# Metaheurísticas constructivas para *Car Sequencing Problem* con Flotas de vehículos especiales

Ignacio Moya*, Joaquín Bautista†, Manuel Chica*, Sergio Damas*, Oscar Cordón*
*Instituto Andaluz Interuniversitario DaSCI (Data Science and Computational Intelligence), Universidad de Granada, España
† Nissan Chair, Universidad Politécnica de Catalunya, España
Emails: imoya@ugr.es, manuelchica@ugr.es, sdamas@ugr.es, joaquin.bautista@upc.edu, ocordon@decsai.ugr.es

***Resumen*—*Car sequencing problem* (CSP) es un problema tradicional de satisfacción de restricciones que refleja los problemas que surgen cuando una serie de vehículos se introducen en una cadena de producción. Sin embargo no considera vehículos no regulares o fuera de catálogo, pese a que en plantas reales pueden llegar a representar entre el 10 % y el 20 % de la producción. Estos vehículos se distinguen de los regulares por ser bajo pedido, estar orientados al servicio público, y/o requerir componentes poco frecuentes. La extensión *Robust CSP* (r-CSP) sí que considera estos vehículos, modelando la incertidumbre que introducen los vehículos no regulares como escenarios de producción. Para resolver r-CSP proponemos emplear metaheurísticas constructivas que ya han sido efectivas en el CSP tradicional: GRASP y VNS. Evaluamos el rendimiento de estos algoritmos en r-CSP extendiendo las instancias disponibles en la literatura añadiendo la semántica de flotas de vehículos especiales. Nuestros experimentos reflejan que GRASP muestra mejor rendimiento en las instancias más sencillas, mientras que VNS se muestra superior en las más complejas.**

***Index Terms*—Metaheurísticas, Líneas de montaje, Secuenciación, Car Sequencing Problem, Flotas de vehículos**

## I. Introducción

*Car Sequencing Problem* (CSP) fue introducido por Parello *et al*. [1] a consecuencia de la transición de la industria hacia la personalización masiva. Es un problema de satisfacción de restricciones que representa las dificultades que surgen al introducir vehículos con opciones diferentes en una misma línea de montaje, trasformando restricciones de tiempo, espacio, o técnicas en opciones binarias de modo que todo vehículo requiera de un conjunto arbitrario de estas opciones. Estas opciones se modelan indicando las paradas que pueden aparecer en una línea de montaje si muchos vehículos seguidos requieren una determinada opción, por lo que cada opción se representa usando un ratio de carga. Este ratio se representa como $p_i/q_i$, donde $p$ es el número de vehículos que pueden requerir la opción $i$ en una secuencia de longitud $q$ sin causar paradas en la línea. El objetivo del CSP es encontrar una secuencia completa de vehículos que no sobrecargue ninguna opción en ninguna de sus subsecuencias.

Sin embargo el CSP no considera vehículos no regulares o fuera de catálogo [2]. Estos vehículos se distinguen de los regulares por ser bajo pedido, estar orientados al servicio público, y/o requerir componentes poco frecuentes. El impacto de estos vehículos en la cadena genera incertidumbre y afecta a la producción, ya que pueden alcanzar entre el 10 % y el 20 % de la producción. Una alternativa para tratar esta incertidumbre es definir distintos escenarios de producción que sean realistas.

La extensión *Robust CSP* (r-CSP) [2] sigue este enfoque, donde la demanda se define en distintos planes de producción y el impacto de la variabilidad se mide observando los conflictos que ocurren cuando se cambia de un plan a otro. En este artículo estudiamos la resolución de r-CSP utilizando como referencia el trabajo anterior para el CSP tradicional, que comúnmente se resuelve usando metaheurísticas. Distintas metaheurísticas se han aplicado con éxito al CSP, de las cuales nosotros destacamos las metaheurísticas constructivas GRASP [3] y VNS [4], [5].

El resto del texto sigue la siguiente estructura. La Sección II revisa la bibliografía relevante relacionada con la resolución del CSP tradicional. Después presentamos brevemente la formalización del r-CSP en la Sección III. La Sección IV describe las metaheurísticas empleadas en nuestros experimentos. En la Sección V presentamos la configuración y los resultados de nuestros experimentos para el r-CSP. Por último, en la Sección VI presentamos nuestras conclusiones.

## II. Trabajo relacionado

Existen diferentes alternativas para resolver el CSP tradicional. Encontramos varios métodos exactos basados en programación con restricciones [6], [7], *integer programming* [8], algoritmos de ramificación y poda [9], o *beam search* [10], [11]. También encontramos muchos ejemplos de resolución usando metaheurísticas [5], [12], [13], [14], [15]. Todos estos métodos se suelen comparar usando el *framework* de referencia CSPLib [16].

De entre los distintos trabajos que utilizan metaheurísticas, podemos destacar los métodos basados en búsqueda local, VNS y GRASP. De entre los trabajos que emplean búsqueda local, el trabajo de Puchta y Gottlieb [17] es especialmente relevante dado que introduce nuevos operadores que han sido empleados posteriormente en otros trabajos [3], [5], [12], [13]. Uno de estos operadores, llamado operador de inversión o *Lin2Opt* se ha convertido en un operador recurrente. Los métodos de búsqueda local normalmente incluyen heurísticas voraces para construir la solución inicial, de las cuales encontramos que la propuesta por Gottlieb *et al*. [12] es la más empleada [5], [13], [14], [18], [19].

