CIMAT

Centro de Investigación en Matemáticas, A.C.

OPTIMIZACIÓN CON COMPONENTES INTERDEPENDIENTES

EL PROBLEMA DEL LADRÓN VIAJERO

T E S I S

Que para obtener el grado de

Maestro en Ciencias

con Orientación en

Ciencias de la Computación

Presenta

Ricardo Nieto Fuentes

Director de Tesis:

Dr. Carlos Segura González

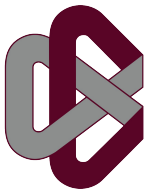
Co-Director de Tesis:

Dr. Sergio Ivvan Valdez Peña

Autorización de la versión final

Guanajuato, Gto., 10 de diciembre de 2018

Centro de Investigación en Matemáticas A.C.



CIMAT A.C

Tesis de Maestría

Optimización con componentes interdependientes

El problema del ladrón viajero

Presenta:
Ricardo Nieto Fuentes

Director de Tesis:
Dr. Carlos Segura González
Co-Director:
Dr. Sergio Ivvan Valdez Peña

Para obtener el grado de:

Maestro en ciencias con especialidad en computación y matemáticas industriales

Diciembre 2018

Resumen

En el área de optimización con metaheurísticas, algunos investigadores [1–3] han sugerido que los modelos utilizados para validar el rendimiento de los optimizadores son demasiado simples al compararse con la complejidad de los problemas reales. Esto es atribuído a que los modelos académicos han omitido la complejidad derivada de la interdependencia que existe entre los subproblemas que componen la mayoría de los problemas reales. El problema del ladrón viajero [3] (*Travelling Thief Problem* — TTP) es un problema sintético y novedoso, diseñado para modelar la interdependencia presente en los problemas reales. Combina dos problemas clásicos ampliamente estudiados por la academia, el problema del vendedor viajero (*Travelling Salesman Problem* — TSP) y el problema de la mochila (*Knapsack Problem* — KP). Desde su publicación, numerosas heurísticas han sido desarrolladas para la solución del TTP, las cuales pueden clasificarse en función de su enfoque en los siguientes grupos: centradas en TSP, codificación completa, sistema CoSolver, exactas y mixtas.

Al analizar las características del TTP y los diferentes mecanismos empleados por las distintas estrategias, se hipotetiza que los enfoques actuales son propensos a centrarse rápidamente en una región del espacio de búsqueda, limitando su capacidad para obtener resultados de alta calidad a largo plazo. Este trabajo se enfoca en la familia de heurísticas dentro de la categoría CoSolver, las cuales alternan entre dos etapas de optimización, una enfocada en la mejora del recorrido y otra en la selección de objetos. En base a la hipótesis de trabajo, se proponen dos heurísticas para evitar el sesgo en la exploración del espacio de búsqueda. La primera estrategia consiste en una metaheurística de trayectoria (REStart with Guided Local Search — RESGLS) basada en la búsqueda local guiada (*Guided Local Search* — GLS), método que penaliza características presentes en el espacio de búsqueda que se hayan explorado en fases anteriores. La segunda consiste en un algoritmo memético (Diversity based Memetic Algorithm — DMA), que incorpora un control explícito de diversidad para forzar la exploración del espacio de búsqueda. Este método, además, incorpora la estrategia basada en la búsqueda local guiada para mejorar la intensificación realizada en cada zona explorada. Ambas propuestas son comparadas frente a algoritmos específicos para el TTP presentes en la literatura.

Los resultados obtenidos, indican que los algoritmos actuales para resolver el TTP se encuentran enfocados en un subconjunto del espacio de búsqueda. En específico, que el sistema CoSolver cae rápidamente en óptimos locales limitando la capacidad de exploración de las heurísticas que lo implementan. Empleando los algoritmos propuestos, se encontraron nuevas mejores soluciones conocidas en ocho instancias. Esto muestra que las estrategias son competitivas y mejoran la capacidad de exploración del sistema CoSolver evitando que la búsqueda local quede atrapada en una zona del espacio de búsqueda.

