Un procedimiento de búsqueda adaptativa voraz aleatorizada con búsqueda por entornos variables y reencademiento de trayectorias para el problema de ruteo y localización capacitado

The Badass Optimizers— SMIO–Hexaly Location Routing Challenge Monterrey 2026

Resumen

Este documento describe de forma completa el algoritmo desarrollado para el *Capacitated Location Routing Problem* (CLRP) del reto SMIO–Hexaly 2026. El método integra cuatro componentes ejecutados secuencialmente: una construcción *GRASP* basada en el algoritmo de ahorros de Clarke-Wright; una búsqueda local de descenso por vecindarios variables (VND) con cinco operadores y *listas de vecinos granulares* que la escalan a instancias grandes; una metaheurística *Variable Neighborhood Search* (VNS) con sacudidas escaladas por k para intensificación; y una fase final de *Path Relinking* que recombina soluciones élite alimentadas por GRASP y VNS. Se especifica cada componente, junto con la corrección necesaria para instancias de matriz asimétrica. La brecha promedio contra la solución de referencia de Hexaly es de -1.65%, manteniendo la factibilidad en todas las instancias.

Índice

1 Introducción

El *Capacitated Location Routing Problem* (CLRP) integra dos decisiones que tradicionalmente se abordan por separado: la localización estratégica de depósitos y el ruteo operativo de vehículos. Dado un conjunto de depósitos potenciales —cada uno con un costo de apertura, una capacidad y un límite de vehículos— y un conjunto de clientes con demanda conocida, el objetivo es decidir simultáneamente qué depósitos abrir, cómo asignar clientes a esos depósitos y qué rutas diseñar, de modo que se minimice el costo total.

La dificultad del CLRP proviene de que ningún paradigma algorítmico domina por sí solo: las decisiones de localización son combinatorias de gran escala y las de ruteo son NP-difíciles. Por ello, los enfoques exitosos suelen ser *matheurísticas* o metaheurísticas híbridas. El algoritmo aquí descrito sigue esa línea: una construcción golosa aleatorizada (GRASP) genera soluciones factibles diversas, una búsqueda local las lleva a óptimos locales, y una búsqueda de vecindario variable (VNS) escapa de esos óptimos locales para intensificar la búsqueda.

El resto del documento se organiza así: la Sección 2 formaliza el problema; la Sección 3 da una visión general del algoritmo; las Secciones 4–7 detallan cada componente; la Sección 9 describe la verificación de factibilidad; y la Sección 10 presenta la comparación experimental antes y después del VNS.

2 Definición del problema

2.1 Conjuntos y parámetros

- $I = \{1, \ldots, m\}$: conjunto de depósitos potenciales.
- $J = \{1, \ldots, n\}$: conjunto de clientes.

- Para cada depósito $i \in I$: coordenadas (x_i^d, y_i^d), costo de apertura f_i, capacidad W_i (demanda total máxima que puede atender) y número máximo de vehículos V_i.
- Para cada cliente $j \in J$: coordenadas (x_j^c, y_j^c) y demanda $q_j > 0$.
- Q: capacidad de cada vehículo (idéntica para toda la flota).
- g: costo fijo por cada ruta despachada.
- $d(a, b)$: distancia de viaje entre dos nodos $a, b \in I \cup J$.

2.2 Variables de decisión

- **Apertura de depósitos:** qué depósitos $i \in I$ abrir ($y_i \in \{0, 1\}$).
- **Asignación de clientes:** cada cliente se asigna a exactamente un depósito abierto.
- **Diseño de rutas:** para cada depósito abierto, un conjunto de rutas que inician y terminan en él y visitan a los clientes asignados.

2.3 Función objetivo

Sea K el conjunto de todas las rutas y $\text{dist}(R_k)$ la distancia de la ruta k (depósito $\to$ primer cliente $\to \cdots \to$ último cliente $\to$ depósito). Se minimiza el costo total

$$\min \underbrace{\sum_{i \in I} f_i \, y_i}_{\text{apertura}} + \underbrace{g \, |K|}_{\text{despacho de rutas}} + \underbrace{\sum_{k \in K} \text{dist}(R_k)}_{\text{distancia}} . \tag{1}$$

2.4 Restricciones

1. **Servicio:** cada cliente es visitado exactamente una vez por exactamente una ruta.
2. **Asignación:** cada ruta opera desde un único depósito abierto, y todos sus clientes están asignados a ese depósito.
3. **Capacidad de vehículo:** $\sum_{j \in R_k} q_j \le Q$ para toda ruta k.
4. **Capacidad de depósito:** $\sum_{j \in S_i} q_j \le W_i$ para todo depósito abierto i, donde S_i es el conjunto de clientes asignados a i.
5. **Límite de vehículos:** el número de rutas que operan desde i no excede V_i, para todo depósito abierto i.