En el caso de VNS y GRASP, las contribuciones más relevantes se aplican a extensiones del CSP tradicional. En

el caso de VNS, destacamos el trabajo propuesto de Ribeiro *et al.* [4]. Su versión de VNS combinada con ILS terminó en segunda posición en el ROADEF'2005 [20], donde el problema a resolver era una extensión del CSP tradicional que incluye nuevas restricciones relacionadas con la pintura. Hemos incluido algunas de las características de su trabajo en nuestro diseño de VNS, como el uso del operador de inserción (explicado en la Sección IV-C) como operador en la perturbación de la solución. En el caso del GRASP podemos encontrar trabajos donde se utiliza para resolver extensiones del CSP tradicional [3], por lo que incluimos estos diseños en nuestro estudio.

## III. Descripción del modelo

En esta sección describimos las particularidades del CSP con flotas de vehículos especiales (r-CSP). Este modelo sigue las siguientes hipótesis de trabajo:

1. Hay dos conjuntos separados de familias de vehículos: los regulares $I_X$ y los especiales $I_{X\prime}$.
2. Existe una demanda de vehículos regulares $D_X$ fija para cada plan de producción $\varepsilon \in E$.
3. Existe una demanda de vehículos especiales $D_{X\prime}$ fija para cada plan de producción $\varepsilon \in E$.
4. La demanda total de vehículos $T(T \equiv D \equiv D_X + D_{X\prime})$, se respeta en todos los planes de producción $\varepsilon \in E$.
5. La demanda para cada vehículo regular $d_i, i \in I_X$ es la misma para cada plan de producción $\varepsilon \in E$. Esto es, $d_{i,\varepsilon} = d_i, \forall \varepsilon \in E$.
6. La demanda de cada vehículo especial $i \in I_{X\prime}$ puede cambiar para dos planes de producción distintos $\{\varepsilon, \varepsilon\prime\} \subseteq E$.
7. Los cambios en la cadena de producción deberían ser tan pocos como sea posible, por lo que las secuencias de producción $\pi_\varepsilon(T)$ y $\pi_{\varepsilon\prime}(T)$ deberían ser lo más similares posible para cada par de planes $\{\varepsilon, \varepsilon\prime\} \subseteq E$. Esto es: $\pi_\varepsilon(T) \approx \pi_{\varepsilon\prime}(T), \forall\{\varepsilon, \varepsilon\prime\} \subseteq E$.
8. Cada vehículo regular mantendrá la misma posición en todas las secuencias $\pi_\varepsilon(T), \forall \varepsilon \in E$.

A continuación incluimos la definición formal de r-CSP en el ámbito de programación lineal entera mixta. También incluimos un conjunto de restricciones que cualquier secuencia válida debe cumplir. Las restricción (1) impone que solo un vehículo (regular o especial) puede asignarse en cada ciclo de producción $t \in [1, T]$ para todos los planes de producción $\varepsilon \in E$. Las restricciones (2) y (3) fuerzan a que se satisfaga la demanda de vehículos regulares ($I_X$) y especiales ($I_{X\prime}$) en cada plan de producción. La restricción (4) determina la producción acumulada de la clase de vehículos regulares $i \in I_X$ hasta el ciclo de fabricación $t$. La restricción (5) es análoga a la anterior, referida a los vehículos especiales. La restricciones (6) y (7) imponen que el número de veces que se requiere la opción $j \in J$ de manera consecutiva en cualquier ciclo de producción $t \in [1, T]$ para todo plan de producción $\varepsilon \in E$ no supere su ratio $p/q$. Las restricciones (8), (9), y (10) definen respectivamente las variables binarias $x_{i,t}$, $x_{i,t,\varepsilon}$, y $z_{j,t,\varepsilon}$. Finalmente, la restricción (11) fija un valor nulo para las variables $Y_{j,0,\varepsilon}(\forall j \in J, \forall \varepsilon \in E)$ en el ciclo virtual $t = 0$.

$$\sum_{i \in I_X} x_{i,t} + \sum_{i \in I_{X\prime}} x\prime_{i,t,\varepsilon} = 1, \forall t \in [1, T], \forall \varepsilon \in E \quad (1)$$

$$\sum_{t \in [1,T]} x_{i,t} = d_i \forall i \in I_X \quad (2)$$

$$\sum_{t \in [1,T]} x\prime_{i,t,\varepsilon} = d_{i,\varepsilon}, \forall i \in I_{X\prime}, \forall \varepsilon \in E \quad (3)$$

$$X_{i,t} - \sum_{\tau \in [1,t]} x_{i,\tau} = 0, \forall i \in I_X, \forall t \in [1, T] \quad (4)$$

$$X\prime_{i,t,\varepsilon} - \sum_{\tau \in [1,t]} x\prime_{i,\tau,\varepsilon} = 0, \quad \forall i \in I_{X\prime}, \forall t \in [1, T], \forall \varepsilon \in E \quad (5)$$

$$Y_{j,t,\varepsilon} - \sum_{i \in I_X} n_{j,i} X_{i,t} - \sum_{i \in I_{X\prime}} n_{j,i} X\prime_{i,t,\varepsilon} = 0, \quad \forall j \in J, \forall t \in [1, T], \forall \varepsilon \in E \quad (6)$$