Abstract

In recent years, researchers in the field of metaheuristics [1–3], have suggested that the current models for algorithm performance validation are unable to represent the complexity of real world problems. One of the most important factors is that current academic models omits the complexity derived from the interdependence between the components of most real world problems. To aliviate this, in 2013 a novel benchmark problem designed to model such interdependence was proposed. The TTP [3] (Travelling Thief Problem) combines two well known classic academic problems, the TSP (Travelling Salesman Problem) and the KP (Knapsack Problem). Various strategies have been devised for the TTP. This strategies can be classified based on their search mechanism in: TSP centered, full encoding, CoSolver system, exact approach and mixed approach.

From a thorough study of the literature and based on the main characteristic of the TTP, this thesis hypothesis is that current approaches are biased to focus in a narrow region of the search space. This limits their capacity to find high quality solutions in the long term. This work is focused on the CoSolver system, the alternation between phases of tour and item selection optimization. Two novel metaheuristics are proposed. First, a trajectory heuristics called RESGLS (Guided Local Search with REStart), where the main idea is to allow the search space exploration by penalizing the characteristics present in the local optimas of the explored regions in previous phases of the search. The second metaheuristic is called DMA (Diversity based Memetic Algorihtm), a memetic algorithm with an explicit diversity control mechanism to enforce the exploration of the search space. A variation of the RESGLS strategy without the restarting phase is used for intensification purposes. Both proposals are compared against the TTP state-of-the-art specific strategies.

The obtained results show that, in fact, the state-of-the-art techniques for the TTP are too focused in a subset of the search space. Specifically, the CoSolver system is prone to get caught in local optima, which limits its exploration potential. The proposed experimental validation improves eight best known solutions. This shows the important benefits of the proposed schemas that allow the exploration with a better treatment of CoSolver's local optimas.

Agradecimientos

A mi familia por su apoyo incondicional que me ha permitido dar un paso más en la vida.

A mi director de tesis, el Dr. Carlos Segura González por su paciencia, apoyo y críticas en cada etapa del desarrollo de esta tesis.

A mi jurado, por el tiempo dedicado en la revisión y evaluación de este trabajo.

Finalmente agradezco los apoyos que me han sido otorgados por parte del CIMAT y el CONACYT pues sin ellos no hubiera sido posible el desarrollo de este trabajo.

Índice general

Índice de figuras

Índice de tablas

Índice de Ecuaciones

Índice de algoritmos

Capítulo 1

Introducción

1.1. Optimización de problemas con componentes interdependientes

La optimización consiste en encontrar el mejor elemento dentro del conjunto de posibles soluciones. Las metaheurísticas son algoritmos generales que aproximan la solución al problema de optimización, utilizadas cuando las técnicas exactas no son aplicables.

Los problemas reales presentan un desafío para las metaheurísticas actuales, algunos factores que deben enfrentar son [1]:

- La cantidad de posibles soluciones prohíbe la completa exploración del espacio de búsqueda.
- No es claro o es demasiado difícil modelarlo, resultando en una sobresimplificación que no representa correctamente el problema original.
- Encontrar una solución puede llegar a ser muy complejo, pues puede contar con restricciones difíciles de cumplir que limiten la región factible a un subconjunto diminuto del espacio de búsqueda.
- Los datos pueden tener ruido o existir incertidumbre en las mediciones.
- Puede estar compuesto por diversas partes, las cuales suelen estar en conflicto.

Para estudiar los problemas reales, la academia ha generado modelos que son una simplificación de los mismos, provocando que los objetivos de la investigación académica y la industria diverjan. Como consecuencia ha surgido una brecha entre el estudio académico de los optimizadores y su aplicación a los problemas reales que enfrenta la industria.

Tradicionalmente los problemas de referencia estudiados en la academia, definen su complejidad como un escalamiento del problema. Basándose en esta definición, para aumentar la dificultad del problema, se hace uso de instancias cada vez más grandes. Sin embargo varios autores hacen ver que existen otros factores que determinan la complejidad de los problemas reales [2]:

- **Combinación**

  Se encuentran conformados por dos o más sub-problemas.

- **Interdependencia**

  Los componentes que lo conforman se encuentran relacionados de manera que la solución de uno tiene influencia sobre la solución de los demás.

Esta diferencia entre el enfoque clásico de la academia y la mezcla de factores que determinan la complejidad en los problemas reales, resulta en una disparidad en el desempeño de las metaheurísticas derivadas de la investigación académica al enfrentarse a problemas más complejos. Esto lleva a la necesidad de desarrollar algoritmos que tomen en consideración los factores que provocan la complejidad de los problemas reales.

