UNIVERSIDAD DE BUENOS AIRES

Facultad de Ciencias Exactas y Naturales

Departamento de Matemática

Tesis de Licenciatura

APLICACIÓN DE TÉCNICAS HEURÍSTICAS A UNA VARIANTE DEL PROBLEMA DE CORTE DE STOCK EN INDUSTRIAS SIDERÚRGICAS

Ricardo Alexis Muñoz Estéfano

Directora: Susana Puddu

Abril del 2010

Agradecimientos

A mi mamá y mi hermana por apoyarme y acompañarme. Su amor y su apoyo son un pilar fundamental en mi vida.

A Susana Puddu por proponerme este problema en el cual ha sido tan grato trabajar. A los jurados por sus valiosas correcciones y observaciones. Al Dr. Vicentini por su colaboración. A Nicolás Vital y Alfredo Cuaradeghini, de Ternium Siderar, por su colaboración en mi comprensión del problema.

A los responsables de la Beca Sadosky, y en especial al Lic. Diego Picco, ya que la misma ha sido fundamental para que pudiera terminar la carrera.

A Florencia Muñoz Estéfano por su valiosa e indispensable ayuda en la elaboración de los gráficos.

A todos los buenos profesores que he tenido durante la carrera.

A todos los que hicieron y hacen posible la educación pública y gratuita en la Argentina.

"Sigan pidiendo y se les dará; sigan buscando, y hallarán; sigan tocando, y se les abrirá. Porque todo el que pide recibe, y todo el que busca halla, y a todo el que toca se le abrirá."

Mateo 7:7, 8.

Índice

Parte I

1. Introducción

Este trabajo aborda una variante de un problema presente en varias industrias (siderurgia, papel, etc.), en las cuales, a partir de objetos grandes, como bobinas o rollos, de cierta materia prima, se deben obtener objetos menores o "ítems" mediante un proceso de corte, pudiendo ser dichos ítems, por ejemplo, bobinas de menor ancho o elementos rectangulares de longitud muy inferior a la de una bobina desplegada. Esto debe realizarse de la forma más eficiente posible, en el sentido de minimizar los costos asociados.

Dicho problema puede traducirse en un problema matemático propio del área de Optimización Combinatoria, conocido como Problema de Corte de Stock ó Cutting Stock Problem (CSP), del cual existen muchas variantes [1].

En este trabajo, nos interesa el caso particular de las industrias siderúrgicas, en las cuales la materia prima es el acero, y utilizamos como referencia la situación particular de una empresa siderúrgica argentina: Ternium Siderar, cuya planta se encuentra ubicada en Florencio Varela, y en la cual se desea optimizar la forma en que son obtenidos ítems rectangulares (chapas), pedidos por clientes, a partir de bobinas de acero (láminas de acero arrolladas, de largo muy superior al ancho), mediante una selección apropiada del stock de bobinas.

Cada una de las formas posibles de cortar los ítems correspondientes a un pedido a partir de una bobina se denomina "patrón de corte". El objetivo principal en la planificación del proceso es el empleo, en la obtención de los ítems pedidos, de bobinas y patrones de corte de las mismas mediante los cuales se puedan cubrir todos los pedidos, minimizando el desperdicio de materia prima, por obvias razones, y la cantidad de tipos diferentes de bobinas a utilizar, lo cual está relacionado con costos de elaboración de las bobinas y de almacenamiento de las mismas, entre otros.

En este tipo de industrias, con grandes niveles de producción y materia prima costosa, pequeñas mejorías en el diseño del proceso de

producción, que impliquen reducciones en el desperdicio de los recursos empleados, pueden generar grandes ahorros económicos. Por este motivo, no solo interesa encontrar la mejor solución posible, es decir, en este caso, la mejor selección posible del stock de bobinas a emplear y de los patrones de corte para las mismas (lo cual, como luego veremos, es, en general y dadas las características del problema, hasta el momento impracticable) sino que suele ser también muy positivo hallar una solución "cercana" a la mejor solución posible, o una que, simplemente, mejore de forma considerable los resultados obtenidos hasta el momento a partir de las soluciones aplicadas al problema por la empresa.

En la primera parte, se describen las características del problema que conducen a su formulación matemática como un problema de Optimización Combinatoria. Se sigue con una breve presentación acerca de qué es un problema de Optimización Combinatoria y cómo los mismos suelen ser resueltos algorítmicamente. Luego, se realiza un resumen de la teoría de NP-completitud, a partir de la cual se intenta formar una noción del grado de dificultad del problema tratado, el cual motiva la utilización de algoritmos de aproximación para el problema, desarrollados mediante técnicas heurísticas.

En la segunda parte, se realiza una descripción de las técnicas heurísticas empleadas (búsqueda local y algoritmos genéticos), seguida de la presentación de los algoritmos desarrollados para el problema en este trabajo. Finalmente, se presentan algunos resultados de la aplicación de los algoritmos desarrollados a instancias del problema, a partir de los cuales se obtienen conclusiones acerca de la eficacia de los algoritmos implementados.

2. Presentación del problema

2.1. Descripción del problema

2.1.1. Obtención de un pedido a partir de una bobina

Se denomina **patrón de corte** a cada secuencia de cortes mediante los cuales, a partir de una bobina de materia prima (a las cuales se denomina "bobinas madres"), se obtienen ítems. Dichos ítems corresponden a pedidos realizados por clientes, siendo, en el caso que nos interesa, elementos rectangulares especificados por sus medidas: ancho, largo y espesor, así como por la cantidad de toneladas demandadas y un conjunto de características técnicas que podríamos clasificar en dos grupos. Por un lado, aquellas características técnicas que llamaremos "primarias": tipo, clase y calidad del acero, y que son propias tanto de los pedidos como de las bobinas madres; y por otro lado, características técnicas que llamaremos "secundarias" y que no serán de interés en nuestro trabajo, como: forma, color, terminación, etc., las cuales solo son propias de los pedidos y son generadas luego de la obtención de estos a partir de las bobinas madres.

Las bobinas madres, por su parte, se especifican a partir de su ancho y espesor (el largo es constante) y sus características técnicas primarias. Cada combinación posible de ancho, espesor y características técnicas primarias determinará lo que denominaremos un **tipo de bobina madre** (**t.b.m.**).

Tanto en el caso de los pedidos como en el de las bobinas madres, las medidas vienen dadas en mms. y el largo y el ancho son enteros. El ancho de cada bobina madre pertenece al intervalo [750, 1650], debido a limitaciones técnicas presentes en su elaboración.

Para que un pedido pueda obtenerse a partir de una bobina madre determinada, ambos deben tener el mismo espesor, iguales características técnicas primarias y medidas que permitan la disposición secuencial del ítem pedido sobre la bobina madre, con sus lados orientados de forma paralela a los de la misma. Para que esto último sea posible, debe ocurrir, asumiendo que el largo de las bobinas desplegadas es siempre muy superior a las medidas de los ítems

rectangulares pedidos, que alguna de las dos medidas correspondientes a ancho y largo del ítem no supere el ancho de la bobina. Es decir, dados un ítem pedido y una bobina madre, con el mismo espesor e iguales características técnicas primarias, y bajo la convención de llamar ancho a la menor y largo a la mayor de las medidas del ítem (sin incluir al espesor), tenemos tres casos posibles:

Sean a y l el ancho y el largo del pedido respectivamente, y A el ancho de la bobina madre.

1) Si a>A, no es posible obtener el pedido a partir de la bobina.

2) Si a≤A y l>A, podemos disponer secuencialmente el ítem pedido sobre la bobina de la forma ejemplificada en la siguiente figura, en la cual la región sombreada corresponde a la porción no aprovechada de la bobina madre en el proceso de corte.

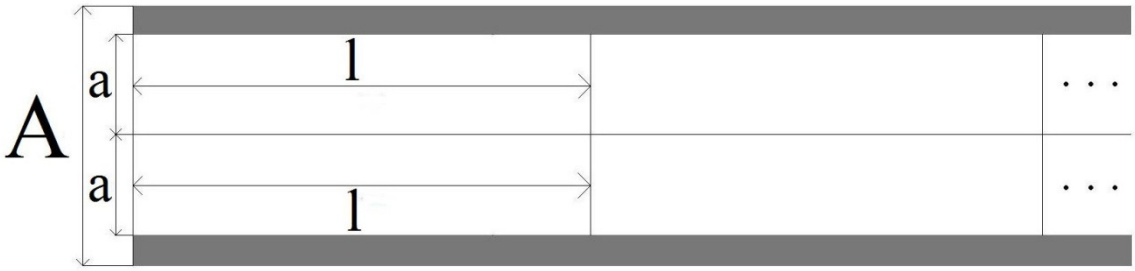


Fig. 1

En el ejemplo graficado en la Figura 1, la relación entre los valores de a y A permite la disposición en paralelo de dos ítems del pedido. En general, se dispondrán en paralelo sobre la bobina madre tantos ítems del pedido a obtener como sea posible.

3) Si a≤A y l≤A, podemos disponer secuencialmente el pedido sobre la bobina de dos formas posible, cada una de las cuales determina un posible patrón de corte.

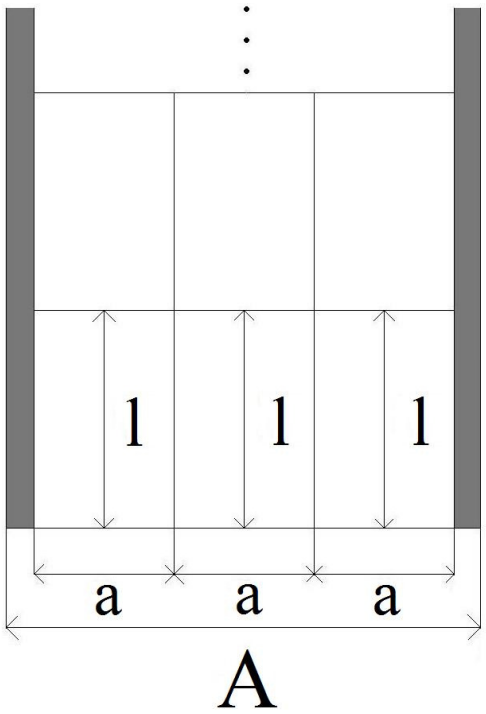


Fig. 2.1

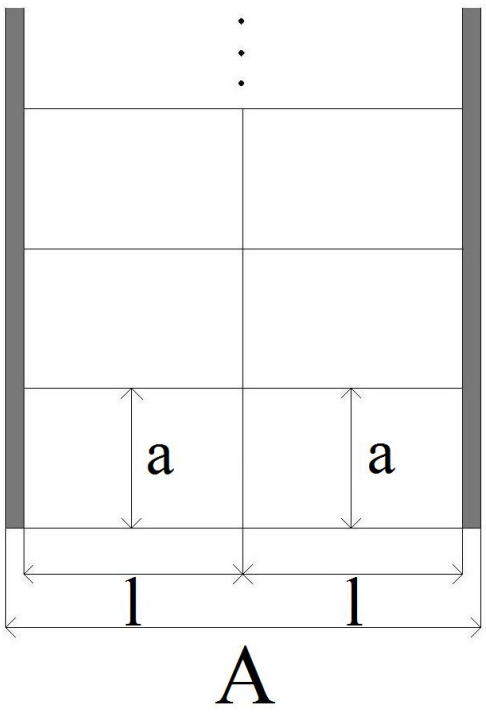


Fig. 2.2

La posibilidad de optar, en el caso 3, entre las dos disposiciones alternativas del pedido sobre la bobina se debe a que las láminas de acero que enrolladas conforman las bobinas son planas y homogéneas, implicando que ambas orientaciones sean factibles. Ante este caso, se optará siempre por la orientación que minimice el desperdicio, cuyo método de determinación se explicará luego.

Debe observarse que, dado que el ancho de las bobinas madres está limitado a ser un entero en el intervalo [750, 1650], para que sea factible que los ítems de un pedido determinado puedan ser obtenidos a partir de alguna bobina madre, su ancho no puede ser superior a 1650 mms.

Cuando es posible, según las condiciones recién detalladas, la forma de cortar los ítems de un pedido a partir de una bobina madre es la siguiente: En una primera etapa, se realiza un corte longitudinal completo de la bobina en sub-bobinas de ancho igual a una de las medidas (L_1) del pedido, las cuales se denominan "flejes". En caso de que el ancho de la bobina madre no sea un múltiplo de L_1, se producirán, además, dos flejes de desperdicio de igual ancho, a ambos lados de la bobina.

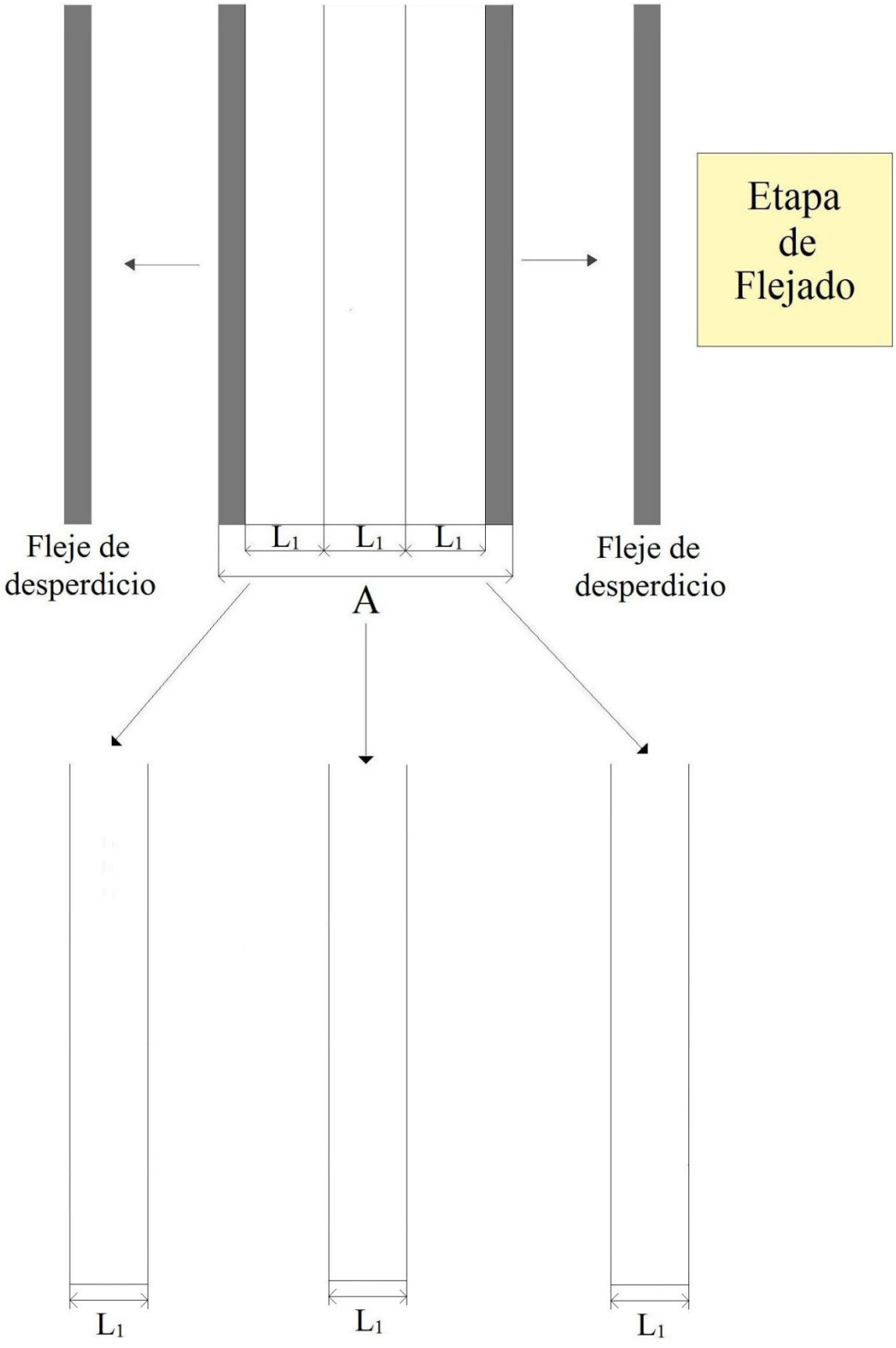

Etapa
de
Flejado
Fleje de
desperdicio
L1
L1
L1
Fleje de
desperdicio
A
L1
L1
L1

Fig. 3.1

En la segunda etapa, se realizan cortes transversales secuenciados, o "guillotinado", sobre los flejes obtenidos en la primera etapa, con una separación entre los cortes igual a la segunda medida del pedido (L_2), obteniéndose como resultado los ítems correspondientes.

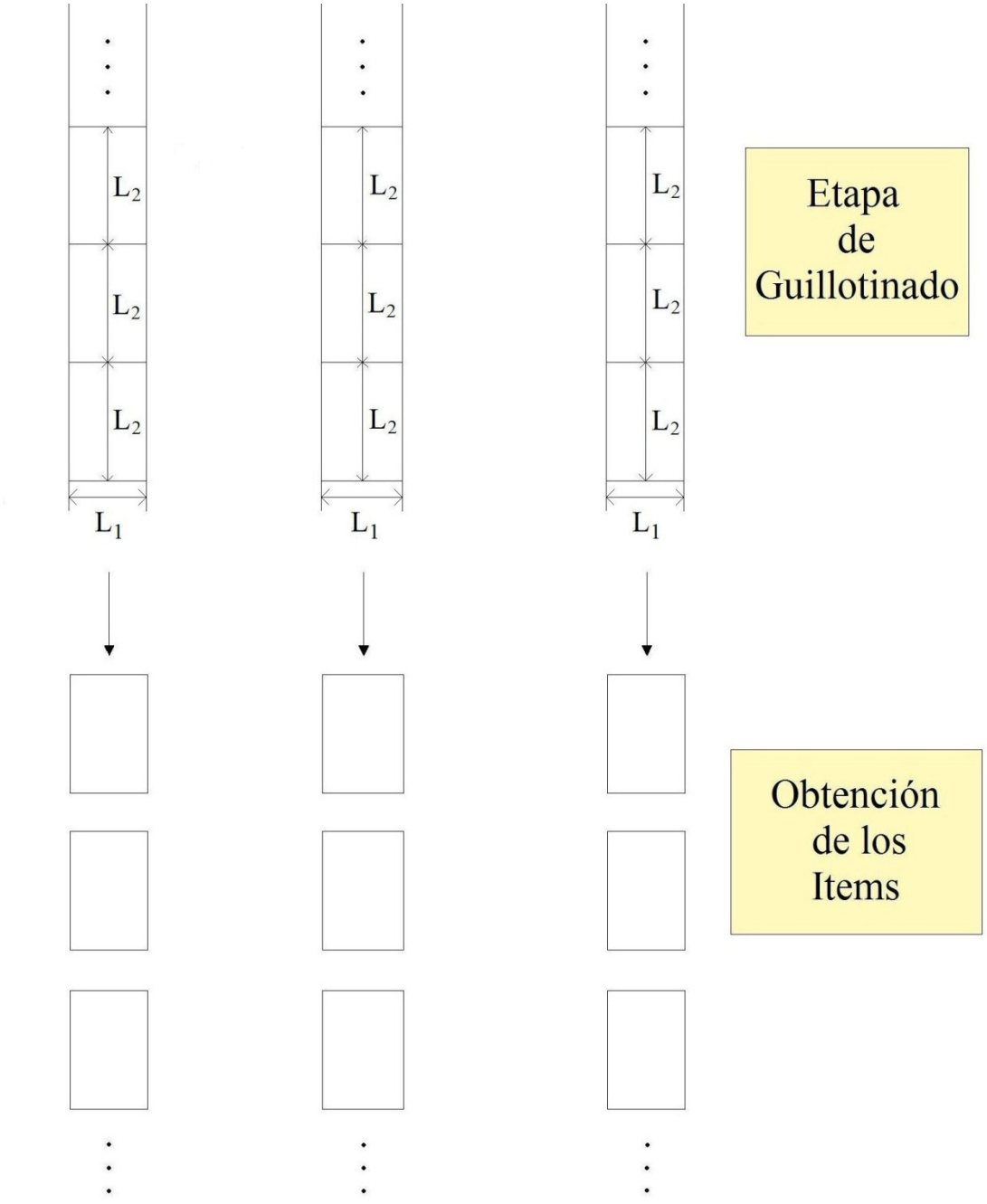


Fig. 3.2

Si ocurre que L_1 coincide con el ancho de la bobina madre (lo cual es muy deseable), el proceso se inicia directamente en la segunda etapa, guillotinando la bobina madre. En este caso, y más en general, en el caso de que el ancho de la bobina madre sea un múltiplo de L_1, no se produce desperdicio de acero durante el proceso de corte.

Usualmente, en el caso de que se realice un proceso de corte de dos etapas, ocurrirá que los flejes obtenidos en la primera etapa excederán lo necesario para cubrir el pedido en cuestión. Es decir, no será necesario guillotinarlos completamente, pudiendo sobrar tramos de fleje o incluso flejes completos. Esto se debe al hecho de que durante el flejado, por cuestiones técnicas, el corte de la bobina madre es completo, es decir, de un extremo a otro de la misma y obteniéndose todos los flejes posibles de ancho L_1, independientemente de qué proporción de la bobina sea suficiente para cubrir la cantidad demandada del pedido a obtener. Del mismo modo, en casos de corte en una sola etapa, suele haber tramos excedentes de la bobina madre empleada.

Estos sobrantes no se considerarán como un desperdicio, dado que los mismos pueden ser almacenados para su posterior utilización para cubrir, por ejemplo, un futuro pedido igual al que se está cubriendo actualmente, lo cual es muy factible, dado que es común que haya clientes que repitan sus pedidos.

De este modo, el desperdicio estará dado, únicamente, por el acero de los flejes de desperdicio que puedan producirse en la etapa de flejado, así como por sobrantes de la etapa de guillotinado que sean demasiado pequeños como para considerar su almacenamiento y posterior utilización. Estos últimos se producen cuando L_2 no es un divisor de la longitud de la bobina madre, de modo tal que el guillotinado completo de los flejes genera sobrantes de longitud igual al resto no nulo del cociente entre la longitud de la bobina madre y la medida L_2. El desperdicio generado a partir de tales sobrantes es insignificante con respecto al resultante a partir de flejes de desperdicio, lo cual nos permite eliminarlo de consideración sin afectar significativamente los resultados obtenidos y evitando una complejización innecesaria del planteo.

2.1.2. Objetivos a considerar en el planteo del problema

Periódicamente, la fábrica debe realizar una renovación limitada de su stock de bobinas madres, en función de poder cubrir, de forma eficiente, los próximos pedidos. Surge entonces la necesidad de decidir qué tipos de bobinas conviene incorporar al mismo, de forma tal que la mayor cantidad posible de los pedidos que surjan próximamente puedan ser cubiertos a partir de bobinas madres en stock, con un bajo desperdicio de acero.

Una vez seleccionado un conjunto de bobinas madres a incorporar al stock, éstas son solicitadas por la fábrica a otra división de la empresa, la cual se encarga de la fabricación de las mismas, pudiendo fabricar bobinas de cualquier ancho que se considere necesario, dentro del rango antes especificado. En caso de surgir, luego de la renovación de stock, un pedido que no pueda ser cubierto con una de las bobinas en stock, habrá que solicitar la elaboración de un tipo de bobina madre particular para tal pedido, implicando costos extras.

En la elección de las bobinas madres a incorporar al stock, debe tenerse en cuenta que una selección de bobinas de muchos tipos distintos genera un gran gasto en términos de elaboración y de almacenamiento, por lo cual se procura solicitar la menor variedad posible de t.b.m.'s. Es decir, no se busca minimizar la cantidad de bobinas madres a utilizar, sino la cantidad de tipos diferentes de las mismas, lo cual representa el principal factor de costo en la elaboración de las bobinas, relacionándose esto con la preparación de los moldes para su fabricación.

Criterios posibles a utilizar como guía para la selección de las bobinas madres a incorporar al stock pueden ser, por ejemplo, los siguientes:

-Dado el carácter periódico de muchos de los pedidos realizados por clientes particulares, elegir el conjunto de bobinas madres en base a los pedidos recibidos dentro de un período determinado, previo a la presente elección.

-Utilizar una lista de pedidos elaborada en base a un análisis de mercado, que permita predecir, con cierto grado de aproximación, los requerimientos que tendrán, en un período determinado, los clientes activos o potenciales de la empresa.

-Contar *a priori* con una lista de pedidos reales a cubrir durante un período determinado. Esta posibilidad, la cual posee claramente un carácter preferencial, no siempre es factible, debiendo recurrirse a los criterios anteriores.

De todos modos, sea cual sea el criterio a emplear, en todos ellos se considera una lista de pedidos y se intenta determinar un conjunto formado por la menor cantidad posible de t.b.m.'s distintos con el cual podría cubrirse dicha lista con un bajo desperdicio de acero. De este modo, cada uno de los criterios mencionados conducirá a instancias, es decir casos particulares, de un mismo problema.

Es importante destacar la relación dual existente entre la cantidad de t.b.m.'s a seleccionar y el desperdicio de acero resultante de la obtención de los pedidos a partir de los mismos, factores determinantes del costo. Dicha dualidad resulta del hecho de que, a medida que consideremos selecciones con mayor cantidad de t.b.m.'s, iremos obteniendo desperdicios cada vez menores, y viceversa (es evidente que para cualquier conjunto de n t.b.m.'s con los cuales pueden cubrirse todos los pedidos de una lista determinada, existe un conjunto formado por n+1 t.b.m.'s con el cual puede cubrirse la lista, generándose un desperdicio no superior). Esto conduce a la necesidad de establecer un "equilibrio" entre el interés por disminuir la cantidad de t.b.m.'s utilizados para cubrir los pedidos y el interés por reducir el desperdicio de acero resultante, debiéndose ponderar los costos asociados a cada uno de estos dos factores.

En base a lo planteado, nuestro problema será el siguiente: Dada una lista de pedidos, seleccionar un conjunto de t.b.m.'s con el cual puedan cubrirse todos los pedidos, minimizando el costo asociado a la obtención de los mismos. Dicho costo estará determinado por la cantidad de t.b.m.'s distintos seleccionados y por el desperdicio de acero asociado.

2.1.3. Clusters de pedidos

Para que dos pedidos distintos p_1 y p_2 puedan ser obtenidos a partir de un mismo t.b.m. b, ambos pedidos deben tener el mismo espesor y

las mismas características técnicas primarias (y a su vez deben coincidir con los de b). Todos los pedidos cuyos espesores coinciden en un valor determinado y comparten un mismo conjunto de características técnicas primarias forman lo que se denomina un **cluster** (anglicismo que significa: "racimo", "grupo", etc.) de pedidos.

Cuando, dados un cluster de pedidos C y un t.b.m. b en particular, haya coincidencia entre sus espesores y sus características técnicas primarias, y por lo tanto sea posible obtener pedidos de C a partir de b (siempre y cuando las medidas del pedido y de la bobina lo permitan), diremos que C y b son **compatibles**. A su vez, diremos que un pedido p en particular es compatible con un t.b.m. b, cuando p puede obtenerse a partir de b. Es decir, el cluster al que pertenece p es compatible con b y las medidas del ítem pedido permiten la disposición secuencial del mismo sobre la bobina madre.

La relación según la cual dos pedidos se encuentran relacionados si y solo si pertenecen a un mismo cluster es una relación de equivalencia sobre el conjunto de pedidos que componen una lista, generando por lo tanto una partición del mismo. De esta forma, podemos separar a un lista dada P de pedidos, en clusters C_1, C_2,…,C_n, convirtiendo nuestro problema de decidir, de forma óptima, qué conjunto B de t.b.m.'s utilizar para cubrir P, en el problema de seleccionar, de forma óptima, un conjunto B_i de t.b.m.'s para cada cluster C_i, de forma tal que se puedan cubrir los pedidos que componen el mismo con el menor desperdicio de acero posible, y buscando a su vez minimizar la cantidad total de t.b.m.'s utilizados para cubrir los diferentes clusters, es decir la suma de los cardinales de los conjuntos B_i (los B_i son conjuntos disjuntos, pues un mismo t.b.m. no puede ser compatible a dos clusters distintos). La solución final para la lista P estará formada por el conjunto B igual a la unión de los conjuntos B_i.