$$Y_{j,t,\varepsilon} - Y_{j,t-q_j,\varepsilon} \leq p_j + T \cdot z_{j,t,\varepsilon}, \quad \forall j \in J, \forall t \in [q_j, T], \forall \varepsilon \in E \quad (7)$$

$$x_{i,t} \in \{0, 1\}, \forall i \in I_X, \forall t \in [1, T] \quad (8)$$

$$x\prime_{i,t,\varepsilon} \in \{0, 1\}, \forall i \in I_{X\prime}, \forall t \in [1, T], \forall \varepsilon \in E \quad (9)$$

$$z_{j,t,\varepsilon} \in \{0, 1\}, \forall j \in J, \forall t \in [q_j, T], \forall \varepsilon \in E \quad (10)$$

$$Y_{j,0,\varepsilon} = 0, \forall j \in J, \forall \varepsilon \in E \quad (11)$$

r-CSP define multisecuencias $\vec{\pi}(E, T)$ compuestas de secuencias individuales $\pi_\varepsilon(T) = (\pi_{1,\varepsilon}, ..., \pi_{T,\varepsilon})$ para cada plan de producción $\varepsilon \in E$, como se muestra en la ecuación (12). Los vehículos regulares $i \in I$ se relacionan con los elementos $\pi_{t,\varepsilon}(\forall t \in [1, T], \forall \varepsilon \in E)$ de la multisecuencia $\vec{\pi}(E, T)$ por las variables binarias $x_{i,t}(\forall i \in I_X, \forall t \in [1, T])$ y $x\prime_{i,t,\varepsilon}(\forall i \in I_{X\prime}, \forall t \in [1, T], \forall \varepsilon \in E)$, como se define en las ecuaciones (13) y (14). Debemos remarcar que todo vehículo regular $(i \in I_X)$ mantiene su posición en la secuencia para cada plan de producción, pero las posiciones ocupadas por vehículos especiales dependen del plan de producción $\varepsilon \in E$. De este modo podemos decir que toda secuencia $\pi_\varepsilon(T)$ tiene un componente común (los vehículos regulares) y un componente exclusivo (sus vehículos especiales).

$$\pi_\varepsilon(T) = \left\{ \begin{array}{c} \pi_1(T) \\ \pi_2(T) \\ ... \\ \pi_\varepsilon(T) \\ ... \\ \pi_{|E|}(T) \end{array} \right\} = \left\{ \begin{array}{c} (\pi_{1,1}, ..., \pi_{t,1}, ..., \pi_{T,1}) \\ (\pi_{1,2}, ..., \pi_{t,2}, ..., \pi_{T,2}) \\ ... \\ (\pi_{1,\varepsilon}, ..., \pi_{t,\varepsilon}, ..., \pi_{T,\varepsilon}) \\ ... \\ (\pi_{1,|E|}, ..., \pi_{t,|E|}, ..., \pi_{T,|E|}) \end{array} \right\} \quad (12)$$

$$x_{i,t} = 1 \Rightarrow \pi_{t,\varepsilon} = i, \forall i \in I_X, \forall t \in [1,T], \forall \varepsilon \in E \quad (13)$$

$$x\prime_{i,t} = 1 \Rightarrow \pi_{t,\varepsilon} = i, \forall i \in I_{X\prime}, \forall t \in [1,T], \forall \varepsilon \in E \quad (14)$$

Siguiendo esta formulación definimos la función objetivo para r-CSP. Esta función, que mostramos en la ecuación (15), minimiza el número de violaciones de restricciones para las opciones $j \in J$ usando sus ratios de carga $p_j/q_j$ para todos los planes de producción $\varepsilon \in E$. Usando este enfoque tratamos r-CSP como un problema de satisfacción de restricciones y conectamos con la formulación original de Parello *et al*. [1].

$$min\ Z = \sum_{j \in J} \sum_{t \in [q_j, T]} \sum_{\varepsilon \in E} z_{j,t,\varepsilon}$$
$$\Leftrightarrow max\ Z\prime = \sum_{j \in J} \sum_{t \in [q_j, T]} \sum_{\varepsilon \in E} (1 - z_{j,t,\varepsilon}) \quad (15)$$

## IV. Métodos

Como ya hemos anticipado, existen distintos casos de éxito de la aplicación de metaheurísticas al CSP tradicional [4], [12], [13], [17], [21]. En muchas de estos casos se aplican metaheurísticas constructivas [22], por lo que proponemos resolver r-CSP también con estos métodos. En concreto, hemos elegido GRASP y VNS.

### A. GRASP

Nuestro diseño de *Greedy Randomized Adaptive Search Procedure* (GRASP) [23] está basado en la heurística propuesta por Gottlieb *et al*. [12]. De este modo, la lista de candidatos para la fase constructiva se compone en tres etapas. En el primer paso, se descartan los vehículos que violarían alguna restricción de carga $p/q$, salvo que no existan vehículos que no violen alguna restricción, que en ese caso deberán incluir el mínimo de nuevas violaciones posible. En el segundo paso, si existe más de un elemento en la lista (muy posible sobretodo en los primeros pasos), esos elementos se ordenan según su valor heurístico usando la heurística de Gottlieb *et al*. [12]. De este modo se favorece incluir primero los vehículos que incluyen opciones más restrictivas. Finalmente la lista de candidatos se trunca dependiendo del parámetro $\alpha \in (0, 1]$, dejando en la lista los elementos con mejor valor heurístico. La selección final en este paso de la fase constructiva será un elemento aleatorio de la lista de candidatos resultante. Exponemos la fase de refinado posterior usando búsqueda local en la Sección IV-C.