2.5 Modelo de distancias y regla de redondeo

Para instancias euclidianas (escalas pequeña y mediana) las distancias se calculan con redondeo *por par*, antes de sumar:

$$d(a, b) = \text{round}\Big(\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2},\, 1 \Big) . \tag{2}$$

El costo de una ruta es la suma de estas distancias ya redondeadas; no se aplica redondeo adicional. La función $\text{round}(\cdot, 1)$ redondea a un decimal con la regla *ties-to-even* (redondeo bancario), idéntica a `round(x,1)` de Python, que usa el verificador oficial. Para instancias grandes con matriz explícita (red vial de Monterrey, posiblemente asimétrica) las distancias se toman directamente de la matriz, sin redondeo.

3 Visión general del algoritmo

El algoritmo opera en **cuatro fases** sobre un presupuesto de tiempo fijo, ejecutadas siempre en el mismo orden. En la **fase de preparación** se lee la instancia, se materializa la matriz de distancias y se construyen las listas de vecinos granulares. En la **fase de construcción** se ejecuta un GRASP multiarranque: cada iteración construye una solución con Clarke-Wright aleatorizado y la mejora con la búsqueda local (VND), conservando la mejor y guardando cada óptimo local factible en un *pool élite*. En la **fase de intensificación por VNS** la mejor solución del GRASP se somete a un ciclo de sacudidas escaladas por k y descenso VND; cada nuevo incumbente alimenta también el pool élite. En la **fase de recombinación por Path Relinking** se toman pares de soluciones élite y se enlazan a nivel de asignación cliente–depósito, quedándose con la mejor solución del camino y refinándola con la búsqueda local. La Figura 1 resume el flujo y el Algoritmo 1 lo formaliza.

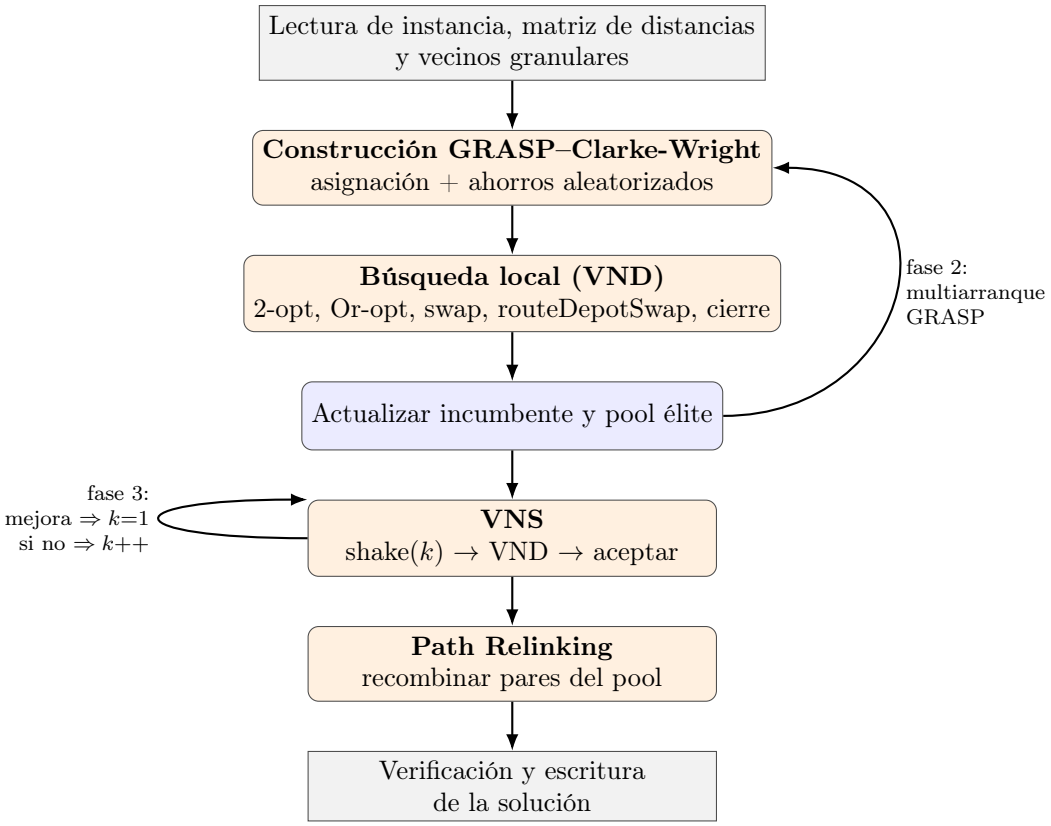

Figura 1: Flujo general: preparación, construcción GRASP multiarranque, intensificación por VNS y recombinación por Path Relinking.

4 Lectura de instancias y modelo de distancias

El lector interpreta el formato de texto del reto (encabezados `NAME`, `CUSTOMERS`, `DEPOTS`, `VEHICLE_CAPACITY`, `ROUTE_FIXED_COST`, `DISTANCE_FORMAT`, seguidos de las secciones de depósitos, clientes y, en su caso, la matriz de distancias). Los identificadores son globales: los depósitos ocupan $1, \ldots, m$ y los clientes $m{+}1, \ldots, m{+}n$.