Ahora, la condición de cubrir todos los pedidos de P a partir del conjunto B de t.b.m.'s a seleccionar implica que ninguno de los conjuntos B_i puede ser vacío, ya que debe seleccionarse al menos un t.b.m. para cada cluster, a fin de poder cubrir los pedidos que lo integran. Esto establece una cota inferior para la cantidad total de t.b.m.'s distintos que deberán seleccionarse, la cual no podrá ser inferior a la cantidad de clusters presentes en P.

Por otro lado, tenemos lógicamente establecida una cota superior para la cantidad total de t.b.m.'s distintos a emplear, dada por la cantidad de pedidos que componen la lista, ya que en caso de poder emplearse dicha cantidad de t.b.m.'s, podría obtenerse de manera trivial una solución óptima con desperdicio 0, simplemente eligiendo para cada pedido un t.b.m. de ancho múltiplo a una de sus medidas. Por lo cual no es necesario considerar la posibilidad de utilizar una cantidad de t.b.m.'s superior a la cantidad total de pedidos.

Ahora, dado que la cantidad de pedidos que componen una lista suele ser muy superior a la cantidad de t.b.m.'s distintos que pueden seleccionarse en la práctica para cubrir la misma, resulta imposible seleccionar para cada pedido un t.b.m. a partir del cual obtener el mismo sin desperdicio, debiendo asignarse a la mayoría de los clusters una cantidad de t.b.m.'s inferior a la cantidad de pedidos que lo componen. Esto conduce a que deban cubrirse pedidos distintos con un mismo t.b.m., generándose, inevitablemente, un desperdicio de materia prima a partir de los patrones de corte utilizados, ya que es muy improbable que puedan seleccionarse, para cada cluster, t.b.m.'s cuyos anchos sean múltiplos de alguna de las medidas de cada uno de los pedidos que habrán de obtenerse a partir de él.

Es importante destacar que, por cuestiones técnicas, no es posible combinar la obtención de diferentes pedidos en un mismo patrón de corte, es decir, la obtención de dos pedidos distintos no puede ser realizada en paralelo, de modo tal que si dos pedidos pertenecientes a un mismo cluster han de obtenerse a partir de un mismo t.b.m., la obtención debe ser hecha en forma sucesiva.

2.1.4. Medición del desperdicio

Dados un pedido p y un t.b.m. b compatibles, el desperdicio de acero resultante de la obtención de p a partir de b es medido de la siguiente manera:

Sean a y l el ancho y el largo de p, respectivamente, y sea t la cantidad de toneladas demandadas de p; sea A el ancho del t.b.m. b compatible con p (o sea, hay coincidencia tanto en el espesor como en

las características técnicas primarias de p y b, y ocurre que A≥a). Tenemos dos casos posibles, según existan una o dos formas de disponer el pedido sobre la bobina:

1) Si l>A: Solo hay una disposición posible del pedido sobre la bobina, es decir, hay un único patrón de corte posible, y el desperdicio resultante D de acero, medido en toneladas, es igual a:

$$D = R_D(A,a).t = \left(\frac{A - \left[\frac{A}{a}\right].a}{\left[\frac{A}{a}\right].a} \right).t$$

Para entender esta fórmula observemos, en primer lugar, que [A/a] (parte entera del cociente entre A y a) indica cuál es la mayor cantidad de ítems del pedido que pueden ser dispuestos en paralelo sobre la bobina madre. Luego, el denominador de $R_D(A,a)$ es igual al ancho de la bobina madre ocupado por los ítems del pedido, mientras que el numerador es igual al ancho no ocupado, o sea, el ancho desperdiciado de la bobina madre durante el proceso de corte. De este modo, $R_D(A,a)$ indica la relación entre lo que se desperdicia y lo que se aprovecha de una bobina madre de ancho A al obtener a partir de ella un pedido de ancho a, y el producto de $R_D(A,a)$ por t, es igual a la cantidad de toneladas de acero desperdiciadas durante la obtención de las t toneladas del pedido.

2) Si l≤A: Hay dos disposiciones posibles del pedido sobre la bobina madre, y optaremos por aquélla que minimice el desperdicio asociado. Por lo tanto, el desperdicio estará dado por:

$$D = \text{mín}\left\{ R_D(A,a).t \; ; \; R_D(A,l).t \right\}$$

Donde $R_D(A,l)$ se determina de forma análoga a $R_D(A,a)$.

2.2. Introducción al planteo matemático del problema

Dada una lista de pedidos P (finita y no vacía), agrupados en q clusters $C_1,...,C_q$, sea el conjunto de t.b.m's U={b: Existe j ($1 \leq j \leq q$)/ b es compatible a C_j}. U es un conjunto finito dado que P lo es y los anchos (en mms.) de los t.b.m.'s solo pueden ser enteros dentro de un rango acotado.

Llamamos **solución factible** (o simplemente solución) del problema dado por la lista P, a cada uno de los subconjuntos de U tal que a partir de los t.b.m.'s que lo componen pueden ser cubiertos todos los pedidos de P. Sea S el conjunto formado por todas las soluciones factibles. S es finito (dado que U lo es) y no vacío. Para ver esto último, consideremos, para cada cluster C_j ($1 \leq j \leq q$), el ancho $A_j = máx_{p \epsilon Cj}\{a(p)\}$ (donde a(p) es el ancho del pedido p). Ahora, el t.b.m. b_j, compatible con C_j, tal que $A(b_j) = máx\{750, A_j\}$ ($A(b_j)$ es el ancho de b_j) pertenece a U y a partir de él pueden ser cubiertos todos los pedidos de C_j. Luego, $s=\{b_{1,...,}b_q\}$ es una solución factible.

Podríamos, ahora, intentar definir una función C sobre S, tal que a cada solución $s \epsilon S$ le asigne el costo asociado a su implementación, el cual dependería de la cantidad de t.b.m.'s incluidos en s y de la cantidad de toneladas de acero desperdiciadas en la obtención de los pedidos de P a partir de tales t.b.m.'s. La cantidad de toneladas de acero desperdiciadas se puede determinar sumando, sobre todos los pedidos $p \epsilon P$, el desperdicio correspondiente a su obtención a partir del t.b.m. compatible, seleccionado en s, que implique un menor desperdicio para p.

Luego, dada una lista de pedidos P, podríamos plantear nuestro problema como el problema de encontrar, dentro del conjunto S de soluciones factibles correspondiente a P, un elemento s cuyo costo asociado C(s) sea mínimo.

Ahora, definir tal función C, podría resultar inconveniente, dada la distinta "naturaleza" de los factores determinantes del costo que prácticamente nos limitaría a expresar el mismo en términos monetarios, como única forma de expresar, mediante una misma medida, los dos factores de costo (no tendría mucho sentido sumar las toneladas de acero desperdiciadas con la cantidad de t.b.m.'s incluidos en la solución como una forma de determinar el costo de la misma).

Tal función quedaría sometida a fluctuaciones económicas sobre el costo monetario de la utilización de cada cantidad posible de t.b.m.'s distintos y el costo monetario asociado al desperdicio de acero, que obligarían a una constante actualización de la función de costo, lo cual, si bien es posible de realizar, puede evitarse mediante la aplicación de la siguiente alternativa, la cual nos permite independizarnos de los costos monetarios asociados a cada factor.

Dado que el costo asociado a la cantidad de t.b.m.'s que componen una solución particular es constante para todas aquellas soluciones compuestas por una misma cantidad de t.b.m.'s, en vez de considerar un conjunto S formado por todas las soluciones factibles para una lista de pedidos P, es decir, todas las selecciones de t.b.m.'s a partir de los cuales pueden cubrirse todos los pedidos de P (pudiendo variar la cantidad de t.b.m.'s en el rango determinado por la cantidad de clusters distintos y la cantidad de pedidos que componen P), consideramos, por separado, los conjuntos S(i) formados por todas las soluciones que incluyan exactamente i t.b.m.'s (con i dentro del rango recién especificado). Para cada uno de estos conjuntos S(i), consideramos el problema de encontrar un elemento s_i en S(i) tal que minimice una función de costo C, la cual mida el desperdicio de acero asociado a una solución. Es decir, hallar, para cada i, una solución s_i tal que $C(s_i) \leq C(s')$, para toda s' en S(i).

Al ser ahora el desperdicio de acero el único factor determinante del valor de la función C que mide el costo asociado a cada solución, podemos medir el desperdicio de acero de la forma que nos resulte más conveniente, pudiendo hacerlo, por ejemplo, simplemente a partir de la cantidad de toneladas desperdiciadas, o mejor aún, a partir del desperdicio relativo de acero, determinado como el cociente entre la cantidad total de toneladas de acero desperdiciadas y la cantidad total de toneladas aprovechadas en la obtención de los pedidos de la lista (es decir, la suma de las demandas de los pedidos), una medida que nos permitiría tener una visión más directa de la efectividad de los métodos de resolución que implementemos, pudiendo comparar mejor los resultados obtenidos en diferentes instancias del problema, entre las cuales la cantidad total de toneladas de acero demandadas podría ser muy variable.

De este modo, dividimos a nuestro problema original en un conjunto finito de sub-problemas, cada uno de los cuales posee una función de costo determinada únicamente por el desperdicio de acero asociado a cada solución, el cual es medido a partir de la relación recién explicada. Una vez hallada una mejor solución para cada uno de estos sub-problemas (o solo para aquellos que resulten relevantes, ya que, por ejemplo, no es práctico considerar soluciones con una cantidad de t.b.m.'s cercana a la cantidad total de pedidos), la solución al problema original consistirá en seleccionar, entre todas estas soluciones, aquélla que resulte más adecuada, a partir de una ponderación del desperdicio asociado y la cantidad de t.b.m.'s seleccionados.

Abordar el problema de esta forma nos permite separar el mismo en dos etapas, de forma tal que la primera etapa, en la cual debe determinarse una selección óptima para cada cantidad posible de t.b.m.'s, contiene la componente esencialmente matemática del problema.

Así, nuestro interés pasa a ser, fundamentalmente, resolver la clase de sub-problemas correspondientes a la primera etapa del planteo, concentrándonos entonces en la resolución del siguiente problema general: Dada una lista de pedidos P a cubrir, un conjunto S(i) formado por todos las selecciones factibles de i t.b.m.'s a partir de los cuales cubrir P, y una función C sobre S(i) que mide el costo asociado a cada solución a partir del desperdicio de acero resultante de su aplicación, hallar un elemento de S(i) que minimice el valor de C. Es decir, hallar s en S(i) tal que $C(s) \leq C(s')$, para toda s' en S(i).

Dado que, dada una lista finita de pedidos P, el conjunto S de soluciones factibles es finito, resulta que cada uno de los subconjuntos S(i) es finito. Luego, la estructura que hemos dado a este problema, a partir del objetivo de minimizar la función C antes definida sobre un conjunto finito de elementos, corresponde a la de un problema de Optimización Combinatoria.

3. Optimización Combinatoria

3.1. Estructura de un problema de Optimización Combinatoria

Un problema de Optimización es, en general, un problema en el cual, dado un conjunto S, cuyos elementos se denominan usualmente "soluciones factibles", o simplemente "soluciones", y una "función de costo" o "función objetivo" C, definida sobre S, se debe, o bien determinar que S es vacío, o en caso contrario, hallar un elemento s en S tal que minimice (o maximice) el valor de C, es decir, hallar $s \epsilon S$ tal que $C(s) \leq C(s')$, para toda s' en S. Cuando el conjunto de soluciones factibles es un conjunto discreto (siendo también en la mayoría de los casos finito), decimos que el problema en cuestión es de Optimización Combinatoria.

Existen muchos problemas de la vida real que pueden formularse matemáticamente mediante este esquema general. Pueden hallarse ejemplos de esto en múltiples actividades, en las cuales debe tomarse una decisión entre muchas posibles, cada una de las cuales posee un costo asociado, lo cual es muy común en la industria y en cualquier ámbito donde sea necesaria la organización racional de recursos.

En general, la dificultad de resolver tales problemas no radica en la determinación de cuáles son las soluciones factibles ni en el cálculo de sus costos, sino en que la gran cantidad de parámetros que suelen involucrar estos problemas deriva en una enorme cantidad de soluciones factibles, de forma tal que suele ser difícil, y hasta imposible en muchos casos, hallar de forma "eficiente" una solución cuyo costo sea óptimo, donde por costo de una solución nos referimos al valor de la función de costo sobre la misma, y por resolución eficiente del problema, a una resolución cuya aplicación demande una cantidad de tiempo y demás recursos que sea "razonable", de forma tal que pueda ser implementada en la práctica.

A continuación, se presentan como ejemplos dos problemas clásicos de Optimización Combinatoria.

3.2. Ejemplos

TSP

El primer ejemplo es el del problema conocido como Traveling Salesman Problem (TSP), o Problema del Viajante de Comercio, el cual es quizás el problema de Optimización Combinatoria que más atención ha concentrado entre los investigadores del área, siendo utilizado para evaluar cada idea que surge en la búsqueda de nuevas técnicas para resolver problemas de Optimización Combinatoria. Puede describirse de la siguiente forma:

Se tienen un conjunto de N ciudades $C=\{c_1, c_2,..., c_n\}$, para el cual cada par de ciudades c_j y c_k se encuentra conectado, de forma tal que se puede viajar desde la ciudad c_j hasta la ciudad c_k, y viceversa; y una matriz D de tamaño NxN, cuyo coeficiente D(j,k) indica el costo de viajar desde c_j a c_k (con D(j,k) no necesariamente igual a D(k,j)).

Se denomina **circuito** a cada una de las formas posibles de recorrer las N ciudades, partiendo de una ciudad determinada, pasando exactamente una vez por cada una de las restantes ciudades, y volviendo finalmente a la ciudad de partida. Cada circuito posee un costo asociado, dado por la suma de los costos de realizar cada uno de los trayectos entre dos ciudades consecutivas que componen el recorrido.

Se busca determinar, entre todos los circuitos posibles sobre las N ciudades, un circuito cuyo costo asociado sea mínimo.

Es posible establecer una relación biunívoca entre el conjunto de circuitos posibles y las permutaciones de los enteros entre 1 y N, según la cual la permutación $(a_1\ a_2\ a_3\ \dots\ a_N)$ se asocia al circuito $c_{a1}\rightarrow c_{a2}\rightarrow c_{a3}\rightarrow\dots\rightarrow c_{aN}\rightarrow c_{a1}$. Luego, el TSP puede plantearse como el problema de determinar, entre todas las posibles permutaciones de los enteros entre 1 y N, una permutación que minimice la función de costo f, definida del siguiente modo:

Dada una permutación p de los enteros entre 1 y N, $p=(a_1\ a_2\ \dots\ a_n)$, $f(p)=D(a_1,a_2)+D(a_2,a_3)+\dots+D(a_{n-1},a_n)+D(a_n,a_1)$.

Para un estudio más detallado de este problema, pueden consultarse [2] y [3].

Programación Lineal Entera

Dada una matriz A$\epsilon\mathbb{R}^{m\times n}$, un vector b$\epsilon\mathbb{R}^{m}$ y un vector c$\epsilon\mathbb{R}^{n}$, hallar x$\epsilon\mathbb{Z}^{n}$ tal que minimice:
$c_1.x_1+\ldots+c_n.x_n$, sujeto a que se verifique A.x=b.

A su vez, una variante importante de este problema es el problema conocido como Programación Lineal Entera 0-1, donde la condición $x\epsilon Z^n$ es reemplazada por $x\epsilon\{0,1\}^n$.

Para un estudio más detallado de este problema, puede consultarse [4].

3.3. Resolución algorítmica

En la resolución de problemas de Optimización Combinatoria se utilizan procedimientos conocidos como algoritmos, los cuales están constituidos por un conjunto finito de instrucciones bien definidas y finitamente expresables, las cuales conducen a una solución. Comúnmente, los mismos son ejecutados por medio de programas escritos en lenguajes de computación para su implementación en computadoras. Suelen estar formados por pasos simples, constituidos por operaciones matemáticas elementales (sumas, restas, productos, comparaciones, etc.), de modo tal que no resulta difícil realizar, en particular, cada uno de los pasos involucrados. La dificultad de su utilización radica, en cambio, en el hecho de que en muchos problemas de Optimización Combinatoria, los mejores algoritmos conocidos a partir de los cuales pueden resolverse dichos problemas están constituidos por cantidades de pasos tan numerosas, que aún con la capacidad de cálculo de las computadoras más avanzadas es imposible ejecutar el algoritmo eficientemente, esto es, empleando una cantidad de recursos computacionales de los cuales realmente se pueda disponer en la práctica.

La imposibilidad de desarrollar, en el caso de muchos problemas, algoritmos eficientes para su resolución, pese a la realización de numerosos esfuerzos, ha llevado a la suposición de que existen problemas inherentemente "intratables", es decir, problemas para los

cuales es en realidad imposible elaborar algoritmos que los resuelvan eficientemente. A partir de esto surge un interés por poder realizar, en el caso de que dicha suposición sea correcta, una clasificación de los problemas, según sean tratables o intratables. En ese sentido, nuestro problema, tal como expondremos luego, pertenece a una clase de problemas conocidos como NP-Hard, los cuales, se supone, no pueden ser resueltos algorítmicamente de forma eficiente. Esta clase incluye a los que pueden considerarse, en cierto sentido que luego desarrollaremos, los problemas de Optimización Combinatoria más difíciles de resolver algorítmicamente. Estas nociones serán formalizadas realizándose una descripción de estudios correspondientes a la Teoría de la Complejidad.

4. Complejidad: Teoría de NP-completitud

A continuación, llevamos a cabo una introducción a esta teoría por medio de una revisión de sus definiciones y resultados fundamentales. Dicho estudio no será ni riguroso formalmente ni exhaustivo, sino que solo será desarrollado en un grado suficiente para generar una noción clara de la dificultad del problema que abordamos en este trabajo. Quien desee profundizar sus contenidos puede hacerlo en [5] y [6].

4.1. Introducción

Uno de los principales objetivos de la teoría de NP-completitud, la cual fue iniciada por Cook en 1971 [6], es poder establecer un modelo teórico a partir del cual sea posible una clasificación de problemas, que formalice la distinción práctica entre aquellos problemas que pueden resolverse algorítmicamente de forma eficiente y aquellos que no, y que la misma no esté basada simplemente en la experiencia práctica al respecto, sino en la identificación de características propias de los problemas que sean determinantes en ese sentido. Los problemas son

tratados aquí como objetos matemáticos en sí mismos, separándolos en clases y estableciendo relaciones entre ellos, resultando necesaria una definición formal y operable de problema, que luego daremos.

Al establecer relaciones entre problemas, se potenciarán las posibilidades de abordar unos a partir de conocimientos adquiridos y técnicas elaboradas en la resolución de otros. De forma tal que este estudio teórico no solo tiene como resultado un mejor conocimiento de lo que puede esperarse en cuanto a la resolución de un problema determinado, saber si es un problema "tratable" o "intratable", sino que también proveerá, en muchos casos, nuevas herramientas para el abordaje del mismo.

En el modelo que expondremos, se clasifica a los problemas según los diferentes "grados de eficiencia" con los cuales pueden ser resueltos mediante algoritmos. Habiendo llegado a este punto, resulta necesario dar algunas definiciones.

Empezamos por definir un **problema** como una cuestión general definida a partir de un conjunto de parámetros junto con dominios dentro de los cuales pueden tomar valores dichos parámetros, más un enunciado en el cual se plantea un requerimiento, que puede ser la determinación de la existencia o no de un objeto matemático que satisfaga un conjunto de relaciones definidas a partir de los parámetros del problema, o determinar la validez de una cierta propiedad enunciada sobre los parámetros. Cada especificación de valores para los parámetros del problema en sus correspondientes dominios constituye una **instancia** del problema. Luego presentaremos una definición más formal del concepto de problema.

Anteriormente definimos a los **algoritmos** como procedimientos constituidos por un conjunto finito de instrucciones bien definidas y finitamente expresables, las cuales conducen a una solución. Diremos que un algoritmo resuelve un problema, cuando conduce a una solución al ser aplicado a cualquier instancia del mismo.

La eficiencia con la cual un algoritmo resuelve una instancia de un problema determinado será medida a partir de la relación entre el "esfuerzo" que requiere la implementación del algoritmo y el "tamaño" de la instancia. En cuanto a qué queremos decir al hablar de esfuerzo y tamaño, debemos empezar por aclarar que, dado que prácticamente todos los algoritmos desarrollados para la resolución de problemas de

Optimización Combinatoria están pensados para ser implementados en computadoras por medio de programas, al hablar de algoritmos, estaremos refiriéndonos en realidad a programas de computadora. El esfuerzo se mide, fundamentalmente, a partir de la cantidad de tiempo y espacio requeridos para la implementación del algoritmo (en una computadora, esto es el tiempo de cómputo y la memoria requeridos para correr el programa correspondiente), pero dado el carácter comúnmente preponderante del primero de estos dos recursos en los análisis de complejidad, nos concentraremos en él. Para simplificar el análisis, mediremos el tiempo a partir de la cantidad de pasos involucrados en la implementación del algoritmo.

Por tamaño de una instancia, nos referimos a la cantidad de símbolos necesarios para su definición, o sea, la longitud de la cadena de símbolos o "input" que codifica la información que debemos ingresar en el programa correspondiente para que éste trabaje.

4.2. Algoritmos de tiempo polinomial

Se considera que un algoritmo resuelve un problema de forma eficiente, si existe un polinomio p tal que, para cada instancia del problema, la implementación del algoritmo conlleva la realización de una cantidad de pasos acotada por el valor de p evaluado en el tamaño de la instancia. Es decir, si definimos para un algoritmo la **función de complejidad** $f:\mathbb{Z}_+ \rightarrow \mathbb{Z}_+$, donde f(n) es la mayor cantidad de pasos involucrados en la aplicación del algoritmo a una instancia de tamaño n, decimos que dicho algoritmo es eficiente, si existe un polinomio p tal que $f(n) \leq p(n)$, para todo n en $\mathbb{Z}_+$.

Esto se puede expresar a partir de la relación O entre funciones, definida de la forma siguiente: Dadas dos funciones f y g, decimos que se cumple que $f(n)=O(g(n))$ (se lee "f es del orden de g") si existe una constante c tal que para todo n en el dominio de las funciones, $f(n) \leq c.g(n)$. A partir de esto, podemos re-expresar lo anterior diciendo que un algoritmo resuelve eficientemente un problema si su función de complejidad es del orden de un polinomio, o sea, es de orden polinomial.

Esta noción de eficiencia está en relación con el hecho de que en la práctica, salvo escasas excepciones, solo se pueden implementar eficientemente algoritmos que involucran cantidades a lo sumo polinomiales de pasos para cada instancia del problema a resolver.

Llamaremos **algoritmo de tiempo polinomial** a un algoritmo que cumpla lo recientemente expuesto. Si dado un problema determinado, existe un algoritmo de tiempo polinomial que lo resuelva, consideraremos a tal problema como "tratable" o "solucionable" (en este último término las comillas aparecen por el hecho de que, si bien basta con que exista un algoritmo que resuelva el problema para que este sea solucionable, deseamos usualmente poder conocer dicha solución, entendiéndolo como una condición necesaria para poder decir que el problema es solucionable, y esto solo es posible si se puede implementar el algoritmo eficientemente).

A continuación se presenta un ejemplo de algoritmo de tiempo polinomial:

Problema: Dada una lista de n números enteros $x_1,\ldots,x_n$, se desea ordenarlos de menor a mayor.

Para esto podemos aplicar el algoritmo denominado Bubble Sort, el cual trabaja en n etapas.

-En la primera etapa, el algoritmo procede del siguiente modo: Se comienza por comparar el primer número de la lista, o sea x_1, con el segundo número de la lista, o sea, x_2. Si se tiene que $x_1 \leq x_2$, entonces no se realiza ningún cambio en la lista; si se tiene que $x_1 > x_2$, se intercambian las posiciones de x_1 y x_2 en la lista. A continuación, el número que ocupe el segundo lugar de la lista (x_2 en el primer caso, x_1 en el segundo) es comparado con x_3, procediéndose de forma similar según el resultado de dicha comparación, es decir, se mantiene la posición de ambos si el número que está en la segunda posición es menor o igual a x_3, y se intercambian sus posiciones en caso contrario. Este procedimiento se repite sucesivamente, hasta alcanzar la última posición de la lista, realizándose de este modo n-1 comparaciones en esta primera etapa, y es posible ver que, al finalizar la misma, el

máximo (o uno de los máximos si hay más de uno), entre los números de la lista, ocupará la última posición.

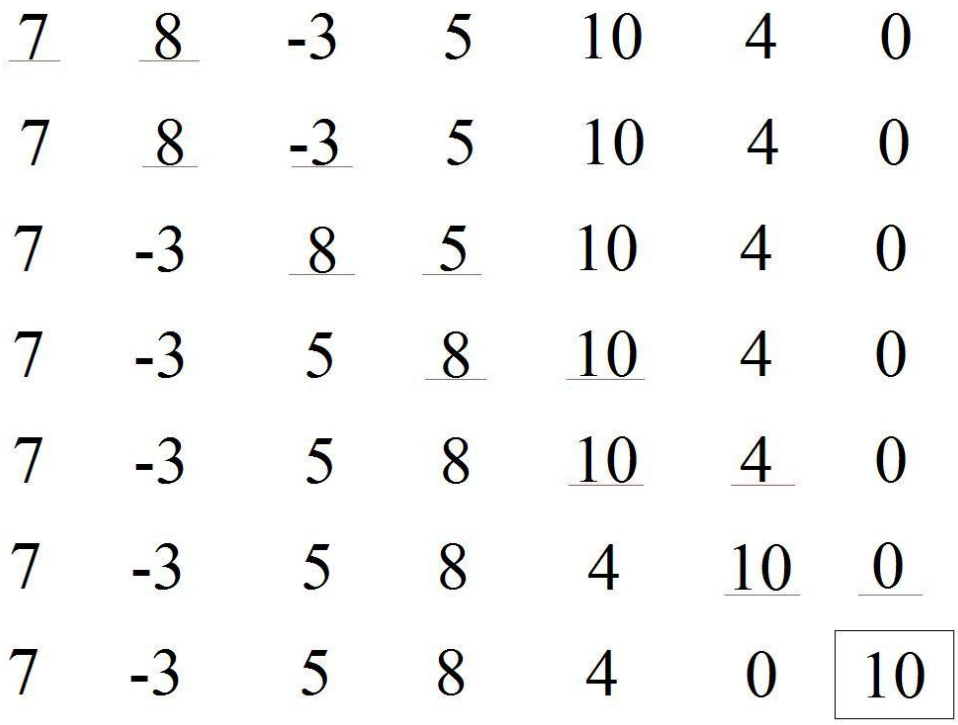


Fig.4: Evolución del orden de una lista de números durante la aplicación de la primera etapa del algoritmo Bubble Sort.

-La segunda etapa consiste en repetir el proceso de comparaciones sucesivas con reordenamientos de la primera etapa, pero solo sobre los n-1 primeros números de la lista, y puede verse, a partir de lo ocurrido en la primera etapa, que al finalizar esta segunda etapa, el segundo número más grande de la lista se encontrará en la posición n-1, tal como se desea.

-Las siguientes etapas consisten de la aplicación de las mismas operaciones que en las dos primeras etapas, solo que en cada etapa debe dejarse fuera de consideración la última posición considerada en la etapa anterior.