### B. VNS

Nuestra implementación de *Variable Neighborhood Search* (VNS) [24] construye su solución inicial siguiendo una estrategia voraz basada en la heurística de Gottlieb *et al*. [12]. A diferencia del diseño usado por GRASP, en este caso el proceso es determinista y siempre se elige el vehículo que incluya el menor número de violaciones posible con el mejor valor de la heurística.

Con respecto al diseño de la fase de perturbación de la solución, usamos un operador distinto al que después se utiliza para la fase de refinado. En concreto elegimos utilizar un operador de inserción, que se aplica $k$ veces durante la fase de perturbación donde $k$ es igual a la iteración actual del algoritmo. Por ejemplo, con $k = 3$ aplicamos tres veces el operador de manera secuencial. Este operador se describe en detalle junto con el resto de operadores de búsqueda local en la Sección IV-C.

### C. Búsqueda local

Elegimos nuestros operadores para la búsqueda local basándonos en trabajo previo [12], [17] y el resultado de nuestros experimentos. Estos operadores son: *Swap*, *Inserción*, y *Inversión*. Para mostrar como estos operadores modifican una secuencia dada usaremos como referencia la cadena $\pi$ descrita en la Ecuación 16. El operador de *Swap* intercambia dos posiciones aleatorias en una secuencia. En la Ecuación 17 se puede ver el resultado de aplicar el operador de *Swap* a la cadena $\pi$, intercambiando los elementos de los índices 3 y 7. El operador de *Inserción* selecciona un índice aleatoriamente y lo mueve a otra posición aleatoria de la cadena, desplazando las posiciones intermedias de manera que mantengan su orden en la secuencia. En la Ecuación 18 el operador de *Inserción* mueve el elemento en la posición 5 a la posición 9. El operador de *Inversión* invierte una subcadena definida por dos índices seleccionados aleatoriamente. En la Ecuación 19 se muestra cómo el operador de *Inversión* invierte una subcadena de $\pi$ entre las posiciones 4 y 6.

$$\pi = 5, 3, 4, 2, 3, 4, 1, 5, 2, 0 \quad (16)$$

$$\pi_{Swap} = 5, 3, \mathbf{1}, 2, 3, 4, \mathbf{4}, 5, 2, 0 \quad (17)$$

$$\pi_{Insert} = 5, 3, 4, 2, 4, 1, 5, 2, \mathbf{3}, 0 \quad (18)$$

$$\pi_{Invert} = 5, 3, 4, \mathbf{4}, \mathbf{3}, \mathbf{2}, 1, 5, 2, 0 \quad (19)$$

Independientemente del operador, nuestra búsqueda local siempre se mueve al primer vecino que mejore la solución (*primer-mejor*). De este modo, en cada paso de la búsqueda local se genera y evalúa un vecino aleatorio de la solución actual, aceptándolo si mejora a la anterior y deshaciendo el cambio si no. Nuestra búsqueda local aplica dos operadores de manera secuencial: primero explora el espacio de búsqueda usando *Swap* hasta que la búsqueda termina; después se repite el proceso utilizando el operador de *Inversión*.

## V. Experimentación

### A. Configuración

Comparamos el rendimiento de las metaheurísticas seleccionadas usando las instancias publicadas en CSPLib [16], que es el *benchmark* generalmente utilizado por la comunidad del CSP [12], [13], [14]. Este benchmark contiene instancias con diferentes grados de complejidad que se podrían dividir en tres categorías: las factibles, las clásicas y las avanzadas. Las instancias factibles son un conjunto de 70 instancias de 200 vehículos con 5 opciones y entre 17 y 30 clases. Podemos argumentar que estas instancias son las más básicas dado que están diseñadas para ser factibles. Estas instancias están divididas en grupos de 10 según el porcentaje de requerimiento de sus opciones. Las instancias menos exigentes de esta categoría tienen un 60 % y los distintos grupos de instancias aumentan su complejidad gradualmente hasta un 90 %. Las instancias clásicas [16] son un grupo de 9 instancias con 100 vehículos con 5 opciones y entre 18 y 24 clases. Este grupo de instancias es más complejo porque sólo 4 de ellas son factibles, dado que su porcentaje de requerimiento de sus opciones está cercano al 100 %. Finalmente las instancias avanzadas, propuestas por [8], son 30 secuencias de 200, 300, y 400 vehículos. Estas instancias tienen configuraciones de opciones y clases similares a las instancias clásicas. Igual que en el caso de las instancias clásicas, las instancias avanzadas están diseñadas para tener un porcentaje de requerimiento muy alto y para la mayoría de ellas no se conoce una solución factible [15].

