UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MIARFID
Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital
Universitat Politècnica de València

Diseño de un algoritmo genético para optimizar el rendimiento de una pesadora de naranjas

Herramientas y Aplicaciones de la Inteligencia Artificial

Autor: Jacobo López Fernández
Juan Antonio López Ramírez

Curso 2019-2020

Índice general

CAPÍTULO 1
Introducción

El problema que nos ocupa se engloba dentro del tratamiento de productos hortofrutícolas, concretamente, de cítricos.

Las pesadoras de naranjas están gestionadas por un dispositivo electrónico que realiza la selección de un conjunto de elementos con un determinado peso para la confección de mallas.

La cinta de alimentación vuelca el contenido de la malla en un grupo de cubetas distribuidas en disposición circular, y que giran a una determinada velocidad de vuelta de carrusel.

El sistema posee un patín de pesado por el que pasa una cubeta en cada momento para determinar el peso del conjunto de elementos que contiene cada cubeta. Además, posee una serie de electroimanes (10 para una pesadora fija) que permiten la apertura simultánea de una serie de cubetas con el fin de obtener una combinación de elementos que cumplan con una condición de peso establecida.

Por último, el sistema hace uso de un sensor que determina el estado (abierto o cerrado) de la cubeta para verificar el correcto funcionamiento de apertura del sistema. Tras el sensor de cubeta abierta, todas las cubetas se cierran mecánicamente para así poder ser llenadas al pasar por debajo de la cinta de alimentación.

La finalidad del presente trabajo consiste en conseguir una combinación de cubetas, gracias a la cual se optimice una determinada condición de peso.

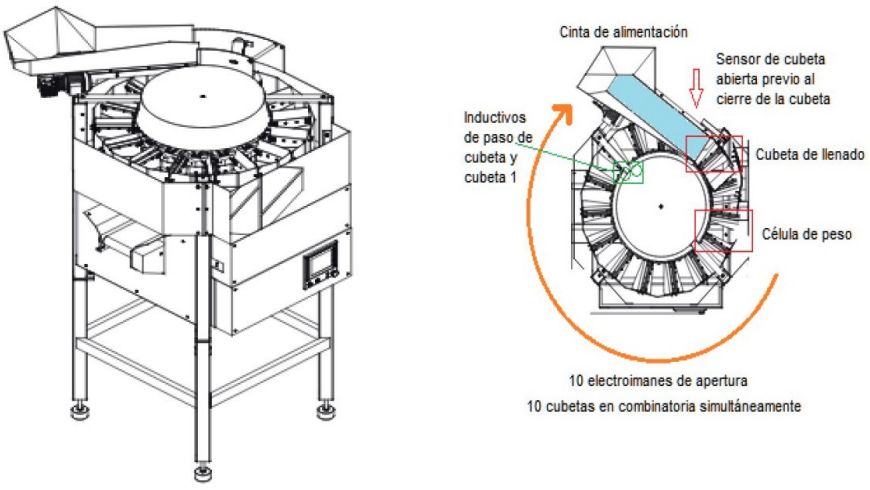

Figura 1.1: Pesadora de naranjas y su funcionamiento.

CAPÍTULO 2
Modelado del problema

Dado que el número de cubetas habilitadas en cada sincronismo de peso es de 10 cubetas, se tratará de encontrar la mejor combinación de 10 cubetas necesaria para obtener una confección, es decir, determinar cuales de las 10 se deben seleccionar para un peso de malla.

Por tanto, la entrada del problema será:

- El peso de la malla.
- El porcentaje máximo de llenado de cubeta respecto al peso de la confección.
- Un vector de 10 elementos con el peso de cada cubeta. Cada uno de los 10 pesos por cubeta se generará de manera aleatoria, pero siempre menor o igual que el porcentaje máximo por cubeta.
- Número mínimo de cubetas a seleccionar.
- Número máximo de cubetas a seleccionar.