1.2. El ladrón viajero

El problema del ladrón viajero o TTP (*Travelling Thief Problem*) fue propuesto en el 2013 [3] para el estudio de problemas de optimización en los que se da una interacción interdependiente entre los componentes que los conforman. El problema tiene dos variantes: Tipo I (mono-objetivo) y Tipo II (multi-objetivo), siendo el más estudiado en la literatura y en el que se enfocará este trabajo el TTP Tipo I, el cual, puede describirse de manera simplificada en el siguiente enunciado [4]:

> Dadas N ciudades y M objetos, con una ganancia p y un peso w, repartidos entre las ciudades. Un ladrón debe visitar las N ciudades, una vez y solo una vez cada una, regresando a la ciudad de origen. Si se tiene un costo ρ por unidad de tiempo, una capacidad máxima W y la velocidad es función de la carga $v(w)$. ¿Cuál es la mejor ruta Ξ y selección de objetos Π para obtener el máximo beneficio?

El problema del ladrón viajero pertenece a la categoría de problemas NP-complejos (Non deterministic Polynomial time — NP) o *NP-hard*, es decir, es tan difícil de resolver como el problema más difícil de decisión verificable en tiempo polinomial [5]. Esto se debe a que se encuentra conformado a su vez por problemas NP-complejos ampliamente estudiados en la investigación académica. En concreto los dos problemas que conforman al TTP son:

- **El problema del vendedor viajero (*Travelling Salesman problem* — TSP)**

  Seleccionar el orden en que se recorren N ciudades para minimizar el tiempo.

- **El problema de la mochila (*Knapsack Problem* — KP)**

  Decidir qué objetos seleccionar cumpliendo la restricción de peso y maximizando la ganancia.

El TTP combina estos problemas y los interrelaciona de manera no lineal empleando el peso de los objetos seleccionados. Conforme se agregan objetos (maximizar la ganancia) la velocidad del ladrón disminuye y toma más tiempo realizar el recorrido. Si por el contrario se eligen pocos objetos, la velocidad aumenta y el tiempo será menor (minimizar el costo de la renta) pero la ganancia será inferior. La interacción no lineal de los componentes fuerza la solución simultánea de ambas, al imposibilitar su descomposición [3].

Por lo anterior este nuevo problema resulta un reto para las heurísticas actuales, permitiendo el estudio y exploración de nuevas estrategias que consideren la interdependencia de los componentes del problema. Desde su publicación en 2013, ha estado presente como parte de los concursos de la conferencia GECCO (*Genetic and Evolutionary Computation Conference*) y del congreso CEC (*Congress on Evolutionary Computation*). En los últimos cinco años se han presentado varias propuestas para resolver el TTP, las cuales pueden clasificarse de la siguiente manera:

- **Centradas en el TSP** [4]

  Parten de una solución del recorrido generada sin considerar los objetos. Para ello se emplean optimizadores del TSP, siendo la más habitual, la heurística Lin Kernighan (LK) [6]. Después, proceden a optimizar la selección de objetos empleando una heurística de trayectoria, sin considerar realizar cambios en la ruta obtenida.

- **Sistema CoSolver** [7]

  Inician con una solución, generada con una estrategia especifica del TSP para el recorrido y alguna heurística constructiva para la selección de objetos. Después iteran entre fases de optimización

del recorrido (sin modificar la selección de objetos) y de optimización de la selección de objetos (sin modificar el recorrido). En la mayoría de los casos la optimización del recorrido se realiza mediante una búsqueda local simple, mientras que la selección de objetos se optimiza con una búsqueda local simple o con metaheurísticas de trayectoria más complejas.

- **Codificación completa** [8]

  Se enfocan en resolver el TTP como un todo. En cada iteración modifican simultáneamente el recorrido y los objetos. Muchos algoritmos poblacionales pertenecen a este esquema y en varios casos son esquemas híbridos que incluyen el sistema CoSolver.

