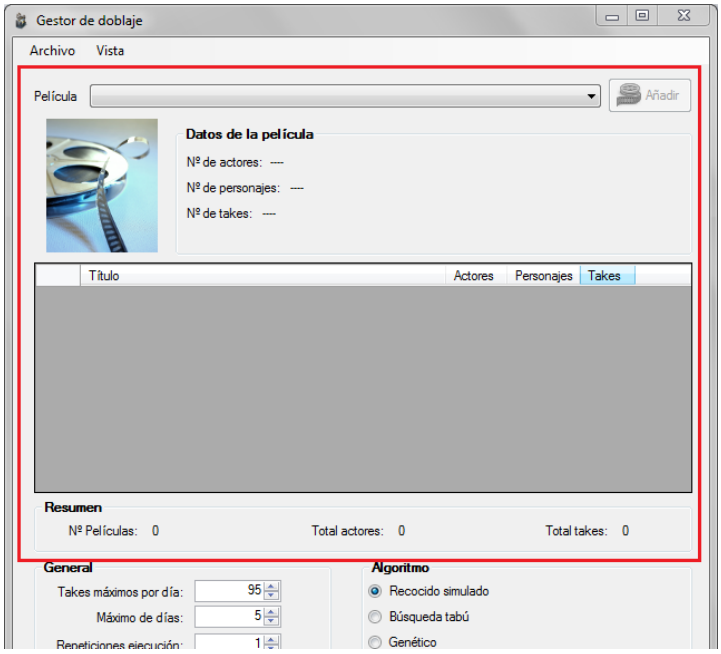

Figura A.3: Ventana principal - Selección de películas

Figura A.4: Ventana principal - Seleccionar una película

Resumen de la lista de películas añadidas a la programación. La Figura A.5 muestra este comportamiento.

Se puede repetir este proceso para todas las películas disponibles en la base de datos.

Si se desea eliminar una película de la programación, se debe pulsar el botón *Eliminar* correspondiente a la película en la tabla de películas añadidas a la programación (véase Figura A.6).

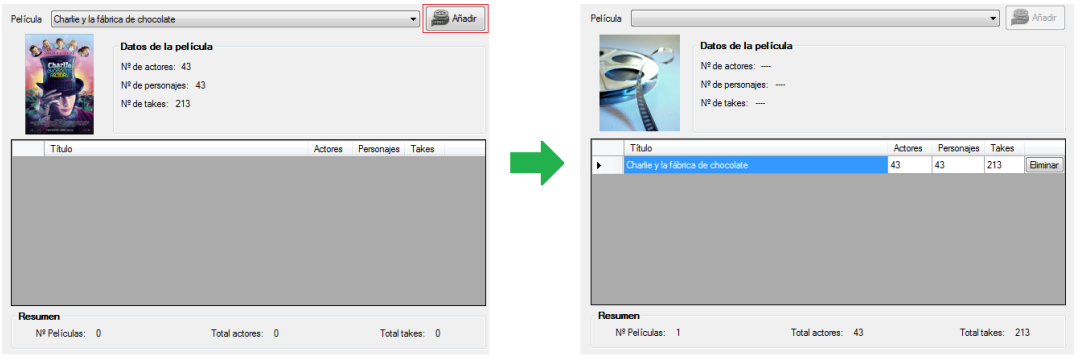

Figura A.5: Ventana principal - Añadir película

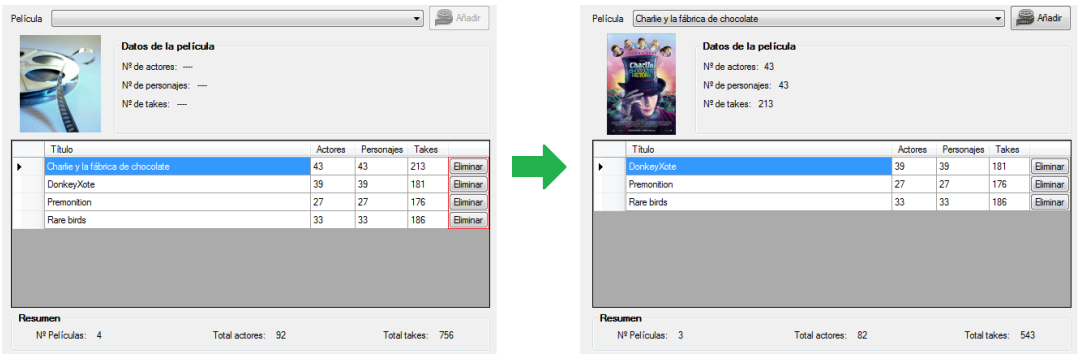

Figura A.6: Ventana principal - Eliminar película

A.1.3. Parámetros de resolución

En la parte inferior de esta ventana, encima del botón *Resolver*, se encuentra este apartado de configuración (véase Figura A.7). En él se deben configurar los parámetros que se tendrán en cuenta a la hora de calcular la programación de las películas, así como el tipo de algoritmo a emplear para buscar dicha programación.

El grupo *General* contiene los parámetros para el cálculo de solución, y son los siguientes:

- *Takes máximos por día:* número de takes máximo que pueden ser doblados en una misma sesión de grabación.
- *Máximo de días:* número de días disponibles para realizar el doblaje completo de la película.