Al finalizar la aplicación de este algoritmo sobre la lista de números, se habrá alcanzado el ordenamiento deseado y la complejidad del algoritmo puede determinarse del siguiente modo:

Dada una instancia del problema compuesta por una lista de n números, en la primer etapa de la aplicación del algoritmo se realizan n-1 comparaciones, con a lo sumo n-1 reordenamientos; en la segunda etapa se realizan n-2 comparaciones, con a lo sumo n-2 reordenamientos, y así sucesivamente, hasta llegar a la etapa n-1, en la que se realizan n-(n-1)=1 comparación, con a lo sumo 1 reordenamiento. De este modo, la cantidad total de pasos realizados por el algoritmo no es superior a: $2.(n-1+n-2+\ldots+1)= 2.(n-1).n/2= (n-1).n<n^2$

De este modo, el algoritmo Bubble Sort resulta ser de tiempo polinomial, ya que su función de complejidad está acotada por el polinomio $p(x)=x^2$.

Cuando un algoritmo que resuelve un problema no es de tiempo polinomial, es decir, su función de complejidad no es del orden de ningún polinomio, o sea, es de orden superior al polinomial, diremos que el mismo es un **algoritmo de tiempo exponencial**. Si dado un problema determinado, no existe un algoritmo de tiempo polinomial que lo resuelva, es decir, solo puede ser resuelto mediante algoritmos de tiempo exponencial, consideraremos a tal problema como "intratable".

4.3. Problemas de decisión. Lenguajes.

En esta teoría se considera, como objetos de estudio, solo a problemas del tipo denominado **problemas de decisión**. Estos son problemas que requieren, como solución, simplemente una respuesta del tipo "sí" o "no". Es decir, en términos de la definición de problema antes presentada, son problemas en los que debe decidirse si existe o no un objeto matemático que satisfaga las relaciones definidas a partir de los parámetros del problema, sin que se requiera que el mismo sea presentado en caso de existir; o son problemas en los que debe decidirse la validez de una propiedad enunciada sobre los parámetros.

La razón de esta elección es que este tipo de problemas pueden ser formalizados de forma precisa y operable a través del concepto de lenguaje, que a continuación definiremos.

Si bien esta restricción a problemas de decisión podría parecer inconveniente, ante nuestro interés de estudiar un problema que no es de decisión sino de Optimización, nuestro principal interés en esta teoría consiste en que la misma nos permita concluir que nuestro problema, bajo supuestos muy plausibles, no puede ser resuelto de forma óptima de manera eficiente, al menos con los recursos de los que se dispone en la actualidad, para luego plantear otras alternativas de resolución, que si bien no estén orientadas a alcanzar soluciones óptimas, aún así puedan resultar provechosas. Como luego veremos, determinar una solución óptima no es necesariamente la única alternativa de interés al abordar un problema de Optimización Combinatoria, especialmente cuando éste está basado en un "problema de la vida real".

Ahora, al igual que ocurre con casi cualquier problema de Optimización Combinatoria, es posible, a partir de nuestro problema, generar un problema de decisión que, en un sentido que luego desarrollaremos, no sea más difícil de resolver que nuestro problema. De modo que si se pudiera demostrar que este problema de decisión es intratable, podríamos concluir que el nuestro, no más sencillo, también lo es.

La forma en que podemos obtener tal problema de decisión a partir de nuestro problema ejemplifica la forma usual en que son generados problemas de decisión a partir de problemas de Optimización Combinatoria, y es la siguiente: Agregamos una constante B (denominada a menudo "budget") a los parámetros que definen nuestro problema original, y planteamos el requerimiento de determinar si existe o no una solución, es decir una selección de un conjunto de t.b.m.'s a partir del cual pueda cubrirse la lista de pedidos, con un costo asociado no superior a B.

Observemos que, de poder resolver nuestro problema de determinar una selección óptima de t.b.m.'s con los cuales cubrir una lista de pedidos, podemos luego resolver el problema de decisión asociado, para cualquier budget B, simplemente comparando con B el costo de la solución óptima obtenida. Es en ese sentido, que entendemos que el

problema de decisión asociado a nuestro problema de Optimización Combinatoria no es, en esencia, más difícil de resolver.

A continuación, formalizaremos el concepto de problema de decisión.

Dado un conjunto finito de símbolos A (o alfabeto), A* representa el conjunto de cadenas o "strings" finitos (o palabras) que pueden formarse a partir de los símbolos de A, incluyendo a la cadena vacía. Se llama **lenguaje** sobre A a cualquier subconjunto L de A*.

Dado un problema de decisión Π, asociamos a él lo que se denomina un **esquema de codificación,** al que llamamos e, el cual consiste de un alfabeto junto con una forma de expresar, a partir de palabras formadas con sus símbolos, las instancias de Π. Aquellas palabras que definen instancias "sí" de Π, es decir instancias para las cuales la respuesta es "sí", componen el lenguaje que asociamos a Π bajo el esquema de codificación e, y lo representamos como L[Π,e].

Será a partir de esta asociación entre problemas de decisión y lenguajes, que, mediante el estudio formal de los segundos, se obtendrán conclusiones acerca de los primeros. En cuanto a los esquemas de codificación que se utilizan para representar problemas de decisión, se entiende que estos son siempre "razonables", en el sentido de que la codificación de los diferentes parámetros que definen el problema se lleva a cabo sin "excesos" en la utilización de símbolos del lenguaje, o sea de forma concisa, y de forma tal que resulte posible de decodificar en tiempo polinomial. Esta definición informal, de naturaleza práctica, y cuya comprensión es dependiente, por lo tanto, de experiencia práctica en la codificación de objetos matemáticos, podría formalizarse a partir de la definición de una forma concisa de codificación para cada objeto matemático que pueda ser un parámetro de un problema de decisión, pudiendo verse un ejemplo de tal formalización en [5]. Los diferentes esquemas de codificación razonables que podamos utilizar para un mismo problema habrán de conducirnos a los mismos resultados en cuanto a su solucionabilidad, básicamente por el hecho de que las longitudes de las palabras que codifiquen las instancias del problema, en los diferentes esquemas de codificación, estarán polinomialmente relacionadas. De modo que nos independizaremos de esta variabilidad, asumiendo en cada caso el uso de un esquema de codificación razonable en particular. Es decir,

supondremos el uso de un esquema de codificación estándar para cada problema.

De este modo, asociaremos a cada problema de decisión Π un único lenguaje L[Π], generado a partir de un esquema de codificación estándar, y cuando hablemos del tamaño de una instancia del problema, nos referiremos a la cantidad de símbolos que componen la palabra de L[Π] que codifica dicha instancia.

4.4. Máquinas de Turing Deterministas

La formalización del concepto de algoritmo, junto con la independencia de tener que considerar computadoras en particular para su implementación, podemos obtenerlas a partir del concepto abstracto de **Máquina de Turing Determinista**, un modelo de cómputo formal desarrollado por Alan Turing en 1936, el cual presentó en su trabajo [7].

Una Máquina de Turing Determinista (MTD) M puede describirse como compuesta de una cinta formada por infinitas celdas, que se suceden regularmente en ambos sentidos de la misma y se encuentran etiquetadas mediante números enteros; junto con un cabezal de lecto-escritura capaz de desplazarse sobre la cinta, una celda a la vez (a izquierda o derecha), leyendo, escribiendo y sobre-escribiendo símbolos de un alfabeto finito determinado sobre las celdas; más un controlador de los finitos estados posibles en los que puede encontrarse el proceso ejecutado por M.

Formalmente, una Máquina de Turing viene dada por una terna (A,E,d):

-A es el alfabeto de la máquina. Es finito y contiene un símbolo distinguido "blanco" b, el cual representa, en realidad, la ausencia de símbolo en una celda.

-E es el conjunto de estados de la máquina. Es finito y contiene un estado e_0 distinguido, que corresponde al estado inicial, y un subconjunto de estados finales distinguidos, llamados estados "halt", los cuales pueden ser: un estado de detención D, uno de aceptación q_s y otro de rechazo q_n.

-d es lo que se llama "función de transición". Determina, a partir del estado en que se encuentre la máquina y el símbolo que esté siendo leído o "escaneado" por el cabezal, cuál es el siguiente paso a ejecutarse, indicando para ello cuál debe ser el nuevo símbolo a escribir sobre la celda escaneada (el cual puede ser el mismo que acaba de ser leído); cuál debe ser el siguiente desplazamiento del cabezal: una celda hacia la izquierda o una celda hacia la derecha; e indicando si debe mantenerse el estado actual o debe cambiarse por otro determinado. Todo esto a menos que la máquina haya alcanzado un estado halt, en cuyo caso se detiene el proceso. Cada una de estas transiciones así determinadas, constituye un "paso" del proceso. La función d podría entenderse como el programa de la máquina.

El proceso realizado por la máquina se inicia a partir del estado inicial e_0. El "input", constituido por una palabra de (A-{b})* (conjunto de todas las palabras formadas a partir del conjunto de símbolos del alfabeto A a excepción del símbolo b (denominado "alfabeto input")), se encuentra a partir de la celda número 1 hacia la derecha, y la cantidad finita de celdas ocupadas por el mismo se define como su tamaño, mientras que todas las celdas restantes se encuentran ocupadas por el símbolo b. Luego, el proceso continúa de acuerdo a lo especificado por la función de transición, de forma indefinida o hasta alcanzar eventualmente un estado halt, en cuyo caso se detiene y la palabra formada por los símbolos presentes en la cinta al finalizar el proceso constituye el "output" de la máquina.

Dadas una MTD M, cuyo alfabeto es A, y una palabra x de (A-{b})* que al ser ingresada como input conduce al estado q_s de aceptación, decimos que x es **aceptada** por M. Si en cambio, x condujera al estado q_n de rechazo, diremos que x es **rechazada** por M. El lenguaje L_M formado por todas las palabras de (A-{b})* que son aceptadas por M es denominado como el **lenguaje reconocido** por M.

Diremos que una Máquina de Turing Determinista M **resuelve** un problema de decisión Π, bajo un esquema de codificación e, cuando alcanza un estado halt a partir de cualquier palabra de e que sea ingresada como input y se cumple que $L_M=L[\Pi,e]$, es decir, el conjunto de palabras que M acepta es el conjunto de palabras que definen instancias "sí" de Π bajo el esquema e.

La forma de operar de una MTD formaliza la noción de algoritmo, aunque, en realidad, dicha formalización se obtiene a partir de aquellas MTD´s tales que siempre, dado cualquier input posible a partir de su alfabeto input, alcanzan un estado de detención. Ya que, en principio, el proceso podría prolongarse infinitamente, sin alcanzar nunca un estado de detención, lo cual no estaría en correspondencia con nuestra noción de algoritmo, según la cual estos están constituidos por una lista finita de pasos que siempre conduce a una solución.

Por lo visto, este modelo de cómputo tiene una estructura muy simple y, en principio, sus posibilidades pueden parecer muy limitadas si uno piensa que lo que se intenta es obtener resultados teóricos que se apliquen a la ejecución de algoritmos en cualquier clase de computadoras conocida. Pero, pese a su simpleza, es posible ver que este modelo de cómputo es tan "fuerte" como cualquier computadora, en el sentido de que cualquier algoritmo que pueda ser ejecutado en tiempo polinomial a través de un programa de computadora puede ser ejecutado en tiempo polinomial por una Máquina de Turing Determinista. Es en ese sentido, que este modelo de Máquinas de Turing Deterministas (o más específicamente, Máquinas de Turing Deterministas de una cinta) es equivalente a las computadoras modernas para los objetivos de esta teoría.

Las Máquinas de Turing Deterministas, junto con otros modelos abstractos de cómputo equivalentes, como el de Máquinas de Turing Deterministas de múltiples cintas o el de Máquinas de Acceso Aleatorio, pueden estudiarse con detalle en [8].

Dada una MTD M y un input x a partir del cual M alcanza un estado halt, el **tiempo** empleado por M sobre x será la cantidad de pasos transcurridos en el correspondiente cómputo hasta alcanzar un estado halt. Si M alcanza un estado halt para cada uno de los inputs posibles, podemos definir una **función de complejidad temporal** para M dada por:

C_M: $\mathbb{Z}_+ \rightarrow \mathbb{Z}_+$,
$C_M(n)$=máx{t: Existe al menos un input x, de tamaño n, tal que el cómputo de M sobre x requiere un tiempo t}

Decimos que M es de **tiempo polinomial** si existe un polinomio p tal que, para todo n en $\mathbb{Z}_+$, $C_M(n) \leq p(n)$.

4.5. Clase P

Definimos la **clase de lenguajes P** como la clase formada por cada lenguaje L para el cual existe una MTD de tiempo polinomial que lo reconoce.

Luego, haciendo uso de la asociación entre problemas de decisión y lenguajes, decimos que un problema de decisión Π, bajo un esquema de codificación e, pertenece a la clase P, si $L[\Pi,e] \epsilon P$, y entendemos que la MTD de tiempo polinomial que reconoce L[Π,e] resuelve Π. Por lo dicho anteriormente, si esto ocurre para un esquema de codificación razonable e, también vale para cualquier otro esquema de codificación razonable, por lo cual diremos simplemente que Π está en P.

Por otro lado, dada la equivalencia entre el modelo de MTD con cualquier otro modelo de cómputo razonable, entendiendo así a cualquier modelo de cómputo que solo pueda realizar una cantidad de trabajo polinomialmente acotada por unidad de tiempo, resulta que si definiéramos la clase P en función de cualquiera de esos otros modelos, de forma similar a la que hemos llevado a cabo para Máquinas de Turing Deterministas, dicha clase estaría formada por los mismos lenguajes, de modo que, en cierta forma, podemos independizarnos también del modelo de cómputo empleado y de la representación de los algoritmos, así como pudimos hacerlo del esquema de codificación de las instancias de un problema.

4.6. Clase NP. Máquinas de Turing No Deterministas

La siguiente clase de problemas que estudiamos es la llamada clase NP, que se encuentra formada por aquellos problemas de decisión Π para los cuales se cumple que, dada una instancia "sí" cualquiera de Π, es decir una instancia para la cual la solución o respuesta es "sí", es posible comprobar en tiempo polinomial que la misma es en efecto una instancia "sí". Observemos que esto no es necesariamente equivalente

a decir que el problema puede ser resuelto en tiempo polinomial, o sea, que podamos para cualquier instancia del problema determinar su solución en tiempo polinomial, sino simplemente que se puede verificar en tiempo polinomial que una instancia tiene como respuesta "sí", si ese es el caso. Esta verificabilidad de tiempo polinomial es posible gracias a la existencia de un objeto matemático asociado a cada instancia "sí" del problema, de tamaño polinomialmente acotado por el tamaño de la misma, a partir del cual puede llevarse a cabo, mediante un algoritmo de tiempo polinomial con respecto al tamaño de la instancia, dicha verificación. Se denomina **certificado sucinto** a tal objeto**.**

Por ejemplo, en el problema TSP antes presentado, en su versión de decisión, en la cual se pregunta por la existencia o no de un circuito cuya longitud no sea superior a un determinado valor B, dada una instancia "sí", es decir una instancia del problema para la cual existe al menos un circuito que cumple la condición enunciada, cualquiera de tales circuitos constituye un certificado sucinto para la instancia, dado que su tamaño está claramente acotado polinomialmente por el tamaño de la instancia, ya que está constituido por algunos de los parámetros que definen la misma, y a partir de él se puede verificar que la instancia tiene como respuesta "sí", simplemente sumando las longitudes de todos los tramos que componen el circuito y comparando luego el resultado con B, y dicha verificación está polinomialmente acotada por el tamaño de la instancia. Por lo tanto, podemos concluir que el problema TSP en su versión de decisión (TSP Decisión) es NP.

Del mismo modo, podemos observar que el problema que planteamos en este trabajo, en su versión de decisión, es un problema NP, ya que dada una instancia "sí" del mismo, la cual tiene como budget un determinado valor B, cualquier selección de un conjunto de t.b.m.'s con un costo asociado no superior a B constituye un certificado sucinto para la instancia.

Observemos, a partir de estos dos ejemplos de problemas NP, que el certificado sucinto no tiene que poder ser explicitado para que se pueda decir que el problema es NP, sino que simplemente se debe saber que existe, conocer cuál es su estructura. Poder determinar un certificado sucinto es, en general, tan complejo en sí mismo como resolver el problema.

Si para cada instancia de un determinado problema de decisión NP pudiéramos, de algún modo, construir todos los posibles certificados sucintos para la misma, es decir, todos aquellos objetos matemáticos, de tamaño polinomialmente acotado a partir del tamaño de la instancia, que pudieran ser un certificado sucinto en el caso de que la instancia fuera una instancia "sí" (en el ejemplo del TSP Decisión recién expuesto, todos los objetos candidatos a ser certificados sucintos de una instancia son todos los circuitos que pueden formarse a partir de las ciudades que componen la instancia; y en el caso de nuestro problema, en su versión de decisión, dada una instancia en particular del mismo, los posibles certificados sucintos serían todas las posibles selecciones factibles de t.b.m.'s), y analizar, a partir de cada uno de ellos, si es posible verificar que la instancia es una instancia "sí", podríamos luego, mediante un algoritmo que hiciera esto, resolver cualquier instancia "sí", o sea, dada una instancia "sí" del problema, este algoritmo nos conduciría a obtener esa respuesta.

El problema es que, en general, la cantidad de certificados sucintos posibles es de orden exponencial, y un algoritmo determinista, es decir un algoritmo del tipo de los que pueden implementarse en la práctica, en los cuales a partir de cada paso queda unívocamente determinado el siguiente paso, de forma tal que el algoritmo procede siguiendo una única línea de cómputo, emplearía una cantidad de tiempo de orden exponencial para realizar la tarea de generar y analizar, sucesivamente, todos los certificados posibles.

En cambio, si fuera posible implementar lo que se conoce teóricamente como algoritmos no deterministas, los cuales tienen, con respecto a los algoritmos deterministas, la capacidad adicional de que a partir de cada paso pueden quedar determinados dos diferentes pasos a seguir en paralelo, de forma tal que el cómputo se ramifique en dos nuevas líneas de cómputo paralelas y así, mediante la aplicación sucesiva de este recurso, el algoritmo se ramifique en una cantidad exponencial de líneas de cómputo con respecto al tiempo transcurrido, podríamos tal vez, dado un problema NP, desarrollar un algoritmo que vaya construyendo y analizando, mediante cómputos paralelos, todos los posibles certificados sucintos, deteniéndose al hallar un primer candidato que, en efecto, sea un certificado sucinto. Y dado que para cualquier instancia "sí" del problema, su certificado

sucinto es de tamaño polinomialmente acotado y puede ser analizado en tiempo polinomial, aquel cómputo, entre todos los cómputos paralelos, que construye y analiza tal certificado podría realizar la verificación deseada en tiempo polinomial y por lo tanto el algoritmo en sí mismo estaría solucionando el problema, para cualquier instancia "sí", en tiempo polinomial. Esto dista, en realidad, de poder entenderse como un proceso de resolución del problema en el sentido clásico, ya que no dice nada al respecto de instancias "no", para las cuales el proceso llevado a cabo por el algoritmo se ejecutaría indefinidamente.

Un algoritmo con tales capacidades es imposible de elaborar y aplicar en la práctica. De esta forma, los algoritmos no deterministas solo pueden ser concebidos de forma abstracta, aunque eso no quita que los mismos puedan servir para obtener importantes conclusiones prácticas, tal como se ve en la teoría brevemente expuesta aquí.

Se dice que un algoritmo no determinista resuelve un problema de decisión Π, si da como respuesta "sí" para todas sus instancias "sí" y no lo hace para instancias "no", y es posible ver que la clase de problemas de decisión NP está constituida por todos aquellos problemas que pueden ser resueltos en tiempo polinomial mediante un algoritmo no determinista, siendo eso equivalente a la propiedad de poseer certificados sucintos para las instancias "sí". Una prueba de esto puede encontrarse en [6].

Para continuar con el estudio teórico, formalizaremos la noción de algoritmo no determinista y de problema de decisión NP. Para ello seguiremos la forma de interpretar el comportamiento de un algoritmo no determinista expuesta en [5], para luego formalizar tales algoritmos a partir del modelo de Máquinas de Turing No Deterministas (MTND), y a partir del mismo formalizar la noción de problemas de decisión NP, mediante la definición de la clase de lenguajes NP.

Un algoritmo no determinista puede entenderse como un algoritmo que trabaja en dos etapas: una "etapa de suposición**"** y una "etapa de chequeo**"**. Dada una instancia I de un problema Π a resolver, en la etapa de suposición se comienza por suponer una estructura S. Luego, S y la codificación de I son tomados como input para la segunda etapa, de chequeo, en la cual son procesados de forma determinista.

Decimos que un algoritmo no determinista resuelve un problema de decisión Π, cuando se cumple que, dada cualquier instancia "sí" del

mismo, existe al menos una estructura S que al ser supuesta en la primera etapa e ingresada junto con la codificación de la instancia como input en la etapa de chequeo, conduce al algoritmo a determinar que en efecto la instancia es "sí", y además esto no ocurre para las instancias "no" del problema, es decir, no existe una estructura S que cumpla esto para una instancia "no" del problema.

Además, decimos que un algoritmo no determinista resuelve un problema de decisión Π en tiempo polinomial, cuando existe un polinomio p tal que para cualquier instancia "sí" I_s de Π existe al menos una suposición S para la cual el algoritmo determina que I_s es una instancia "sí" en una cantidad de pasos acotada por $p(|I_s|)$, donde $|I_s|$ es el tamaño de la codificación de I_s.

Ahora formalizaremos el concepto de algoritmo no determinista mediante el modelo de **Máquinas de Turing No Deterministas (MTND)**. Por tal puede entenderse a una Máquina de Turing tal como la hemos definido en el caso determinista, con un alfabeto, un conjunto finito de estados y un cabezal de lecto-escritura, con las mismas características y elementos distinguidos que antes, pero a la cual se le ha agregado un cabezal de solo escritura, es decir, que solo tiene la capacidad de escribir símbolos, el cual actúa solo sobre las celdas etiquetadas con números enteros negativos. Una vez ingresado el input x (desde la celda 1 hacia la derecha, igual que antes), esta máquina trabaja en dos etapas: en la primera, a la cual se denomina **etapa de suposición**, actúa el cabezal de solo escritura, el cual, de forma totalmente arbitraria, escribe una palabra a partir de la celda -1 hacia la izquierda, utilizando los símbolos del alfabeto de la Máquina hasta detenerse luego de una cantidad de pasos también arbitraria (pudiendo en principio nunca detenerse), quedando escrita una palabra cualquiera de A*. Al detenerse el cabezal, si es que esto ocurre, finaliza la primer etapa y comienza la segunda etapa, denominada **etapa de chequeo**, en la cual actúan el controlador de estados, a partir del estado inicial q_0, y el cabezal de lecto-escritura, que trabajan de la misma forma que en una Máquina de Turing Determinista, llevando a cabo un cómputo a partir de x y de la palabra surgida de la etapa de suposición. Dado un input x, cada palabra distinta surgida de la etapa de suposición lleva a que la máquina, en principio, compute de forma distinta y diremos que una MTND M

acepta un input x, cuando existe alguna palabra S que, al ser supuesta, conduce luego el cómputo de la máquina, en la etapa de chequeo, al estado de aceptación q_s. A su vez, definiremos como **lenguaje reconocido** por M, al conjunto L_M de palabras de (A-{b})* que son aceptadas por M, y diremos que M requiere un **tiempo** t para aceptar una palabra x de L_M, si t es la menor cantidad de pasos empleados por M para aceptar x, sobre todos los cómputos posibles de M que conducen a aceptar x, es decir, sobre todos los cómputos correspondientes a palabras S que, al ser supuestas, conducen luego a M a aceptar x. Luego, podemos definir la **función de complejidad temporal** para M de la siguiente manera:

CTM: $\mathbb{Z}_+ \rightarrow \mathbb{Z}_+$,
CTM(n)=máx({1} U {t: Existe x en LM tal que |x|=n y M emplea un tiempo t en aceptar x})

Diremos que M es de **tiempo polinomial,** si existe un polinomio p tal que CTM(n)≤p(n), para todo n en $\mathbb{Z}_+$.

Llamaremos NP al siguiente conjunto de lenguajes: NP={L: existe una MTND M de tiempo polinomial para la cual L_M=L}.

Diremos que un problema de decisión Π, bajo un esquema de codificación e, pertenece a la clase NP, si L[Π,e] (o sea el lenguaje formado por aquellas palabras que codifican instancias "sí" de Π, bajo el esquema de codificación e) pertenece a NP, y del mismo modo que antes, cuando haya un esquema de codificación e razonable para el cual esto se cumpla, dada la equivalencia de este tipo de esquemas para los objetivos de este estudio, diremos simplemente que Π está en NP.

Además, es posible ver que si definimos, de forma similar a la de recién, la clase de lenguajes NP a partir de cualquier modelo de cómputo equivalente al de MTD con el agregado de una etapa de suposición, obtendremos el mismo conjunto de lenguajes que en el caso desarrollado a partir del modelo de MTND, de modo tal que podemos independizar la clase NP del modelo de MTND.

4.7. Relación entre las clases P y NP

Habiendo definido ya las clases de problemas P y NP, observemos una primera relación entre ambas dada por el hecho de que todo problema de la clase P pertenece a su vez a NP, es decir, se cumple que la clase P está incluida en la clase NP. Formalmente, esto puede verse de la siguiente forma: Dado un lenguaje L en P, existe una Máquina de Turing Determinista M que reconoce L en tiempo polinomial, y podemos, a partir de ella, construir una MTND M´ que trabaje en la etapa de chequeo de forma idéntica a M sin tener en cuenta la palabra supuesta en la primer etapa, la cual reconocerá L.

Más informalmente, podemos pensar a cualquier problema Π de la clase P, como un problema de la clase NP, a cuyas instancias "sí" podemos asociar como certificado sucinto cualquier objeto matemático, de tamaño tan pequeño como sea necesario para satisfacer la condición de que esté acotado polinomialmente por el tamaño de la instancia, y utilizar, como proceso de verificación de que la instancia es "sí", al algoritmo de tiempo polinomial que resuelve Π (el cual existe por pertenecer Π a la clase P), sin necesidad de que el mismo emplee el certificado sucinto.

Ahora, una de las cuestiones sin solución más importantes de la Matemática y la Computación, en la actualidad, es decidir si es cierto o no que se cumple que todo problema de NP está en P, es decir, si vale o no que la clase NP está incluida en la clase P y por lo tanto, sumado esto a lo anterior, vale que P=NP.

Dada la existencia de muchos problemas de decisión que se sabe están en NP, y para los cuales muchos investigadores han realizado numerosos esfuerzos, a través de varios años, en la búsqueda de algoritmos de tiempo polinomial para su resolución, sin que esto se haya podido lograr, y dado que la capacidad de los algoritmos no deterministas de realizar en paralelo cantidades arbitrarias de cómputos parece ubicarlos en un nivel superior al de los algoritmos deterministas, en el sentido de que resulte esperable que esa capacidad superior de cómputo permita solucionar en tiempo polinomial problemas que no pueden solucionarse de ese modo con algoritmos deterministas, ha surgido con fuerza la conjetura de que existen problemas de NP que no están en P.

Si la conjetura recién planteada, la cual no ha podido ser probada, resultara cierta, entonces el conjunto de problemas de decisión NP podría separase en dos subconjuntos no vacíos: El conjunto P formado por aquellos problemas de decisión solucionables por medio de algoritmos de tiempo polinomial, es decir, aquellos que son tratables; y por otro lado, el conjunto de problemas intratables de NP (NP\P), para los cuales no existen algoritmos de tiempo polinomial que los resuelvan. Luego, dado un problema de NP, sería de mucho interés poder distinguir a cuál de los dos subconjuntos pertenece, para saber qué puede esperarse de su abordaje práctico, sabiendo que existiría la posibilidad concreta de que el mismo fuera intratable.