1.3. Hipótesis

Diferentes algoritmos han sido desarrollados para resolver el TTP, entre ellas, el sistema CoSolver ha destacado al obtener valores objetivo competitivos frente a estrategias más complejas como lo son los algoritmos meméticos pertenecientes a la categoría de codificación completa. El sistema CoSolver ha sido uno de lo esquemas más exitosos en el rango de instancias pequeño a mediano en el TTP, pero han tenido poco éxito en su aplicación a instancias de mayor tamaño.

En base a los algoritmos y resultados presentes en la literatura, se hipotetiza que dos de los factores limitantes en el desempeño del sistema CoSolver son:

- **Carece de una estrategia para escapar de óptimos locales**

  En cada iteración de la búsqueda, el sistema CoSolver optimiza uno de los dos subproblemas hasta encontrar un óptimo local. Una vez encontrado, el óptimo es utilizado como solución inicial en la optimización del segundo subproblema. Note que cada fase de optimización solo explora una pequeña vecindad del óptimo local encontrado por la fase anterior, en consecuencia su capacidad de exploración es poca.

- **La dependencia al recorrido inicial generado por el optimizador de TSP**

  Como consecuencia de intercalar la optimización de los subproblemas, se espera que el sistema CoSolver tenga una fuerte dependencia a la solución inicial, pues ésta determina la región del espacio de búsqueda explorada. Los optimizadores actuales emplean heurísticas específicas del TSP sesgando las soluciones iniciales a los recorridos con menor distancia.

1.4. Objetivo

El objetivo de este trabajo de tesis es, mejorar el desempeño presentado por el sistema CoSolver, al incorporar mecanismos para mejorar el manejo de óptimos locales.

Para ello el trabajo se dividió en dos partes. La primera se enfoca en las metaheurísticas de trayectoria y plantea los siguientes objetivos:

- Incorporar en el sistema CoSolver, un mecanismo clásico de heurísticas de trayectoria para escapar de óptimos locales.
- Mostrar que un mejor manejo de los óptimos locales tiene un impacto positivo en la calidad de las soluciones obtenidas, al comparar la implementación frente a algoritmos de la literatura.

La segunda parte se enfoca en un algoritmo memético y plantea los siguientes objetivos:

- Diseñar un algoritmo poblacional que incorpore el sistema CoSolver con manejo de óptimos locales para mejorar la capacidad de intensificación de la población.

- Promover la exploración del espacio de búsqueda del sistema CoSolver, al incluir un mecanismo explícito de control de diversidad.

1.5. Contribución

En este trabajo se presentan dos nuevos enfoques para lidiar con óptimos locales al resolver el TTP.

- **RESGLS**

  Un algoritmo CoSolver basado en GLS [9], el cual es un mecanismo clásico para escapar de óptimos locales, en combinación con una estrategia de reinicio [10] para promover la exploración. La novedad de esta nueva estrategia es ser el primer esquema que utiliza la optimización del recorrido para evitar que el sistema CoSolver permanezca en una zona del espacio de búsqueda.

- **DMA**

  Un algoritmo memético [11] que utiliza la propuesta RESGLS para la fase de intensificación e implementa un mecanismo de control de diversidad elitista, basado en la estrategia de reemplazamiento con control de diversidad dinámico multi-objetivo [12] (*Replacement with Multi-objective based Dynamic Diversity Control strategy* — RMDDC), para promover la exploración. Este algoritmo es el primer esquema en considerar el control de diversidad para promover la exploración del espacio de búsqueda del TTP.

1.6. Organización del trabajo

La organización del trabajo es la siguiente: El Capitulo 1 motiva la investigación del TTP y muestra los objetivos de esta tesis. El Capitulo 2 presenta una revisión de los conceptos clave de optimización combinatoria utilizados en el problema del ladrón viajero. Después en el Capítulo 3 se describe formalmente el TTP y se presenta una revisión del estado del arte de los algoritmos para resolverlo. En el capítulo 4 se describe el diseño e implementación de las propuestas de esta tesis. La descripción y discusión de los resultados del diseño experimental se presentan en el Capítulo 5 y finalmente en el Capítulo 6 se discuten los resultados y se concluye el trabajo proponiéndose el posible trabajo futuro derivado de esta tesis.