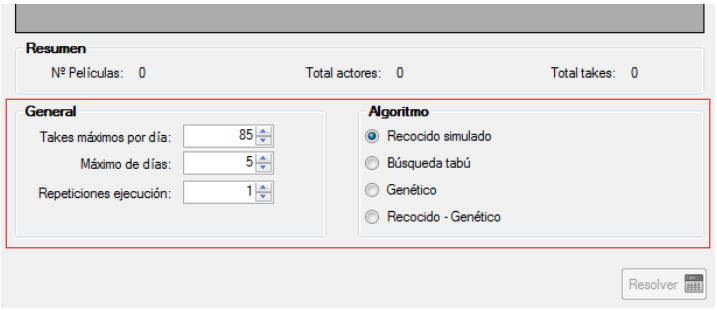

Figura A.7: Ventana principal - Parámetros de resolución

- *Repeticiones de ejecución:* número de veces que se ha de repetir el algoritmo antes de dar por finalizada la búsqueda de la programación.

El grupo *Algoritmo* permite seleccionar el algoritmo empleado para realizar la búsqueda de la solución entre los disponibles. Estos son:

- *Recocido simulado:* algoritmo de recocido simulado.
- *Búsqueda tabú:* algoritmo de búsqueda tabú.
- *Genético:* algoritmo genético.
- *Recocido - Genético:* algoritmo genético con una población inicial creada usando el algoritmo de recocido simulado.

A.1.4. Botón *Resolver*

Este botón, situado en la parte inferior derecha de la ventana (véase Figura A.8), lanza la ejecución del algoritmo de búsqueda indicado. Cuando se lanza un algoritmo se muestra la *Ventana de ejecución* (Sección A.4) que irá actualizando los datos del mismo en tiempo real.

Al abrir la ventana este botón está deshabilitado, y sólo se habilita si la lista de películas añadidas a la programación no está vacía.

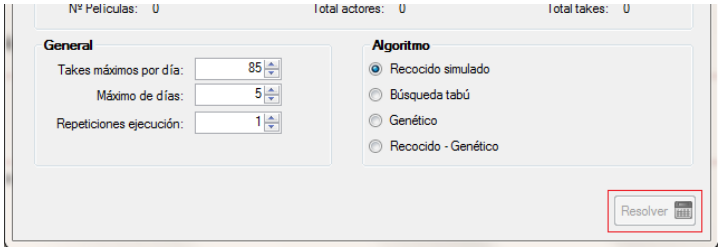

Figura A.8: Ventana principal - Botón Resolver

A.2. Ventana principal (Vista de experto)

La *Vista de experto* es un modo de la aplicación orientado a usuarios de la misma que quieren variar los parámetros de configuración de los algoritmos. Se puede cambiar el modo de la aplicación desde el menú *Vista* en la barra de menús (véase Sección A.1.1).

La Figura A.2 muestra el aspecto de la Ventana principal en modo experto cuando arranca la aplicación.

En esta vista se encuentran dos diferencias con respecto a la Vista de usuario explicada en la Sección A.1:

- Se añade una barra que indica el paso en el que se encuentra la configuración de la aplicación para realizar la ejecución. Estos pasos son: *Inicio* y *Configuración*.
- Se cambia el botón *Resolver* (ver Sección A.1.4) por el botón *Siguiente* que permitirá avanzar del paso *Inicio* al paso *Configuración*. Este botón tiene el mismo comportamiento en cuanto a habilitación que el botón *Resolver*.

El funcionamiento de toda la ventana es el mismo que en el modo usuario considerando que las Secciones *Selección de películas* (A.1.2) y *Parámetros de resolución* (A.1.3) forman parte del paso *Inicio* de este modo, y que el botón *Siguiente* lleva al paso de *Configuración* en lugar de lanzar el algoritmo como hace el botón *Resolver* (A.1.4).

A.2.1. Paso Configuración

En este paso el usuario de la aplicación en modo experto podrá modificar los parámetros de los algoritmos. La información mostrada en este paso dependerá de qué

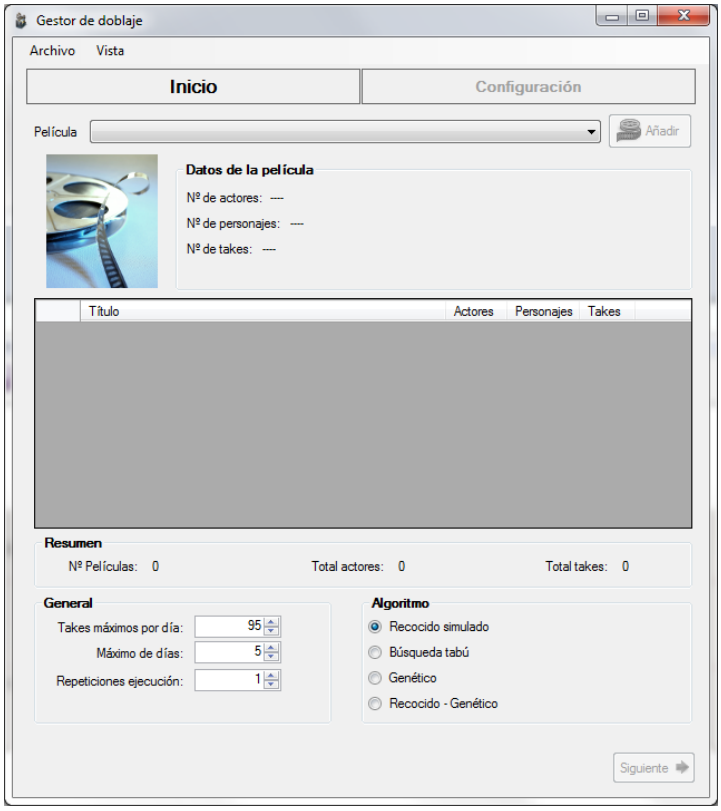

Figura A.9: Ventana principal (Vista de experto)

algoritmo se ha seleccionado en el paso Inicio, pero en la parte inferior siempre se encontrarán dos botones que permiten salir de este paso (ver Figura A.10).