La salida del problema será un vector de 10 elementos, donde cada uno de ellos tendrá un valor de 1 (cubeta activada) o 0 (cubeta desactivada).

Para resolverlo, se ha elegido la técnica del algoritmo genético, cuya implementación se explicará a continuación.

CAPÍTULO 3

Diseño e implementación del algoritmo genético

Un algoritmo genético está basado en la evolución natural y la genética. Al igual que la población evoluciona de forma natural (mejorando su adaptación), un algoritmo genético lo hace a partir de una población de soluciones iniciales, intentando producir nuevas soluciones que sean mejores que las anteriores.

El proceso de evolución es guiado por decisiones probabilísticas basadas en la aptitud de los individuos (marcada por una función de evaluación o *fitness*) hasta obtener un buen individuo, esto es, una buena solución global al problema.

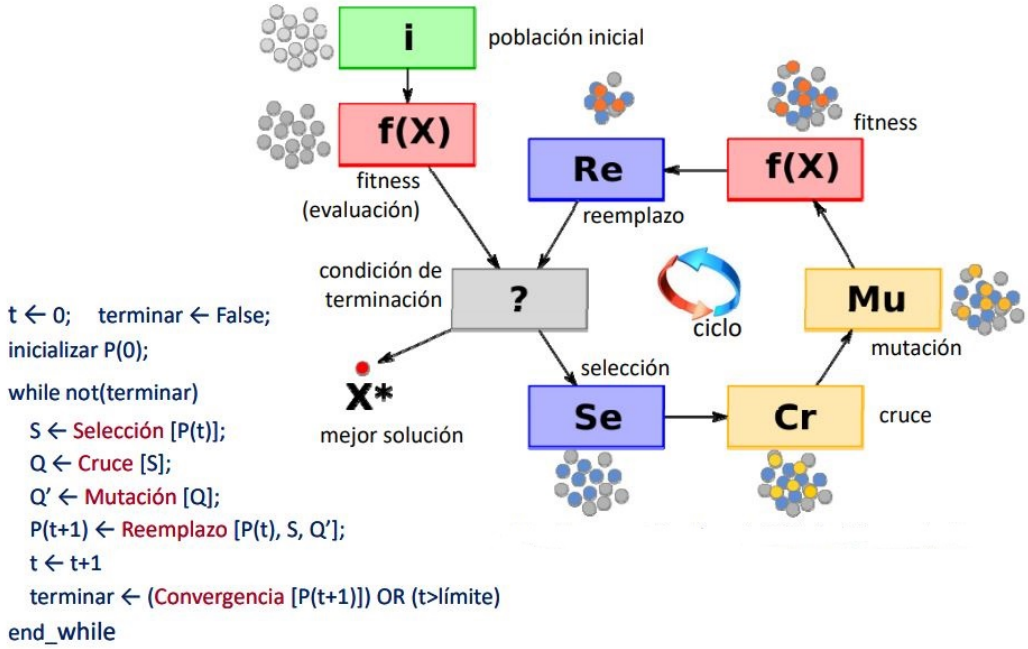

Figura 3.1: Estructura del algoritmo que se ha seguido para implementar nuestro genético.

3.1 Esquema general del algoritmo genético

Nuestro algoritmo realiza el esquema tradicional de un genético (y que se puede apreciar en la figura 3.1), con la modificación de que, a cada iteración, devuelve la mejor solución que lleva hasta ahora y la iteración en la que se encontró dicha solución.

Una vez el algoritmo finaliza, devuelve la solución óptima, si la ha encontrado; o la mejor solución que ha sido capaz de obtener.

De esta forma, la función se ve influenciada por el número máximo de iteraciones, encontrar la solución óptima al problema o encontrar un posible mínimo local sin conseguir mejorar la menor aptitud del conjunto de soluciones halladas.

