VRP-Informe
Diseño y Análisis de Algoritmos

DANIEL HERNÁNDEZ DE LEÓN

May 2022

1 Introduction

Esta práctica se basa en la generación de soluciones con componente aleatorio, ayudándonos de búsquedas locales para maximizar o minimizar la función objetivo.

Se utilizarán los algoritmos **Greedy**, **Grasp** (Greedy Randomized Adaptive Search) y **Gvns** (General Variable Neighborhood Search)

2 Greedy

2.1 Explicación

Para la realización de una búsqueda Greedy, añadimos el punto de depósito a todos los vehículos, luego escogemos el cliente más cercano y así hasta que todos los clientes estén visitados y por último volvemos a añadir el nodo de depósito.

Gracias a este método podemos obtener buenas soluciones en tiempos bastante rápidos.

2.2 Pseudocódigo

```
function GREEDY
    solution ← depot
    while allCusomtersNotVisited do
        solution ← closeCustomer
    end while
    solution ← depot
    return solution
end function
```

3 Grasp

3.1 Explicación

Para la realización de una búsqueda GRASP, generamos una solución Greedy aleatoria con un conjunto de mejores candidatos, así nos aseguramos tener buenas soluciones con un grado de aleatoriedad para poder inspeccionar el espacio de soluciones.

Tras generar esa solución, lo siguiente es buscar un mínimo local con una heurística como reinserción entre rutas, reinserción dentro de la ruta, intercambio entre rutas, intercambio dentro de la ruta, intercambio 2-opt, etc.

Repetimos esta secuenca x iteraciones quedándonos con la mejor solución.

3.2 Pseudocódigo

```
function GRASP(numberIterations, numberCandidates)
    bestSolution ← constructSolution(numberCandidates)
    for numberIterations do
        solution ← constructSolution(numberCandidates)
        solution ← localSearch(solution)
        if solution < bestSolution then
            bestSolution ← solution
        end if
    end for
    return bestSolution
end function
```

4 Gvns

4.1 Explicación

Para la realización de una búsqueda GVNS, primero construimos una solución inicial y luego iteramos x veces como en GRASP generando una solución y luego para evitar estancarnos en un mínimo local realizamos un movimiento aleatorio y buscamos el mejor vecino con una heurísticas si ya es el mejor vecino, utilizamos otra, si encontramos un mejor vecino, volvemos a utilizar la primero heurística (y así hasta terminar con nuestro conjunto de heurísticas).

Al terminar el búsqueda local comparamos si es mejor que la solución inicial, si lo es volvemos a buscar, si no hacemos 2 movimientos aleatorios (y así hasta el máximo número de movimientos aleatorios).

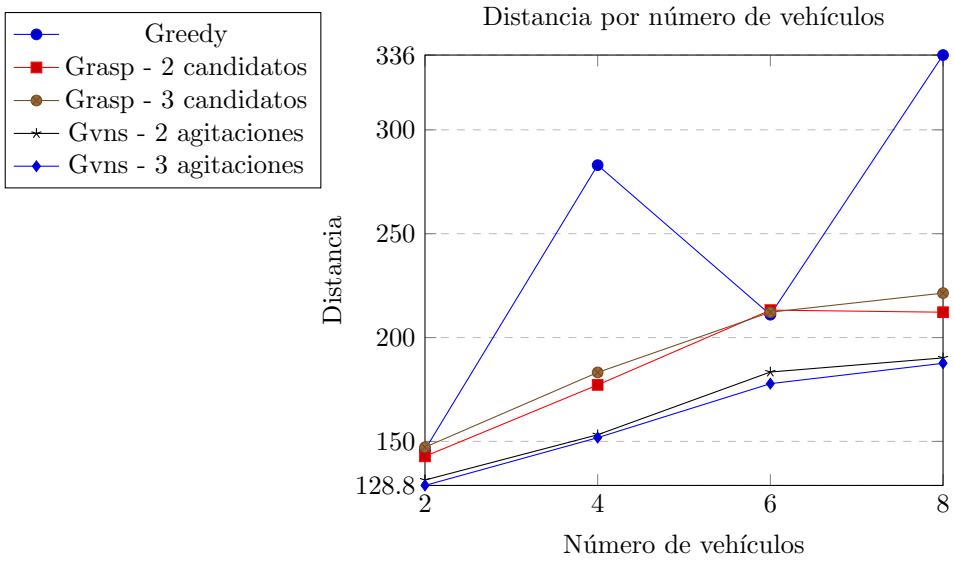
4.2 Pseudocódigo

```
function GVNS(numberIterations, maxShakes)
    bestSolution ← constructSolution()
    for numberIterations do
        solution ← localSearch(solution)
        for i ← 1; i ≤ maxShakes; i ← i + 1 do
            shaked ← shake(solution, i)
            localMinimum ← vnd(shaked)
            if localMinimum < solution then
                solution ← localMinimum
                i ← 0
            end if
        end for
        if solution < bestSolution then
            bestSolution ← solution
        end if
    end for
    return bestSolution
end function
```