Para acelerar los millones de consultas de distancia que hacen la construcción y la búsqueda local, se materializa una **matriz de distancias** de tamaño $(m{+}n) \times (m{+}n)$: en instancias euclidianas se calcula una vez con la Ecuación (2); en instancias con matriz explícita se copia directamente. Así, cada consulta $d(a, b)$ es $O(1)$. Esta materialización también permite que el mismo código funcione, sin cambios, para distancias simétricas y asimétricas.

Algoritmo 1 Esquema general (cuatro fases)

Entrada: instancia I, presupuesto de tiempo T, fracción GRASP γ, fracción PR π, K vecinos granulares, $k_{\max}$, pool P
Salida: mejor solución factible encontrada

▷ **Fase 1: Preparación**
cargar I, materializar matriz de distancias, construir listas de K vecinos
$x^\star \leftarrow \varnothing, \quad f^\star \leftarrow +\infty, \quad \mathcal{E} \leftarrow \varnothing$ ▷ $\mathcal{E}$: pool élite
$t_1 \leftarrow \gamma T, \quad t_2 \leftarrow t_1 + (T - t_1)(1 - \pi)$

▷ **Fase 2: GRASP**
mientras tiempo transcurrido $< t_1$ **hacer**
  $x \leftarrow$ ClarkeWrightGRASP(I)
  $x \leftarrow$ BusquedaLocal(x)
  si x factible **entonces**
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{x\}$ ▷ recortando al tamaño P por peor costo
    si costo(x) $< f^\star$ **entonces** $x^\star \leftarrow x,\ f^\star \leftarrow$ costo(x)
    fin si
  fin si
fin mientras

▷ **Fase 3: VNS**
$x^\star \leftarrow \mathrm{VNS}(x^\star,\ t_2,\ \mathcal{E})$ ▷ cada nuevo incumbente entra a $\mathcal{E}$

▷ **Fase 4: Path Relinking**
$x^\star \leftarrow$ PathRelinking($\mathcal{E},\ T$)
devolver $x^\star$

4.1 Listas de vecinos granulares

Antes de comenzar la construcción se precomputa, para cada cliente $c \in J$, una lista con sus K vecinos más cercanos (por distancia dirigida saliente):

$$\mathcal{N}_K(c) = \{v_1, \ldots, v_K\} \quad \text{tal que} \quad d(c, v_1) \le d(c, v_2) \le \cdots \le d(c, v_K). \tag{3}$$

Estas listas son la base del *granular search* [**?**]: los operadores de la búsqueda local que reubican o intercambian clientes solo consideran posiciones adyacentes a esos K vecinos, no todas las rutas y posiciones. Esto reduce el costo de cada barrido de $O(n^2)$ a $O(n \cdot K)$, lo que hace viable escalar a instancias grandes ($n \ge 500$). El valor de K es adaptativo por tamaño: $K = 20$ para $n \le 300$, $K = 25$ para $300 < n \le 700$, $K = 30$ para $700 < n \le 1500$ y $K = 35$ para $n > 1500$. Se construyen una única vez tras la lectura, en tiempo $O(n^2 \log K)$, y se cachean para el resto de la ejecución.

5 Construcción: GRASP–Clarke-Wright

5.1 El marco GRASP

Un GRASP (*Greedy Randomized Adaptive Search Procedure*) construye soluciones repitiendo una construcción golosa *aleatorizada*: en cada paso, en lugar de tomar siempre el mejor candidato, se forma una **lista restringida de candidatos** (RCL) con los candidatos suficientemente buenos según una *función greedy*, y se elige uno al azar. El parámetro $\alpha \in [0, 1]$ regula el tamaño de la RCL: $\alpha = 0$ es puro voraz (determinista) y valores mayores introducen diversidad entre iteraciones. La construcción del CLRP procede en dos fases, cada una con su propia función greedy.

5.2 Fase 1: asignación cliente–depósito

La primera fase asigna cada cliente a un depósito. La función greedy es la **distancia** cliente–depósito:

$$g(j, i) = d(i, j) \quad \text{(se minimiza)}. \tag{4}$$

Los candidatos de un cliente j son los depósitos con capacidad residual suficiente para q_j. Es clave que la capacidad efectiva de un depósito se acota por

$$\widehat{W}_i \;=\; \min(W_i,\; V_i \cdot Q)\,, \tag{5}$$

pues un depósito nunca puede atender más que V_i cargas completas de vehículo; acotar así reduce drásticamente la probabilidad de violar el límite de vehículos tras el ruteo. La RCL se forma por umbral de distancia $\text{thr} = d_{\min} + \alpha_{\text{asig}}(d_{\max} - d_{\min})$, eligiendo al azar un depósito entre los que quedan por debajo del umbral. Los clientes se procesan en orden aleatorio para diversificar entre iteraciones. En la implementación, $\alpha_{\text{asig}} = 0.30$ por defecto (opción `-aa`).