- Botón *Atrás*: permite volver al paso Inicio y modificar la información que en él se encuentra.
- Botón *Resolver*: lanza la ejecución del algoritmo con todos los parámetros configurados. Una vez se pulsa este botón se guarda la configuración seleccionada para el algoritmo como configuración por defecto, con lo que las siguientes ejecuciones, tanto en modo usuario como en modo experto, utilizarán estos parámetros. Al igual que ocurría con este botón en la *Vista de usuario*, al pulsar el botón se mostrará la *Ventana de ejecución* (Sección A.4).

A continuación se muestran las diferentes configuraciones que se obtienen según el algoritmo que se haya seleccionado en el paso *Inicio*.

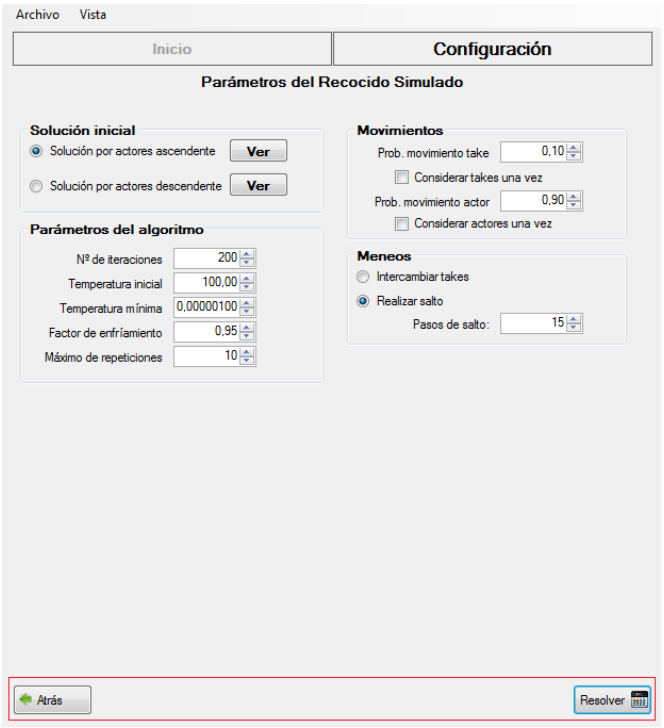

Figura A.10: Ventana principal - Paso Configuración

A.2.1.1. Configuración del algoritmo de recocido simulado

La Figura A.11 muestra la configuración del algoritmo de recocido simulado que se verá en el paso *Configuración* cuando se haya seleccionado en el paso *Inicio* dicho algoritmo.

En el grupo *Solución inicial* se debe escoger el mecanismo de generación de la solución inicial del algoritmo que puede ser *Solución por actores ascendente* o *Solución por actores descendente*. Estas soluciones se pueden ver desde esta ventana pulsando el botón *Ver* correspondiente que abrirá la Ventana de solución (Sección A.5) mostrando esta solución.

En el grupo *Parámetros del algoritmo* se pueden configurar los valores de los parámetros que empleará el algoritmo en su ejecución. Estos parámetros son:

- *Nº de iteraciones:* número de veces que se va a repetir el proceso de recocido simulado.

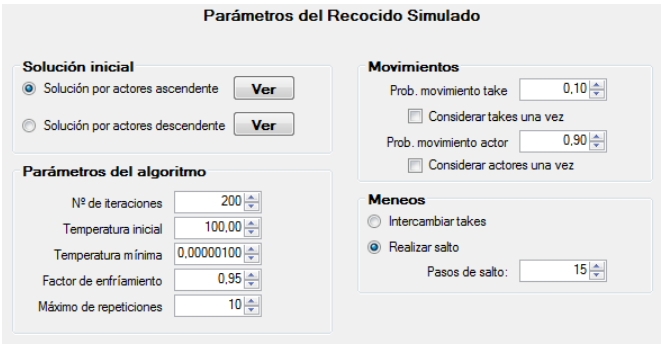

Figura A.11: Ventana principal - Configuración recocido simulado

- *Temperatura inicial:* valor de la temperatura de la que parte el algoritmo.
- *Temperatura mínima:* valor mínimo de la temperatura a la que se da por finalizada la iteración del algoritmo.
- *Factor de enfriamiento:* factor que determina como se decrementa la temperatura en cada paso del algoritmo.
- *Máximo de repeticiones:* número máximo de repeticiones de la solución actual antes de ejecutar el mecanismo de escape del algoritmo.

El grupo *Movimientos* permite configurar cómo se deben tomar estos en la ejecución del algoritmo. Se deben configurar las dos probabilidades de los movimientos que determinarán la frecuencia de ejecución de cada uno de ellos. Los marcadores *Considerar takes/actores una vez* indican si se debe ponderar el número de apariciones de cada take/actor a la hora de elegirlos en el movimiento.

En el grupo *Meneos* se puede seleccionar el movimiento que se ejecutará como mecanismo de escape del algoritmo. Si se selecciona el movimiento *Realizar salto* se podrá variar el número de pasos cambiando el valor en *Pasos de salto*.

A.2.1.2. Configuración del algoritmo de búsqueda tabú

El paso *Configuración* mostrará la información de la Figura A.12 cuando se selecciona el algoritmo de búsqueda tabú en el paso *Inicio*.