Si bien hasta el momento no ha podido probarse que el subconjunto NP\P es no vacío, sí se ha logrado probar lo siguiente para muchos problemas de decisión Π de NP:

$$P \neq NP \rightarrow \Pi \text{ pertenece a } NP\backslash P.$$

O sea, si realmente existen problemas intratables en NP, entonces Π es intratable. Este es el resultado que buscamos ver que se aplica a nuestro problema en su versión de decisión, y será la forma débil en que veremos que el mismo es intratable. La prueba de este tipo de resultados se basa en el concepto de **transformación polinomial** entre problemas, el cual se enunciará formalmente a continuación.

4.8. Transformación polinomial

Dados dos alfabetos A_1 y A_2, y dos lenguajes: L_1 incluido en A_1^* y L_2 incluido en A_2^*, decimos que L_1 se transforma polinomialmente en L_2, si existe una función $f:A_1^* \rightarrow A_2^*$ (a la cual se denomina transformación polinomial (**t.p.**) desde L_1 a L_2) tal que puede ser computada por una MTD de tiempo polinomial, y cumple que: para todo $x \epsilon A_1^*$, $x \epsilon L_1$ si y solo si $f(x) \epsilon L_2$. La notación que utilizamos para representar esto es: $L_1 \Theta L_2$.

Lema 1: Dados L_1 incluido en A_1^* y L_2 incluido en A_2^*, si $L_1 \Theta L_2$ entonces: L_2 en P → L_1 en P.

Dem.: Si L_2 está en P, existe una MTD M_2 que reconoce a L_2 en tiempo polinomial, y dado que $L_1 \Theta L_2$, existe una t.p. desde L_1 a L_2: $f:A_1^* \rightarrow A_2^*$ (f cumple que: $x \epsilon L_1$ si y solo si $f(x) \epsilon L_2$, es decir, $x \epsilon L_1$ si y solo si M_2 reconoce f(x)), la cual es computada por una MTD M_f.

Consideremos ahora, una MTD M_1 tal que, dado un input $x \epsilon A_1^*$, opera primero como M_f obteniendo f(x), y a partir de allí opera como M_2. Dado que: $x \epsilon L_1$ si y solo si $f(x) \epsilon L_2$, resulta que M_1, resultante de componer M_2 y M_f, reconoce a L_1, y del hecho de que la composición de polinomios da como resultado un polinomio se sigue que M_1 opera en tiempo polinomial.

Dados dos problemas de decisión Π_1 y Π_2, bajo esquemas de codificación e_1 y e_2 respectivamente, diremos que Π_1 se transforma polinomialmente en Π_2 ($\Pi_1 \Theta \Pi_2$), si $L[\Pi_1,e_1] \Theta L[\Pi_2,e_2]$.

De este modo, bajo la suposición de que nos limitamos a considerar esquemas de codificación estándar, podemos entender la relación $\Pi_1 \Theta \Pi_2$ como determinada por la existencia de una función $f:I(\Pi_1) \rightarrow I(\Pi_2)$, donde $I(\Pi_i)$ es el conjunto de instancias de Π_i, que sea computable por un algoritmo de tiempo polinomial y cumpla que, dada una instancia I de Π_1, I es una instancia "sí" si y solo si f(I) es una instancia "sí" de Π_2.

Por el Lema 1, llevado al nivel de problemas, dados dos problemas Π_1 y Π_2 tales que $\Pi_1 \Theta \Pi_2$, Π_2 en P → Π_1 en P. Luego, si un problema Π_1 puede transformarse polinomialmente en un problema Π_2, resulta que si Π_2 puede resolverse en tiempo polinomial, o hablando en términos prácticos, puede resolverse, también podrá resolverse Π_1. A su vez, si Π_1 es intratable también lo será Π_2, de modo que podemos interpretar, a partir de la relación $\Pi_1 \Theta \Pi_2$, que Π_2 es "al menos tan difícil de resolver" como Π_1.

En caso de que, además de valer $\Pi_1 \Theta \Pi_2$, valga $\Pi_2 \Theta \Pi_1$, diremos que Π_1 y Π_2 son **polinomialmente equivalentes**, y entenderemos que tienen la misma dificultad.

4.9. Clase NP-c

Es posible ver que la relación Θ es transitiva. Para ello basta con observar, en términos formales, que si f_1 constituye una t.p. desde un lenguaje L_1 a otro L_2, y f_2 es una t.p. desde L_2 a un lenguaje L_3, entonces la composición $f_2 \circ f_1$ es una t.p. entre L_1 y L_3.

De la transitividad de Θ se sigue que la relación, entre problemas NP, de ser polinomialmente equivalentes es una relación de equivalencia, ya que, además de ser transitiva a partir de esto último, es simétrica, por definición, y obviamente reflexiva, y por lo tanto separa a NP en clases de equivalencia. Una de estas clases está formada por el conjunto P, a cuyos integrantes podríamos considerar como los problemas más "sencillos" de NP, ya que cualquier problema que pueda transformarse polinomialmente en un problema de P estará a su vez en P. En el otro extremo, existe una clase de problemas que podemos considerar como los más "difíciles" de NP, es la clase de problemas llamados NP-completos. Todo problema Π de esta clase cumple que, dado cualquier otro problema Π' en NP, Π' puede transformarse polinomialmente en Π. De este modo, si cualquier problema de esta clase, denotada usualmente como NP-c, pudiera resolverse en tiempo polinomial, entonces todos los problemas de NP podrían ser resueltos a su vez en tiempo polinomial, mientras que de existir un problema intratable en NP, entonces todos los problemas de NP-c serían intratables.

Más formalmente, un lenguaje L es NP-completo, si está en NP y para todo otro lenguaje L' en NP se cumple que L'ΘL. Informalmente, diremos que un problema de decisión Π es NP-completo, si está en NP y para todo problema Π' en NP vale que Π'ΘΠ.

Tenemos entonces que de existir problemas intratables en NP, tal como se conjetura, los problemas de la clase NP-c serían intratables. Es decir, dado un problema Π en NP-c, se cumple para Π el resultado antes enunciado: $P \neq NP \rightarrow \Pi \in NP \setminus P$, y la cuestión central, antes planteada, de decidir si es cierto que $P \neq NP$, es decir, decidir si existen problemas intratables en NP, podría expresarse ahora, en función de cualquier problema Π en NP-c, mediante la siguiente pregunta: ¿Es Π intratable? En caso afirmativo, también resulta afirmativa la respuesta para la cuestión anterior. Si en cambio la respuesta es no, entonces

ningún problema en NP puede ser intratable, resultando negativa la respuesta a la pregunta anterior.

De este modo, para probar que nuestro problema en su versión de decisión es intratable, asumiendo la conjetura de que existen problemas intratables, bastará con saber que el mismo es un problema NP-completo. Ante esto, surge el interrogante de cómo es posible probar que un problema determinado es NP-completo, teniendo en cuenta que para ello habría que probar que cada problema en NP puede transformarse polinomialmente en él. Luego, surge el interrogante de si existe en efecto algún problema NP-completo. La respuesta a estas preguntas viene dada por el Teorema Fundamental de Cook, en el cual se prueba que un problema de decisión propio de la lógica Booleana, conocido como Satisfiability (SAT), es NP-completo.

Una vez conocido que un problema Π_1 es NP-completo, podremos probar que otro problema Π_2 es NP-completo si logramos ver que se cumple $\Pi_1 \Theta \Pi_2$. Y es así que, a partir de la demostración de que SAT es NP-completo, se ha demostrado mediante ese recurso la pertenencia de muchos otros problemas a la clase NP-c, y a medida que esto ocurre, al incrementarse la cantidad de problemas NP-completos conocidos, aumentan las alternativas posibles para probar que nuevos problemas son NP-completos.

A continuación, presentamos el problema de decisión SAT y enunciamos sin demostración el Teorema de Cook, cuya demostración puede ser consultada en [5], [6].

4.10. Problema SAT. Teorema Fundamental de Cook

Problema SAT

Dado un conjunto $B=\{x_1,\ldots,x_n\}$ de variables booleanas, las cuales pueden tomar dos valores: V(verdadero) y F(falso), cada forma posible de asignar un valor a cada una de las variables se denomina **asignación**. Se define un operador de negación sobre las variables de B, representado por el símbolo ¬ y tal que: $\neg x_i=V$ si y solo si $x_i=F$; llamamos **literal** a cada una de las expresiones x_i y $\neg x_i$, para cada

variable x_i en B. Se define luego la operación **suma** o disyunción entre literales, simbolizada por +, tal que: $l_1+\ldots+l_k=V$ si y solo si existe j ($1\leq j\leq k$) tal que $l_j=V$. Cada expresión formada por literales ligados por la operación suma se denomina **cláusula**. Se define además una operación entre clausulas llamada **producto** o conjunción, representada por * y tal que: $C_1*C_2*\ldots*C_k=V$ si y solo si $C_j=V$, para todo j ($1\leq j\leq k$).

Una expresión booleana formada por cláusulas ligadas por la operación producto se denomina **expresión normal**, y dada una asignación A para las variables de B, le corresponderá un valor V ó F, determinado a partir de los literales y las operaciones que conformen la expresión, de acuerdo a las definiciones recién dadas. En caso de que el valor que le corresponde a una expresión normal E sea V, diremos que E se **satisface** bajo la asignación A.

Una instancia del problema SAT queda definida como sigue: Dada una expresión normal E sobre un conjunto B de variables booleanas, decidir si existe o no una asignación para las mismas tal que se satisface E.

Una instancia I de SAT será una instancia "sí", cuando exista al menos una asignación que satisfaga la expresión normal correspondiente a I, y puede verse que cualquier asignación que cumpla esto constituirá en sí misma un certificado sucinto de I. Luego, el problema SAT pertenece a NP.

Teorema Fundamental de Cook: El problema SAT es NP-Completo.

Dem.: Puede estudiarse en [5], [6].

5. Planteo matemático del problema

5.1. Parámetros del problema

Una lista de pedidos P contiene esencialmente la siguiente información: un código para cada pedido, en el que se especifican sus características técnicas (primarias y secundarias); las medidas (en

mms.), o sea ancho, largo y espesor, y las demandas (en tons.) para cada pedido.

A partir de dicha lista, generamos una matriz que contenga la información relevante para el problema. Esta matriz, que llamamos M, es una matriz de n filas y 4 columnas, donde n es la cantidad de pedidos. En la fila i ($1 \leq i \leq n$), se encuentra la información referente a un pedido p_i, expresada de la siguiente forma: el coeficiente M_{i1} indica el número de cluster al que pertenece p_i. Para poder determinar el número de cluster correspondiente a cada pedido, es necesario, previamente, agrupar los pedidos en clusters a partir de la identificación, mediante sus códigos de características técnicas, de aquellos pedidos que tienen iguales características técnicas primarias y un mismo espesor. A los q clusters resultantes de esto los denominamos C_1, C_2, …, C_q, y asignamos las filas de la matriz M a los pedidos de la lista de forma tal que en las primeras filas aparezcan los pedidos del cluster C_1, luego los del cluster C_2, y así sucesivamente. Los coeficientes M_{i2} y M_{i3} indican, respectivamente, el ancho y el largo del pedido p_i. El coeficiente M_{i4} indica la cantidad de toneladas demandadas para p_i y será siempre un valor racional positivo.

Al agrupar los pedidos en clusters, previamente al armado de la matriz M, es común que aparezcan varios clusters compuestos por un único pedido ("clusters unitarios"). Para cada uno de estos clusters puede asignarse, de forma trivial, un t.b.m. a partir del cual obtener el único pedido que lo compone con desperdicio 0. Para ello, basta con seleccionar un t.b.m. cuyo ancho sea múltiplo del ancho o el largo del pedido.

Con el objetivo de disminuir el tamaño de las instancias del problema y aumentar la eficacia de los algoritmos implementados, los clusters unitarios son apartados de la lista original de pedidos, pudiendo resolverse por separado, de forma sencilla según lo descripto, el problema de asignar t.b.m.'s a los mismos.

Es decir, dada una lista de pedidos determinada, consideramos el problema de seleccionar, de forma óptima, t.b.m.'s a partir de los cuales cubrir los pedidos correspondientes a los clusters no unitarios de la lista, los cuales serán representados en la matriz M.

$M =$

Número de cluster	Ancho (mms.)	Largo (mms.)	Demanda (tons.)
1	900	1350	14
1	750	2000	21.2
2	1130	1250	32.4
2	1200	1650	48.1
2	610	830	21
3	1000	3000	102
3	1100	1900	14.5
3	950	1100	31.2
3	1000	1520	9

Fig.5: Ejemplo de una matriz obtenida a partir de una lista de pedidos divididos en tres clusters. Las cantidades de pedidos y clusters son muy inferiores a las usuales, siendo el único objetivo de esta figura ejemplificar la estructura de una matriz de pedidos M.

Dada la matriz M, lo último que se necesita para definir una instancia del problema es especificar cuál es la cantidad m de t.b.m.'s que se seleccionarán para cubrir los pedidos, teniendo en cuenta que m no puede ser superior a la cantidad de pedidos que componen la lista, ni inferior a la cantidad de clusters distintos.

Recordando que cada t.b.m. queda determinado por su ancho, espesor y características técnicas primarias, y que cada uno de los clusters en los cuales se encuentran separados los pedidos de una lista contiene a todos los pedidos que comparten un mismo espesor y un mismo conjunto de características técnicas primarias, una vez divididos los pedidos de la lista en clusters, podemos pensar que cada t.b.m. a considerar queda determinado conociendo su ancho y el cluster con el cual es compatible, o sea, con cuyos pedidos tiene coincidencia en el espesor y características técnicas primarias. De este modo, podemos

especificar cada t.b.m. simplemente a partir de su ancho, siempre y cuando sea claro que el mismo es compatible a un cluster determinado.

Una solución factible s consiste de una selección de m t.b.m.'s con los cuales se pueden cubrir los pedidos especificados en M, recordando que los anchos de dichos t.b.m.'s tienen que respetar el rango especificado, y teniendo en cuenta que, para cada cluster, debe haber al menos un t.b.m. compatible cuyo ancho no sea inferior al mayor de los anchos entre los pedidos que lo componen. De otra forma, el pedido que verifica el máximo ancho no podría cubrirse a partir de esa selección de t.b.m.'s. Cada pedido se obtendrá a partir del t.b.m. compatible seleccionado que implique un menor desperdicio de acero, teniendo en cuenta las dos formas en que el pedido puede ser dispuesto sobre la bobina, siempre que sea posible.

El costo que asociaremos a cada solución estará dado por la relación porcentual entre el desperdicio total de acero (la suma de los desperdicios correspondientes a la obtención de cada pedido) y la cantidad total de acero aprovechada para la obtención de los pedidos (la cual es igual a la suma de las demandas de los pedidos que componen la lista). Entonces, si C(s) es el costo asociado a s, tenemos que: $C(s)=(D/T)\times 100$, donde D es el desperdicio de acero (en tons.) y T el aprovechamiento.

Observemos que, dado un pedido p, no es posible que se genere en su obtención un desperdicio superior al aprovechamiento, ya que, para que esto ocurriera, se deberían disponer los ítems de p sobre la bobina correspondiente de forma tal que la dimensión del pedido con respecto a la cual se realice el flejado (ancho o largo) ocupe menos de la mitad del ancho de la bobina, pero en este caso, podría utilizarse parte del ancho no aprovechado para disponer otro fleje del pedido y no sería necesario su descarte. De este modo, el costo asociado a una solución nunca será superior a 100.

Como ya se ha dicho, la función de costo utilizada tiene la ventaja de permitirnos apreciar, de forma más directa, la eficacia de los algoritmos que implementemos para resolver el problema, pudiendo comparar los resultados obtenidos sobre diferentes instancias del problema (entre las cuales la cantidad total de toneladas demandadas podría variar mucho) mejor que en el caso en que la función de costo

da como resultado la cantidad total de toneladas de acero desperdiciadas.

Las siguientes observaciones nos conducen a una importante reducción del conjunto de soluciones factibles para cada instancia del problema:

-En primer lugar, notemos que, dado un cluster C, no tiene sentido considerar para el mismo t.b.m.'s cuyos anchos sean inferiores al menor de los anchos de pedidos que integran C.

-En segundo lugar, supongamos que tenemos una selección factible de t.b.m.'s con los cuales cubrir los pedidos, y consideremos, dentro de la misma, un t.b.m. b en particular, de ancho A. El t.b.m. b será compatible con un cluster determinado C_i y habrá un subconjunto no vacío C_{ib} de pedidos de C_i (respetando la condición impuesta por la primera observación) que pueden obtenerse a partir de b. Consideremos un pedido $p \epsilon C_{ib}$, para el cual el ancho desperdiciado a_d de una bobina madre de tipo b, en su obtención, es mínimo sobre los pedidos de C_{ib}, y supongamos que $a_d>0$. Notemos ahora, que si redujéramos el ancho A de la bobina restándole a_d (o reduciéndolo hasta 750mms. en caso de que esta última operación nos diera un ancho inferior a 750mms.), obtendríamos una bobina madre tal que a partir de ella sería posible cubrir los mismos pedidos que con la bobina original, pero con un desperdicio inferior en cada caso. A partir de esto, podemos concluir que no es necesario considerar t.b.m.'s cuyos anchos no son múltiplos de alguna de las medidas de los pedidos que integran el cluster con el cual son compatibles o iguales a 750mms.

De este modo, hemos logrado reducir el espacio de soluciones factibles de una forma que será decisiva para la aplicación de nuestras técnicas de resolución (a partir de aquí, solo consideraremos como soluciones factibles a aquellas que estén compuestas por t.b.m.'s que respeten las observaciones anteriores), pues dada una matriz de pedidos, podremos listar, sin demasiado "esfuerzo" computacional, los anchos de t.b.m.'s posibles para cada cluster C, simplemente considerando el ancho mínimo (750mms.), en caso de que éste no sea inferior al menor ancho para un pedido en C, y los múltiplos de las medidas de los pedidos de C, que estén dentro del rango 750-1650 mms.

Asociaremos a cada cluster C_i el conjunto AC_i, formado por todos los anchos a ser considerados, teniendo en cuenta las observaciones anteriores, para t.b.m.'s compatibles con dicho cluster, y dentro de este conjunto destacaremos un subconjunto ACP_i formado por todos aquellos anchos a partir de los cuales podría obtenerse cualquiera de los pedidos que componen C_i, es decir, aquellos anchos de t.b.m.'s que son superiores o iguales a los anchos de todos los pedidos de C_i. A los anchos contenidos en ACP_i los llamaremos "anchos padres", y para comprender la importancia de los mismos, debe recordarse que toda selección de t.b.m.'s debe contener al menos un t.b.m. con un ancho de este tipo para cada cluster.

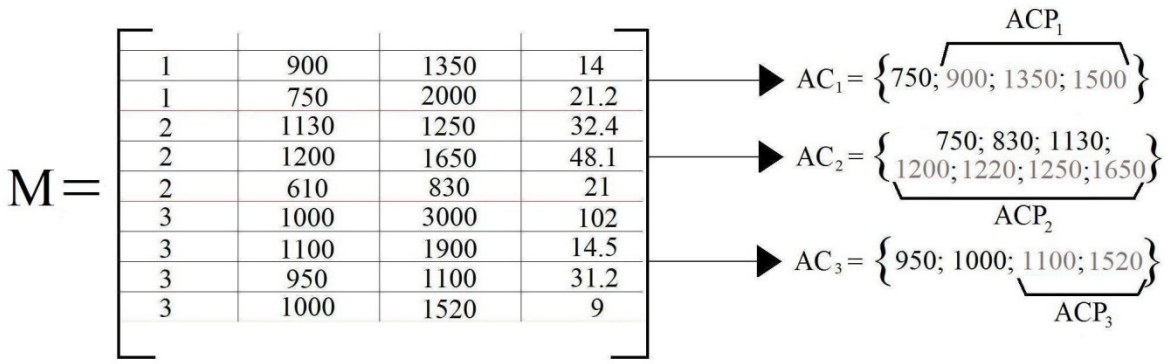


Fig.6: Determinación de los conjuntos de anchos de t.b.m.'s candidatos para cada cluster de la matriz M de la Figura 5. En cada conjunto AC_i se especifican en tono más claro los correspondientes anchos padres.

De este modo, dada una matriz M de n pedidos, los cuales se dividen en q clusters ($q \leq n$), y siendo m la cantidad de t.b.m.'s a seleccionar para cubrir los pedidos ($q \leq m \leq n$), una vez determinados cuáles son los anchos de t.b.m.'s a considerar como opción para cada cluster, una solución puede construirse comenzando por seleccionar un ancho padre para cada uno de los q clusters y luego seleccionando m-q anchos de t.b.m.'s entre los anchos candidatos restantes de todos los clusters (incluyendo los anchos padres que no hayan sido seleccionados en la primera etapa). La primera etapa de la selección

tiene como objetivo garantizar la factibilidad de la solución obtenida, ya que al realizar dicha selección de anchos padres, habrá sido seleccionado para cada cluster al menos un t.b.m. compatible con todos los pedidos del mismo. Mediante este proceso de construcción de soluciones, puede generarse cada una de las soluciones factibles para la instancia del problema definida a partir de M.

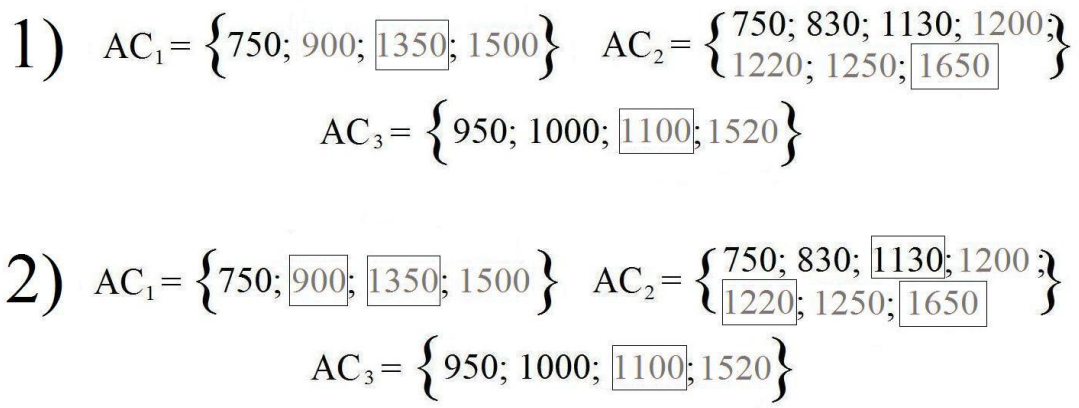


Fig.7: Construcción de una selección factible de 6 t.b.m.'s para cubrir la lista de pedidos representada en la matriz M de la Figura 5. En la primer etapa se selecciona un ancho padre de t.b.m. para cada uno de los tres clusters; en la segunda etapa se seleccionan tres anchos de t.b.m.'s entre todos los anchos no seleccionados en la primer etapa.

Cluster 1

Pedidos \ Anchos selec.			900		1350	
a	l	t	Desperdicio		Desperdicio	
900	1350	14	$D_1 = 0$	—	$D_1 = 7$	$D_2 = 0$
750	2000	21.2	$D_1 = 4.24$	—	$D_1 = 16.96$	—

Cluster 2

Pedidos \ Anchos selec.			1130		1220		1650	
a	l	t	Desperdicio		Desperdicio		Desperdicio	
1130	1250	32.4	$D_1 = 0$	—	$D_1 = 2.58$	—	$D_1 = 14.91$	$D_2 = 10.37$
1200	1650	48.1	—	—	$D_1 = 0.80$	—	$D_1 = 18.04$	$D_2 = 0$
610	830	21	$D_1 = 17.90$	$D_2 = 7.59$	$D_1 = 0$	$D_2 = 9.87$	$D_1 = 7.40$	$D_2 = 20.75$

Cluster 3

Pedidos \ Anchos selec.			1100	
a	l	t	Desperdicio	
1100	1900	14.5	$D_1 = 0$	—
950	1100	31.2	$D_1 = 4.93$	$D_2 = 0$
1000	1520	9	$D_1 = 0.9$	—

Fig.8: Dada la solución de la Figura 7, en estas tablas se especifica para cada cluster, para cada uno de sus pedidos, cuál es el desperdicio asociado a su obtención a partir de cada uno de los t.b.m.'s compatibles seleccionados. Primero, usando como primer medida de corte el ancho del pedido (Fig.2.1), luego, usando como tal al largo del pedido (Fig. 2.2). Un guión indica que la orientación correspondiente no es factible. Se distinguen en tono más claro los desperdicios mínimos para cada pedido, a partir de los cuales se determina qué t.b.m. y qué patrón de corte se utilizará para cada pedido. El costo asociado a esta solución es aprox. 2.69, es decir, se desperdicia una cantidad de acero igual al 2.69% del acero aprovechado. Para determinarlo, se suman los desperdicios mínimos para cada pedido (en el ejemplo: 0+4.24+0+0+0+0+0+0.9=5.14); esa cantidad es dividida por el total de toneladas producidas (en el ejemplo:

14+21.2+32.4+48.1+21+14.5+31.2+9 = 191.4), y el resultado es multiplicado por 100 ((5.14/191.4)x100~=2.69).

5.2. Codificación de soluciones

Definiremos ahora una forma de representación de soluciones, que nos permita operar con las mismas de manera más simple. Para esto, será esencial que dicha representación, además de expresar de forma unívoca las características específicas de cada solución, lo haga de una forma concisa y fácilmente decodificable.

De aquí en adelante, utilizaremos la siguiente forma de representación para las soluciones de una instancia dada:

Dada una instancia definida por una matriz M de n pedidos, divididos en q clusters, para la cual se deben seleccionar m t.b.m.'s distintos a partir de los cuales cubrir los pedidos, sean AC_1,…,AC_q los conjuntos de anchos de t.b.m.'s a ser considerados para cada cluster, a los cuales llamaremos también "anchos candidatos". Cada uno de estos conjuntos contiene a su correspondiente subconjunto de anchos padres.

Sean w_1,…,w_q los cardinales de AC_1,…,AC_q, respectivamente. Luego, consideramos un conjunto de vectores de k componentes, con $k=w_1+…+w_q$, o sea una componente por cada ancho candidato, de forma tal que las primeras w_1 componentes correspondan a los anchos candidatos del primer cluster, ordenados de menor a mayor; las siguientes w_2 componentes correspondan a los anchos candidatos del segundo cluster, ordenados de menor a mayor; y así sucesivamente. Además, consideramos que el valor correspondiente a cada componente solo puede ser 1 ó 0, de forma tal que un 1 en una componente determinada representará la selección del t.b.m. con el ancho correspondiente, mientras que el valor 0 representará lo contrario.

Las soluciones para la instancia, en la cual, recordemos, deben seleccionarse m t.b.m.'s, estarán representadas por aquellos vectores, pertenecientes al conjunto recién definido, que poseen exactamente m componentes iguales a 1 (las restantes son iguales a 0) y que cumplen

además que, para cada cluster, hay al menos una componente, entre aquellas correspondientes a sus anchos padres (las cuales se encuentran al final de cada grupo de componentes correspondientes a un cluster, dado el orden creciente usado para la asignación), que es igual a 1.

Para simplificar, nos referiremos a cada grupo de componentes correspondientes a los anchos candidatos de t.b.m.'s para un cluster de pedidos, como un "cluster de componentes". Además, denominaremos a cada uno de los dos subconjuntos en los que se divide un cluster de componentes, a saber, el correspondiente a los anchos padres y el correspondiente a anchos no padres, como un "subcluster de componentes", distinguiendo entre los dos tipos de subclusters mediante las denominaciones: "subcluster de componentes padres" y "subcluster de componentes no padres".

