draft2

April 9, 2024

5.1 Codificación de soluciones

Existen diferentes maneras de codificar las soluciones para el problema de llenado de contenedores. Algunos de los más interesantes son por ejemplo Bortfeldt y otros [1] que propusieron una codificación basada dobles o simplemente dos listas de igual longitud, la primera representa la secuencia de llenado de los paquetes y la segunda indica la rotación de cada paquete. De manera similar Xianbo y otros [2] propusieron el uso de la primera lista como la secuencia de llenado de cada tipo de paquete y la segunda como la rotación de cada tipo de paquete.

Sin embargo, para las restricciones en nuestro problema, esta codificaciones no son adecuadas ya que necesitamos considerar que las cantidades de un tipo de paquete no son fijas sino que pueden variar de un rango mínimo a un máximo. Por lo tanto, proponemos una codificación basada en 3 listas de igual longitud, la primera lista representa la secuencia de llenado de cada tipo de paquete, la segunda lista indica la rotación de cada tipo de paquete y la tercera lista indica la cantidad de paquetes de cada tipo que se llenan en el contenedor.

Por ejemplo, si tenemos 3 tipos de paquetes $T = \{1, 2, 3\}$, todos los tipos con un mínimo de 0 paquetes y el tipo 1 con un máximo de 10 paquetes $q_1 \in [0, 10]$, el tipo 2 con un máximo de 30 paquetes $q_1 \in [0, 30]$ y el tipo 3 con un máximo de 20 paquetes $q_1 \in [0, 20]$, una solución factible para el problema de llenado de contenedores podría ser la siguiente:

```
[ 2,  1,  3]
[ 0,  1,  0]
[27,  9, 18]
```

La primera lista indica que el tipo 2 se llena primero, luego el tipo 1 y finalmente el tipo 3. La segunda lista indica que el tipo 2 no se rota, el tipo 1 se rota y el tipo 3 no se rota. La tercera lista indica que se llenan 27 paquetes del tipo 2, 9 paquetes del tipo 1 y 18 paquetes del tipo 3. En la figura Fig.3 se muestran dos ejemplos cómo se usa la codificación propuesta para representar soluciones para el problema.

Estas tres listas son necesarias para el algoritmo de llenado presentado en la sección 2, el cuál se encargará de llenar el contenedor de forma determinista y de acuerdo a las restricciones del problema, es decir dada una lista siempre se llenará el contenedor de una única manera. Una vez que el contenedor se ha llenado y si aún quedan paquetes por llenar se irán descartando los paquetes que ya no caben lo que implica actualizar o corregir la lista de cantidades de paquetes de cada tipo, de este modo se garantiza que la solución sea factible.

Otra ventaja de esta codificación es la flexibilidad que ofrece para adaptar el problema a diferentes restricciones, por ejemplo si se desea considerar una cantidad fija de paquetes de un tipo, simplemente se puede fijar el mínimo y el máximo de ese tipo de paquete. De igual manera, la estructura

soporta si se desea considerar otros tipos de rotaciones, además de las consideradas en el presente trabajo.

Este tipo de codificación permite representar soluciones factibles para el problema de llenado de contenedores, ya que considera la cantidad de paquetes de cada tipo que se llenan en el contenedor. Además, esta codificación es fácil de implementar y de entender, lo que facilita la implementación del algoritmo de llenado y del algoritmo de optimización para resolver el problema.

5.2 Población inicial

Para generar la población inicial se propone un mecanismo simple de generación de soluciones aleatorias. Dado un conjunto de tipos de paquetes $T = \{1, 2, ..., n\}$, donde cada tipo de paquete $t_i \in T$ tiene un mínimo y un máximo de paquetes a insertar en el contenedor $t_i \in [t_i^{min}, t_i^{max}]$, se generan soluciones aleatorias de la siguiente manera:

1. Se selecciona aleatoriamente un tipo de paquete $t_i \in T$.
2. Se selecciona aleatoriamente un tipo de rotación $r_i \in R, R \in \{0, 1\}$.
3. Se selecciona aleatoriamente una cantidad de paquetes $q_i \in [t_i^{min}, t_i^{max}]$.
4. Se repiten los pasos 1 al 3 hasta que se hayan seleccionado todos los tipos de paquetes.
5. Se repiten los pasos 1 al 4 hasta que se hayan generado N soluciones.

El algoritmo de generación de la población inicial se muestra en el Algoritmo 1. En este algoritmo, T es el conjunto de tipos de paquetes, R es el conjunto de rotaciones, N es el tamaño de la población, t_i^{min} y t_i^{max} son el mínimo y el máximo de paquetes de cada tipo, respectivamente, y P es la población inicial.

Algorithm 1 Generación de la población inicial

Require: $T, R, N, t_i^{\min}, t_i^{\max}$
Ensure: P
1: $P \leftarrow \emptyset$
2: **for** $i = 1$ to N **do**
3: $S \leftarrow \emptyset$
4: **for** each $t_i \in T$ **do**
5: $r_i \leftarrow$ rotación aleatoria de t_i
6: $q_i \leftarrow$ cantidad aleatoria de paquetes de t_i
7: $S \leftarrow S \cup \{(t_i, r_i, q_i)\}$
8: **end for**
9: $P \leftarrow P \cup \{S\}$
10: **end for**
11: **return** P

De esta manera, se generan soluciones aleatorias que cumplen con las restricciones del problema, es decir, se garantiza que la cantidad de paquetes de cada tipo que se insertan en el contenedor esté dentro del rango definido por el mínimo y el máximo de paquetes de cada tipo. Además, se garantiza que la cantidad de paquetes de cada tipo que se insertan en el contenedor sea un número entero.