En el grupo *Solución inicial* se debe escoger el mecanismo de generación de la

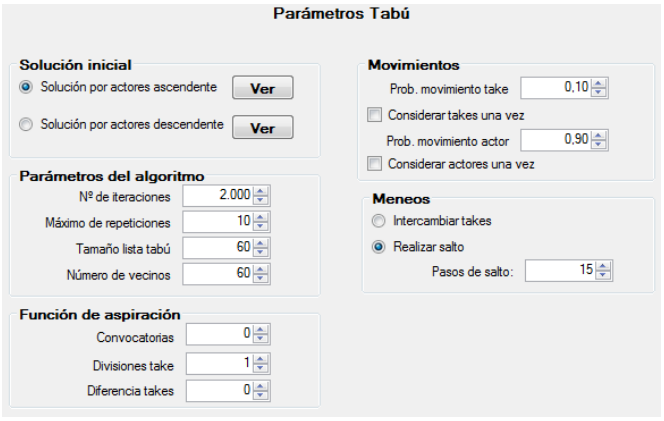

Figura A.12: Ventana principal - Configuración búsqueda tabú

solución inicial del algoritmo que puede ser *Solución por actores ascendente* o *Solución por actores descendente*. Estas soluciones se pueden ver desde esta ventana pulsando el botón *Ver* correspondiente que abrirá la Ventana de solución (Sección A.5) mostrando esta solución.

En el grupo *Parámetros del algoritmo* se pueden configurar los valores de los parámetros que empleará el algoritmo en su ejecución. Estos parámetros son:

- *Nº de iteraciones:* número de veces que se va a repetir el proceso de recocido simulado.
- *Máximo de repeticiones:* número máximo de repeticiones de la solución actual antes de ejecutar el mecanismo de escape del algoritmo.
- *Tamaño lista tabú:* número máximo de elementos que puede contener la lista tabú.
- *Número de vecinos:* número de movimientos diferentes que se generarán en cada iteración del algoritmo.

En el grupo *Función de aspiración* se pueden configurar los valores de los parámetros de la función objetivo que deben mejorarse para ejecutar un movimiento de la lista tabú. Estos valores son:

- *Convocatorias:* diferencia de convocatorias entre la nueva programación y la actual.

- *Divisiones take:* diferencia en el máximo número de divisiones de un take entre la nueva programación y la actual.
- *Diferencia takes:* diferencia entre la máxima diferencia de takes entre la programación actual y la nueva programación.

El grupo *Movimientos* permite configurar cómo se deben tomar estos en la ejecución del algoritmo. Se deben configurar las dos probabilidades de los movimientos que determinarán la frecuencia de ejecución de cada uno de ellos. Los marcadores *Considerar takes/actores una vez* indican si se debe ponderar el número de apariciones de cada take/actor a la hora de elegirlos en el movimiento.

En el grupo *Meneos* se puede seleccionar el movimiento que se ejecutará como mecanismo de escape del algoritmo. Si se selecciona el movimiento *Realizar salto* se podrá variar el número de pasos cambiando el valor en *Pasos de salto*.

A.2.1.3. Configuración del algoritmo genético

Al seleccionar el algoritmo genético en el paso *Inicio*, el paso *Configuración* mostrará la información de la Figura A.13.

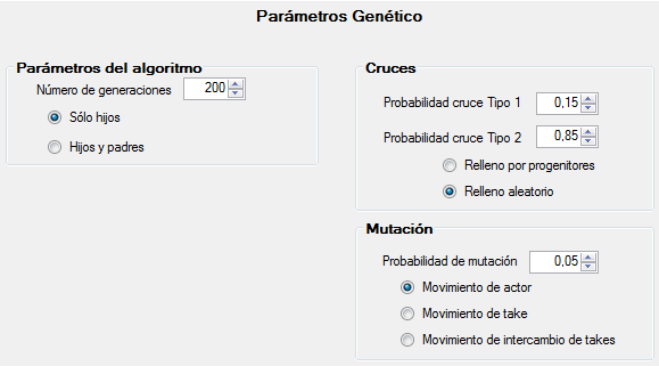

Figura A.13: Ventana principal - Configuración genético

En el grupo *Parámetros del algoritmo* se pueden configurar los valores que se usarán en la ejecución del algoritmo. El único parámetros que se puede configurar es el *Número de generaciones*, que es el total de generaciones que se crearán antes de dar por finalizada la ejecución del algoritmo. El tipo de las nuevas generaciones puede ser *Sólo hijos*, donde las nuevas generaciones estarán compuestas únicamente por nuevas programaciones, o

Hijos y padres, donde los padres usados para generar la descendencia se incluyen en la nueva generación además de los hijos.

En el grupo *Cruces* se deben establecer todos los valores referentes a los cruces usados en el algoritmo. Se deben configurar las probabilidades de usar cada uno de los dos cruces implementados. Para el segundo cruce hay que configurar también el tipo de relleno que se realizará tras añadir la información común a los progenitores. Los posibles rellenos son:

- *Relleno por progenitores:* los huecos de la programación se rellenarán siguiendo el orden establecido en uno de los progenitores, que será elegido al azar para cada par take-actor.
- *Relleno aleatorio:* la programación se completará incluyendo todos los pares take-actor que falten en la misma de forma aleatoria.