5.3 Fase 2: Clarke-Wright aleatorizado por depósito

Para los clientes de cada depósito se aplica el algoritmo de ahorros de Clarke-Wright, aleatorizado. Se parte de una ruta por cliente y se fusionan rutas mientras haya ahorro positivo. La función greedy es el **ahorro clásico**:

$$s(i, j) = d(o, i) + d(o, j) - d(i, j), \tag{6}$$

donde o es el depósito e i, j son extremos de rutas distintas. Intuitivamente, $s(i, j)$ mide cuánto se ahorra al unir dos rutas conectando i con j en lugar de que cada cliente salga y regrese solo al depósito. Los candidatos son las fusiones factibles (ambos en extremos y carga combinada $\leq Q$) con ahorro positivo, ordenadas de forma descendente. La RCL usa el umbral $\text{thr} = s_{\max} - \alpha_{\text{ahorro}}(s_{\max} - s_{\min})$ y se elige una fusión al azar entre las de mayor ahorro. Con $\alpha_{\text{ahorro}} = 0$ se recupera el Clarke-Wright determinista. En la implementación, $\alpha_{\text{ahorro}} = 0.20$ por defecto (opción `-as`).

El ahorro simétrico de la Ecuación (6) se usa solo para *ordenar y elegir*; el costo real de la fusión se evalúa probando las cuatro concatenaciones posibles de los extremos de ambas rutas y tomando la de menor distancia. Esto mantiene la construcción correcta incluso con distancias **asimétricas**.

5.4 Reparación del límite de vehículos en construcción

Si tras Clarke-Wright un depósito conserva más de V_i rutas, se realizan fusiones *forzadas* (aunque el ahorro sea negativo) respetando Q, eligiendo en cada paso el par de rutas con mayor ahorro, hasta cumplir V_i o hasta que la capacidad lo impida.

6 Búsqueda local (VND)

La búsqueda local lleva una solución a un óptimo local recorriendo varios vecindarios. Es de **primera mejora**: en cuanto un operador halla un movimiento que reduce el costo, lo aplica y se reinicia el barrido. Su eficiencia descansa en tres ideas: (i) ningún movimiento recalcula el costo total, sino solo su **delta** (aristas que quita menos aristas que pone, más los términos de g y f_i afectados); (ii) dos arreglos con la carga y el número de rutas por depósito se mantienen de forma incremental, de modo que las verificaciones de factibilidad entre depósitos son $O(1)$; y (iii) los operadores que mueven o intercambian clientes usan las **listas de vecinos granulares** de la Sección 4.1: en vez de escanear todas las rutas y posiciones, un cliente u solo se considera para insertarse o intercambiarse junto a sus vecinos $\mathcal{N}_K(u)$. Un índice cliente $\rightarrow$ (ruta, posición) permite resolver esas posiciones en $O(1)$.

Algoritmo 2 ClarkeWrightGRASP

Entrada: instancia; parámetros $\alpha_{\text{asig}}, \alpha_{\text{ahorro}}$
Salida: solución factible

Fase 1: asignar cada cliente (en orden aleatorio) a un depósito de su RCL de distancia, respetando $\widehat{W}_i$
para cada depósito abierto o **hacer**
    inicializar una ruta $[c]$ por cada cliente c asignado a o
    calcular los ahorros $s(i,j)$ de la Ec. (6) y ordenarlos
    mientras exista fusión factible con ahorro positivo **hacer**
        construir la RCL de fusiones y elegir una al azar
        fusionar tomando la mejor de las 4 concatenaciones de extremos
    fin mientras
    reparar V_o con fusiones forzadas si hay más de V_o rutas
fin para
devolver unión de las rutas de todos los depósitos

6.1 2-opt intra-ruta

Dentro de una sola ruta, invierte un segmento para deshacer cruces. Reemplazar las aristas (a,b) y (c,d) por (a,c) y (b,d) mejora si

$$d(a,c) + d(b,d) \;<\; d(a,b) + d(c,d). \tag{7}$$

6.2 Or-opt: reubicación de cadenas

Mueve una cadena de 1 a 3 clientes consecutivos de una ruta a otra posición —en la misma ruta o en otra, incluso de otro depósito—, probando inserción directa e invertida. Para una cadena con extremos (f,ℓ) y longitud interna L, el costo de sacarla o insertarla entre dos nodos p, q es

$$\Delta(p,f,\ell,q) = d(p,f) + L + d(\ell,q) - d(p,q). \tag{8}$$

El delta total combina el ahorro por remover y el costo por insertar; si la ruta origen queda vacía, se descuenta g y, si con ella se cierra un depósito, también f_i. En modo granular, las posiciones de inserción se restringen a las adyacentes a los vecinos $\mathcal{N}_K(f) \cup \mathcal{N}_K(\ell)$.