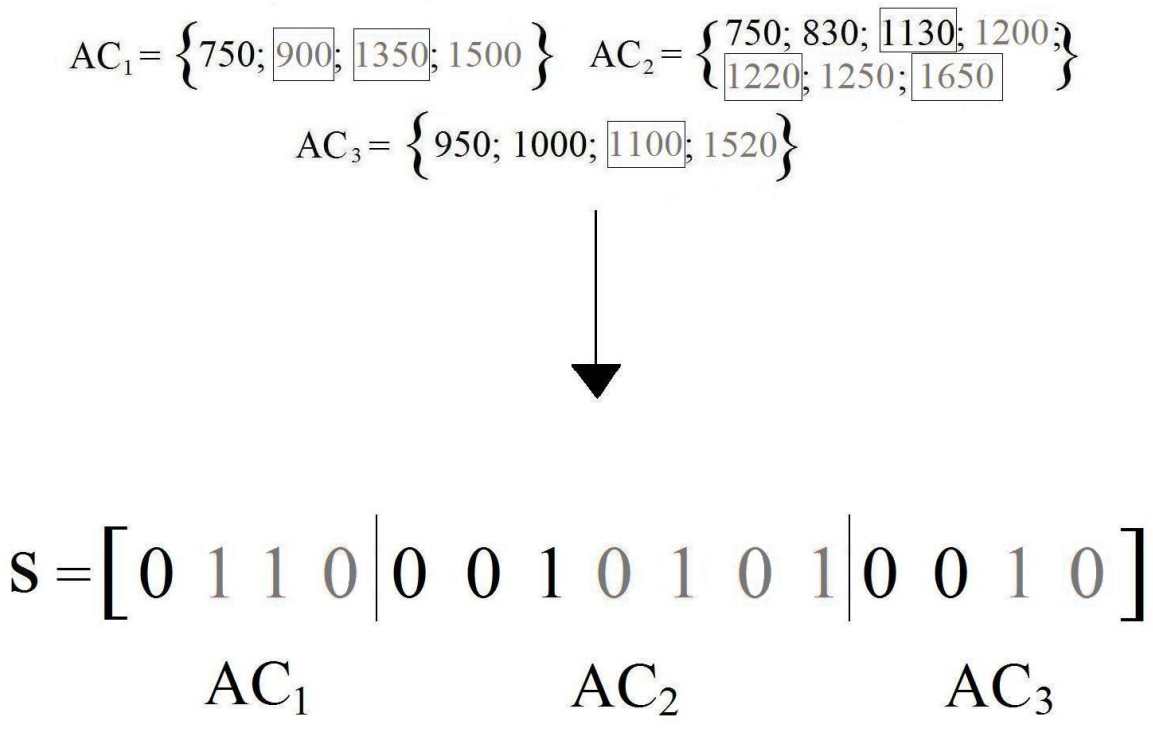


Fig.9: Representación vectorial de la solución de la Figura 7.

De este modo, el problema de determinar, para una matriz M de pedidos, una selección de m t.b.m.'s distintos, a partir de los cuales cubrir todos los pedidos con un mínimo costo asociado, se puede pensar como el problema de determinar, entre todos aquellos vectores que codifican soluciones factibles, de acuerdo al criterio de codificación recién expuesto, un vector cuyo costo asociado sea mínimo, siendo el costo de cada vector igual al costo de la solución representada.

5.3. Clasificación del problema

Nuestro problema, como ya hemos dicho, constituye un caso particular del problema conocido como Cutting Stock Problem (CSP), el cual fue planteado por primera vez por Kantorovich en 1939, siendo publicado recién en 1960 en [9]. A partir de entonces, ha sido formulado, a través de los años, de numerosas y variadas formas, llegando a ser uno de los problemas matemáticos vinculados a la industria más estudiados. En 1990, Dyckhoff presentó en [1] una clasificación de las diferentes variantes del CSP, basada en la identificación de ciertos elementos fundamentales en el planteo del CSP, los cuales juegan un papel central en la generación de variantes. Dichos elementos se detallan a continuación:

1) La dimensión del problema, esto es, la cantidad de dimensiones involucradas en la elaboración de los patrones de corte, siendo las alternativas más comunes las de 1 y 2 dimensiones.

2) Restricciones en cuanto a la asignación de materia prima a los pedidos, tales como el requerimiento de cubrir todos los pedidos o el de usar todo la materia prima en stock.

3) El surtido de los objetos grandes, denominándose así a las unidades de materia prima (en nuestro caso las bobinas madres de acero). Esto está relacionado con la cantidad de unidades y la variedad de medidas presentes en los objetos grandes.

4) El surtido de los ítems pequeños, denominándose así a los productos finales (en nuestro caso las chapas pedidas). Esto está relacionado con la cantidad de unidades y la variedad de medidas presentes en los ítems pequeños.

Dyckhoff elaboró una tipología, asignando un símbolo a cada una de las categorías posibles que surgen a partir de los cuatro factores enunciados, correspondiendo a nuestro problema el código: 2-V-D-M.

El 2 en la codificación se debe al hecho de que nuestro problema es bidimensional, ya que existen dos dimensiones relevantes en la elaboración de los patrones de corte; V representa el hecho de que, con respecto a las restricciones sobre la asignación de materia prima a los pedidos, en nuestro problema deben ser cubiertos todos los pedidos; D significa que, en cuanto a la variedad de objetos grandes, en nuestro problema existen muchos de ellos y de medidas (ancho) variables; y por último, M representa el hecho de que, con respecto al surtido o la variedad de pedidos, en nuestro problema estos son muchos y de dimensiones muy variables.

Por otro lado, otros aspectos destacables en la formulación de nuestro problema están en el hecho de que los cortes sean de tipo guillotina y ortogonal, significando esto que cada corte debe ser paralelo a uno de los lados de la pieza a cortar y debe ser completo.

Dado que puede probarse que aún los problemas más sencillos de esta clasificación son, en su versión de decisión, NP-completos, según puede verse en [10] y [11], se deduce que en particular nuestro problema, en su versión de decisión, es NP-completo. Por lo tanto, nuestro problema de Optimización Combinatoria, no más sencillo que el problema de decisión asociado, resulta también intratable algorítmicamente.

Aquellos problemas Π, sean o no problemas de decisión, tales que pueden ser considerados al menos tan difíciles como un problema de decisión Π' que sea NP-completo, de modo tal que de existir un algoritmo de tiempo polinomial que resuelva Π entonces ha de existir un algoritmo de tiempo polinomial que resuelva Π', son denominados NP-Hard.

Si bien es posible continuar la presentación teórica antes realizada mediante la formalización del concepto de problema NP-Hard y la

generalización del concepto de transformación polinomial mediante el concepto de reducción polinomial, aplicable a una clase más general de problemas que la de los problemas de decisión, incluyendo a problemas de Optimización Combinatoria, consideramos que el desarrollo teórico realizado es suficiente para formar una noción del grado de dificultad de nuestro problema, la cual justifique el tratamiento que a continuación comenzaremos a dar al mismo. Quien desee estudiar el concepto de problema NP-Hard puede hacerlo en [5].

Parte II

6. Técnicas Heurísticas

Una vez conocido que nuestro problema es NP-Hard, y por lo tanto intratable bajo el supuesto de que P es distinto de NP, surge la alternativa de resignar la posibilidad de hallar una solución óptima y abocarnos a la elaboración de algoritmos capaces de hallar soluciones al menos "aproximadas", en el sentido de que su costo no difiera mucho del costo asociado a una solución óptima, y que tales algoritmos, como contrapartida de su inexactitud, trabajen en tiempo polinomial.

Existen muchos problemas NP-Hard para los cuales se han desarrollado algoritmos de aproximación, muchas veces con considerable éxito. En algunos casos, es posible determinar el grado de aproximación que habrán de tener las soluciones resultantes. Pero esto es, en general, muy difícil de lograr, debiendo conformarnos con la suposición, o "esperanza", de que los mismos conducirán a soluciones con un "buen" grado de aproximación.

Por otro lado, si bien no es posible, en la mayoría de los casos, determinar el grado de aproximación de las soluciones, siempre es posible comparar la calidad de las soluciones obtenidas mediante diferentes algoritmos de aproximación aplicados al mismo problema.

De forma tal que, a medida que la cantidad de los mismos crezca y los costos de las soluciones resultantes mejoren, la comparación de los resultados de nuevos algoritmos con los resultados de los ya existentes para el mismo problema constituirá un criterio cada vez más fuerte para determinar su eficacia. Además, en el caso de problemas reales, por ejemplo industriales, es posible comparar la calidad de las soluciones obtenidas para el problema por medio de algoritmos de aproximación con la de las soluciones implementadas hasta el momento por la empresa. Un algoritmo que logre mejorar considerablemente el costo asociado a la solución actual difícilmente no será un buen algoritmo para el problema.

Por último, es posible, en muchas ocasiones, determinar al menos cotas para el costo de una solución óptima, pudiéndose tal vez medir la calidad de las soluciones aproximadas que se obtengan, a partir de la comparación con dicha cota.

Las técnicas involucradas en la elaboración de algoritmos cuya eficacia *a priori* está basada únicamente en la suposición o "esperanza", por así decirlo, de que los mismos pueden conducir a una solución aproximada, basándose en el hecho de que el criterio utilizado en su elaboración proviene de alguna experiencia o captura algún aspecto fundamental del problema, son conocidas como **técnicas heurísticas**.

Existen varios ejemplos de técnicas de este tipo. Un estudio detallado de varias de ellas puede realizarse en [12].

En este trabajo realizamos dos tratamientos heurísticos del problema abordado.

Otros abordajes del CSP en la industria siderúrgica pueden encontrarse en [13] y en [14], mientras que un abordaje del CSP en otro contexto (sobre madera) es llevado a cabo en [15].

7. Algoritmos de Búsqueda Local

7.1. Introducción

Nuestro primer abordaje puede incluirse en el conjunto de técnicas conocidas como **búsqueda local**. Estas técnicas se desarrollan en torno a un concepto fundamental, que es el de cercanía o "vecindad" entre soluciones. Esto es, dada una instancia de un problema de Optimización Combinatoria, con sus correspondientes conjunto o espacio de soluciones S y función de costo C sobre S, definimos un criterio de cercanía o vecindad entre soluciones, usualmente basado en la forma de representación matemática que hayamos dado a las mismas, y a partir del mismo definimos una función E:S→Partes(S), comúnmente denominada **función de entorno** o **función de vecindad**, que asocia a cada solución s del conjunto S, un subconjunto E(s) de S, formado por todas las soluciones que son vecinas de s, al cual llamaremos **vecindad** o **entorno** de s.

Una vez dada de este modo, mediante el criterio de vecindad empleado, una estructura al espacio de soluciones S, lo que sigue es la elaboración de algoritmos que exploren S en busca de "buenas" soluciones para el problema, es decir, soluciones cuyo costo asociado sea cercano al óptimo, aprovechando para ello dicha estructura, la cual guiará nuestros "desplazamientos" en S de la forma más eficiente posible.

De este modo, los algoritmos de búsqueda local constituyen una alternativa racional a la computacionalmente impracticable exploración exhaustiva del espacio de soluciones.

Para empezar, podríamos decir que el comportamiento general de un algoritmo de búsqueda local queda descripto del siguiente modo:

- Se comienza por considerar una solución inicial s_0, perteneciente al espacio de soluciones S.

- Se explora la vecindad de s_0 en busca de una solución s_1 tal que $C(s_1)<C(s_0)$. Una vez obtenida s_1, se pasa a explorar su vecindad en busca de una solución s_2 tal que $C(s_2)<C(s_1)$, y así sucesivamente. (Luego veremos que existe una clase de algoritmos de búsqueda local que presentan una diferencia importante en este punto).

- El algoritmo se detiene al alcanzar una solución s_n para la cual no exista ninguna solución en su vecindad con un costo inferior. En este caso, se dice que se ha alcanzado un **mínimo local** para la función de costo en dicha solución s_n. Dado que la exploración, tal como ha sido definida, no puede continuar generando progresos, s_n es presentada como la solución aproximada determinada por el algoritmo de búsqueda.

Debe observarse que un mínimo local no tiene por qué ser una solución óptima ni estar cerca de una, en términos de costo. El principal desafío en la elaboración de un algoritmo de búsqueda local consiste en definir un criterio de vecindad y diseñar una técnica de exploración, tales que el algoritmo no conduzca a mínimos locales de baja calidad, o sea, cuyo costo se encuentre alejado del costo de una solución óptima. Los múltiples esfuerzos en pos de cumplir tal objetivo central y las variadas características de los problemas abordados por esta técnica general han conducido al desarrollo de numerosas variantes en la elaboración de algoritmos de búsqueda local.

Estas variantes surgen, en primer lugar, en torno a la definición del criterio de vecindad a emplear, ya que, por lo general, existen múltiples criterios de vecindad posibles para un mismo conjunto de soluciones. En tal punto, debe tenerse en cuenta que si las vecindades resultantes del criterio elegido contienen pocas soluciones, será grande la probabilidad de caer rápidamente en mínimos locales de baja calidad. Mientras que si las vecindades resultan de gran tamaño, la exploración se tornará costosa computacionalmente.

En cuanto a cómo habrán de ser exploradas las vecindades, tenemos dos alternativas principales: Dada una solución cuya vecindad deseamos explorar en busca de una mejor solución, podemos, o bien realizar una exploración completa de la vecindad, quedándonos finalmente con la mejor solución que hayamos encontrado, o bien

explorar la misma solo hasta hallar una primera solución con un costo inferior, quedarnos con ella y pasar a explorar su vecindad, en busca de una nueva mejor solución.

También se originan variantes a partir de la elección de la solución inicial, la cual puede ser realizada simplemente al azar, o de forma tal que la misma cumpla ciertos requisitos, por ejemplo, relacionados con su costo. Así, podríamos desear que la solución inicial tuviera un cierto grado de calidad y utilizar como tal, por ejemplo, a la solución final generada por algún otro algoritmo de aproximación.

Por otro lado, pueden llevarse a cabo bifurcaciones en la exploración del espacio de soluciones, tomando diferentes caminos a partir de una solución, en un determinado punto del algoritmo, con la esperanza de minimizar así la probabilidad de que el algoritmo se detenga antes de lo deseado al alcanzar pronto un mínimo local. Dando como solución final a la mejor solución obtenida entre las diferentes vías.

Por último, se debe destacar un importante factor de variación, dado por la posibilidad de que el algoritmo de búsqueda local acepte una nueva solución sin que la misma mejore necesariamente el costo de la solución presente, aceptándose, por ejemplo en determinadas etapas del algoritmo, incrementos en el costo, a cambio de minimizar las posibilidades de "estancarse" en un mínimo local de baja calidad. Así, habiendo alcanzado un mínimo local, la aceptación, como nueva solución, de una solución vecina que implique un aumento del costo, podría permitirnos "escapar" de esa vecindad y continuar la exploración en otras vecindades, partiendo de esa nueva solución, las cuales conduzcan, tal vez, a mejores soluciones aproximadas.

A partir de aquí, denominaremos a aquellos algoritmos de búsqueda local que solo acepten, como una nueva solución, a una que mejore el costo de la solución presente, como algoritmos de búsqueda local de **mejora continua**.

7.2. Algoritmos de Búsqueda Local para el problema

En nuestro caso, recordemos que las soluciones a una instancia de nuestro problema están representadas por vectores de k componentes, con exactamente m componentes iguales a 1 y el resto iguales a 0,

donde m es la cantidad de t.b.m.'s a seleccionar y k es la cantidad total de anchos de t.b.m.'s candidatos a ser seleccionados, entre los q clusters de pedidos de la instancia. Las componentes se encuentran agrupadas, de acuerdo a un orden preestablecido a partir de una numeración de los clusters de pedidos, en lo que denominamos clusters y subclusters de componentes, y debe cumplirse que cada subcluster de componentes padres tenga al menos una de sus componentes igual a 1. El conjunto de tales vectores será nuestro conjunto S de soluciones.

El criterio de vecindad que utilizaremos en S será el siguiente: Dado un vector solución s en S, definimos como su **j-vecindad $E_j(s)$**, con $1 \leq j \leq m$, al subconjunto de S formado por todos los vectores que coinciden con s salvo por la ubicación de, a lo sumo, j 1's en sus componentes. Nos concentraremos en el caso particular en que j=1, es decir, utilizaremos principalmente 1-vecindades $E_1(s)$ (o simplemente E(s)), compuestas por todas aquellas soluciones que difieren de s en la ubicación de un 1 en sus componentes.

Dado que, en general, la cantidad de pedidos que componen una instancia da lugar a que los vectores solución tengan una importante cantidad de componentes, este criterio de vecindad dará lugar a vecindades grandes, por lo cual, el tipo de exploración que utilizamos en los algoritmos de búsqueda local desarrollados en este trabajo es aquella en la que se exploran las vecindades solo hasta hallar una primera solución con un costo inferior al de la solución presente.

Si consideramos 1-vecindades, dado un vector solución cualquiera, podemos obtener a partir de él cualquiera de los vectores solución que forman parte de su vecindad simplemente permutando la ubicación de un 1 y un 0 en sus componentes, teniendo cuidado, al realizar este cambio, de que el vector resultante siga conteniendo al menos un 1 para cada subcluster de componentes padres. Es decir, no toda permutación entre un 1 y un 0 del vector es válida para generar una solución vecina.

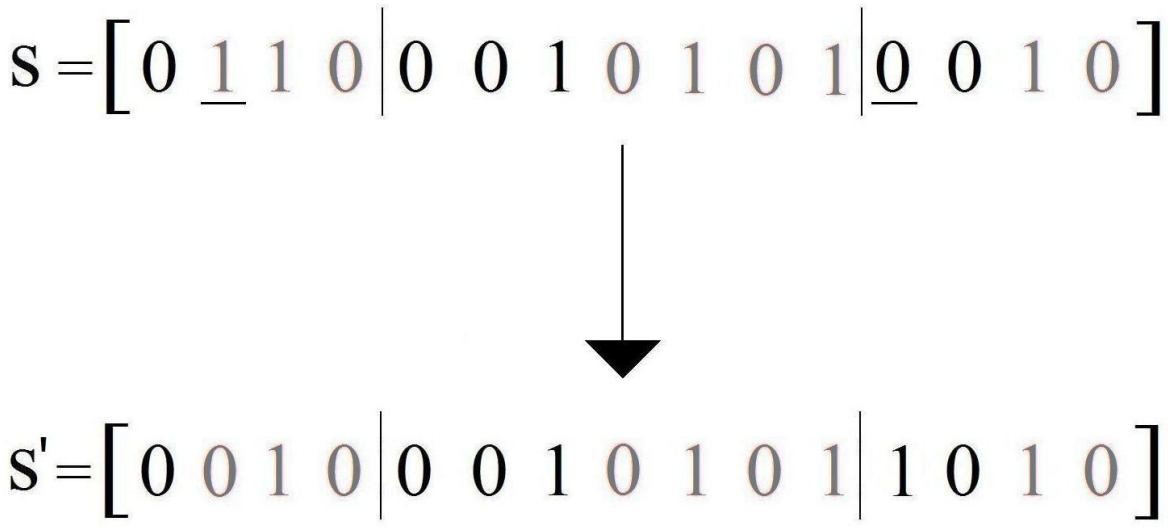


Fig.10: Los vectores s y s' representan soluciones vecinas según el criterio recién descripto.

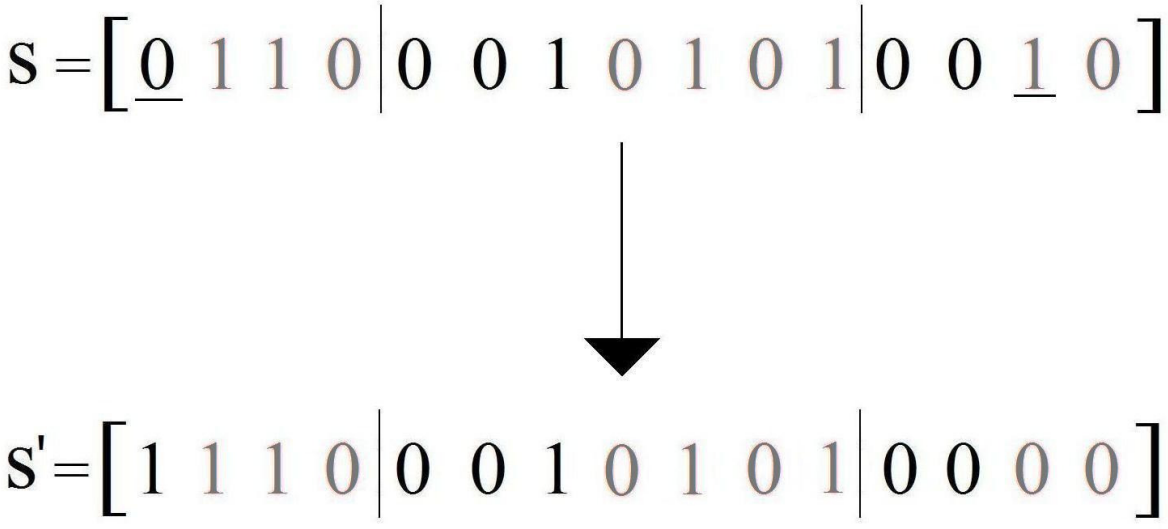


Fig. 11: Los vectores s y s' no representan soluciones vecinas, pues s' no representa una solución del problema, ya que no cumple el requerimiento de tener, al menos, una componente igual a 1 en el tercer subcluster de componentes padres.

7.2.1. Algoritmo AR

Antes de elaborar nuestro primer algoritmo de búsqueda local para el problema, empezamos por elaborar un algoritmo muy simple, de exploración random (al azar) del conjunto de soluciones. El mismo consiste en ir tomando, al azar, en forma sucesiva, una cantidad i de soluciones en S, evaluar su costo e ir quedándonos con la solución de menor costo obtenida hasta el momento.

Dada una lista de pedidos, representada por una matriz M, y siendo que se desea seleccionar una cantidad m de t.b.m.'s para cubrir tales pedidos, este algoritmo, al cual denominamos simplemente Algoritmo Random (AR), funciona del siguiente modo:

```
AR(M,m)=[solucion, costo]

solucion=sol(M,m);
costo=C(solucion);
    for k=1:i
      s=sol(M,m);
      f=C(s);
         if  f<costo
            solucion=s;
            costo=f;
         end
    end
end
```

donde i es la cantidad de iteraciones a realizar; *sol* es una función tal que, a partir de M y m, construye al azar una solución factible; mientras que la función *C* calcula el costo asociado a una solución.

sol(M,m)=solucion

solucion=*ceros*(C);
//solucion se inicializa como un vector de 0's con C componentes, donde C es la //cantidad total de anchos de t.b.m.'s candidatos para la instancia definida por M y //m.
for k=1:q
//q es la cantidad de clusters.
    z=*azar*(uc(k)-pc(k));
//pc(k) y uc(k) indican cuáles son la primer y la última componente de s, respec., //que corresponden al k-ésimo subcluster de componentes padres. *azar*(j) es una //función que determina, al azar, un entero entre 0 y j.
   solucion(uc(k)+z)=1;
end
contador=0;
while contador<m-q
    r=*azar*(C-1);
    if solucion(r+1)==0
      solucion(r+1)=1;
      contador=contador+1;
    end
end
end

La finalidad de elaborar AR es que el mismo nos dé un punto de referencia para medir la calidad de las soluciones que obtengamos con nuestros algoritmos heurísticos. De algún modo, el resultado obtenido con este algoritmo constituye una cota inferior para la calidad de las soluciones a obtener por algoritmos heurísticos, pues ningún algoritmo bien elaborado debería ser de calidad inferior a éste. Otro punto de referencia, tal vez el más importante, estará dado por el conocimiento de cuáles son las cantidades de t.b.m.'s que se utilizan para cubrir los pedidos en la práctica, así como el desperdicio de acero resultante.

7.2.2. Algoritmo BL_1

Nuestro primer algoritmo de búsqueda utilizará 1-vecindades y será de mejora continua. Dado que las vecindades que se utilizan contienen grandes cantidades de soluciones, la exploración exhaustiva de las mismas resultaría muy costosa, por lo cual, cada vecindad es explorada solo hasta hallar una primera solución que mejore el costo.

Este criterio de exploración tiene un inconveniente. El mismo radica en que no se puede garantizar, *a priori,* que en la vecindad de una solución determinada hallaremos otra solución que mejore el costo, y ocurriría que, al alcanzar un mínimo local, deberíamos realizar una exploración completa de su vecindad, hasta poder concluir que, en efecto, es un mínimo local. Pero dicha exploración completa, como ya se ha dicho, sería muy costosa.

La primera alternativa que seguimos, ante esta dificultad, es la de fijar, antes de aplicar el algoritmo, la cantidad de evaluaciones de soluciones a realizar. Es decir, habiendo realizado, entre las sucesivas exploraciones de vecindades que ocurran durante la ejecución del algoritmo, una cantidad preestablecida i de evaluaciones de vectores solución, el algoritmo se detiene. A cada una de estas evaluaciones de vectores solución, seguida de la comparación de su costo con el costo de la solución presente, la consideramos una iteración del algoritmo.

Al fijar una cantidad de iteraciones, se evita, por un lado, que el algoritmo se "estanque" explorando la vecindad de un mínimo local o una vecindad en la cual existan soluciones que mejoren el costo, pero la probabilidad de hallarlas sea muy baja debido a su escasa proporción con respecto a la cantidad de soluciones que componen la vecindad, tornando muy costosa la exploración.

Por otro lado, puede suceder que, al detenerse el algoritmo habiendo cumplido dicha cantidad fija de iteraciones, hayamos alcanzado una solución a partir de la cual podríamos progresar si el algoritmo continuara. Es decir, podría ocurrir que el algoritmo se detenga antes de lo necesario, con un consecuente deterioro del costo de la solución final. Ante esto, tenemos la alternativa de volver a aplicar el algoritmo, utilizando como solución inicial la solución final obtenida en la aplicación anterior del mismo. En función de esta idea, podríamos, por ejemplo, llevar, durante la ejecución del algoritmo, un

registro de la cantidad de iteraciones transcurridas sin obtener una solución que mejore el costo, y a partir del mismo, decidir si es conveniente repetir su aplicación o no.

Dada una matriz M representando una lista de pedidos, y una cantidad m de t.b.m.'s a ser seleccionados para cubrir tales pedidos, el primer algoritmo de búsqueda local que utilizamos, al cual denominamos simplemente Algoritmo de Búsqueda Local 1 (BL_1), trabaja del siguiente modo:

Partiendo de una solución inicial generada al azar, se explora su vecindad hasta alcanzar una nueva solución con un costo inferior. Luego, la exploración continuará en la vecindad de esta nueva solución, y así sucesivamente. Esto hasta que se cumpla una cantidad preestablecida de iteraciones.

```
BL_1(M,m)=[solucion, costo]

solucion=sol(M,m);
costo=C(solucion);
    for k=1:i
      s=sol(M,m);
      f=C(s);
         if  f<costo
            solucion=s;
            costo=f;
         end
    end
end
```

donde la función *vecino*, aplicada a una solución, genera al azar una solución 1-vecina de la misma.

Al final de este trabajo, se presentan los resultados de la aplicación de este algoritmo, así como de los otros algoritmos desarrollados en este trabajo, sobre una lista de pedidos.

7.2.3. Algoritmo BL_2

Una característica esencial en la elaboración de BL_1 es que el criterio de vecindad utilizado es tal que nos permite movernos a través del espacio de soluciones sin grandes restricciones. Es decir, dada una solución inicial cualquiera, se puede, en principio, alcanzar cualquier otra solución del espacio con un mejor costo, o dicho de otro modo, alcanzar y explorar cualquier región del espacio de soluciones. En cambio, el criterio de vecindad que a continuación se describe no presenta esta característica.

Definimos un nuevo criterio de vecindad, dado por una función E_p:S→Partes(S), según el cual, dos vectores solución s y s' son vecinos, si s' se obtiene a partir de s permutando los valores de las componentes correspondientes a uno de los subclusters de componentes de s. Es decir, s y s' coinciden en los valores de las componentes correspondientes a cada uno de sus subclusters a excepción de uno, para el cual se cumple que hay igual cantidad de 1's en ambos vectores, pero distribuidos de diferente modo.