El grupo *Mutación* contiene toda la configuración de las mutaciones que aplicará el algoritmo. Se debe configurar la probabilidad de que surja una mutación al generar un hijo, así como el movimiento que se aplicará para generar dicha mutación. Dicho movimiento debe ser uno de los siguientes: *Movimiento de actor*, *Movimiento de take* o *Movimiento de intercambio de takes.*

A.2.1.4. Configuración del algoritmo de recocido simulado - genético

En la Figura A.14 se puede ver la información mostrada al avanzar al paso *Configuración* desde el paso *Inicio* habiendo seleccionado el algoritmos de recocido simulado - genético.

Los parámetros que se muestran en este paso de configuración son la unión de los parámetros del algoritmo de recocido simulado que se definen en la Sección A.2.1.1, y los parámetros del algoritmo genético que se detallan en la Sección A.2.1.3.

A.3. Ventana de gestión de películas

La ventana de gestión de películas (véase Figura A.15) permite administrar los datos de las películas que hay cargadas en la base de datos de la aplicación. Desde esta

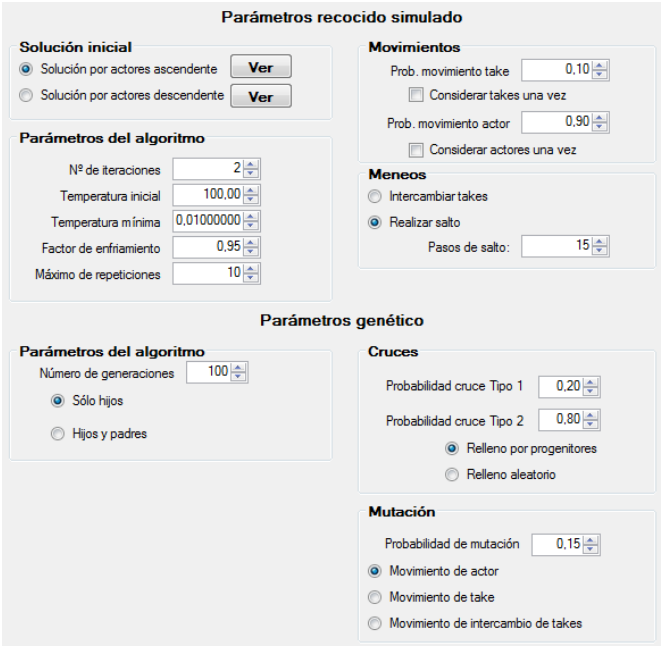

Figura A.14: Ventana principal - Configuración recocido simulado - genético

ventana el usuario podrá ver o editar los datos registrados de la películas, y además añadir nuevas o eliminar alguna de las existentes.

El desplegable de la parte superior, contiene los títulos de todas las películas registradas en la base de datos en ese momento. Al actualizar la selección de dicho desplegable se actualizarán los datos inmediatamente inferiores al mismo, y además se habilitarán los botones *Actualizar imagen* y *Eliminar* (véase Figura A.16).

Los datos mostrados se corresponden con los datos resumen de la película seleccionada, y los dos botones habilitados tienen las siguientes funcionalidades:

- *Actualizar imagen*: permite seleccionar un nuevo fichero de imagen que se actualizará en la base de datos de películas. Se tratará de una imagen representativa de la misma.
- *Eliminar*: borra todos los datos de la película de la base de datos. Al pulsarlo se mostrará una Ventana de proceso (Sección A.6)

En el grupo *Nueva película* de la ventana (Figura A.17) se puede registrar una película

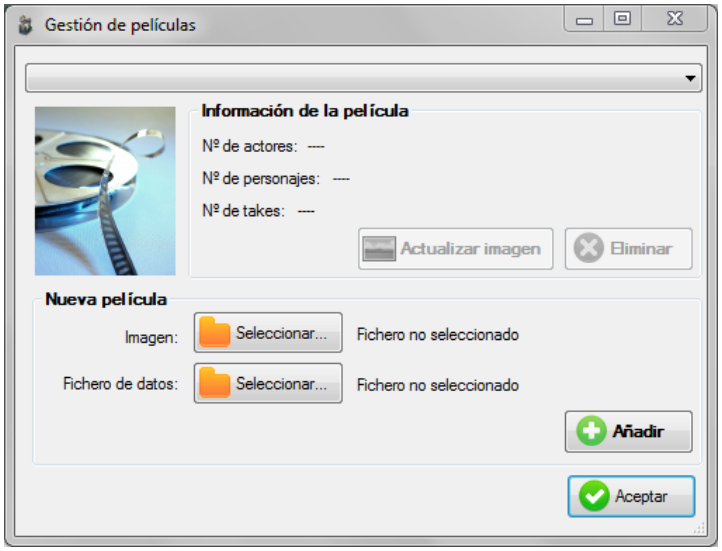

Figura A.15: Ventana de gestión de películas

Figura A.16: Ventana de gestión de películas - Seleccionar película

nueva en la base de datos. Para ello se debe seleccionar al menos un *Fichero de datos*, con el formato correcto de entrada, para cargar los datos de la misma. Adicionalmente se puede seleccionar una *Imagen* que será usada para representar la película en la aplicación. Una ver se haya finalizado de seleccionar los ficheros correspondientes, pulsando en el botón *Añadir* se procederá a registrar la película, mostrando una Ventana de proceso (Sección A.6). Una vez concluido el proceso, el título de la película aparecerá en el desplegable de la parte superior de la ventana.

La ventana posee un botón *Aceptar* en la parte inferior de la misma, que la cerrará volviendo a la vista principal de la aplicación.