El diseño general de nuestro algoritmo genético se ha implementado mediante un método en *Python* con cuatro parámetros de entrada, que son:

- *maxIter*. Establece el número de iteraciones máximas que ha de realizar nuestro algoritmo.
- *pobIni*. Establece el número de individuos que forman la población inicial.
- *probCruce*. Determina el porcentaje de la población que será seleccionada para el cruce (en tanto por uno).
- *probMutacion*. Determina el porcentaje de la nueva generación, originada en el cruce, que será seleccionada para la mutación (en tanto por uno).

3.2 Codificación del individuo y generación de la población inicial

En el problema de las naranjas, un posible individuo sería un vector de 10 elementos, donde cada uno tiene un valor de 1 (si esa cubeta está activada) o 0 (si la cubeta está desactivada). Como restricción se ha añadido que la suma de todos los componentes del individuo sea mayor que el número mínimo de cubetas y menor que el número máximo de cubetas. Esto se debe a que, como los componentes están a 0 o 1, la suma de todos indica cuantas cubetas están activadas.

Inicialmente, un individuo es una lista de 10 elementos inicializados a 0. Se recorre la lista elemento a elemento y, aleatoriamente, se decide si el elemento pasa a ser 1 o cambia a 0.

Esto se realiza hasta que el individuo tiene tantos unos como el número máximo de cubetas, o hasta que se recorren los 10 elementos. En este segundo caso, se comprueba si tiene tantos o más unos que el número mínimo de cubetas. En caso afirmativo, se deja al individuo; en caso negativo, se vuelve a recorrer la lista asignando aleatoriamente unos y ceros.

Por tanto, un posible individuo de la población y que representa que las cubetas 2, 4, 6 y 8 están activadas sería:

$$individuo = [0, 1, 0, 1, 0, 1, 0, 1, 0, 0] \tag{3.1}$$

Una vez el individuo cumple las restricciones, se añade a una lista que representa la población del algoritmo genético y se repite el proceso de generación y adición de individuos a la población.

Este proceso se repite hasta que la longitud de la población alcanza el valor del parámetro *pobIni* que se le había pasado al algoritmo como entrada.

3.3 Función de evaluación

La función de aptitud la hemos definido como la distancia entre el peso de naranjas que deja pasar el individuo con sus cubetas activadas y el tamaño de la confección que se busca. Por tanto, un individuo es más 'apto' cuanto menor valor tenga su función de aptitud (se está tratando un problema de minimización).

Como se busca aproximar al peso objetivo de la confección por exceso, se ha tenido en cuenta que tengan prioridad aquellos pesos que sobrepasen el peso de la malla, antes que los que no lo alcanzan. Por ejemplo, si el peso de la confección es 100 y un individuo deja pasar 130, dicho individuo tiene una mejor aptitud que otro que deja pasar 98 de peso, aunque este esté mas próximo al peso exacto de la malla.

De esta forma, la aptitud se calcula de dos formas:

- Si el peso que deja pasar el individuo (*weight*) es mayor o igual que el peso objetivo de la malla (*pesoMalla*), la aptitud es la resta de *weight* menos *pesoMalla*.
- Si el peso que deja pasar el individuo (*weight*) es menor que el peso objetivo de la malla (*pesoMalla*), la aptitud se calcula como:

$$apt = pesoMalla * probCubeta * 100 - weight \tag{3.2}$$

  Donde *probCubeta* es el porcentaje máximo de llenado de cubeta respecto al peso de la confección.

Así, los individuos que dejan pasar más peso que el de la confección son mejores (priorizando aquellos que son más pequeños dentro de este subconjunto) que los que no lo alcanzan (priorizando aquellos que son más grandes dentro de este subconjunto).

De esta manera, tenemos una función que se pretende minimizar, de modo que la aptitud del individuo será óptima cuando su función de aptitud sea 0.

3.4 Selección, cruce, mutación y reemplazo

Antes de la selección de individuos que serán sometidos a cruce, primero ordenamos toda la población actual en función de la aptitud de cada uno, de mejor a peor. Posteriormente, seleccionamos, con un porcentaje de *probCruce*, los mejores individuos. Por ejemplo, si hay 100 individuos y *probCruce* es 0.8, se seleccionan los 80 mejores.