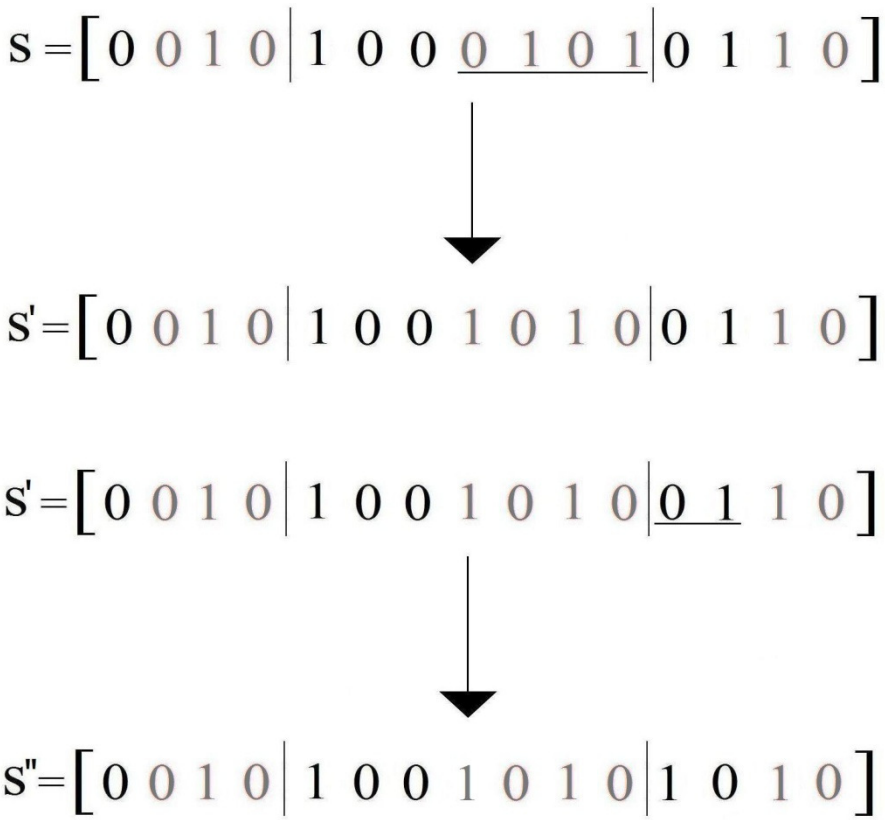


Fig. 12: La solución s' es vecina de s según Ep, pues se obtiene a partir de s permutando el segundo subcluster de componentes padres. A su vez, s'' es vecina de s', pues se obtiene a partir de ésta permutando el tercer subcluster de componentes no padres.

Este nuevo criterio de vecindad no es útil para la elaboración de un algoritmo de búsqueda local que, al igual que BL_1, parta de una solución inicial cualquiera y explore el espacio de soluciones, moviéndose a través de las vecindades definidas a partir de tal criterio, en busca de una "buena" solución final, dado que este criterio de vecindad divide al conjunto de soluciones en subconjuntos desconectados, cada uno de ellos formado por todas las soluciones que coinciden en la cantidad de componentes iguales a 1 presentes en cada cluster de componentes (es decir, coinciden en la cantidad de t.b.m.'s asignados a cada cluster), habiendo, a su vez, coincidencia en la cantidad de componentes iguales a 1 presentes en cada subcluster de componentes. Con el término desconectados se hace referencia al hecho de que, dados dos de estos subconjuntos S_1 y S_2, no existe una solución s en S_1 que sea vecina de una solución s' en S_2 según E_p.

Por otro lado, dada una solución final s_f de BL_1, la cual suponemos es una buena solución del problema, podemos pensar que, en particular, la forma en que se ha asignado la cantidad de t.b.m.'s a cada cluster es satisfactoria en sí misma, dado que ésta es una característica no definitiva, pero esencial, para la calidad de una solución, y luego, esperar que pueda existir alguna otra solución, con esa misma distribución de cantidades de t.b.m.'s para cada cluster, con un costo inferior al de s_f.

De este modo, desarrollamos un algoritmo cuyo objetivo es mejorar la calidad de las soluciones finales obtenidas por BL_1, al cual llamamos BL_2, y el cual trabaja, a partir de una solución inicial, del mismo modo que BL_1, pero empleando el nuevo criterio de vecindad definido.

Dada una matriz M, la cual representa una lista de pedidos; una cantidad m de t.b.m.'s a ser seleccionados para cubrir tales pedidos; y

siendo s la solución final obtenida tras la aplicación de BL_1 y c el costo asociado a la solución s,

```
BL_2(M,m,s,c)=[s,c]

for k=1:i
    solucion=vec_2(s);
    costo=C(solucion);
    if costo<c
       c=costo;
       s=solucion;
    end
end
```

donde la función *vec_2* (s) genera al azar una solución vecina de s, de acuerdo al criterio de vecindad dado por E_p.

Éste es un ejemplo que evidencia la flexibilidad de los algoritmos de búsqueda local, la cual nos permite, entre otras cosas, la combinación y potenciación entre diferentes algoritmos.

Dicha flexibilidad nos permitirá también, dado un algoritmo de búsqueda local como BL_1, aplicar modificaciones en su elaboración orientadas a mejorar su rendimiento, tales como las que se presentan a continuación.

7.2.4. Algoritmo BL_3

El criterio utilizado en la elaboración de algoritmos de búsqueda local de mejora continua, según el cual, una solución presente es reemplazada por otra si y solo si el costo de la segunda es inferior resulta muy natural e intuitivo y fue una característica central en la elaboración de los primeros algoritmos de búsqueda local.

Posteriormente, surgió una alternativa a este criterio, de naturaleza también intuitiva, pero más compleja, basada en la idea de que, muchas veces, tal como ocurre en situaciones reales, un retroceso con

respecto a la dirección de progreso puede ser el primer paso para un posterior avance significativo.

Siguiendo esta idea, fueron desarrollados los **algoritmos de búsqueda local de umbral,** en los cuales, para que una nueva solución sea aceptada, no siempre es una condición necesaria que la misma mejore el costo de la solución presente.

Cada uno de estos algoritmos tiene asociado un conjunto de valores reales no negativos u_j, llamados **umbrales** o **thresholds,** correspondiendo el umbral u_j a la iteración j, del siguiente modo: En la iteración j, una solución s', vecina de la solución presente s, es aceptada si y solo si $C(s')-C(s)<u_j$. (Notar que los algoritmos de mejora continua pueden pensarse como un caso particular de este tipo de algoritmos, en el cual $u_j=0$, para todo j).

De este modo, si para una iteración j ocurre que $u_j>0$, cabe la posibilidad de que, en esa iteración, se seleccione una nueva solución cuyo costo sea superior al de la solución presente. El objetivo de esto es que, mediante una selección adecuada de los valores de los umbrales, se minimice la probabilidad de que el algoritmo alcance prontamente un mínimo local de baja calidad. Observemos para entender esto que, habiéndose alcanzado un mínimo local, es decir, habiéndose alcanzado una solución tal que ninguna solución vecina de la misma tenga un costo inferior, la existencia de umbrales positivos para las siguientes iteraciones nos permitiría, tal vez, "escapar" de este mínimo local, seleccionando una nueva solución entre sus vecinas, la cual posea un costo superior, pero a partir de la cual se puedan alcanzar, posteriormente, nuevas soluciones de mejor calidad.

Una alternativa común, en la elección de los umbrales, es que los mismos conformen una sucesión de valores no negativos ($u_J\geq 0$), no creciente ($u_J\geq u_{J+1}$) y convergente a 0.

En esta alternativa de elección de los umbrales, en cada iteración se acepta o no un determinado incremento en el costo, según el valor del correspondiente umbral.

Un modelo más complejo de algoritmo de búsqueda local con umbrales es el correspondiente al **algoritmo de recocido simulado**, cuyo nombre es debido a una analogía con un proceso industrial que sirvió como inspiración para su diseño. En este algoritmo, los umbrales son variables aleatorias que siguen una determinada función de

probabilidad sobre $\mathbb{R}_+$. Aquí, cualquier incremento en el costo tiene una probabilidad positiva de ser aceptado (siendo ésta una de las características distintivas de este algoritmo de búsqueda), aunque la función de probabilidad es elegida de forma tal que haya una baja probabilidad de aceptar nuevas soluciones que impliquen grandes incrementos en el costo, mientras que la probabilidad de aceptación aumenta para soluciones que impliquen incrementos pequeños.

A continuación, se describe un nuevo algoritmo para el problema, al cual denominamos BL_3, desarrollado a partir del agregado de umbrales a BL_1.

Los valores de los umbrales son asignados durante la ejecución del algoritmo, según el estado de progreso del mismo.

A menos que indiquemos lo contrario, los umbrales en cada iteración serán iguales a 0. Cuando se alcanza una solución a partir de la cual no es posible obtener una solución vecina con un costo inferior tras una cantidad k de iteraciones (es decir, al alcanzar un posible mínimo local), se comienza a dar valores positivos a los umbrales de las siguientes iteraciones hasta lograr, eventualmente, "escapar" de esta solución. Esto se hace del siguiente modo: Habiendo alcanzado una solución s a partir de la cual no puede obtenerse una solución vecina de costo inferior tras k iteraciones, se da a los umbrales de las iteraciones siguientes valores positivos tales que se permita la aceptación de nuevas soluciones, de costo superior al de s, con la condición de que el costo de tales soluciones no supere al de s en más de una cantidad igual a un determinado porcentaje del costo de s. Esto durante una cierta cantidad r_1 de iteraciones, al cabo de las cuales se tendrá, generalmente, una solución s'_1 de costo superior al de s. Luego, a partir de s'_1 se busca, utilizando umbrales iguales a 0, durante una cantidad k_1 de iteraciones, una solución s" cuyo costo sea inferior al de s. A su vez, como esto podría no conducir a una solución mejor que s, se mantiene, durante el transcurso de todo el algoritmo, un registro de la mejor solución hallada hasta el momento (en este caso s), la cual será presentada como solución final.

Si, transcurridas k_1 iteraciones, alcanzamos una solución s" de costo inferior al de s, la registramos como la mejor solución hallada, en lugar de s, y continuamos la exploración a partir de s" con umbrales iguales

a 0, hasta que sea nuevamente necesario dar valores positivos a los umbrales.

En caso contrario, de no hallarse tal solución s" transcurridas k_1 iteraciones, volvemos a s y a la exploración de su vecindad, utilizando para las siguientes iteraciones umbrales más altos (permitiendo un incremento del costo igual a un porcentaje más elevado del costo de s), en base a la idea de que, tal vez, debemos alejarnos más de s para dar un nuevo mejor rumbo a la exploración.

Una vez concluida la aplicación de estos nuevos umbrales, transcurridas r_2 iteraciones, buscamos, a partir de la solución alcanzada s'_2, una solución s" que mejore el costo de s, del mismo modo que antes.

Este proceso se repite una cantidad determinada de veces, incrementando sucesivamente el valor de los umbrales.

La ejecución de este algoritmo, al igual que ocurría con los algoritmos anteriormente presentados, está limitada por una cantidad total de iteraciones.

Al igual que con BL_1, al finalizar la ejecución de BL_3, podemos tomar la solución final obtenida y utilizarla como solución inicial del algoritmo BL_2.

A continuación, se ejemplifica el funcionamiento de BL_3 a partir del caso más simple en el cual solo se da una vez valores positivos a los umbrales para intentar superar un mínimo local. El caso general puede deducirse a partir de este caso.

Dada una matriz de pedidos M para la cual se desean seleccionar m t.b.m.'s,

BL_3(M,m)=[s,c]

```
s=sol(M,m);
c=C(s);
contador=0;
iteraciones=0;
while iteraciones<i
    if contador<k
       iteraciones=iteraciones+1;
       solucion=vecino(s);
       costo=C(solucion);
```

```
        if costo<c
          s=solucion;
          c=costo;
          contador=0;
        end
      end
      if contador==k
        s'1=s;
        for j=1:r
            solucion=vecino (s'1);
            costo=C (solucion);
            if costo< c(1+p1) // se permite un incremento en el costo no
              s'1=solucion;    // superior al porcentaje p1 de c.
            end
        end
      c'1=C (s'1);
      iteraciones=iteraciones+r;
      if k<contador<k+k1
        iteraciones=iteraciones+1;
        solucion=vecino(s'1);
        costo=C (solucion);
        if costo<c'1
          s'1=solucion;
          c'1=costo;
          if c'1<c
            contador=0;
            c=c'1;
            s=s'1;
          end
        end
      end
      if contador≥k+k1
        contador=0;
      end
contador=contador+1;
end
end
```

Por último, agregamos que hemos realizado estudios de algoritmos de búsqueda local basados en j-vecindades, con j>1, pero no se obtuvieron resultados de calidad superior a los obtenidos a partir de trabajar con 1-vecindades.

La teoría referente a algoritmos de búsqueda local puede ser estudiada con profundidad en [16].

8. Algoritmos Genéticos

8.1. Introducción

Nuestro segundo tratamiento heurístico del problema consiste de la utilización de lo que se conoce como **algoritmos genéticos**. Estos fueron desarrollados por Holland y su grupo de trabajo, en la Universidad de Michigan, en los ’60 y ’70 [17].

Los algoritmos genéticos se basan en el establecimiento de analogías con conceptos de la Biología Evolutiva y la Genética de Poblaciones. El conjunto de soluciones del problema es considerado como el conjunto de **individuos** posibles de una **población,** y a partir del costo de cada solución, se asocia al individuo correspondiente un valor denominado **fitness,** el cual mide la adaptación del individuo a su entorno y su preponderancia dentro de la población, o sea, en términos de soluciones, lo adecuado de aplicar la solución correspondiente al problema en cuestión y la calidad relativa de la misma con respecto a las demás soluciones.

A cada individuo se asocia un objeto matemático, que codifica los parámetros de la solución correspondiente. Este es entendido como el **cromosoma** del individuo, el cual codifica las características del mismo. Dicho objeto matemático, suele ser un vector de componentes numéricas, en muchos casos tomando valores iguales a 0 ó 1.

Cada una de las unidades de codificación dentro del cromosoma del individuo, por ejemplo cada una de las componentes en el caso de un vector, se denomina **gen**. A su vez, cada valor posible de un gen es llamado **alelo,** y cada ubicación posible de un gen dentro del cromosoma es un **locus**. Las asignaciones posibles de valores a los

genes constituyen los **genotipos** posibles del individuo, y el conjunto de características codificadas por un genotipo es lo que se denomina el **fenotipo** del individuo.

A partir de estas analogías, los algoritmos genéticos trabajan, en términos generales, del siguiente modo: Dado un problema de Optimización Combinatoria, habiendo establecido, a partir del conjunto de soluciones más una representación adecuada de las mismas, el conjunto de individuos (o individuos solución) posibles, con sus correspondientes cromosomas, y habiendo elaborado una función de fitness sobre los individuos a partir de la función de costo del problema, se selecciona una población formada por una determinada cantidad de individuos, la cual, dada la gran cantidad de individuos solución que suelen haber, suele constituir una proporción muy baja de la cantidad total de individuos posibles. Luego, se procede a seleccionar sucesivamente, a partir de los fitness de los individuos de la población, una proporción de los mismos para su "reproducción" y la generación de nuevos individuos, a partir de un proceso que simulará las características principales de los mecanismos de reproducción biológica.

La reproducción es generada a través de lo que se denomina un operador de **cruza** o **crossover,** el cual segmenta los cromosomas de los individuos "padres", ensamblándolos luego, de forma entremezclada, de manera tal que se formen nuevos individuos, cuyos cromosomas tengan igual estructura que los de sus padres y contengan genes de ambos. Luego, a partir de estos nuevos individuos, se realiza una renovación de la población, la cual puede ser completa o parcial, seleccionando, por ejemplo, entre los nuevos descendientes, aquellos de mejor fitness, e incorporándolos a la población, en lugar de antiguos integrantes de bajo fitness.

El objetivo es que la selección de los individuos para la reproducción, el operador de cruza y la renovación de la población sean diseñados de forma tal que los segmentos de los cromosomas que codifican características valiosas, en términos del fitness de los individuos, se propaguen en la población, reuniéndose en los cromosomas de los individuos de las nuevas generaciones, cuyos integrantes tendrán, de este modo, valores de fitness superiores a los de sus predecesores y

representarán, por lo tanto, soluciones de mejor calidad para el problema.

Ahora, el modelo así descripto presenta una limitación, dada por el hecho de que las características valiosas que pueden propagarse y combinarse en la población están limitadas, en general, a pertenecer al conjunto de características valiosas pertenecientes al "paquete genético" de la primera generación, la cual, al estar compuesta por una pequeña proporción del total de individuos posibles, puede carecer de muchas de las características valiosas posibles. De esta forma, los mecanismos descriptos, al ser aplicados a una población en particular, nos permitirían, solamente, alcanzar algo así como un mínimo local dentro del conjunto de todos los individuos solución posibles, correspondiente al mejor individuo que puede generarse a partir del paquete genético formado por los genes contenidos en los cromosomas de la primera generación de la población, y cuyo fitness no tiene por qué ser cercano al de un individuo óptimo (entendiendo por individuo óptimo al correspondiente a una solución óptima del problema).

Para entender mejor esto último, pensemos, por ejemplo, en el siguiente hecho: Supongamos que los cromosomas de los individuos, tal como suele ocurrir, son vectores de 1's y 0's, donde cada componente es un gen, y utilizamos un operador de cruza común, el cual corta y re-ensambla los cromosomas de los individuos padres para formar nuevos individuos. Supongamos ahora, que todos los vectores de la primera generación poseen el mismo alelo para un gen determinado, o sea, el mismo valor para una misma componente en particular. Luego, toda la descendencia posible, todos los integrantes de las futuras generaciones, obtenidos a partir de cruzas, tendrán ese mismo alelo, el cual podría, tal vez, codificar una característica poco deseable. O visto de otro modo, en caso de que el alelo alternativo codificara una característica valiosa, sería imposible que dicha característica llegara a estar presente en algún individuo de la población.

Es por ello, que se implementa en los algoritmos genéticos otro operador, siempre en analogía con procesos naturales, conocido como operador de **mutación**, el cual actúa seleccionando, usualmente al azar, genes de individuos de la población y alterando con cierta

probabilidad su valor, dando así variabilidad genética a la población y permitiendo, en principio, el acceso a cualquier característica deseable. El mismo suele aplicarse luego de la cruza, sobre los descendientes generados, y la probabilidad de que cada gen en particular sea mutado suele ser baja, para no contrarrestar el efecto de la cruza al agregar una componente de azar demasiado grande en la generación de nuevos individuos, la cual atente contra la heredabilidad, elemento esencial del proceso.

8.2. Ejemplo de Algoritmo Genético

A continuación, se presenta un ejemplo de algoritmo genético. El mismo es muy simple y está basado en un problema que dista mucho de requerir un tratamiento heurístico, pero que es muy adecuado para ejemplificar, de forma sencilla, las características principales de un algoritmo genético, por lo cual, es muy común encontrar ejemplos similares a éste en la bibliografía relativa al tema. Este ejemplo está elaborado en base a uno presente en [12].

Sea la función $f(X)=X^3-57X^2+720X+2800$.

Supongamos que deseamos determinar, mediante un algoritmo genético, un máximo para f sobre los valores enteros pertenecientes al intervalo [0,31]. (Analíticamente puede determinarse que f posee un máximo en 8 y un mínimo en 30, dentro del intervalo [0,31]).

Para ello, podríamos asociar a cada entero z, en dicho intervalo, un individuo i(z), cuyo cromosoma sea un vector de 5 componentes, conteniendo la representación binaria de z, y cuyo fitness F(i(z)) sea simplemente igual a f(z). Luego, sobre el total de 32 individuos posibles, podríamos construir una población inicial de 5 individuos, y aplicar sobre ellos operadores de cruza y mutación, generando nuevos individuos, con la esperanza de que los mismos tengan un fitness superior. Esto podríamos hacerlo del siguiente modo:

-Dada la población presente de 5 individuos, se asigna a cada individuo i_j $(1\leq j\leq 5)$, a partir de su fitness, una probabilidad p_j de ser

seleccionado para formar una pareja con otro individuo de la población, la cual es igual a la razón entre su fitness y la suma de los fitness de todos los individuos de la población.

-Se asocia al individuo i_j el valor $r_j=p_1+\ldots+p_j$.

-Se forman 5 parejas $P_k=(P_{k,1},P_{k,2})$, $(P_{k,i}\in\{i_1,\ldots,i_5\}$, $P_{k,1}\neq P_{k,2})$, del siguiente modo:

-Se seleccionan al azar, utilizando una distribución de probabilidad uniforme, números z_k $(1\leq k\leq 5)$ en el intervalo [0, 1].

-Se selecciona como primer individuo de la pareja P_k al individuo i_j, con j tal que: $r_{j-1}<z_k\leq r_j$, $(r_0=0)$.

-El segundo individuo de P_k es seleccionado al azar.

-Dado que se forman 5 parejas a partir de 5 individuos, habrá individuos que formen parte de más de una pareja e incluso puede formarse una misma pareja más de una vez (lo cual significa que tal pareja generará descendencia más de una vez).

-Luego, a partir de cada pareja, se realiza una cruza obteniendo un nuevo individuo, de la forma que a continuación será detallada.

Supongamos, por ejemplo, que partimos de la siguiente población inicial:

Número de individuo	Cromosoma	Entero representado	Fitness (evaluación en f(x))	Probabilidad de ser seleccionado
1	0 0 0 1 0	2	4020	0.356
2	0 1 1 0 1	13	4724	0.418
3	1 1 0 1 1	27	370	0.033
4	1 0 1 0 1	21	2044	0.181
5	1 1 1 1 1	31	134	0.012

Luego, a partir de las probabilidades de ser seleccionado de cada individuo, formamos cinco parejas, seleccionando el primer individuo de cada pareja a partir de dichas probabilidades, y el segundo al azar. Obtuvimos lo siguiente:

Parejas (se especifican a partir del número asignado a cada individuo en la primera columna de la tabla):

$P_1=(i_2, i_1)$; $P_2=(i_2, i_3)$; $P_3=(i_2, i_3)$; $P_4=(i_3, i_5)$; $P_5=(i_1, i_2)$

Luego, se efectúa una cruza en cada pareja del siguiente modo:

-Se selecciona, al azar, un mismo punto de corte en los cromosomas de ambos padres. Por ejemplo, para la primera pareja, realizamos el corte entre el tercer y el cuarto locus, obteniéndose dos pares de segmentos de igual longitud:

i_2= 0 1 1 | 0 1 $\qquad$ i_1=0 0 0 | 1 0

0 1 1 $\qquad$ 0 1 $\qquad$ 0 0 0 $\qquad$ 1 0

Luego, unimos el primer segmento obtenido a partir del cromosoma del primer integrante de la pareja (i_2 en el ejemplo), con el segundo segmento obtenido a partir del cromosoma del segundo miembro de la pareja (i_1 en el ejemplo), obteniendo así un nuevo individuo.

h= 0 1 1 1 0

Del mismo modo, realizamos las cruzas para las restantes parejas, obteniendo un nuevo individuo por cada pareja.

Finalmente, aplicamos, sobre cada uno de los nuevos individuos, un operador de mutación consistente en alterar en cada gen, con una probabilidad igual a 0.05, el alelo presente.

Luego, los cinco individuos así generados se incorporan a la población en lugar de los cinco integrantes anteriores. Así, por ejemplo, nosotros obtuvimos la siguiente nueva población (en la cual

se aplicaron mutaciones en los individuos obtenidos a partir de la segunda y la cuarta pareja):

Número de individuo	Cromosoma	Entero representado	Fitness (evaluación en f(x))	Probabilidad de ser seleccionado
1	0 1 1 1 0	14	4452	0.189
2	0 1 0 1 0	10	5300	0.225
3	0 1 0 1 1	11	5154	0.219
4	0 1 1 1 1	15	4150	0.176
5	0 0 0 1 1	3	4474	0.190

Puede observarse cómo ha habido un aumento del fitness promedio entre la primera y la segunda generación. De todos modos, este ejemplo no está orientado a la obtención de conclusiones sobre la eficacia de los algoritmos genéticos, sino a ejemplificar la forma de proceder de los mismos. A su vez, debe tenerse en cuenta que los elementos constituyentes de este algoritmo, como los operadores de cruza y de mutación, los criterios de formación de parejas y de renovación de la población utilizados, son solo casos particulares entre los variados tipos posibles.

8.3. Algoritmos Genéticos para el problema

A continuación, se detallan los criterios empleados en la elaboración de los algoritmos genéticos que utilizamos en este trabajo.

8.3.1. Representación de soluciones. Fitness.

Anteriormente, representamos a las soluciones del problema por medio de vectores de 1's y 0's, donde cada componente del vector estaba asociada a un ancho de t.b.m. candidato a ser seleccionado para un cluster. Un 1 en una componente significaba la elección del ancho de t.b.m. correspondiente, en la solución representada por el vector; mientras que un 0 indicaba lo contrario. La forma en que se asignaban las componentes del vector a los distintos t.b.m.'s era la siguiente: las primeras componentes correspondían a los anchos de t.b.m.'s candidatos a ser seleccionados para el primer cluster, ordenados de menor a mayor; luego seguían las componentes correspondientes a los anchos candidatos para el segundo cluster, y así sucesivamente. De forma tal que en cada grupo de componentes correspondiente a un cluster determinado (o cluster de componentes), aparecían, en primer lugar, las componentes correspondientes a anchos no padres, y luego las componentes correspondientes a anchos padres, formando cada uno de estos conjuntos lo que denominábamos como un subcluster de componentes. Además, debía cumplirse que para cada subcluster de componentes padres, hubiera al menos una componente igual a 1.

Ahora, para cada solución de una instancia del problema, entenderemos que el vector de 1's y 0's asociado a la misma mediante esta forma de representación, constituye el cromosoma de un individuo que asociaremos a la solución. Cada componente del vector será un gen del cromosoma y codificará una característica del individuo, equivalente a la elección del t.b.m. correspondiente en la solución asociada. Los valores 0 y 1 serán los dos alelos posibles para cada gen. La presencia del alelo 1 indicará que el gen está "activo", es decir, la característica codificada por el gen está presente en el

fenotipo del individuo, mientras que la presencia del alelo 0 indicará lo contrario. Además, cada grupo de genes correspondiente a un subcluster de componentes padres debe contener al menos un gen activo, pudiéndose entender, profundizando así la analogía, que es necesario que uno de estos genes esté activo para la supervivencia del individuo. Llamaremos "grupo esencial de genes" a cada uno de estos grupos de genes correspondientes a un subcluster de componentes padres, y "grupo no esencial de genes" al correspondiente a un subcluster de componentes no padres.

El fitness de cada individuo será determinado a partir del costo de la solución correspondiente, del siguiente modo: Dado un individuo i, asociado a una solución s con costo C(s), el fitness del individuo i será F(i)=100-C(s), siendo, por lo tanto, un valor entre 0 y 100 (recordemos que el costo de una solución, es decir, el desperdicio relativo asociado a la misma, no puede ser superior a 100), y dependerá, por lo tanto, de cuáles sean los t.b.m.'s que compongan la solución, o sea, dependerá de qué componentes del vector asociado a la solución sean iguales a 1.

De este modo, el fitness del individuo, o sea, hablando en términos de la analogía empleada, la medida de la capacidad de adaptación del individuo a su entorno y de la preponderancia dentro de su población, estará determinado por los genes que se encuentren activos en su cromosoma, y su valor será mayor mientras menor sea el costo asociado a la solución correspondiente, de forma tal que el objetivo original de hallar soluciones con el menor costo posible será equivalente a la búsqueda de individuos con el mayor fitness posible.

8.3.2. Operadores de cruza y de mutación empleados

Habiendo definido ya la forma en que representaremos a las soluciones como individuos de una población, describimos a continuación los operadores de cruza y de mutación utilizados, así como los criterios empleados para la formación de las parejas y la renovación de la población.