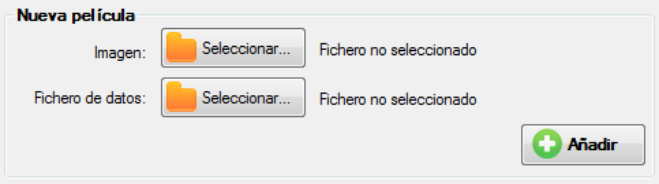

Figura A.17: Ventana de gestión de películas - Nueva película

A.4. Ventana de ejecución

La *Ventana de ejecución* (ver Figura A.18) muestra datos importantes de la ejecución de los algoritmos que permiten llevar un control de cómo se está llevando a cabo la ejecución.

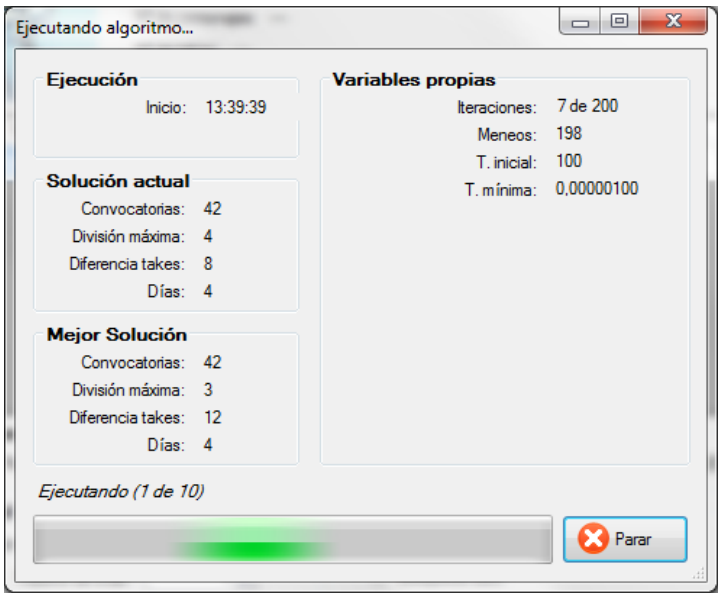

Figura A.18: Ventana de ejecución

Esta ventana contiene una parte común a todos los algoritmos, la columna izquierda y la parte inferior, y una parte variable según el algoritmos que se esté ejecutando, contenida dentro del grupo *Variables propias*. A continuación se detallará la parte común y posteriormente la parte variable dependiendo del algoritmo.

A.4.1. Parte común

La parte común contiene toda la información mostrada en la ventana salvo el grupo *Variables propias* que es variable según el algoritmo en ejecución.

El grupo *Ejecución* tiene un único parámetro *Inicio* que indica la hora en la que se ha lanzado el algoritmo, lo que permite determinar el tiempo que lleva ejecutándose.

En el grupo *Solución actual* se muestran los datos de la solución con la que actualmente está trabajando el algoritmo, y en el grupo *Mejor solución* se muestran los datos de la mejor solución encontrada hasta el momento en esa ejecución del algoritmo. Si se han planificado dos o más ejecuciones, esta solución se establece a la mejor solución inicial con cada nueva ejecución.

Los valores indicados en estos dos grupos son:

- *Convocatorias*: número total de convocatorias para la programación.
- *División máxima*: número máximo de divisiones por take de la programación.
- *Diferencia takes*: diferencia máxima de takes entre dos sesiones de grabación para la programación.
- *Días*: número total de sesiones de grabación no vacías de la programación.

En la parte inferior de la ventana se muestra una barra de ejecución, que indica que se está trabajando en el algoritmo, donde se indica el número de la ejecución actual sobre el total programado (véase Figura A.19). La *Ventana de ejecución* se cerrará automáticamente una vez se haya completado el número total de ejecuciones, abriendo la *Ventana solución* (Sección A.5) con la mejor solución encontrada hasta el momento.

Al lado de la barra de progreso de la parte inferior se encuentra el botón *Parar*, que permite detener la ejecución del algoritmo, cerrando la *Ventana de ejecución* y mostrando la *Ventana de solución* (Sección A.5) con la mejor solución encontrada hasta ese momento.

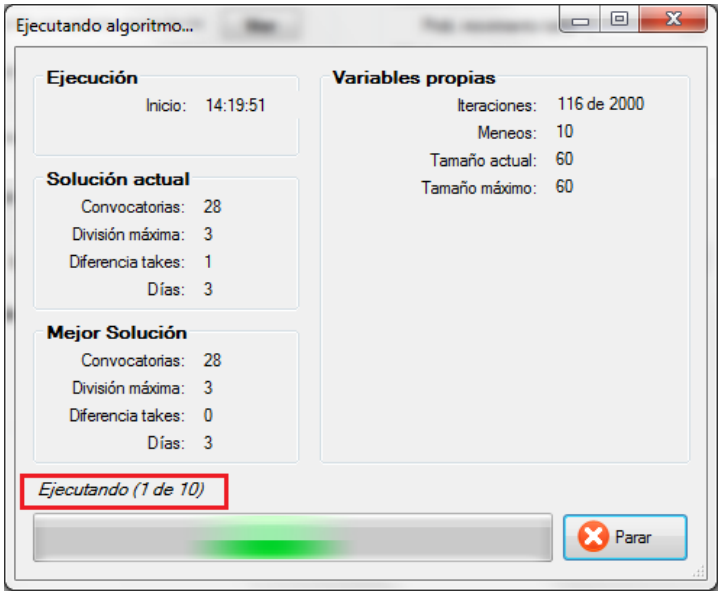

Figura A.19: Ventana de ejecución - Progreso

A.4.2. Parte variable

El grupo Variables propias de la ventana contiene la información variable según el algoritmo que se esté ejecutando. A continuación se detallarán las variables que aparecerán según el algoritmo.