Como estas instancias no consideran demanda parcial incierta, extendemos su semántica generando apéndices que incluyan flotas de vehículos especiales para las instancias ya existentes. De esta manera maximizamos la compatibilidad con los trabajos anteriores. El enfoque que seguimos es el siguiente: para cada instancia existente, consideramos como no regulares el 20 % de sus clases de vehículos. La suma de la demanda original de los vehículos que pasan a tratarse como no regulares será la demanda de vehículos especiales en el plan de producción de referencia. En concreto, las clases de vehículos no regulares serán las últimas clases definidas en la instancia original. Por ejemplo, tomando secuencia original con 10 clases, las clases de vehículos 1 a 8 seguirán siendo vehículos regulares y las clases 9 y 10 serán consideradas no regulares. De este modo, la demanda de vehículos que consideran flotas de vehículos especiales depende de la demanda para las clases 9 y 10. Para terminar, se generan aleatoriamente 10 planes de producción utilizando una distribución uniforme. Continuando el ejemplo anterior, si la demanda original de las clases 9 y 10 era de 20 vehículos cada una, la suma de ambas será la demanda del plan de referencia y se generan 10 combinaciones aleatorias de 40 vehículos en total.

En cuanto a los cálculos de fitness, en el caso del CSP tradicional los valores de fitness representan el número de violaciones de restricciones $p/q$ en una secuencia. Este es un caso especial de la función objetivo definida en la Ecuación 15 con un único plan de producción (i.e. $|E| = 1$). Podemos denominar a esta función objetivo $regular(\pi_\varepsilon)$. Calcular el fitness para $|E| > 1$ implica generar y calcular cada combinación de planes de producción y vehículos no regulares contando las nuevas violaciones ($\forall \varepsilon \in E$). Dado que generar cada posible combinación para todas las secuencias implicaría tiempos de ejecución muy costosos, en su lugar optamos por una solución intermedia: simular este valor siguiendo un enfoque similar a las *simheurísticas* [25]. De este modo aproximamos el valor generando aleatoriamente $|M|$ secuencias posibles para cada plan de producción partiendo de la secuencia encontrada usando el plan de referencia $\pi_\varepsilon$ y calculamos la media del mínimo de violaciones para cada plan $\varepsilon \in E$. La función $fleets(\pi_\varepsilon)$, definida en la Ecuación 20, es por tanto una aproximación para calcular la función objetivo de r-CSP (definida en la Ecuación 15 de la Sección III). Nuestra función de fitness final para resolver r-CSP combina el número de violaciones sobre el plan de referencia con la aproximación de las violaciones de las flotas de vehículos especiales usando el parámetro $\theta$. Este parámetro $\theta$ se incluye para potenciar la relevancia del plan de referencia con respecto al resto de planes de producción. Ajustamos los parámetros $\theta$ y $|M|$ durante la experimentación y finalmente se fijan a $\theta = 0{,}6$ y $|M| = 30$. En la Ecuación 21 mostramos la función objetivo final que utilizamos en nuestros experimentos ($fitness(\pi_\varepsilon)$).

$$fleets(\pi_\varepsilon) = \frac{min \sum_{j \in J} \sum_{t \in [q_j, T]} \sum_{\varepsilon \in E} z_{j,t,\varepsilon}}{|M|}, \quad \forall m \in M \tag{20}$$

$$fitness(\pi_\varepsilon) = \theta * regular(\pi_\varepsilon) + (1 - \theta) * fleets(\pi_\varepsilon) \tag{21}$$

Tanto GRASP como VNS ejecutan durante su búsqueda local un máximo de 10,000 pasos, y contemplan 400 iteraciones respectivamente. De este modo, nuestra experimentación fija un criterio de parada de 4,000,000 evaluaciones para las metaheurísticas, que se ejecutan 30 veces utilizando diferentes semillas. Además, fijamos el parámetro $\alpha = 0,15$ para las ejecuciones de GRASP. En cuanto a la implementación, hemos implementado cada metaheurística en Java y hemos empleado JAMES [26] para la búsqueda local.

### B. Comparación algorítmica para el r-CSP

En esta sección discutimos el rendimiento de las metaheurísticas sobre las instancias modificadas para incluir flotas de vehículos especiales. Por razones de espacio los resultados sobre estas instancias se muestran divididos en las Tablas I (instancias básicas desde 60-01 hasta 85-10) y II (instancias básicas entre 90-01 y 90-10, instancias clásicas p*, e instancias avanzadas pb_200-*, pb_300-*, y pb_400-*).

En la Tabla I vemos que prácticamente todos los valores resaltados son de GRASP, tanto en mínimo valor como en valor promedio de $fitness$. La superioridad de GRASP en estas instancias podría deberse a la baja carga de sus opciones. Además con la configuración de parámetros elegida puede que

Tabla I
Valores mínimos y promedio de las 30 ejecuciones de cada metaheurística aplicada a las instancias entre 60-01 y 85-10.