6.3 Intercambio inter-ruta

Intercambia dos clientes u, v de rutas distintas. Con vecinos (p_u, q_u) y (p_v, q_v), mejora si

$$d(p_u,v) + d(v,q_u) + d(p_v,u) + d(u,q_v) \;<\; d(p_u,u) + d(u,q_u) + d(p_v,v) + d(v,q_v), \tag{9}$$

y ambas rutas siguen respetando Q (y W_i si cruzan depósitos). En modo granular, para cada u solo se prueban intercambios con $v \in \mathcal{N}_K(u)$.

6.4 `routeDepotSwap`: reasignación de una ruta a otro depósito

El costo de una ruta se descompone en dos *piernas de depósito* (depósito → primer cliente y último cliente → depósito), la cadena interna entre clientes y su parte del costo de apertura. Reasignar la ruta *completa* a otro depósito *sin reordenar* a los clientes deja intacta la cadena interna: solo cambian las dos piernas y, en su caso, costos de apertura/cierre. Sea i el depósito actual, i_2 el nuevo, f el primer cliente y ℓ el último. El delta es

$$\Delta = \big[d(i_2,f) + d(\ell,i_2)\big] - \big[d(i,f) + d(\ell,i)\big] + f_{i_2}\,[\,i_2 \text{ cerrado}\,] - f_i\,[\,\text{es la última ruta de } i\,]. \tag{10}$$

El operador recorre cada ruta contra cada otro depósito ($O(|K| \cdot m)$ por barrido), y como cada evaluación es $O(1)$ resulta muy económico. Llena un vacío que los operadores por cliente no cubren: tras las fusiones de Clarke-Wright, el "centro de masa" de una ruta se desplaza y con frecuencia otro depósito queda más cerca de *ambos* extremos, o conviene consolidarla en un depósito ya abierto. Este operador re-aloja la ruta entera de una sola vez.

6.5 Cierre y consolidación de depósitos

Intenta vaciar por completo cada depósito abierto reinsertando todos sus clientes en rutas de otros depósitos (mediante inserción más barata, $d(p,u) + d(u,q) - d(p,q)$); acepta el cambio solo si el costo total disminuye. Es el operador que captura el ahorro del costo de apertura f_i.

6.6 Corrección para distancias asimétricas

Los operadores que *invierten* un tramo de ruta requieren cuidado especial cuando las distancias son asimétricas ($d(a,b) \neq d(b,a)$), como en las instancias grandes con matriz explícita. Al invertir un segmento no solo cambian las dos aristas frontera: también se invierten todas las aristas *internas* del segmento, cuyo costo cambia. Un delta que solo considere las aristas frontera (válido en el caso simétrico) queda incorrecto y puede llevar a aceptar y deshacer el mismo movimiento en un ciclo infinito. Por ello, cuando `DISTANCE_FORMAT: FULL_MATRIX`, el 2-opt y la inserción invertida del Or-opt suman explícitamente el costo interno en el sentido invertido:

$$\text{interno}_{\text{inv}} = \sum_{t} d(c_{t+1}, c_t), \tag{11}$$

frente al interno directo $\sum_t d(c_t, c_{t+1})$ del caso original. Así cada movimiento reduce el costo real y la búsqueda converge. La lectura de la matriz sigue la misma indexación dirigida que el verificador oficial (matriz$[a-1][b-1]$), por lo que no introduce error alguno.

6.7 Reparación del límite de vehículos

No optimiza, sino que *garantiza factibilidad*: si un depósito supera V_i, disuelve su ruta más ligera reinsertando sus clientes en otras rutas (mismo u otro depósito), o en una ruta nueva de un depósito con vehículos disponibles.

6.8 Estrategia de control

Los operadores se organizan como un *Variable Neighborhood Descent* (VND): se aplican en cadena y, cuando el conjunto "fino" se estanca, se intenta el cierre de depósitos; si éste consigue algo, se relanza toda la cadena. El proceso termina en un óptimo local respecto a todos los vecindarios (Algoritmo 3).

Algoritmo 3 BusquedaLocal (VND)

Entrada: solución x
Salida: óptimo local respecto a todos los vecindarios

```
reparar Vehiculos(x)
repetir
    repetir
        aplicar 2-opt, Or-opt, swap y routeDepotSwap a primera mejora
    hasta ningún operador fino mejore
    intentar cierre/consolidación de depósitos
hasta ni los operadores finos ni el cierre logren mejora
devolver x
```

7 Metaheurística: Variable Neighborhood Search

7.1 Concepto

La búsqueda local se detiene en un óptimo local que lo es *solo respecto a sus vecindarios*. El VNS explota una observación central:

> Un óptimo local para un vecindario casi nunca es óptimo local para otro vecindario distinto.

Por tanto, en lugar de rendirse, el VNS cambia de vecindario de manera sistemática. Combina tres ingredientes: una **sacudida** (*shake*) que salta de forma aleatoria y controlada a una solución cercana pero distinta; una **búsqueda local** que desciende a un nuevo óptimo local; y una regla de aceptación con k **variable**: si el nuevo óptimo mejora, se adopta y se reinicia $k = 1$ (saltos pequeños para intensificar); si no, se aumenta k (saltos mayores para diversificar).