A.4.2.1. Algoritmo de recocido simulado

Las variables propias de este algoritmo se muestran en la Figura A.20 y son las siguientes:

- *Iteraciones*: indica el número actual y el máximo de iteraciones realizadas del algoritmo.
- *Meneos*: muestra el número total de meneos que lleva realizados el algoritmo.
- *T. inicial*: temperatura inicial del algoritmo. Este parámetro no varía durante la ejecución.
- *T. mínima*: temperatura mínima en la que se da por finalizada la ejecución del algoritmo. Este parámetro no varía durante la ejecución.

Figura A.20: Ventana de ejecución - Variables recocido simulado

A.4.2.2. Algoritmo de búsqueda tabú

El algoritmo de búsqueda tabú muestra las variables indicadas en la Figura A.21 y explicadas a continuación:

- *Iteraciones*: muestra el número actual y total de iteraciones del algoritmo.
- *Meneos*: muestra el número total de meneos que lleva realizados el algoritmo.
- *Tamaño actual*: número de elementos contenidos en la lista tabú actualmente.
- *Tamaño máximo*: número de elementos máximo que pueden estar contenidos en la lista tabú. Este parámetro no varía durante la ejecución.

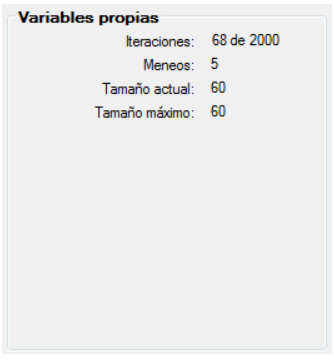

Figura A.21: Ventana de ejecución - Variables búsqueda tabú

A.4.2.3. Algoritmo genético

La Figura A.22 muestra las variables que se podrán ver en la ejecución del algoritmo genético. Dichas variables son las siguientes:

- *Generaciones*: número actual y total de las generaciones que se crean para el algoritmo.
- *Prob. Mutación*: probabilidad de realizar una mutación al generar un descendiente. Este parámetro no varía durante la ejecución.
- *Mov. Mutación*: movimiento que se realiza cuando se produce una mutación. Este parámetro no varía durante la ejecución.

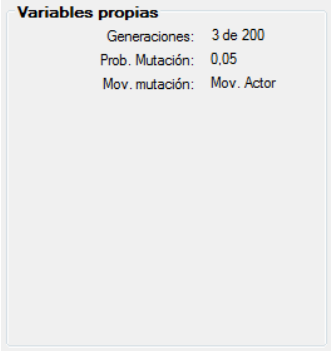

Figura A.22: Ventana de ejecución - Variables genético

A.4.2.4. Algoritmo de recocido simulado - genético

Las variables mostradas en la ejecución de este algoritmo dependerán del algoritmo que se esté ejecutando en cada momento. Así, al generar la población inicial se mostrarán las variables propias del algoritmo de recocido simulado explicadas en la Sección A.4.2.1, y cuando se comience a iterar sobre dicha población se mostrarán las variables propias del algoritmo genético detalladas en la Sección A.4.2.3.

A.5. Ventana de solución

Esta ventana, mostrada en la Figura A.23, contiene la programación de una solución detallando cómo se reparten los actores en ella y el número de takes en los que participan en cada sesión.

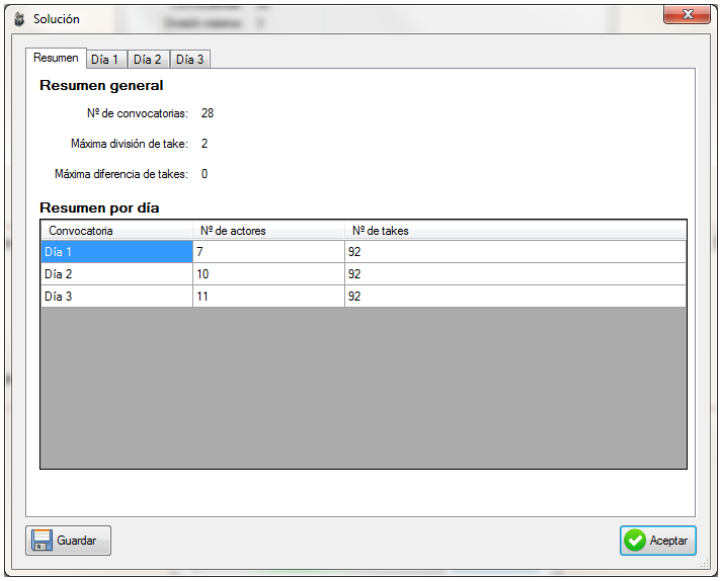

Figura A.23: Ventana de solución

La ventana contiene un panel de pestañas, en la parte superior, donde se muestra el resumen de la programación y la planificación para cada uno de los días. Además, en la parte inferior, hay dos botones con las siguientes funcionalidades:

- El botón *Guardar* permite salvar la programación mostrada en la ventana en un fichero de Excel en el formato de salida de la aplicación.
- El botón *Aceptar* cierra la ventana. Al cerrar la ventana la solución no será guardada, con lo que no se podrá volver a recuperar a no ser que hubiera sido salvada antes pulsando el botón *Guardar*.

A.5.1. Pestaña *Resumen*

La pestaña Resumen, mostrada en la Figura A.24, contiene la información resumida de la programación encontrada.