Capítulo 2

Optimización con meta-heurísticas

2.1. Conceptos básicos en metaheurísticas

El problema de optimización consisten en la selección del mejor elemento (con respecto a un criterio) de un conjunto de soluciones candidatas. Un problema de optimización puede representarse de la siguiente manera [13]:

$$\text{Encontrar } \{s \in F \mid f(s) \preceq f(s'),\ \forall s' \in F\} \tag{2.1}$$

La solución s es un elemento que pertenece al conjunto soluciones factibles F (aquellas que cumplan con las restricciones impuestas al problema) y $f(.)$ es el criterio a optimizar. Finalmente $a \preceq b$ denota que a es "mejor" que b, por ejemplo, al minimizar el símbolo $\preceq$ corresponde a $(\leq)$, al maximizar a $(\geq)$ y en optimización multi-objetivo corresponde a la dominancia de Pareto (las componentes de un vector a son mejores o iguales que las de un vector b y al menos una componente de a es estrictamente mejor).

Los problemas de optimización pueden clasificarse en función de la codificación de sus soluciones. Aquellos problemas cuya solución se encuentra en los valores reales [14] y aquellos en los cuales su solución se encuentra codificada en variables discretas [15]. Los problemas abordados en este trabajo de tesis pertenecen a esta segunda categoría de optimización, a la cual se referirá como optimización combinatoria.

Los algoritmos para solucionar los problemas de optimización se pueden clasificar en exactos o aproximados [16]. Los algoritmos exactos garantizan encontrar una solución óptima, entre ellos se encuentran programación dinámica y ramificación y poda. Por otro lado, los algoritmos aproximados son utilizados cuando las técnicas exactas no son aplicables, garantizan una solución suficientemente buena en un tiempo razonable. Estos algoritmos son particularmente útiles en problemas NP-Complejos (no existe un algoritmo que los solucione en tiempo polinomial), donde los algoritmos exactos no son escalables. En los métodos aproximados se sacrifica la precisión de encontrar la solución óptima por encontrar soluciones aceptables en un margen de tiempo razonable.

Existen dos familias de métodos aproximados: heurísticas, los cuales son específicas a un problema o instancia y las metaheurísticas que son algoritmos diseñados para ser aplicados a una amplia gama de problemas [17]. A su vez los métodos exactos, pueden clasificarse en función del manejo de la solución [18]:

- **Métodos de trayectoria**

  Estos métodos manejan en todo momento una sola solución, la cual va sufriendo modificaciones o remplazos a lo largo de las iteraciones.

- **Métodos poblacionales**

  Estos métodos manejan en todo momento un conjunto de soluciones, entre las cuales se comparte información en su modificación o remplazo a lo largo de las iteraciones.

2.2. Métodos de trayectoria

Los métodos de trayectoria operan en cada instante del algoritmo sobre una única solución y reciben su nombre debido a que las modificaciones realizadas a la solución forman un camino en el espacio de búsqueda. En el Algoritmo {1} se muestra el pseudocódigo de este tipo de esquemas.

Algoritmo 1: Esquema general de un algoritmo de trayectoria

```
def método-trayectoria (solución inicial)
    s* = s = solución inicial
    hasta que se cumpla el criterio de paro
        C = soluciones candidatas de la solución s
        s = seleccionar una nueva solución de C
        s* = s si s ≺ s*
    fin
    regresa la mejor solución encontrada s*
fin
```

Los componentes básicos para un método de trayectoria son: la forma de generar nuevas soluciones candidatas y el criterio de actualización de la nueva solución.

Para generar nuevas soluciones candidatas diferentes algoritmos emplean distintas estrategias como lo son: de forma aleatoria, empleando heurísticas de construcción o definiendo una vecindad. La definición de una vecindad $N(s)$ puede expresarse en términos de un conjunto de modificaciones M. De esta manera, el conjunto $N(s) \subset S$ es el conjunto de soluciones válidas, resultante de aplicar una modificación $m \in M$ a la solución s. Es importante notar que la definición de vecindad depende intrínsecamente de la codificación de las soluciones y del problema, siendo posible definir más de una vecindad para una codificación específica del problema.

La definición de la vecindad permite definir el concepto de óptimo local. Dada una solución s, un óptimo local es aquella solución $s^l \in N(s) \mid f(s^l) \preceq f(s')$ para toda solución $s' \in N(s) \setminus s^l$. A su vez el óptimo global s^* (2.1) debe ser el óptimo local para todas las vecindades posibles.