Una vez están seleccionados, el cruce se hará eligiendo al mejor individuo como padre y al resto como madre, de manera que se generarán n-1 hijos, siendo n el tamaño de los seleccionados. El cruce que se ha realizado consiste en obtener una solución con los componentes impares del padre y los pares de la madre. Como es probable que el hijo tenga más componentes a 1 que el máximo de cubetas o menos que el mínimo, se recorre la lista añadiendo o eliminando componentes. El índice del componente que será eliminado o insertado se hace de forma aleatoria para no priorizar el orden ascendente o descendente de la lista.

Dentro de estos nuevos individuos, la mutación que se ha realizado es la de Intercambio Recíproco. Esta consiste en seleccionar dos genes al azar y permutarlos. En nuestro caso, primero se ha de tener en cuenta que el individuo tenga todos los componentes a 1, en cuyo caso se elegirá uno aleatoriamente y se cambiará a 0. En cualquier otro caso, se elegirán al azar dos elementos y se permutarán, habiendo comprobado previamente que ambos elementos son distintos, de lo contrario la mutación sería trivial. El porcentaje de

individuos que se selecciona para mutación viene determinado por *probMutacion*, como se ha explicado anteriormente.

Por último, se aplica reemplazo por Estado Estacionario, es decir, se descartan los peores individuos de la población para añadir a la nueva generación (tanto los que han mutado como los que no). Por tanto, el número de descartados es el número de nuevos individuos.

3.5 Convergencia

La convergencia del algoritmo genético se ha implementado como un método en Python cuyo objetivo es el de comprobar si un individuo es o no óptimo, de manera que el genético finalizará en caso afirmativo. Su esquema algorítmico es:

```
def convergencia(pueblo):
    for individuo in pueblo:
        if aptitud(individuo)==0:
            optimo = individuo
            return True, optimo
    return False, None
```

Listing 3.1: Convergencia.

CAPÍTULO 4
Resultados y conclusiones

Debido a que, en el peor de los casos, la talla del problema es 2^{10}, hay 1024 posibles soluciones. Para que nuestro algoritmo converja, es necesario comprobar si se ha generado un individuo que satisface el problema, es decir, si su función de evaluación es 0. En caso afirmativo, el algoritmo finalizará devolviendo esa solución, independientemente de la iteración por la que se encuentre ejecutando.

Una vez que ejecutamos de forma continuada nuestro algoritmo, podemos observar que la solución se encuentra, en la mayoría de los casos, en las primeras iteraciones, de modo que el parámetro de *maxIter* lo ajustamos a un valor pequeño (10).

Para un valor grande de *pobIni*, la solución que se encuentra suele tener una aptitud menor. Esto puede deberse al hecho de que la talla del problema no es de un tamaño desorbitado y es muy posible que, de manera arbitraria, el individuo que se genere en una población inicial sea precisamente el óptimo.

Sin embargo, como estamos teniendo en cuenta el tiempo de cómputo, no debemos establecer un valor muy alto para *pobIni*, ya que en ese caso tardará más tiempo en generar individuos debido a las restricciones que han de cumplir y a que no se permiten repetidos.

Por ello, hemos establecido el valor máximo de la población inicial a 100.

También se ha podido comprobar que el resto de argumentos de entrada (*probCruce* y *probMutacion*) no son muy relevantes a la hora de mejorar la aptitud de la solución o el tiempo de cómputo, de manera que se les ha asignado un valor arbitrario de 0.9 y 0.5, respectivamente.

Una vez tenemos los parámetros del algoritmo genético establecidos, hemos ejecutado 10000 iteraciones de este para comprobar el mejor y el peor tiempo obtenido, tanto para soluciones óptimas como para las que no (anotando en este caso, la función de evaluación mejor y peor que se obtiene, además del tiempo).