| Instancias | GRASP | | VNS | |
|---|---|---|---|---|
| | Min | Med | Min | Med |
| 60-01 | **3.08** | **3.63** | 4.64 | 5.97 |
| 60-02 | **59.48** | **60.50** | 62.92 | 63.73 |
| 60-03 | **4.32** | **4.66** | 6.00 | 6.82 |
| 60-04 | **30.56** | **31.23** | 31.28 | 32.87 |
| 60-05 | **1.48** | **2.40** | 3.44 | 4.62 |
| 60-06 | **2.24** | **3.01** | 2.68 | 3.56 |
| 60-07 | **1.72** | **2.18** | 2.92 | 3.74 |
| 60-08 | **3.52** | **4.22** | 6.56 | 7.71 |
| 60-09 | **2.96** | **3.44** | 3.88 | 5.03 |
| 60-10 | **2.92** | **3.71** | 9.24 | 10.81 |
| 65-01 | **6.28** | **7.04** | 7.24 | 7.82 |
| 65-02 | **59.96** | **60.72** | 62.68 | 62.68 |
| 65-03 | **5.60** | **6.24** | 9.68 | 11.02 |
| 65-04 | **26.60** | **27.57** | 27.32 | 29.10 |
| 65-05 | **4.96** | **5.53** | 5.52 | 7.42 |
| 65-06 | **5.20** | **5.99** | 8.76 | 10.13 |
| 65-07 | **5.08** | **5.68** | 7.60 | 9.05 |
| 65-08 | **5.24** | **5.99** | 8.04 | 9.07 |
| 65-09 | **2.32** | **2.75** | 3.60 | 4.59 |
| 65-10 | **4.00** | **4.99** | 7.16 | 9.43 |
| 70-01 | **7.24** | **8.56** | 11.20 | 11.91 |
| 70-02 | **58.56** | **60.03** | 61.04 | 62.07 |
| 70-03 | **10.88** | **11.53** | 12.76 | 13.60 |
| 70-04 | **10.48** | **11.33** | 11.04 | 11.94 |
| 70-05 | **12.28** | **12.81** | 14.88 | 15.96 |
| 70-06 | **3.28** | **3.94** | 6.56 | 7.38 |
| 70-07 | **8.92** | **9.86** | 12.12 | 13.88 |
| 70-08 | **8.32** | **8.99** | 9.44 | 10.46 |
| 70-09 | **6.48** | **7.52** | 7.36 | 10.01 |
| 70-10 | **7.84** | **8.82** | 11.16 | 13.88 |
| 75-01 | **10.80** | **11.28** | 11.84 | 13.22 |
| 75-02 | **106.32** | **108.29** | 109.92 | 113.16 |
| 75-03 | **15.88** | **16.42** | 17.40 | 18.74 |
| 75-04 | **17.64** | **19.02** | 18.60 | 20.34 |
| 75-05 | **8.92** | **9.83** | 10.80 | 11.92 |
| 75-06 | **9.52** | **10.08** | 12.40 | 13.84 |
| 75-07 | **14.04** | **14.86** | 14.96 | 16.77 |
| 75-08 | **17.96** | **19.12** | 20.52 | 21.65 |
| 75-09 | **6.64** | **7.95** | 9.12 | 10.52 |
| 75-10 | **11.72** | **12.97** | 16.52 | 18.39 |
| 80-01 | **6.40** | **7.11** | 7.80 | 8.41 |
| 80-02 | **110.92** | **112.89** | 113.12 | 115.61 |
| 80-03 | **13.24** | **13.82** | 15.64 | 16.67 |
| 80-04 | **34.72** | **35.68** | 34.88 | 36.65 |
| 80-05 | **19.16** | **20.09** | 23.28 | 24.79 |
| 80-06 | **9.44** | **10.40** | 12.76 | 14.29 |
| 80-07 | **15.40** | **16.36** | 19.20 | 20.86 |
| 80-08 | **13.60** | **14.68** | 14.48 | 15.82 |
| 80-09 | **86.40** | **87.62** | 89.40 | 92.24 |
| 80-10 | **12.28** | **13.79** | 15.64 | 16.53 |
| 85-01 | **6.20** | **7.02** | 7.76 | 8.63 |
| 85-02 | **29.76** | **30.53** | 32.56 | 34.16 |
| 85-03 | **36.76** | **37.64** | 38.40 | 41.20 |
| 85-04 | **23.48** | **24.02** | 26.12 | 27.24 |
| 85-05 | **50.20** | **51.20** | 52.08 | 53.26 |
| 85-06 | 61.84 | **62.65** | **61.16** | 62.94 |
| 85-07 | **11.60** | **12.15** | 13.80 | 14.51 |
| 85-08 | **19.72** | **20.56** | 21.00 | 22.00 |
| 85-09 | **55.04** | **56.34** | 55.00 | 56.93 |
| 85-10 | **22.20** | **23.73** | 25.72 | 27.85 |

Tabla II
Valores mínimos y promedio de las 30 ejecuciones de cada metaheurística aplicada a las instancias 90-*, clásicas, 200-*, 300-*, y 400-*.