7.2 Sacudida (*shake*)

La sacudida define una familia de vecindarios ordenados por agresividad creciente con k, de modo que el "k variable" no solo aumenta la *cantidad* de perturbación sino que cambia su *naturaleza*:

- $k = 1, 2$: movimientos ligeros —reubicar un cliente al azar, intercambiar dos clientes, o reasignar una ruta completa a otro depósito (idea de `routeDepotSwap`)—, uno o tres según k.
- $k = 3$: **abrir un depósito cerrado**, migrando hacia él sus clientes más cercanos (cada uno en una ruta nueva que la búsqueda local fusiona después).
- $k = 4$: **cerrar un depósito abierto**, redistribuyendo todos sus clientes por inserción más barata en otros depósitos (abriendo uno nuevo si hace falta).
- $k \geq 5$: **intercambio de depósitos**, abriendo uno cerrado y cerrando otro abierto.

Los movimientos de $k \geq 3$ son perturbaciones estructurales grandes: reconfiguran qué depósitos están abiertos y obligan a la búsqueda local a *recalcular las rutas* desde una topología distinta — justo el tipo de sacudida agresiva que permite explorar decisiones de localización, no solo de ruteo. Todos los movimientos respetan Q y W_i; si uno viola V_i, la reparación interna de la búsqueda local lo corrige, y si un movimiento agresivo resulta infactible se revierte y se aplica una sacudida ligera. Un k mayor produce, así, un punto de reinicio más alejado y estructuralmente más distinto.

7.3 k variable, alimentación del pool élite y GVNS

Como la búsqueda local interna es en sí un VND, el esquema resultante es un *General VNS* (GVNS): VND por dentro y sacudidas por fuera. El VNS se aplica en la fase 3 sobre la mejor solución del GRASP hasta agotar el tiempo asignado. Un detalle clave para la fase siguiente: **cada óptimo local factible que produce el VNS se inserta en el pool élite $\mathcal{E}$** (con el criterio descrito en la Sección 8). Esto asegura que el Path Relinking posterior disponga de soluciones *fuertes y diversas*, no solo de las soluciones de GRASP; sin esta alimentación, el pool se queda con material débil y el PR pierde eficacia. El Algoritmo 4 formaliza el ciclo.

8 Recombinación: Path Relinking

8.1 Concepto y motivación

Los primeros tres componentes son métodos de *trayectoria* (una única solución evoluciona con perturbación y descenso). Path Relinking (PR) añade una dimensión *poblacional*: mantiene

Algoritmo 4 VNS (GVNS básico con alimentación de pool)

Entrada: solución inicial x, $k_{\max}$, fecha límite, pool $\mathcal{E}$
Salida: mejor solución encontrada

$x \leftarrow \textsc{BusquedaLocal}(x)$; $x^\star \leftarrow x$
insertar $x^\star$ en $\mathcal{E}$
repetir
  $k \leftarrow 1$
  mientras $k \le k_{\max}$ **y** no se agote el tiempo **hacer**
    $x' \leftarrow \textsc{Shake}(x^\star, k)$ ▷ perturbación de fuerza k
    $x'' \leftarrow \textsc{BusquedaLocal}(x')$
    si x'' factible **entonces**
      insertar x'' en $\mathcal{E}$ ▷ alimentación de material élite
      si costo(x'') < costo$(x^\star)$ **entonces**
        $x^\star \leftarrow x''$; $k \leftarrow 1$ ▷ mejora: reiniciar
        continuar
      fin si
    fin si
    $k \leftarrow k + 1$ ▷ sin mejora: sacudir más fuerte
  fin mientras
hasta se agote el tiempo
devolver $x^\star$

un *pool élite* de soluciones buenas y diversas, y genera nuevas soluciones **recombinando pares**. Dadas una solución inicial x_s y una solución guía x_g, PR construye un camino $x_s = z_0, z_1, \ldots, z_T = x_g$ transformando z_i en z_{i+1} mediante movimientos que reducen la *distancia* entre z_i y x_g; en cada paso se evalúa el costo y al final se devuelve el mejor $z_i^\star$ del camino, opcionalmente pulido con búsqueda local.

Para el CLRP definimos la distancia entre dos soluciones a nivel de *asignación cliente–depósito*: si $a_x(c)$ denota el depósito al que la solución x asigna al cliente c, entonces

$$d_{\text{asig}}(x, y) = \big|\{c \in J : a_x(c) \neq a_y(c)\}\big|. \tag{12}$$

Esta elección es natural para el LRP porque captura las decisiones *estratégicas* (qué depósitos están abiertos y quién sirve a quién), que son precisamente las que ni las trayectorias del VNS ni la búsqueda local por sí solas recombinan sistemáticamente.