Por otro lado tres ejemplos de estrategias ampliamente utilizadas como criterio de actualización (selección) son:

- **Mejor solución**

  Seleccionar la mejor solución de entre todas las soluciones de la vecindad $N(s)$ que mejoran a s.

- **Primera mejora**

  Se define un orden de las soluciones en $N(s)$, explorándose en el orden establecido, se mueve la solución actual s a la primer solución s' que mejore s.

- **Selección aleatoria**

  Se selecciona de manera aleatoria entre las soluciones de $N(s)$ que mejoran a s.

A continuación se describen los mecanismos clásicos de las metaheurísticas de trayectoria, ofreciéndose más detalle sobre la búsqueda local guiada GLS y la búsqueda local iterada (*Iterated Local Search* — ILS), pues son dos métodos utilizados en esta tesis para abordar el TTP.

2.2.1. Búsqueda local

En el Algoritmo {2} se esquematiza la búsqueda local, la cual es la más antigua y simple de las metaheurísticas de trayectoria, siendo la base de métodos más complejos.

Algoritmo 2: Búsqueda local

def *búsqueda-local (s = solución inicial)*
  hasta que *se cumpla el criterio de parada*
    $N(s)$ = generar la vecindad de s
    s = seleccionar un elemento de $N(s)$
    romper si $\nexists s' \in N(s) \mid s' \prec s$
  fin
  regresa s
fin

El funcionamiento de la búsqueda local es el siguiente: mientras no se cumpla el criterio de paro (usualmente que no exista un elemento en la vecindad que mejore la solución actual), la búsqueda se mueve a una nueva solución candidata, de esta forma se va modificando poco a poco la solución hasta llegar a un óptimo local.

El éxito de la búsqueda local se debe a que es un método que es muy fácil de implementar y diseñar. Proporcionando, para muchos problemas, resultados con una calidad aceptable en un tiempo relativamente corto. Sin embargo se pueden identificar dos problemas importantes. El primero es que converge a un óptimo local y en muchos problemas existe una gran cantidad de óptimos locales, con la posibilidad de que muchos de ellos tengan una calidad muy baja. El segundo problema es que tiene una dependencia muy importante a la solución inicial.

2.2.2. Estrategias para escapar de óptimos locales

Para solventar los principales inconvenientes de la búsqueda local, se han propuesto varias metaheurísticas de trayectoria más complejas. Usualmente se pueden incluir en alguno de los siguientes tres grupos de esquemas:

Iterar con diferentes soluciones

Como se muestra en el Algoritmo {3}, estas estrategias están basadas en aplicar una heurística de trayectoria a diferentes soluciones, una vez que una zona ha sido explorada, con la intención de encontrar nuevos óptimos locales.

Algoritmo 3: Estructura de los algoritmos de reinicio

def *estrategia-reinicio (t = duración de búsqueda)*
  $s^* = s$ = generar una nueva solución
  hasta que *se cumpla el criterio de duración*
    s = generar una nueva solución
    s = realizar una búsqueda local empleando s
    $s^* = s$ **si** $s \prec s^*$
  fin
  regresa s^*
fin

Entre los métodos más representativos de esta familia se encuentran:

- **MultiStart** [19]

  Una vez que se encuentra un óptimo local se genera una nueva solución.

- **Búsqueda local iterada** [20]

  Una vez que se encuentra un óptimo local se perturba (modifica) la solución actual y se prosigue con la búsqueda.

- **GRASP** [21]

  Una vez que se encuentra un óptimo local se genera una nueva solución empleando un algoritmo codicioso aleatorizado.

Aceptar vecinos que no mejoran

Esta familia de métodos se basa en intentar escapar del óptimo local realizando movimientos incluso si no mejoran la solución s. El Algoritmo {4} muestra la forma de este mecanismo, donde el criterio de admisibilidad controla la aceptación de soluciones que empeoren la solución, con esperanza de mover la solución lo suficiente para escapar de la región actual y llegar a un nuevo óptimo local.