| Instancias | GRASP | | VNS | |
|---|---|---|---|---|
| | Min | Med | Min | Med |
| 90-01 | **12.76** | **14.27** | 13.60 | 14.35 |
| 90-02 | **26.28** | **27.37** | 26.44 | 30.13 |
| 90-03 | **18.52** | **21.91** | 21.36 | 23.05 |
| 90-04 | **13.40** | **14.35** | 14.28 | 15.35 |
| 90-05 | 42.40 | 44.85 | **40.52** | **42.15** |
| 90-06 | **61.80** | **63.43** | 66.56 | 69.52 |
| 90-07 | 26.48 | 28.57 | **25.92** | **27.22** |
| 90-08 | **65.60** | **67.01** | 67.20 | 68.80 |
| 90-09 | **44.20** | **45.46** | 45.48 | 46.80 |
| 90-10 | **38.04** | **38.97** | 39.08 | 40.22 |
| p10_93 | 15.88 | 17.53 | **10.72** | **11.87** |
| p16_81 | 10.28 | 11.07 | **5.52** | **6.65** |
| p19_71 | 9.52 | 10.94 | **7.08** | **7.59** |
| p21_90 | 4.28 | 5.75 | **3.20** | **3.88** |
| p26_82 | 12.80 | 13.53 | **10.16** | **10.98** |
| p36_92 | 12.68 | 13.36 | **10.32** | **10.93** |
| p41_66 | 6.40 | 7.29 | **6.08** | **7.03** |
| p4_72 | 10.08 | 11.17 | **6.20** | **7.35** |
| p6_76 | 7.28 | 7.83 | **7.16** | **7.58** |
| pb_200_01 | 53.40 | 54.83 | **46.76** | **48.84** |
| pb_200_02 | 20.68 | 22.87 | **14.68** | **15.73** |
| pb_200_03 | 43.32 | 46.02 | **38.40** | **39.86** |
| pb_200_04 | 25.64 | 27.65 | **19.72** | **21.25** |
| pb_200_05 | 23.44 | 24.75 | **20.92** | **22.07** |
| pb_200_06 | 24.52 | 25.70 | **20.96** | **22.09** |
| pb_200_07 | 13.76 | 15.57 | **7.84** | **8.23** |
| pb_200_08 | 30.28 | 32.29 | **27.24** | **29.33** |
| pb_200_09 | 20.44 | 22.35 | **17.72** | **18.32** |
| pb_200_10 | 34.76 | 37.17 | **34.32** | **35.44** |
| pb_300_01 | 71.36 | 73.56 | **57.88** | **60.97** |
| pb_300_02 | 44.16 | 47.69 | **30.00** | **32.87** |
| pb_300_03 | **49.76** | **52.42** | 52.76 | 54.17 |
| pb_300_04 | 55.08 | 57.41 | **44.52** | **47.26** |
| pb_300_05 | 71.52 | 74.93 | **70.40** | **74.24** |
| pb_300_06 | 44.16 | 47.46 | **41.12** | **44.09** |
| pb_300_07 | 62.00 | 64.68 | **45.52** | **51.22** |
| pb_300_08 | 38.40 | 41.16 | **27.28** | **29.87** |
| pb_300_09 | 46.56 | 48.17 | **41.60** | **43.82** |
| pb_300_10 | 113.16 | 116.20 | **106.48** | **110.25** |
| pb_400_01 | 84.84 | 89.38 | **79.12** | **82.13** |
| pb_400_02 | 70.64 | 74.18 | **65.20** | **67.85** |
| pb_400_03 | 56.12 | 57.66 | **47.12** | **49.85** |
| pb_400_04 | 106.04 | 108.47 | **95.44** | **97.38** |
| pb_400_05 | 61.36 | 65.55 | **47.76** | **52.32** |
| pb_400_06 | 87.16 | 89.49 | **72.32** | **75.24** |
| pb_400_07 | 83.72 | 88.40 | **73.84** | **78.20** |
| pb_400_08 | 45.80 | 48.24 | **40.60** | **43.71** |
| pb_400_09 | 145.52 | 150.82 | **131.64** | **137.13** |
| pb_400_10 | 126.72 | 132.31 | **120.28** | **124.20** |

GRASP tenga algo más de diversidad que VNS, al cual la heurística voraz de su fase constructiva puede estar guiando a un óptimo local donde el plan de referencia es factible. Esta explicación es consistente con los resultados de la Tabla II, donde GRASP reduce mucho su rendimiento en favor de VNS. En este caso VNS consigue prácticamente la mayoría de los mejores resultados desde p10_93 en adelante. Dado que estas instancias tienen un requerimiento de sus opciones mayor, se incrementa la importancia de encontrar una secuencia factible para el plan de referencia. Además GRASP sólo consigue uno de los mejores resultados en instancias que tengan más de 200 vehículos, lo que podría indicar que la escalabilidad del problema afecta a su rendimiento.

## VI. Conclusiones

En este artículo hemos propuesto cómo solucionar r-CSP usando metaheurísticas constructivas, específicamente GRASP y VNS. Para ello hemos presentado nuestra propuesta para generar instancias de r-CSP a partir de las instancias de CSP tradicionales, que hemos tomado del benchmark de referencia CSPLib. Los resultados al aplicar las metaheurísticas usando las instancias extendidas para r-CSP muestran que GRASP es más competitivo en las instancias para las cuales es más sencillo encontrar una secuencia factible para el plan de referencia. En cambio pierde efectividad gradualmente al incrementar la dificultad de las instancias y finalmente es sobrepasado por VNS.

Nuestro trabajo futuro se centrará en incluir nuevas metaheurísticas constructivas como *Ant Colony Optimization* [27], que podrían además combinarse con nuestros procedimientos de búsqueda local ya presentados. Futuras extensiones de nuestro trabajo también considerarán la definición de métricas de robustez más avanzadas que evalúen la calidad de las soluciones más allá de la violación de restricciones del CSP tradicional.