Utilizaremos por separado, en diferentes algoritmos, dos operadores de cruza distintos, los cuales estarán acompañados por un mismo operador de mutación. Ambos operadores de cruza actuarán sobre una pareja de individuos, cuyos integrantes llamaremos "padres", de forma tal que se obtengan dos nuevos individuos, a los cuales llamaremos "hijos". En cada generación, las parejas se forman a partir de aquella mitad de la población formada por los individuos con mejor fitness, los cuales son agrupados, al azar, en parejas (se utilizan tamaños de población múltiplos de cuatro), obteniéndose dos descendientes por cada pareja, los cuales renovarán la población reemplazando a la mitad de la población con menor fitness.

El primer operador de cruza utilizado, al cual llamaremos "cruza simple", corresponde a uno de los tipos más comunes, en el cual se selecciona, en la estructura de los cromosomas de los individuos padres i_1 e i_2, un mismo punto de corte, separando ambos cromosomas en dos segmentos de longitudes n_1 y n_2. El punto de corte, por motivos que luego serán detallados, habrá de encontrarse siempre en el límite entre los genes correspondientes a dos subclusters de componentes consecutivos. Luego, los segmentos de longitudes n_1 y n_2 obtenidos a partir del cromosoma de i_1 se ensamblan, respectivamente, con los segmentos de longitudes n_2 y n_1 obtenidos del cromosoma de i_2.

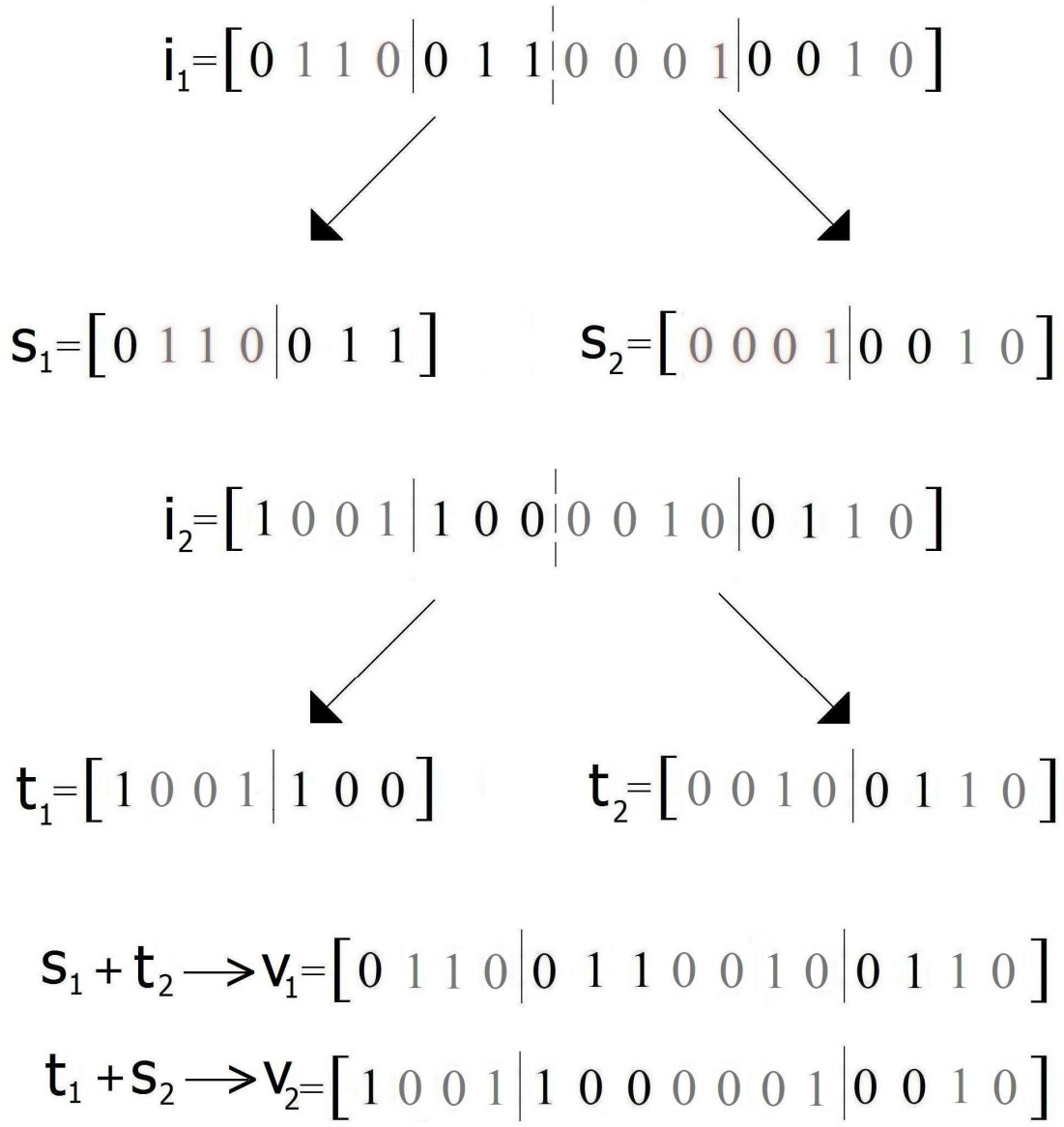


Fig. 13: En la primera etapa del proceso de cruza, los cromosomas de i_1 e i_2 son separados, a través de un mismo punto de corte, en dos pares de segmentos de igual longitud, los cuales se entremezclan formando dos nuevos cromosomas, v_1 y v_2.

Debe tenerse en cuenta que, para que los dos vectores obtenidos a partir del operador de cruza puedan ser considerados como los cromosomas de dos individuos solución, deben tener la misma cantidad de componentes iguales a 1 que los cromosomas de los individuos padres, ya que en cada instancia consideramos soluciones

formadas por una cantidad fija de t.b.m.'s. Debe cumplirse además, que en cada grupo esencial de genes haya al menos un gen activo. Es para garantizar esto último, que se agrega la condición de que el punto de corte deba hallarse siempre en el límite entre dos subclusters de componentes.

De este modo, luego del corte y ensamblado, cada uno de los cromosomas obtenidos tendrá, para cada segmento de su estructura correspondiente a un grupo esencial de genes, el conjunto de alelos de uno de sus padres, el cual contendrá, por lo tanto, al menos un alelo 1, tal como puede verse en el ejemplo de la Figura 13.

En cuanto al requerimiento de que los cromosomas de los nuevos individuos contengan la misma cantidad de genes activos que los de sus padres, debemos tener en cuenta que, al cortar y ensamblar de este modo, ocurre, generalmente, que el cromosoma correspondiente a cada hijo contiene una cantidad de genes activos mayor o menor que la requerida (esto también puede verse en el ejemplo de la Figura 13). Para corregir esto, se eliminan o agregan en los cromosomas de los hijos (al azar), según corresponda, alelos 1, hasta alcanzar la cantidad necesaria, teniendo cuidado, en el caso de tener que eliminar alelos 1, de no eliminar un alelo 1 que sea el único correspondiente a un grupo esencial de genes.

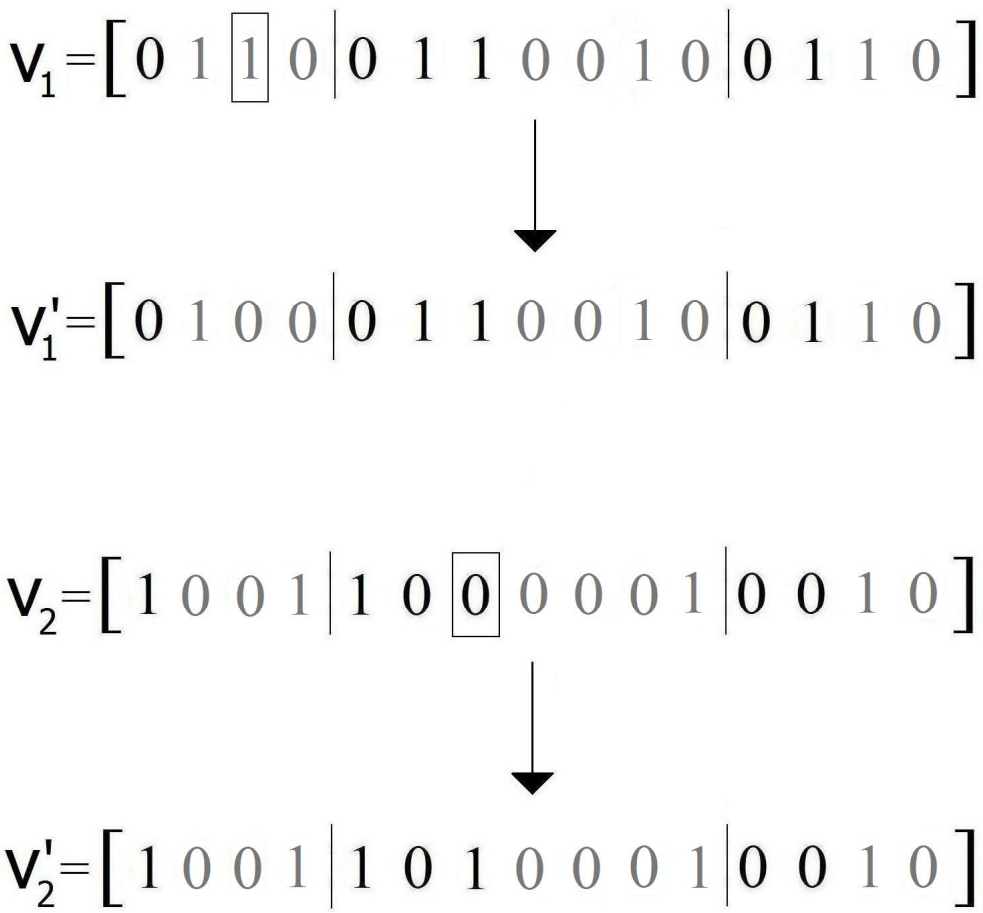


Fig. 14: En la segunda etapa del proceso de cruza, se eliminan o agregan, según corresponda, alelos 1 de los cromosomas obtenidos, hasta cumplir el requerimiento de que estos posean la misma cantidad de genes activos, o sea la misma cantidad de alelos 1, que los cromosomas padres.

La segunda etapa de este proceso de cruza genera por sí misma variabilidad genética, por ejemplo, al agregar, en uno de los cromosomas de los hijos, un alelo 1 en un locus donde no se hallaba presente este alelo en los cromosomas de ambos padres, o al eliminar un alelo 1 presente en un mismo locus de los cromosomas de ambos. Pero esto no resulta suficiente para evitar importantes limitaciones en la variabilidad genética, y por ende, una rápida convergencia del algoritmo (por convergencia se entiende el hecho de que la población llegue a estar formada únicamente por repeticiones de un mismo individuo), dado que la probabilidad de aplicación de este factor de

variabilidad se reduce al progresar el algoritmo, pues al mejorar el fitness de los individuos, acumulando estos cada vez mayor cantidad de las características buenas presentes en la población, sus cromosomas comienzan a tornarse cada vez más similares, coincidiendo, en particular, en la concentración de alelos 1 en cada sector del cromosoma, reduciéndose las probabilidades de que los nuevos individuos resultantes de las cruzas contengan más o menos alelos 1 de los requeridos. De esta forma, la presencia de este factor de variabilidad, se diluye cuando más debe estar presente un factor generador de variabilidad: en etapas avanzadas del algoritmo, cuando hay tendencia a la convergencia. Es por esto, que implementamos un operador de mutación que refuerce este proceso, cambiando al azar, luego de la cruza, la posición de un alelo 1 en el cromosoma (teniendo cuidado, al hacer el cambio, de no quitar el único alelo 1 de un grupo esencial de genes).

De este modo, las mutaciones no dependen de las diferencias en la distribución de 1's en los cromosomas de los padres, y no dejarán de aplicarse al comenzar a converger la población. (Observemos que este operador de mutación toma un vector solución y genera un vector solución vecino del mismo, de acuerdo al criterio de 1-vecindad utilizado anteriormente en la búsqueda local).

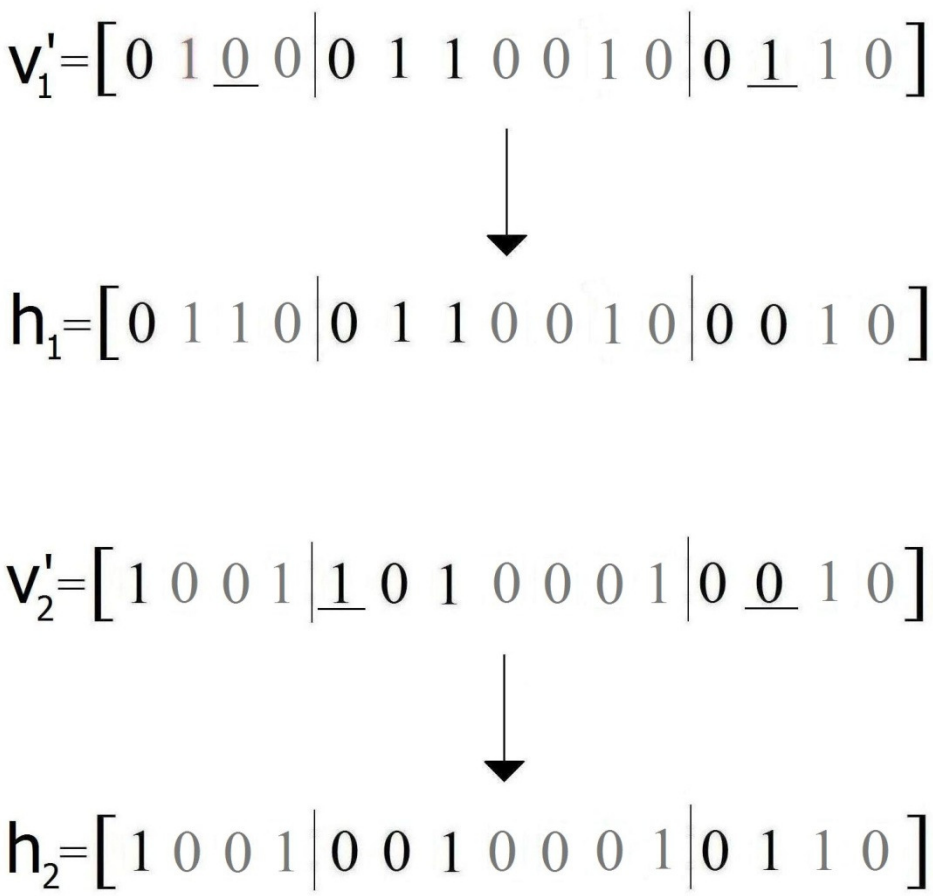


Fig. 15: Una vez terminado el proceso de cruza, los cromosomas resultantes son mutados, alterando al azar la ubicación de un alelo 1 en cada uno de ellos. Tras esto, se obtienen finalmente los dos individuos hijos.

El segundo operador de cruza, al cual denominaremos "cruza múltiple", corresponde a una complejización del primero, en la cual, en lugar de un único punto de corte, se utilizan múltiples puntos de corte en la estructura de los cromosomas. Dados los cromosomas de dos individuos padres, se dividen los mismos en múltiples segmentos, uno por cada subcluster de componentes presente en la estructura del vector solución, cortando en cada punto de los cromosomas que sea límite entre los genes asociados a las componentes de dos subclusters consecutivos. Luego, se forman dos individuos ensamblando los segmentos resultantes al azar, de forma tal que cada uno de los dos

segmentos obtenidos, correspondientes a una misma región de los cromosomas de los padres, sea asignado a un hijo diferente, mezclándose así los segmentos de ambos padres generando dos hijos, cuyos cromosomas tienen igual estructura que los cromosomas de sus padres y que verifican la condición de tener al menos un gen activo por cada grupo esencial.

Por último, se debe, al igual que con el operador de cruza simple, eliminar o agregar alelos 1 en los cromosomas obtenidos, hasta alcanzar la cantidad requerida de alelos 1, para finalmente aplicar el mismo operador de mutación utilizado en el caso anterior.

Los criterios empleados para la formación de las parejas y la renovación de la población son los mismos que en el caso anterior.

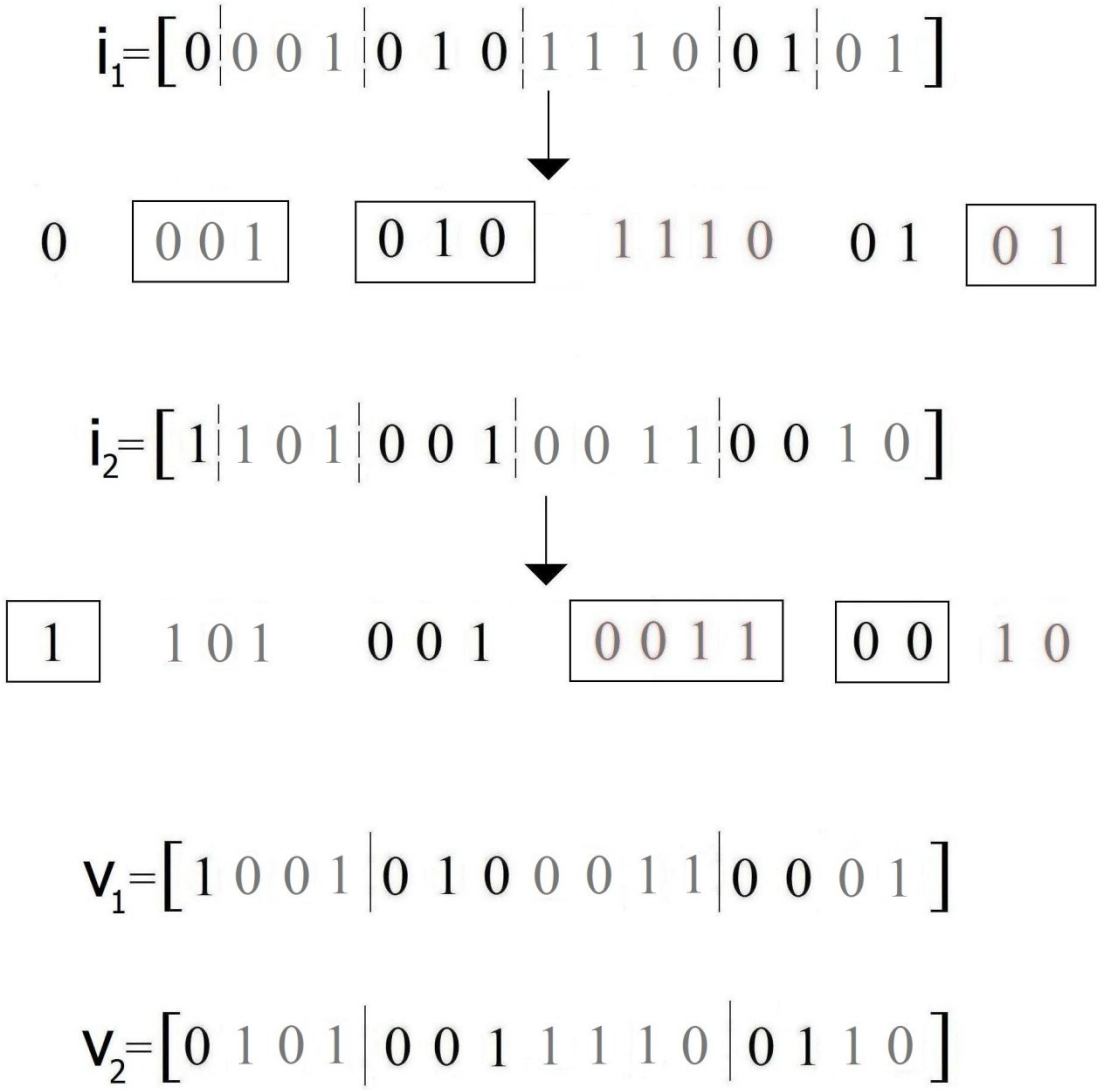


Fig. 16: En la primera etapa de la cruza múltiple, los cromosomas de los individuos padres son segmentados, de acuerdo a la división en subclusters de componentes presente en los vectores solución. Los segmentos resultantes son entremezclados formando dos vectores, a partir de los cuales, siguiendo los mismos pasos que con el primer operador de cruza, se obtendrán los cromosomas de dos individuos hijos.

8.3.3. Tamaño de población. Población inicial.

-En cuanto al tamaño de población, es muy usual que el mismo sea constante, alternativa que seguiremos en este trabajo. Mientras que, en lo que respecta a cuál habrá de ser el tamaño fijo de las poblaciones, debe tenerse en cuenta que tamaños muy grandes implican elevados costos computacionales, a la vez que, con tamaños de población chicos, se corre peligro de que haya una rápida convergencia a mínimos locales de baja calidad, dado que las poblaciones serán relativamente pobres en buenas características. En este último caso, la calidad de las soluciones obtenidas dependerá, en alto grado, de lo eficaz que sea el operador de mutación utilizado en proveer variabilidad genética.

Dado que en nuestro caso, como es común, los cromosomas resultan muy extensos, utilizaremos poblaciones de tamaños no proporcionales a la longitud de los mismos. Aunque, con la esperanza de poder extraer alguna conclusión con respecto a la influencia del tamaño de la población en la eficiencia del algoritmo, utilizaremos, por separado, dos tamaños de población distintos: Un tamaño chico: 20 y uno considerablemente superior: 96.

-Por último, debemos determinar cuál será el criterio que empleemos para generar la población inicial. Para ello, existen básicamente dos alternativas.

La primera consiste en tomar individuos al azar. Ante esto, es difícil esperar que la población inicial cuente con una cantidad considerable de buenas características. Pero se garantiza, por otro lado, la diversidad genética de la población, elemento importante para evitar la rápida convergencia del algoritmo.

La otra alternativa es aplicar algún método de selección que garantice que los individuos que compongan la población inicial correspondan a soluciones de una cierta calidad, utilizando, por ejemplo, las soluciones brindadas por algún otro algoritmo, garantizándose así la presencia de una buena cantidad de características deseables en el paquete genético de la población inicial, pero corriendo el riesgo, por otro lado, de que la población así generada, no posea un buen grado de diversidad genética, ocasionando la rápida convergencia del algoritmo.

La opción que utilizaremos en los algoritmos genéticos que empleemos en este trabajo será la de generar la población inicial a partir de la aplicación del primer algoritmo de búsqueda local presentado en este trabajo, es decir BL_1, utilizando una cantidad de iteraciones muy inferior a lo usual para este algoritmo. De forma tal que los individuos tengan un caudal de características deseables considerablemente superior al que tendrían si fueran seleccionados al azar. Pero buscando evitar, a la vez, una excesiva pérdida de variabilidad genética, la cual podría producirse con soluciones avanzadas de BL_1, que pudieran tener estructuras similares.

9. Aplicación de los algoritmos desarrollados

A continuación, se presentan los resultados obtenidos mediante la aplicación de los diferentes algoritmos desarrollados, sobre instancias del problema definidas a partir de una lista real de pedidos provista por Ternium Siderar. Los pedidos que componen la lista se encuentran agrupados en 44 clusters distintos (clusters no unitarios).

Esta lista de 44 clusters fue cubierta en la práctica utilizando 88 t.b.m.'s, es decir, con un promedio exacto de 2 t.b.m.'s por cluster. El desperdicio de acero resultante fue, aproximadamente, de un 3%.

Lista de pedidos

Para reducir el tamaño de la presentación, la lista de pedidos no se presenta en forma matricial, sino que se enumeran por separado los diferentes clusters. Cada uno de los pedidos es especificado mediante una terna, compuesta por el ancho y el largo (en mms.) y la demanda (en tons.) del pedido. Es decir, toda la información necesaria para la aplicación de los algoritmos. Para esta misma lista de pedidos se buscan, en forma sucesiva, soluciones compuestas por cantidades crecientes de t.b.m.'s (desde 45 a 110). Es decir, son resueltas varias instancias del problema dadas por la misma lista de pedidos y diferentes cantidades de t.b.m.'s a seleccionar.