En la parte superior contiene un *Resumen general* con los siguientes datos de la solución encontrada:

- *Nº de convocatorias*: número total de convocatorias de los actores.

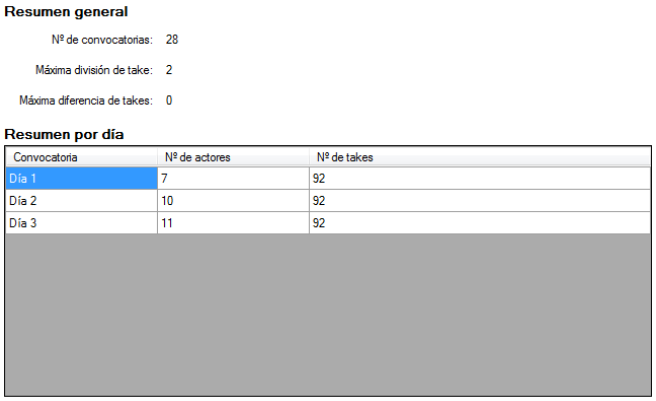

Figura A.24: Ventana de solución - Pestaña Resumen

- *Máxima división de take*: número máximo de divisiones de un mismo take en la programación.

- *Máxima diferencia de takes*: diferencia máxima existente entre el número de takes grabados en dos sesiones diferentes de la programación.

Se muestra además un *Resumen por día* de la programación, donde para cada sesión de grabación se mostrará el *Nº de actores* y el *Nº de takes*.

A.5.2. Pestaña Día N

La *Ventana de solución* contiene una pestaña de este tipo por cada día de grabación que exista en la convocatoria. En el título de la pestaña se mostrará el número de día que representa la solución (ver Figura A.25).

Figura A.25: Ventana de solución - Pestañas

En cada una de estas pestañas se mostrará la información que se puede ver en la Figura A.26. En ella hay dos tablas que muestran la información detallada de la programación para ese día.

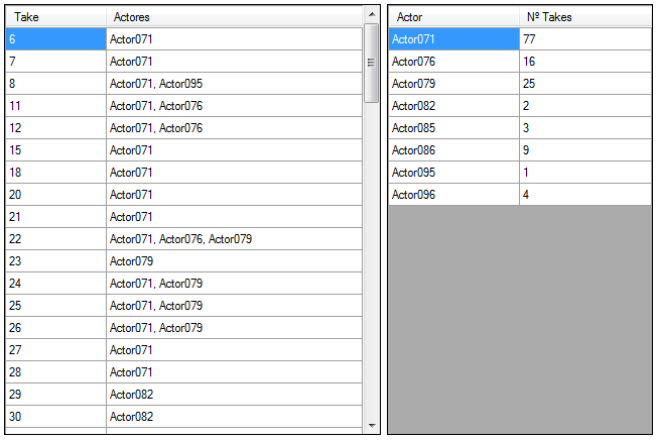

Figura A.26: Ventana de solución - Pestaña Día N

En la lista de la izquierda se incluyen todos los takes que se graban para este día, indicando los actores que tienen que intervenir en él.

La lista de la parte derecha muestra el resumen para cada actor en el día en cuestión, indicando el número de takes que debe grabar cada uno.

A.6. Ventana de proceso

Esta ventana se muestra cuando se está llevando a cabo un proceso dentro de la aplicación, y bloquea la aplicación mientras este se lleva a cabo. La Figura A.27 muestra esta ventana.

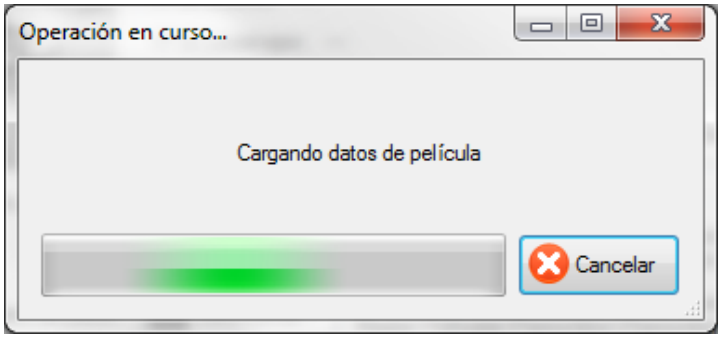

Figura A.27: Ventana de proceso

La ventana muestra un mensaje describiendo el tipo de operación que se está llevando

a cabo y un botón Cancelar que permite detener dicho proceso. Este botón no siempre está habilitado, ya que hay operaciones que no pueden ser canceladas. En la Figura A.28 se muestra esta ventana con el botón Cancelar deshabilitado.

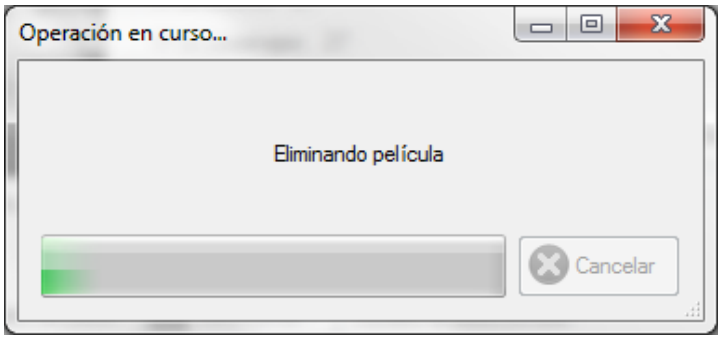

Figura A.28: Ventana de proceso - Cancelar deshabilitado