8.2 Gestión del pool

El pool $\mathcal{E}$ tiene capacidad P (por defecto 8). Al intentar insertar una solución z:

1. Si z tiene el mismo costo (dentro de 10^{-4}) que alguna solución ya en $\mathcal{E}$, se descarta como duplicado.
2. Si $|\mathcal{E}| < P$, se agrega.
3. Si $|\mathcal{E}| = P$ y costo(z) mejora al peor elemento del pool, ese peor elemento se reemplaza por z.

El pool se alimenta tanto de las soluciones que produce el GRASP en la fase 2 como de los óptimos locales que produce el VNS en la fase 3, de modo que combina diversidad (múltiples arranques aleatorizados) con calidad (óptimos locales fuertes).

8.3 Operador de enlace

Dado un par (x_s, x_g) del pool, el enlace opera así (Algoritmo 5): sea $D = \{c \in J : a_{x_s}(c) \neq a_{x_g}(c)\}$ el conjunto de clientes con asignación distinta. En un orden aleatorizado sobre D, se toma cada cliente c y se **mueve al depósito que tiene en la guía** x_g, insertándolo por inserción más barata en alguna ruta existente de ese depósito, o en una ruta nueva si ninguna admite al cliente. Cada movimiento respeta Q y W_i; si un cliente no puede moverse, se salta y se continúa. Se lleva registro de la mejor solución $z^\star$ vista durante el camino; al final, $z^\star$ se pule con la búsqueda local (VND) y se intenta insertar en el pool. En cada iteración de la fase 4 se elige la mejor solución del pool como uno de los extremos y otra aleatoria como el otro, y el enlace se aplica en ambos sentidos (hacia adelante y hacia atrás) para explorar caminos distintos.

Algoritmo 5 PathRelinking sobre el pool $\mathcal{E}$

Entrada: pool $\mathcal{E}$ (tamaño ≥ 2), fecha límite
Salida: mejor solución del pool tras la recombinación

mientras no se agote el tiempo **y** $|\mathcal{E}| \geq 2$ **hacer**
  $x_b \leftarrow \arg\min_{x \in \mathcal{E}} \text{costo}(x)$
  elegir $x_r \in \mathcal{E} \setminus \{x_b\}$ al azar
  for all $(x_s, x_g) \in \{(x_b, x_r), (x_r, x_b)\}$ **hacer**
    $D \leftarrow \{c \in J : a_{x_s}(c) \neq a_{x_g}(c)\}$
    $z \leftarrow$ copia de x_s; $\quad z^\star \leftarrow z$
    for all $c \in D$ en orden aleatorio **hacer**
      mover c al depósito $a_{x_g}(c)$ en z (inserción más barata; respetar Q, W)
      si costo$(z) <$ costo$(z^\star)$ **entonces** $z^\star \leftarrow z$
      fin si
    fin para
    $z^\star \leftarrow$ BusquedaLocal$(z^\star)$
    intentar insertar $z^\star$ en $\mathcal{E}$
  fin para
fin mientras
devolver $\arg\min_{x \in \mathcal{E}} \text{costo}(x)$

8.4 Rol dentro del pipeline

El PR se aplica *después* del VNS por dos razones. Primero, el pool necesita material fuerte para que el enlace produzca soluciones competitivas —enlaces entre soluciones débiles casi nunca mejoran—, y los óptimos locales del VNS son exactamente ese material. Segundo, si el PR compite con el VNS por tiempo de manera muy agresiva (fracciones altas de π), el rendimiento empeora en instancias donde el VNS aún no ha convergido; con la fracción por defecto $\pi = 0.20$, el PR aporta mejoras adicionales sin perjudicar la fase de intensificación.

9 Factibilidad y verificación

Toda solución candidata se somete a un verificador interno que replica las seis condiciones del reto: (1) cada cliente aparece en exactamente una ruta; (2) cada ruta se asigna a un depósito abierto; (3) la carga de cada ruta no excede Q; (4) la carga de cada depósito no excede W_i; (5) el número de rutas por depósito no excede V_i; y (6) el costo reportado coincide con el recomputado dentro de una tolerancia de 10^{-4}. En la fase experimental, además, todas las soluciones se validaron con el *código oficial* del reto: el costo reportado por nuestro solver coincide exactamente con el recomputado por el verificador oficial en todas las instancias, lo que confirma que la regla de redondeo de la Sección 2.5 está correctamente implementada.

10 Resultados computacionales

10.1 Instancias

Se emplearon las diez instancias de práctica del reto (Tabla 1), que cubren escalas pequeña y mediana con distintos patrones de holgura de capacidad, distribución de clientes y estructura de acoplamiento depósito/vehículo.

Tabla 1: Instancias de práctica.