Los resultados obtenidos, con una confección de peso 1000 y 40 % de llenado máximo de cubeta respecto a dicho peso, han sido los siguientes:

Cubetas min	**Cubetas max**	**Mejor (ms)**	**Peor (ms)**
5	5	2.71	24.28
4	6	2.54	46.58
3	7	2.51	43.61

Tabla 4.1: Resultados con respecto a soluciones óptimas.

Cubetas min	Cubetas max	Mejor (ms)	Peor (ms)	Mejor (aptitud)	Peor (aptitud)
5	5	23.97	60.10	3	19
4	6	24.62	62.05	1	19
3	7	23.50	60.67	2	13

Tabla 4.2: Resultados con respecto a buenas soluciones no óptimas.

A la vista de estos resultados, podemos observar que las soluciones óptimas se encuentran en mejor tiempo que las que no lo son. Esto a se debe a que es más sencillo encontrar una solución óptima en la población inicial antes que obtenerla por los procesos del genético (cruce, mutación, etc.).

Además, dentro de las soluciones no óptimas, se pueden apreciar unos valores de la función de evaluación bastante asequibles, teniendo en cuenta que se está minimizando, que en ningún momento superan la barrera de 20 y que las mejores soluciones dentro de este subgrupo cuentan con una aptitud muy cercana a 0. Recordemos que si, por ejemplo, la aptitud de la solución es 1 y el peso de malla es 1000, eso quiere decir que el individuo deja pasar un peso de 1001.

CAPÍTULO 5
Trabajos relacionados

Como hemos podido observar, la Inteligencia Artificial tiene sus aplicaciones ya no solo con los productos hortofrutícolas, sino dentro del campo de la agricultura en general.

La IA aplicada en la agricultura es una herramienta tecnológica que puede sonar como algo muy nuevo en el sector agrícola, pero hoy en día es una práctica que se está implementando en todo el mundo y está beneficiando a muchas personas dedicadas a este tipo de áreas.

En primer lugar, está el problema de planificación óptima en terrenos destinados a la agricultura en la India[1]. Este problema es afectado por restricciones tales como: coste derivado por el tipo de técnica de cultivo, presupuesto máximo por cada tipo de técnica de cultivo, coste por el tipo de producto cultivado, presupuesto para el tipo de producto cultivado, terreno total disponible, precio competitivo en el mercado para cada tipo de producto a cultivar, precio mínimo sostenible para cada unidad de cultivo, beneficio e inversión totales, rendimiento del cultivo, etc... Finalmente, se obtienen los tamaños óptimos de cada área de cultivo que encajarán en el terreno. Este problema, por lo tanto, se compone de dos problemas lineales: minimizar el coste de cultivar el tipo de cultivo x en el área y; y obtener el máximo beneficio de cultivar el cultivo x en el tipo de terreno y (pudiendo ser este húmedo o seco).

En segundo lugar, se encuentra el problema de optimización en la recogida de frutas en función de su calidad y su valor económico derivado en función de la misma. Esta técnica se aplicó al caso de la recolección de tomate en condiciones Vietnamitas[2]. En este caso, se controlaron el crecimiento y desarrollo (calidad) de los tomates en función del tamaño y el color de los mismos durante las estaciones de crecimiento de invierno y verano. Permitiendo así, obtener las fechas óptimas y la frecuencia de cosecha de los vegetales, priorizando el valor del producto siempre que sea posible. Esto permite a los agricultores apurar más las fechas de recolección y obtener una gran cantidad de producto que está en la misma etapa de maduración para su venta. El beneficio obtenido en esta tarea es independiente del beneficio total de la granja, pues depende de la técnica de recogida total elegida.

[1] https://www.researchgate.net/publication/322152469_Optimal_Programming_Problems_for_Crop_Planning_and_Agricultural_Resource_Management

[2] https://www.ncbi.nlm.nih.gov/pubmed/28473843