5 Resultados

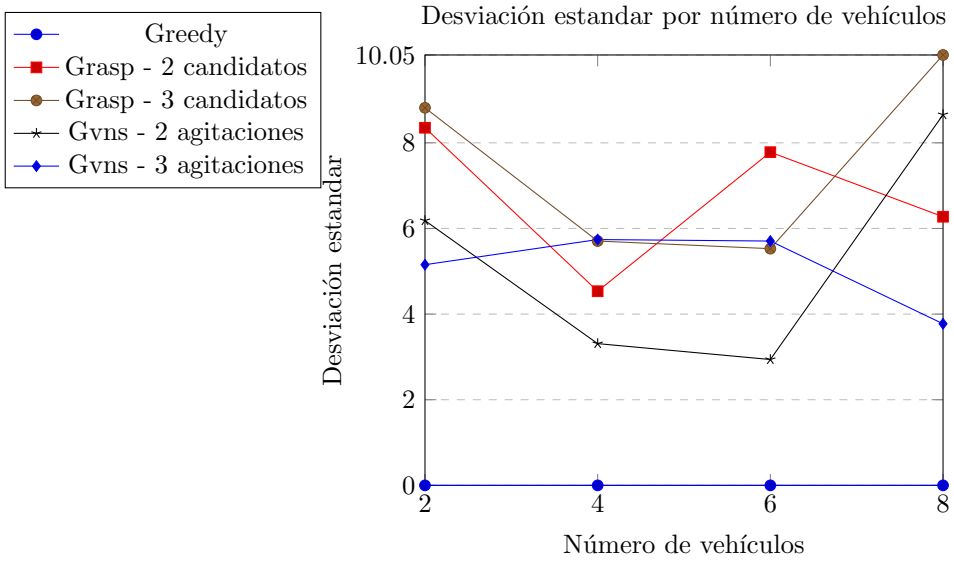
Se han realizado 5 ejecuciones por cada instancia, y en GRASP y GVNS como máximo 1000 iteraciones y 100 iteraciones sin mejora. GRASP se evaluó con reinserción entre rutas, mientras que GVNS utiliza todas las heurísticas y GRASP se evaluó con 2 y 3 candidatos, mientras que GVNS se evaluó con 2 y 3 máximas agitaciones.

5.1 Distancias medias



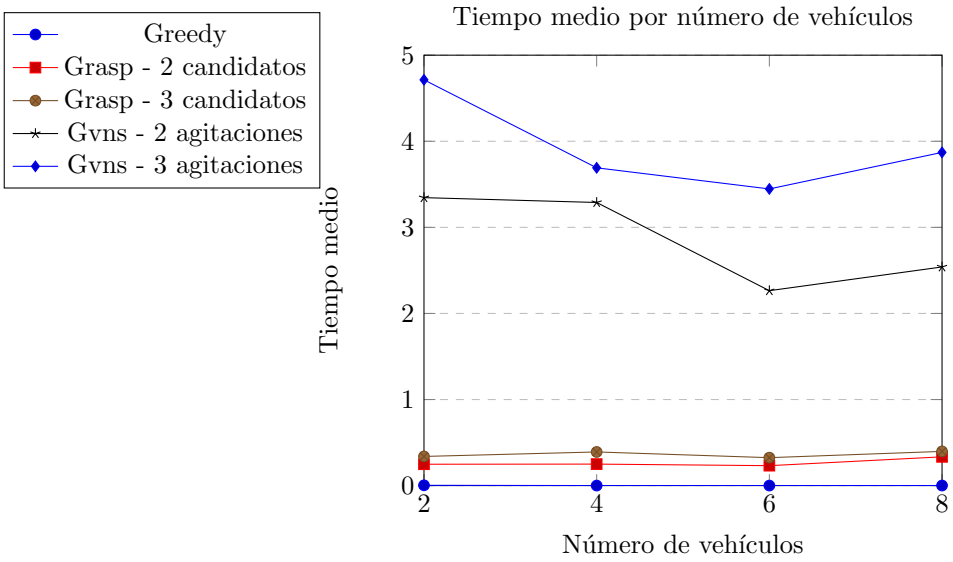
Podemos observar como el método Greedy para 2 o 6 vehículos nos dan buenos resultados pero para 4 y 8 no, con GRASP vemos como en el peor caso nos da soluciones como greedy y en el mejor de los casos divide del coste del greedy, y con respecto al número de candidatos añadirle número de candidatos hace que las soluciones que genera empeoren ligeramente. Con GVNS vemos que en todos los casos es mejor que GRASP y Greedy, con respecto al número máximo de agitaciones se observa que a mayor número de agitaciones se mejoran ligeramente las soluciones.

5.2 Desviación Estandar



Viendo esta gráfica podemos sacar en claro que utilizando un algoritmo Greedy siempre nos va a devolver el mismo coste sin variación alguna ya que es un algoritmo exacto (al contrario con GRASP y GVNS que son aproximados), y con respecto a las variaciones de GRASP y GVNS podemos intuir que GVNS genera soluciones con valores más similares en cada ejecución que los que genera GRASP, que serían más "aleatorios".

5.3 Tiempo Medio



En este último gráfico observamos que el algoritmo Greedy es el más rápido con diferencia, el GRASP también es bastante rápido aunque no tanto como el Greedy y el GVNS vemos como es el más lento de los 3 que en el mejor caso tarda más de 2 segundos mientras que el peor caso del grasp no llega a 0.4 segundos.

También vemos como GRASP con 2 candidatos tarda ligeramente menos que con 3 y que GVNS con 3 agitaciones como máximo tarda mucho más que con 2.

6 Conclusiones

Para concluir, teniendo en cuenta los anteriores gráficos, sacamos como conclusiones que el algoritmo que mejor resultados da es el GVNS y también el más lento, el más rápido es el greedy pero el que da peores resultados. El más polivalente es el GRASP ya que es casi tan rápido como el Greedy pero con soluciones ligeramente mejores, aunque más aleatorias que el GVNS.

También está la opción de aumentar el número de shakes máximos del GVNS bajando el número máximo de iteraciones pero eso implica que aumentar el número de agitaciones aumenta el tiempo de ejecución, asi que daría los mismos tiempos más o menos, y aumentar el número de candidatos en GRASP sería empeorar en tiempos y soluciones