Instancia	Escala	Clientes	Depósitos
`mock_small_consolidation`	Pequeña	90	5
`mock_small_loose`	Pequeña	220	11
`mock_small_loose_clustered`	Pequeña	240	12
`mock_small_asym_depot_binding`	Pequeña	300	12
`mock_small_moderate`	Pequeña	320	14
`mock_small_asym_vehicle_binding`	Pequeña	340	14
`mock_small_tight`	Pequeña	360	15
`mock_medium_loose`	Mediana	640	21
`mock_medium_moderate`	Mediana	820	26
`mock_medium_tight`	Mediana	900	28

10.2 Configuración experimental

Cada instancia se resolvió **5 veces**, con semillas $s = 1, \ldots, 5$ y un presupuesto de 240 segundos de tiempo de pared por corrida, conservando la mejor solución factible. Se usó la configuración por defecto del solver: $\alpha_{\text{asig}} = 0.30$, $\alpha_{\text{ahorro}} = 0.20$, fracción GRASP $\gamma = 0.30$, fracción PR $\pi = 0.20$, $k_{\max} = 5$ —con lo que se activan las sacudidas agresivas de depósitos—, tamaño de pool $P = 8$ y K vecinos granulares adaptativo ($K = 20$ para $n \leq 300$, $K = 25$ para $n \leq 700$, $K = 30$ para $n \leq 1500$). El reparto de tiempo por corrida es aproximadamente 36 s para el GRASP (fase 2), 67 s para el VNS (fase 3) y 17 s para el Path Relinking (fase 4). La columna *Hexaly* es la solución de referencia publicada por el reto; la brecha se define como gap = (costo − Hexaly)/Hexaly, de modo que un valor negativo indica que se supera a Hexaly.

10.3 Resultados

La Tabla 2 presenta la mejor solución de las 5 semillas por instancia. El método *iguala o supera* a la referencia industrial de Hexaly en **10 de las 10** instancias, con una brecha promedio de -1.65%.

10.4 Escalado a instancias grandes

Para verificar el escalado del método a instancias fuera del conjunto de prueba (que llegan a $n = 900$) se generaron dos instancias adicionales con el generador oficial en modo `FULL_MATRIX` y distancias simuladas asimétricas: $n = 500$, $m = 15$ y $n = 1500$, $m = 30$. Ambas producen soluciones factibles según el verificador oficial. La Tabla 3 muestra el efecto de las listas de vecinos granulares en $n = 1500$ (semilla 1): antes de introducirlas, el número de iteraciones GRASP alcanzadas a esa escala era muy bajo (4 iteraciones en 120 s) y duplicar el tiempo apenas mejoraba 0.06%; con vecinos granulares se alcanzan 26 iteraciones en el mismo tiempo y 51 en 240 s, y la mejora al duplicar el tiempo pasa a 0.9%. Es decir, el escalado del método con el presupuesto de tiempo, que se había perdido en instancias grandes, se restaura.

Tabla 2: Mejor solución de 5 semillas ($s = 1, \ldots, 5$; 240 s por corrida) frente a la referencia de Hexaly. Brecha negativa significa que se supera a Hexaly. Todas las soluciones son factibles según el verificador oficial.

Instancia	Mejor costo	Semilla	Hexaly	Brecha
`consolidation`	16 271.05	1	16 271.05	**0.00%**
`loose`	29 101.97	2	29 600.47	**−1.68%**
`loose_clustered`	23 702.31	4	23 730.63	**−0.12%**
`asym_depot`	36 433.35	1	37 436.32	**−2.68%**
`moderate`	39 543.51	3	40 181.44	**−1.59%**
`asym_vehicle`	40 464.40	4	41 559.40	**−2.63%**
`tight`	40 738.82	3	41 676.97	**−2.25%**
`medium_loose`	55 507.35	5	56 457.52	**−1.68%**
`medium_moderate`	71 386.71	1	73 582.52	**−2.98%**
`medium_tight`	64 194.25	7	64 802.84	**−0.94%**
Promedio				**−1.65%**

Tabla 3: Efecto de las listas de vecinos granulares en la instancia n=1500, m=30 (semilla 1). Todas las soluciones son factibles según el verificador oficial.

Tiempo	Sin granular	Con granular	Iteraciones GRASP
120 s	7 361 002	7 406 181	$4 \to 26$
240 s	7 356 340	**7 336 990**	$8 \to 51$

11 Conclusiones

Se presentó un algoritmo completo para el CLRP que integra cuatro fases ejecutadas en el orden fijo Preparación → GRASP → VNS → Path Relinking: construcción GRASP–Clarke-Wright, búsqueda local VND con cinco operadores acelerada por listas de vecinos granulares, metaheurística GVNS con sacudidas escaladas por k que abarcan reestructuración de depósitos, y una recombinación por Path Relinking sobre un pool élite alimentado por el GRASP y por los incumbentes del VNS. La factibilidad se verifica internamente y se contrasta contra el verificador oficial del reto en todas las instancias reportadas. La brecha promedio contra la referencia industrial es de -1.65% con 5 semillas y 240 s por corrida, igualando o superando a Hexaly en 10 de las 10 instancias. Las listas de vecinos granulares hacen viable escalar el método a instancias de $n = 1500$ manteniendo la factibilidad y permitiendo que el tiempo adicional se traduzca en mejoras reales.