C_1= {(1500, 5440, 35.31); (1245, 4120, 10.7); (1245, 4380, 21.4)}

C_2= {(1500, 1500, 21); (1090, 1830, 9.45)}

C_3= {(1170, 1965, 20.44); (1270, 2340, 11.242)}

C_4= {(1100, 1780, 6.132); (900, 1500, 0.954); (300, 300, 0.064)}

C_5= {(1000, 2555, 1.836); (1000, 2000, 3.18); (1220, 2440, 5.3); (1220, 1303, 0.9392)}

C_6= {(1220, 1348, 12.826); (475, 1000, 43.0542)}

C_7= {(1220, 2440, 3.18); (1000, 2000, 2.12)}

C_8= {(1000, 2600, 10.6); (1220, 2600, 4.24); (1220, 2440, 1.59)}

C_9= {(1220, 2250, 5.7781); (387, 387, 7.5691)}

C_{10}= {(605, 625, 3); (690, 790, 21.7766); (635, 740, 3); (530, 675, 14.8824); (190, 1060, 4.0619); (270, 900, 9.8046)}

C_{11}= {(1220, 1940, 14.522); (1220, 1860, 2.65); (1220, 2200, 4.452); (150, 843, 3.1882)}

C_{12}= {(1220, 2440, 1.5); (1220, 2440, 3.604); (250, 750, 6.3058); (280, 760, 4.8627)}

C_{13}= {(1485, 1822, 54.0346); (300, 300, 0.009)}

C_{14}= {(1000, 1454, 2); (300, 300, 0.01)}

C_{15}= {(630, 1292, 7.8129); (1020, 1292, 12.6496); (1245, 1292, 11.0294); (395, 1292, 3.5353)}}

C_{16}= {(970, 1513, 1.2943); (797, 970, 1.7045); (970, 1491, 3.1895); (970, 1270, 21.7247); (873, 970, 18.6719); (667, 970, 3.1387); (970, 1330, 0.5692); (970, 1109, 0.7113); (712, 970, 1.8274); (600, 1163, 0.6148); (1220, 1220, 0.6561); (1000, 1270, 1.1204); (561, 970, 1.1045); (506, 970, 1.8402)}

C_{17}= {(618, 1285, 41.2912); (1015, 1285, 55.6193); (1236, 1285, 21.4608); (1266, 1285, 4.2305); (435, 1008, 1.8251); (412, 1285, 14.1824)}

C_{18}= {(640, 1395, 1.65); (890, 1395, 0.55); (1180, 1395, 2.2); (1205, 1395, 1.1); (685, 1473, 12.2012); (725, 1706, 50.8365); (725, 1736, 21.3796); (725, 1866, 26.8741); (774, 796, 0.5962); (716, 925, 3.2021); (796, 898, 0.6908); (796, 923, 1.067); (796, 1099, 12.6984); (716, 1142, 11.8668); (643, 652, 7.095); (643, 732, 2.959); (603, 652, 1.5202); (690, 732, 1.221); (263, 716, 0.9032); (279, 716, 2.8732); (391, 716, 0.6776); (410, 716, 1.848); (443, 796, 3.41); (461, 796, 15.994); (358, 796, 0.4343)}

C_{19}= {(739, 839, 7.6076); (769, 801, 1.8898); (839, 1622, 8.25); (583, 687, 2.662)}

C_{20}= {(1220, 2270, 2.968); (1220, 2550, 1.378); (1300, 2055, 8.1631)}

C_{21}= {(1220, 2440, 1.626); (1000, 2464, 0.742); (900, 2440, 37.3862); (1300, 2100, 50.88); (1000, 2464, 2.12)}

C_{22}= {(1000, 2220, 2.12); (1000, 1610, 21.2); (1220, 2440, 31.8)}

C_{23}= {(1220, 2470, 23.9473); (1045, 2220, 6.2879)}

C_{24}= {(1000, 2825, 1.1124); (1220, 2440, 5.0191)}

C_{25}= {(1515, 1620, 188.5789); (1340, 1380, 22.4524); (1340, 1380, 28.2725); (300, 300, 0.008)}

C_{26}= {(1300, 1420, 12.9809); (1300, 1340, 11.477); (1300, 1340, 11.9092); (500, 1300, 19.3053)}

C_{27}= {(1650, 1685, 29.9527); (610, 1520, 16.88)}

C_{28}= {(1440, 1800, 29.3191); (955, 1700, 31.5649)}

C_{29}= {(718, 1420, 1.134); (440, 560, 5.2784); (310, 550, 2.5); (300, 850, 34.3957)}

C_{30}= {(1100, 2200, 5.67); (1050, 1430, 34.02); (1010, 1330, 37.6857); (500, 870, 10.0246)}

C_{31}= {(1470, 1530, 11.7018); (510, 1630, 20.0457); (190, 410, 5.0028); (240, 1325, 6.9293); (300, 300, 0.016)}

C_{32}= {(1400, 1480, 65.4386); (300, 300, 0.008)}

C_{33}= {(1000, 1300, 34.4827); (415, 770, 3.4); (300, 300, 0.008)}

C_{34}= {(885, 1630, 6.7964); (1345, 1960, 42.7197)}

C_{35}= {(1150, 1440, 19.2814); (1150, 2105, 5.171); (300, 300, 0.011)}

C_{36}= {(1340, 1430, 70.2801); (300, 300, 0.016)}

C_{37}= {(716, 802, 5.7238); (716, 1032, 7.8883); (716, 831, 2.0744); (649, 770, 0.7467); (607, 3700, 3.7587); (607, 3942, 3.2184); (607, 4102, 2.7287); (685, 1892, 8.3601); (810, 2105, 9.8951); (810, 2335, 20.1224); (810, 2712, 6.3741); (810, 1407, 2.7564); (810, 1036, 4.0583); (607, 4102, 2.0671); (607, 3942, 52.0544); (607, 3700,

92.0128); (607, 3540, 28.491); (623, 1078, 1.4817); (623, 988, 3.933); (743, 1102, 13.432); (623, 1391, 6.0778); (623, 1086, 0.128); (743, 1110, 7.5833); (690, 1886, 6.4825); (532, 716, 2.0864); (561, 716, 1.4009); (623, 996, 6.9579); (449, 623, 5.9219); (565, 1418, 3.4208); (441, 623, 0.3361); (585, 795, 4.6338); (515, 554, 0.6768); (490, 1288, 1.5127); (490, 1665, 1.9538); (441, 623, 1.3502); (388, 743, 1.2893); (396, 743, 8.6109); (585, 1591, 4.4154)}

C_{38}= {(783, 1521, 1.7843); (783, 1951, 2.0979); (814, 1436, 4.331); (1204, 1436, 6.5978); (1220, 1428, 4.749); (726, 1089, 1.5268); (796, 802, 2.7863); (796, 818, 2.1296); (796, 923, 3.2064); (606, 4804, 3.2413); (606, 3541, 24.235); (606, 4000, 9.5209); (711, 936, 0.9324); (642, 1315, 9.1956); (642, 1084, 1.2005); (866, 2578, 3.3613); (866, 1281, 1.4406); (866, 1259, 1.4406); (711, 1151, 1.1655); (816, 1715, 61.3904); (579, 1436, 2.2665); (439, 642, 0.3601); (390, 726, 0.5582); (358, 796, 0.3109); (443, 796, 0.9616); (513, 1390, 0.9604); (529, 866, 0.7203); (529, 543, 0.6002); (513, 1174, 0.8403); (463, 1021, 1.4406)}

C_{39}= {(622, 1337, 1.4552); (892, 1337, 4.7719); (1152, 1337, 2.6962)}

C_{40}= {(400, 720, 4.9453); (375, 680, 2.5)}

C_{41}= {(1290, 1300, 32.422); (1275, 1300, 46.96); (1275, 1300, 20.5524); (1290, 1300, 15.7341)}

C_{42}= {(1054, 1840, 3.5845); (475, 1000, 17.4105)}

C_{43}= {(220, 330, 1.0359); (180, 1145, 2.4); (420, 520, 6.1667); (230, 480, 1.5599); (340, 440, 2.6401); (300, 300, 0.008)}

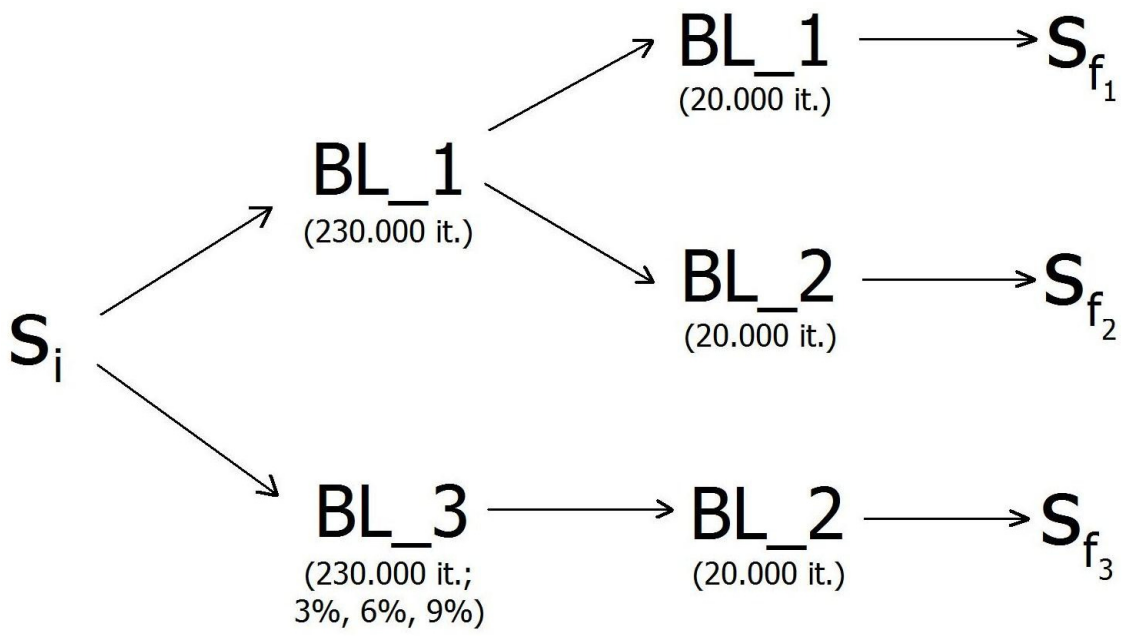
C_{44}= {(270, 305, 60.7064); (300, 300, 0.01)}

Tablas de resultados

La Tabla 1, que a continuación aparece, contiene los resultados de la aplicación, sobre la lista de pedidos presentada, de los algoritmos de búsqueda local desarrollados y del algoritmo random, para cantidades de t.b.m.'s que van desde 45 a 110.

En cada caso, se realizaron 30.000 iteraciones de AR, cuyos resultados aparecen en la segunda columna de la tabla. Mientras que los algoritmos de búsqueda local fueron empleados del siguiente modo:

Para cada cantidad de t.b.m.'s considerada, se genera, al azar, una solución inicial s_i, y a partir de ella se siguen diferentes caminos que conducen a tres diferentes soluciones finales:



En primer lugar, se aplican 230.000 iteraciones del algoritmo BL_1, obteniéndose una solución s', a partir de la cual se siguen dos caminos:

-Por una lado, se aplican, a partir de s', 20.000 iteraciones más de BL_1, obteniéndose una solución final s_{f1} (cuyo valor de costo se especifica en la columna 3 de la tabla).

-Por otro lado, se aplican, a partir de s', 20.000 iteraciones de BL_2, obteniéndose una solución final s_{f2} (cuyo valor de costo se especifica en la columna 4).

El objetivo de esto es comprobar qué tan útil puede ser cambiar el criterio de vecindad empleado en la búsqueda local, al alcanzar un punto en el que el progreso, con el criterio de vecindad anterior, se torna dificultoso (como ocurre, en este caso y para cada cantidad de t.b.m.'s considerada, según observaciones realizadas, transcurridas 230.000 iteraciones del primer algoritmo).

En segundo lugar, se aplican, a partir de la solución inicial s, 230.000 iteraciones del algoritmo BL_3, obteniéndose una solución s", a partir de la cual se aplican 20.000 iteraciones de BL_2, alcanzándose una solución final s_{f3} (cuyo valor de costo se especifica en la columna 5).

En cada caso, BL_3 trabaja realizando una búsqueda de mejora continua, es decir, con umbrales 0, hasta transcurrir 5.000 iteraciones sin lograr progresos, en cuyo caso se comienzan a emplear umbrales positivos durante 500 iteraciones, de forma tal que se acepten soluciones con un costo que sea hasta un 3% superior al de la mejor solución hallada s. Luego, a partir de la solución s' obtenida de tal modo, se realizan 5.000 iteraciones de mejora continua. Si esto no conduce a una solución mejor que s, se retorna a la solución s y se repite el proceso anterior, con la diferencia de que se tolera hasta un 6% de incremento en el costo. Si esto tampoco conduce a una solución mejor que s, se repite el proceso con una tolerancia de hasta un 9% de incremento en el costo. Si esto último tampoco funciona, se continúa con búsqueda de mejora continua a partir de s durante 5.000 iteraciones y de no haber mejorías se reinicia el proceso de dar valores positivos a los umbrales.

En la columna 5 de la tabla, se especifican, además de los costos de las soluciones finales halladas mediante la combinación de BL_3 y BL_2, las cantidades de veces, para cada caso, en que la aplicación de alguno de los tres niveles de tolerancia en el incremento del costo condujo a una solución mejor.

Tabla 1: Algoritmos de Búsqueda Local

Cantidad de t.b.m.'s	Algoritmo Random, 30.000 iteraciones	BL_1, 230.000 iteraciones+…		BL_3, 230.000 iteraciones, con niveles de tolerancia: 3%, 6%, 9% + 20.000 iteraciones BL_2
		…+20.000 iteraciones de BL_1	…+20.000 iteraciones de BL_2	
45	8.0342	5.3351	5.3351	5.3351 (0-0-0)
46	7.9442	4.6712	4.6712	4.6712 (0-2-0)
47	7.3666	4.0165	4.0165	4.0165 (2-1-0)
48	6.9123	3.3760	3.3760	3.3760 (0-0-1)
49	6.7919	2.9717	2.9717	2.9717 (1-1-0)
50	7.1662	2.7089	2.7089	2.7089 (3-1-0)
51	6.3645	2.4509	2.4509	2.4509 (2-0-1)
52	6.5432	2.2074	2.2074	2.2074 (2-0-1)
53	6.1927	2.2501	2.2501	2.0024 (1-1-0)
54	6.1781	1.8041	1.8041	1.8041 (2-0-2)
55	6.4742	1.6325	1.6325	1.6325 (2-0-0)
56	5.3070	1.4905	1.4905	1.4918 (1-0-0)
57	5.8576	1.4626	1.4626	1.3493 (2-0-1)
58	5.2299	1.2095	1.2095	1.2081 (0-0-0)
59	4.9979	1.0843	1.0843	1.0830 (1-0-1)

60	5.6744	1.0530	1.0530	0.9898 (4-1-0)
61	4.7936	0.9585	0.9585	0.9213 (1-0-0)
62	5.0581	0.8566	0.8566	0.8566 (1-0-0)
63	4.6511	0.8031	0.8031	0.8031 (1-1-0)
64	5.2906	0.7960	0.7960	0.7509 (3-0-0)
65	4.8159	0.6875	0.6875	0.6875 (1-2-0)
66	4.3195	0.6725	0.6725	0.6384 (3-1-0)
67	4.5108	0.6091	0.6091	0.5905 (0-2-2)
68	4.5505	0.5638	0.5592	0.5464 (3-2-1)
69	4.2304	0.5043	0.5043	0.5043 (3-0-1)
70	4.7754	0.5269	0.5269	0.4668 (0-1-1)
71	4.5319	0.4425	0.4425	0.4375 (2-2-1)
72	4.4306	0.4128	0.4082	0.4012 (1-2-1)
73	4.2036	0.3732	0.3732	0.3732 (2-1-1)
74	3.8345	0.3445	0.3445	0.3439 (3-2-0)
75	4.0001	0.3249	0.3204	0.3197 (2-0-0)
76	3.7842	0.2978	0.2978	0.2910 (1-4-0)
77	3.9263	0.2753	0.2753	0.2675 (2-0-1)
78	3.2115	0.2513	0.2467	0.2451 (3-2-0)
79	3.6549	0.2427	0.2427	0.2242 (2-0-1)
80	3.3932	0.2103	0.2042	0.2042 (3-1-0)

81	3.6385	0.2102	0.2038	0.1855 (3-1-1)
82	3.5208	0.1696	0.1696	0.1679 (3-3-0)
83	3.4342	0.1709	0.1639	0.1526 (2-4-1)
84	3.4281	0.1427	0.1374	0.1374 (2-2-0)
85	3.1713	0.1293	0.1240	0.1256 (3-2-0)
86	3.5169	0.1254	0.1200	0.1114 (2-1-1)
87	3.2376	0.1206	0.1179	0.1000 (3-0-2)
88	2.8402	0.1159	0.1106	0.0919 (4-2-0)
89	2.8615	0.0946	0.0946	0.0812 (1-0-0)
90	2.2962	0.0818	0.0818	0.0735 (3-0-1)
91	3.0810	0.0741	0.0715	0.0672 (2-1-1)
92	3.0226	0.0697	0.0697	0.0616 (5-2-1)
93	2.7522	0.0701	0.0627	0.0608 (1-1-1)
94	2.8410	0.0545	0.0538	0.0543 (5-0-0)
95	2.7620	0.0527	0.0482	0.0457 (3-0-0)
96	2.5234	0.0602	0.0585	0.0462 (8-2-0)
97	2.6805	0.0383	0.0383	0.0385 (4-4-1)
98	2.6560	0.0399	0.0381	0.0353 (1-2-0)
99	2.5598	0.0344	0.0344	0.0307 (8-0-1)
100	2.8414	0.0383	0.0362	0.0319 (0-4-0)
101	2.5682	0.0437	0.0431	0.0248 (2-1-1)

102	2.4921	0.0237	0.0227	0.0228 (3-2-0)
103	2.1448	0.0227	0.0227	0.0202 (4-1-0)
104	2.4857	0.0241	0.0218	0.0183 (6-2-0)
105	2.2258	0.0168	0.0168	0.0157 (5-1-2)
106	2.2028	0.0170	0.0170	0.0150 (5-1-1)
107	2.4866	0.0119	0.0119	0.0120 (1-1-1)
108	2.0261	0.0127	0.0099	0.0099 (1-5-0)
109	2.2284	0.0108	0.0099	0.0081 (2-2-0)
110	1.9727	0.0073	0.0073	0.0065 (1-0-0)

En la Tabla 2, que a continuación aparece, se especifican los resultados de la aplicación, sobre la lista de pedidos presentada, de los algoritmos genéticos desarrollados.

Las cantidades de iteraciones empleadas para cada tamaño de población son tales que los tiempos de ejecución de estos algoritmos sean similares entre sí, y a su vez sean similares a los de los algoritmos de búsqueda local antes empleados.

Tabla 2: Algoritmos Genéticos

Cantidad de t.b.m.'s	Población: 20 Iteraciones: 16.000 Cruza simple	Población: 96 Iteraciones: 3.200 Cruza simple	Población: 20 Iteraciones: 16.000 Cruza múltiple	Población: 96 Iteraciones: 3.200 Cruza múltiple
45	5.3351	5.3351	5.3351	5.3351
46	4.6712	4.6712	4.6712	4.6712
47	4.0165	4.0165	4.0165	4.0165
48	3.3760	3.3760	3.3760	3.3760
49	2.9717	2.9717	2.9717	2.9717
50	2.7089	2.7136	2.7136	2.7089
51	2.4509	2.4509	2.4509	2.4509
52	2.2074	2.2074	2.2074	2.2074
53	2.0024	2.0024	2.0024	2.0024
54	1.8041	1.8041	1.8041	1.8041

55	1.6325	1.6339	1.6325	1.6339
56	1.4918	1.4905	1.4918	1.4918
57	1.3506	1.3493	1.3506	1.3506
58	1.2095	1.2081	1.2081	1.2081
59	1.0843	1.0843	1.0843	1.0843
60	0.9912	0.9912	0.9912	0.9912
61	0.9605	0.9213	0.9605	0.9213
62	0.8566	0.8566	0.8566	0.8566
63	0.8031	0.8031	0.7974	0.8031
64	0.7398	0.7398	0.7661	0.7509
65	0.6875	0.6875	0.7018	0.6875
66	0.6705	0.6384	0.6384	0.6384
67	0.6142	0.5950	0.6149	0.5905
68	0.5464	0.5509	0.5464	0.5464
69	0.5043	0.5043	0.5217	0.5043
70	0.4668	0.4668	0.4668	0.4668
71	0.4375	0.4355	0.4370	0.4324
72	0.4057	0.4038	0.4335	0.4038
73	0.3971	0.3796	0.3770	0.3764
74	0.3491	0.3445	0.3432	0.3445
75	0.3198	0.3198	0.3198	0.3198

76	0.2963	0.2910	0.2963	0.2963
77	0.2737	0.2721	0.2721	0.2721
78	0.2451	0.2483	0.2467	0.2467
79	0.2349	0.2312	0.2304	0.2242
80	0.2087	0.2068	0.2095	0.2066
81	0.1981	0.1901	0.1879	0.1855
82	0.1815	0.1687	0.1868	0.1733
83	0.1656	0.1533	0.1545	0.1533
84	0.1427	0.1374	0.1539	0.1381
85	0.1302	0.1301	0.1293	0.1256
86	0.1138	0.1121	0.1121	0.1138
87	0.1016	0.1062	0.1000	0.1070
88	0.1043	0.0926	0.0972	0.0919
89	0.0882	0.0820	0.0812	0.0828
90	0.0792	0.0797	0.0834	0.0735
91	0.0755	0.0720	0.0695	0.0672
92	0.0659	0.0619	0.0657	0.0639
93	0.0581	0.0553	0.0644	0.0606
94	0.0559	0.0510	0.0564	0.0520
95	0.0498	0.0479	0.0487	0.0484
96	0.0417	0.0423	0.0480	0.0423

97	0.0382	0.0398	0.0389	0.0387
98	0.0362	0.0385	0.0364	0.0347
99	0.0318	0.0396	0.0331	0.0331
100	0.0288	0.0326	0.0273	0.0277
101	0.0270	0.0247	0.0257	0.0266
102	0.0229	0.0223	0.0245	0.0224
103	0.0238	0.0202	0.0219	0.0199
104	0.0184	0.0178	0.0203	0.0194
105	0.0157	0.0160	0.0171	0.0156
106	0.0136	0.0138	0.0139	0.0139
107	0.0117	0.0117	0.0132	0.0118
108	0.0111	0.0100	0.0099	0.0099
109	0.0099	0.0081	0.0081	0.0081
110	0.0108	0.0067	0.0088	0.0065

A continuación, se presenta, como ejemplo, una de las soluciones obtenidas, especificando, para cada cluster, los t.bm.'s seleccionados para cubrir los pedidos que lo componen. Corresponde a la aplicación del algoritmo genético con operador de cruza simple, para un tamaño de población igual a 96, con una selección de 88 t.b.m.'s. Esta solución tiene un costo asociado, según puede verse en la Tabla 2, de 0.0926.

C_1= {1245; 1500}

C_2= {1090; 1500}

C_3= {1170; 1270}

C_4= {1100}

C_5= {1000; 1220}

C_6= {950; 1220}

C_7= {1000; 1220}

C_8= {1000; 1220}

C_9= {1161; 1220}

C_{10}= {1270; 1350; 1380}

C_{11}= {1220}

C_{12}= {840; 1250}

C_{13}= {1485}

C_{14}= {1000}

C_{15}= {1292}

C_{16}= {970; 1270}

C_{17}= {870; 1285}

C_{18}= {796; 1304; 1432; 1450; 1473}

C_{19}= {839; 1166}

C_{20}= {1220; 1300}

C_{21}= {900; 1000; 1300}

C_{22}= {1000; 1220}

C_{23}= {1045; 1220}

C_{24}= {1000; 1220}

C_{25}= {1380; 1515}

C_{26}= {1300}

C_{27}= {1220; 1650}

C_{28}= {955; 1440}

C_{29}= {1120; 1500}

C_{30}= {1010; 1050; 1100}

C_{31}= {1330; 1530}

C_{32}= {1400}

C_{33}= {1300}

C_{34}= {1345; 1630}

C_{35}= {1150}

C_{36}= {1340}

C_{37}= {1214; 1246; 1380; 1432; 1486; 1620}

C_{38}= {866; 1212; 1284; 1436; 1592; 1632}

C_{39}= {1337}

C_{40}= {1440}

C_{41}= {1300}

C_{42}= {1054; 1425}

C_{43}= {1040; 1440}

C_{44}= {1220}

10. Conclusión

Resulta natural y esperable que existan diferencias considerables entre los resultados obtenidos a partir de la aplicación de los algoritmos desarrollados en este trabajo y los resultados obtenidos en la práctica, a favor de los primeros. Pues mientras que estos se aplicaron sobre una lista de pedidos conocida a priori, en la práctica los pedidos fueron cubiertos a medida que iban surgiendo. Incluso puede observarse que los resultados arrojados por AR son similares a los obtenidos por la empresa.

Aún así, la diferencia generada por los algoritmos heurísticos aplicados es de tal magnitud, que consideramos que la misma sugiere fuertemente que la aplicación de estos algoritmos sobre modelos que predigan la demanda a cubrir podría conducir a importantes ahorros en el proceso productivo de la empresa considerada. En términos generales, el desperdicio promedio de acero en la empresa se puede estimar en un 3%, mientras que la cantidad de t.b.m.'s empleados es de aproximadamente 2 t.b.m.'s por cluster no unitario. Por otro lado, a partir de los algoritmos presentados se obtienen, en promedio, desperdicios de alrededor de 3% con menos de 1.3 t.b.m.'s por cluster no unitario; y desperdicios inferiores a 0.6% con aprox. 2 t.b.m.'s por cluster no unitario.

En cuanto al comportamiento relativo de los algoritmos, las siguientes observaciones extraídas de las tablas presentadas, reforzadas por lo observado a partir de la aplicación de los algoritmos a listas de pedidos generadas aleatoriamente, permiten obtener algunas conclusiones al respecto:

-Comparando el rendimiento de los algoritmos de búsqueda local empleados, se observa que la combinación de BL_3 y BL_2, en la cual se utilizan umbrales y cambio de criterio de vecindad en la exploración, obtuvo resultados superiores a los de las otras dos alternativas seguidas (BL_1 y BL_1+BL_2) en 42 casos, y obtuvo resultados de menor calidad en solo 6 casos. A su vez, la combinación de BL_1 y BL_2, es decir, la combinación de los dos criterios de vecindad diferentes, arrojó resultados superiores a los de BL_1 (en el cual se utiliza un único criterio de vecindad) en 24 casos, y en ningún caso dio un resultado inferior.

Sumando a esta observación otras similares realizadas a partir de listas de pedidos generadas aleatoriamente, se puede concluir que el cambio del criterio de vecindad en etapas avanzadas de la exploración conduce a mejorías en los resultados con una frecuencia muy elevada. Por otra parte, el agregado de umbrales positivos a la exploración genera resultados superiores a los obtenidos en la exploración sin umbrales, en una gran proporción de los casos.

-Es interesante observar, a partir de la Tabla 2, que si se compara, en cada caso, el mejor resultado obtenido por los dos algoritmos que emplean cruza simple con el mejor resultado obtenido a partir de los dos algoritmos que usan cruza múltiple, los primeros dieron mejores resultados en 19 casos y los segundos fueron superiores también en 19 casos.

Por otro lado, si se compara el rendimiento de los algoritmos que utilizaron un tamaño de población igual a 20, con los que utilizaron una población de tamaño 96, los primeros arrojaron mejores resultados en 11 casos, mientras que los segundos lo hicieron en 29 casos. Al respecto, debe recordarse que, tal como se especifica en la Tabla 2, al

cambiar el tamaño de población de 20 a 96, se redujo de manera inversamente proporcional la cantidad de iteraciones, de forma tal que los algoritmos trabajen en cantidades de tiempo similares, y así poder realizar una mejor comparación de la eficacia en ambos casos.

-Comparando los resultados obtenidos por los algoritmos de búsqueda local con los obtenidos por los algoritmos genéticos, se observa que los primeros arrojaron mejores resultados en 14 casos, mientras que los segundos lo hicieron en 17 casos (se compara, en cada caso, el mejor resultado obtenido por los algoritmos de búsqueda local con el mejor resultado obtenido por los algoritmos genéticos). Si bien ambos condujeron a mejores resultados en cantidades similares de casos, puede observarse que los algoritmos genéticos fueron mostrando un comportamiento más favorable, con respecto al de la búsqueda local, al irse incrementando la cantidad de t.b.m.'s a seleccionar y, por lo tanto, el tamaño del espacio de soluciones (en las últimas 15 instancias, los algoritmos genéticos arrojaron mejores resultados en 11 casos). En cambio, la exploración con umbrales parece incrementar su eficacia relativa cuando el tamaño del espacio de soluciones es menor.

11. Bibliografía

[1] Dyckhoff, H.: *A typology of cutting and packing problems*. European Journal of Operational Research, 44, pp. 145-149, 1990.

[2] Cook, William J.; Cunningham, W.H.; Pulleyblank, W.R.; Schrijver, A.: *Combinatorial Optimization*; Wiley-Interscience, 1998.

[3] Lawler, E.L.; Lenstra, J.K.; Rinooy Kan, A.H.G.; Shmoys, D.B.: *The traveling Salesman Problem: a guided tour of Combinatorial Optimization*; Wiley, 1985.

[4] Nemhauser, G.L.; Wolsey, L.A.: *Integer and Combinatorial Optimization*; Wiley-Interscience, 1988.

[5] Garey, M.R.; Johnson, D.S.: *Computers and Intractability. A guide of the theory of NP-Completeness*; Freeman, 1979.

[6] Cook, Stephen: *The complexity of theorem proving procedures*, Proceedings of the Third Annual ACM Symposium on Theory of Computing, ACM, New York, pp. 151-158; 1971.

[7] Turing, Alan: *On computable numbers, with an application to the Entscheidungsproblem*, Proceedings of the London Mathematical Society, Series 2, 42, pp. 230-265, 1936.

[8] Papadimitriou, Christos H.: *Computational Complexity*; Addison-Wesley, 1994.

[9] Kantorovich, L.V.: *Mathematical methods of organizing and planning production*, Management Science, 6, pp. 366-422, 1960.

[10] Blazewicz, J.; Drozdowski, B.; Soniewicki, B.; Walkowiak, R.: *Two dimensional cutting problem: basic complexity results and algorithms for irregular shapes*, Found. Contr. Eng., 14, 1989.

[11] Fowler, R.J.; Paterson, M.S.; Tatimoto, S.L.: *Optimal packing and covering in the plane are NP-complete*. Information Processing Letters, 12, 1981.

[12] Reeves, C.R.: *Modern Heuristic Techniques for Combinatorial Optimization*, Wiley, 1993.

[13] Karelahti, J: *Solving the cutting stock problem in the steel industry*. Master's Thesis, Department of Engineering Physics and Mathematics, Helsinki University of Technology, 2002.

[14] Olsen, S.M.: *Using simulation to examine cutting policies for a steel firm*. Degree of Master of Science in Industrial Engineering, University of Pittsburgh, 2003.

[15] Ignacio Ojea: *Un algoritmo para el problema de Corte de Stock en Dos Dimensiones por Matching Iterado*. Tesis de Licenciatura en Cs. Matemáticas, Directora: Susana Puddu, Departamento de Matemática, Universidad de Buenos Aires, 2008.

[16] Aarts, E.; Lenstra, J.K.: *Local Search in Combinatorial Optimization*; Wiley-Interscience, 1997.

[17] Holland, John H.: *Adaptation in Natural and Artificial Systems*; University of Michigan Press, 1975.