## Agradecimientos

Este trabajo está financiado por el Ministerio de Economía y Competitividad bajo los proyectos NEWSOCO (ref. TIN2015-67661-P) y FHI-SELM2 (ref. TIN2014-57497-P), incluyendo Fondos Europeos de Desarrollo Regional (ERDF).

## Referencias

[1] B. D. Parrello, W. C. Kabat, and L. Wos, "Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem," *Journal of Automated reasoning*, vol. 2, no. 1, pp. 1–42, 1986.

[2] J. Bautista Valhondo, "Modelos y métricas para la versión robusta del car sequencing problem con flotas de vehículos especiales," *Dirección y organización*, vol. 60, no. 2016, pp. 57–65, 2016.

[3] J. Bautista, J. Pereira, and B. Adenso-Díaz, "A GRASP approach for the extended car sequencing problem," *Journal of Scheduling*, vol. 11, no. 1, pp. 3–16, 2008.

[4] C. C. Ribeiro, D. Aloise, T. F. Noronha, C. Rocha, and S. Urrutia, "An efficient implementation of a VNS/ILS heuristic for a real-life car sequencing problem," *European Journal of Operational Research*, vol. 191, no. 3, pp. 596–611, 2008.

[5] U. Golle, "Fitness landscape analysis and design of metaheuristics for car sequencing," *On the Car Sequencing Problem: Analysis and Solution Methods*, p. 100, 2011.

[6] M. Bergen, P. Van Beek, and T. Carchrae, "Constraint-based vehicle assembly line sequencing," *Advances in Artificial Intelligence*, pp. 88–99, 2001.

[7] M. Siala, E. Hebrard, and M.-J. Huguet, "A study of constraint programming heuristics for the car-sequencing problem," *Engineering Applications of Artificial Intelligence*, vol. 38, pp. 34–44, 2015.

[8] M. Gravel, C. Gagne, and W. L. Price, "Review and comparison of three methods for the solution of the car sequencing problem," *Journal of the Operational Research Society*, vol. 56, no. 11, pp. 1287–1295, 2005.

[9] M. Fliedner and N. Boysen, "Solving the car sequencing problem via branch & bound," *European Journal of Operational Research*, vol. 191, no. 3, pp. 1023–1042, 2008.

[10] J. Bautista, J. Pereira, and B. Adenso-Díaz, "A beam search approach for the optimization version of the car sequencing problem," *Annals of Operations Research*, vol. 159, no. 1, pp. 233–244, 2008.

[11] U. Golle, F. Rothlauf, and N. Boysen, "Iterative beam search for car sequencing," *Annals of Operations Research*, vol. 226, no. 1, pp. 239–254, 2015.

[12] J. Gottlieb, M. Puchta, and C. Solnon, "A study of greedy, local search, and ant colony optimization approaches for car sequencing problems," in *Applications of evolutionary computing*. Springer, 2003, pp. 246–257.

[13] S. Morin, C. Gagné, and M. Gravel, "Ant colony optimization with a specialized pheromone trail for the car-sequencing problem," *European Journal of Operational Research*, vol. 197, no. 3, pp. 1185–1191, 2009.

[14] A. Zinflou, C. Gagné, and M. Gravel, "Crossover operators for the car sequencing problem," in *Evolutionary Computation in Combinatorial Optimization*. Springer, 2007, pp. 229–239.

[15] ——, "Genetic algorithm with hybrid integer linear programming crossover operators for the car-sequencing problem," *INFOR: Information Systems and Operational Research*, vol. 48, no. 1, pp. 23–37, 2010.

[16] I. P. Gent and T. Walsh, "CSPLib: a benchmark library for constraints," in *Principles and Practice of Constraint Programming–CP'99*. Springer, 1999, pp. 480–481.

[17] M. Puchta and J. Gottlieb, "Solving car sequencing problems by local optimization," in *Applications of Evolutionary Computing*. Springer, 2002, pp. 132–142.

[18] B. Estellon, F. Gardi, and K. Nouioua, "Two local search approaches for solving real-life car sequencing problems," *European Journal of Operational Research*, vol. 191, no. 3, pp. 928–944, 2008.

[19] C. Solnon, "Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization," *European Journal of Operational Research*, vol. 191, no. 3, pp. 1043–1055, 2008.

[20] C. Solnon, V. D. Cung, A. Nguyen, and C. Artigues, "The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem," *European Journal of Operational Research*, vol. 191, no. 3, pp. 912–927, 2008.

[21] M. Prandtstetter and G. R. Raidl, "An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem," *European Journal of Operational Research*, vol. 191, no. 3, pp. 1004–1022, 2008.

[22] E.-G. Talbi, *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

[23] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *Journal of global optimization*, vol. 6, no. 2, pp. 109–133, 1995.

[24] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[25] A. A. Juan, J. Faulin, S. E. Grasman, M. Rabe, and G. Figueira, "A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems," *Operations Research Perspectives*, vol. 2, pp. 62–72, 2015.

[26] H. De Beukelaer, G. F. Davenport, G. De Meyer, and V. Fack, "JAMES: An object-oriented Java framework for discrete optimization using local search metaheuristics," *Software: Practice and Experience*, vol. 47, no. 6, pp. 921–938, 2017.

[27] M. Dorigo and T. Stützle, "Ant colony optimization: overview and recent advances," *Handbook of metaheuristics*, 2010.