5.3 Selección

Para seleccionar las soluciones que se utilizarán en cada generación del algoritmo evolutivo, se propone un mecanismo de selección basado en el método de torneo usando un tamaño de torneo

de 2, también conocido como torneo binario. Anand y otros [3] lo describen como uno de los más eficientes métodos de selección para algoritmos evolutivos. Para usar el método de torneo binario, se seleccionan aleatoriamente dos soluciones de la población y se selecciona la mejor solución de las dos. Este proceso se lleva a cabo junto con el operador de cruce para lo cual cada padre se selecciona mediante un torneo binario.

En este algoritmo Alg.2., P es la población, N es el tamaño de la población, $P1$ y $P2$ son los padres seleccionados.

Algoritmo 2: Selección de padres

```
Entrada: $P$, $N$
Salida: $P1$, $P2$
1: $P1 \leftarrow$ solución aleatoria de $P$
2: $P2 \leftarrow$ solución aleatoria de $P$
3: for $i = 1$ to $N$ do
4:     $S1 \leftarrow$ solución aleatoria de $P$
5:     $S2 \leftarrow$ solución aleatoria de $P$ - $S1$
6:     if $S1$ > $P1$ then
7:         $P1 \leftarrow S1$
8:     end if
9:     $S3 \leftarrow$ solución aleatoria de $P$
10:     $S4 \leftarrow$ solución aleatoria de $P$ - $S3$
11:     if $S3$ > $P2$ then
12:         $P2 \leftarrow S3$
13:     end if
14: end for
15: return $P1$, $P2$
```

De esta manera, se generan dos listas de padres que se utilizarán en el operador de cruce para generar una nueva población. Este método de selección procura que siempre se tienda a seleccionar soluciones mejores aunque tampoco se descartan soluciones peores, lo que permite mantener la diversidad en la población.

5.4 Cruce

Como resultado de la selección de padres, se generan dos listas de padres que se utilizarán en el operador de cruce para generar una nueva población. Para el operador de cruce se propone un mecanismo de cruce basado en el cruce de un punto. Según Umbarkar y otros [4], para este tipo de operador, se selecciona un punto aleatorio en la solución y se intercambian las partes de las dos soluciones que están a la izquierda y a la derecha del punto. Este operador de cruce es simple y fácil de implementar, y se ha demostrado que es eficiente para resolver problemas de optimización combinatoria.

Cabe mencionar que el operador de cruce se aplica a una proporción de la población, dependiendo de la tasa de cruce P_CROSS, que se define como la probabilidad de que dos soluciones seleccionadas se crucen. De esta manera, se garantiza que la población se mantenga diversa y que se generen nuevas soluciones que puedan ser mejores que las soluciones actuales.

En este algoritmo Alg.3., $P1$ y $P2$ son los padres seleccionados, $C1$ y $C2$ son los hijos generados, N es el tamaño de la población y T es el conjunto de tipos de paquetes.

Algoritmo 3: Cruce de padres

```
Entrada: $P1$, $P2$, $N$, $T$
Salida: $C1$, $C2$
1: $C1 \leftarrow \emptyset$
2: $C2 \leftarrow \emptyset$
3: $p \leftarrow$ punto aleatorio en la solución
4: for $i = 1$ to $N$ do
5:     if $i \leq p$ then
6:         $C1 \leftarrow C1 \cup P1[i]$
7:         $C2 \leftarrow C2 \cup P2[i]$
8:     else
9:         $C1 \leftarrow C1 \cup P2[i]$
10:         $C2 \leftarrow C2 \cup P1[i]$
11:     end if
12: end for
13: return $C1$, $C2$
```

Aunque este algoritmo de cruce es genérico y aplicado a muchos tipos de problemas, se puede adaptar para el problema de llenado de contenedores y para considerar la codificación propuesta en la sección 5.1. ya que realizar el cruzamiento simple podría generar soluciones que no sean factibles, por ejemplo repitiendo un tipo de paquete que ya se ha llenado en el contenedor. Por lo tanto, se propone un mecanismo de cruce que garantice que las soluciones generadas sean factibles. Para ello, se propone usar la primera partición de la solución de un padre y completar la solución con tipos de paquetes que no se han seleccionado en la primera partición. De esta manera, se garantiza que la solución generada sea factible y que contenga todos los tipos de paquetes.

Entonces, el algoritmo de cruce se modifica de la siguiente manera:

```
Entrada: $P1$, $P2$, $N$, $T$
Salida: $C1$, $C2$
1: $C1 \leftarrow \emptyset$
2: $C2 \leftarrow \emptyset$
3: $p \leftarrow$ punto aleatorio en la solución
4: for $i = 1$ to $N$ do
5:     if $i \leq p$ then
6:         $C1 \leftarrow C1 \cup P1[i]$
7:         $C2 \leftarrow C2 \cup P2[i]$
8:     else
9:         for each $t_i \in T$ do
10:            if $C1$ no contiene $P2[i]$ then
11:                $C1 \leftarrow C1 \cup P2[i]$
12:            end if
13:            if $C2$ no contiene $P1[i]$ then
14:                $C2 \leftarrow C2 \cup P1[i]$
15:            end if
16:        end for
17:    end if
18: end for
19: return $C1$, $C2$
```