Algoritmo 4: Estructura de los algoritmos que aceptan vecinos sin mejora

```
def aceptar-vecinos-sin-mejora (t = duración de la búsqueda)
    s* = s = generar una nueva solución
    hasta que se cumpla el criterio de paro
        s = siguiente solución admisible del vecindario N(s)
        s* = s si s ≺ s*
        actualizar el criterio de admisibilidad
    fin
    regresa la mejor solución encontrada s*
fin
```

Los dos algoritmos más representativos de esta categoría son:

- **Recocido Simulado** [22]

  Basado en principios de la mecánica estadística del recocido para la obtención de estructuras cristalinas. Consiste en recorrer la vecindad de forma aleatoria y hacer que la probabilidad de aceptación sea función de la calidad de la solución candidata. En concreto esta probabilidad es 1 cuando se mejora la solución actual y es inversamente proporcional a la diferencia del criterio de mejora entre movimiento propuesto y la mejor solución actual.

- **Búsqueda Tabú** [23]

  Se emplea una estructura de memoria que almacena los movimientos realizados previamente a los que se les denomina movimientos tabú, con la intención de no realizar modificaciones que regresen la solución a un estado anterior. De las soluciones no tabú se elige la mejor solución o aquella que cumpla algún criterio específico (denominado criterio de aspiración).

Cambiar el paisaje del problema

El paisaje representa la relación entre la aptitud y el espacio de búsqueda. Se puede definir como un grafo dirigido, donde cada vértice corresponde a una solución del espacio de búsqueda, el peso de cada nodo a la aptitud asignada por la función objetivo y las aristas están dadas por la definición de

vecindad [24]. El paisaje es dependiente del problema y está fuertemente ligado tanto a la representación seleccionada como a su vecindad (como se encuentran conectadas las soluciones). El concepto de paisaje permite estudiar a las metaheurísticas de trayectoria como estrategias para navegar esta estructura en búsqueda de la mejor solución.

En el Algoritmo {5} se esquematiza la familia de metaheurísticas enfocadas en modificar esta representación para promover nuevos movimientos una vez que se ha encontrado un óptimo local.

Algoritmo 5: Estructura de los algoritmos que cambian el paisaje del problema

def *modificación-del-paisaje* ($\hat{g}(s)$ = *nuevo modelo del paisaje*)
  $s^* = s$ = generar solución inicial
  hasta que *se cumpla el criterio de paro*
    s = óptimo local empleando el modelo $\hat{g}(s)$
    $s^* = s$ **si** $s \prec s^*$
    modificar el modelo $\hat{g}(s)$
  fin
  regresa *la mejor solución encontrada* s^*
fin

Entre los algoritmos clásicos de esta familia se encuentran:

- **Búsqueda local guiada** [9]

  Consiste en penalizar las características presentes en el óptimo local modificando la función objetivo. De esta forma la búsqueda evita centrarse en una sola región del espacio.

- **Suavizado** [25]

  Consiste en modificar la función objetivo para eliminar óptimos locales, se generan modelos que aproximan el problema pero son más fáciles de resolver y poco a poco se va cambiando de modelo hasta llegar al problema original.

- **Vecindario variable** [26]

  Esta búsqueda local selecciona de un conjunto de vecindades la vecindad a aplicar, una vez que se ha encontrado un óptimo local en esa vecindad se selecciona una nueva vecindad. Se basa en el hecho de que diferentes vecindades tienen diferentes óptimos locales y el óptimo global es óptimo local en todas las vecindades.

2.2.3. Búsqueda local guiada (GLS)

La búsqueda local guiada, pertenece a la categoría de cambiar el paisaje del problema. Ofrece un mecanismo de exploración que define una función de aptitud, basada en la función objetivo, la cual se modifica durante la búsqueda con la intención de aumentar la exploración. Esta estrategia ha sido aplicada con cierto grado de éxito al TSP [27] y se utiliza en este trabajo de tesis pues es fácilmente integrable dentro del sistema CoSolver para resolver el TTP.

Esta metaheurística requiere de la definición de un conjunto de características que identifican a las soluciones y cuenten con un costo que tenga un impacto en la función objetivo. Por ejemplo en el TSP las aristas son ampliamente utilizadas como conjunto de características, pues distinguen a la solución (un recorrido tiene o no una arista) y cuentan con un costo definido por su distancia afectando el valor de la función objetivo [28].

La función de aptitud penalizada $\hat{f}(s)$ se define como:

$$\hat{f}(s) = f(s) + \lambda \sum_{i \in \Phi(s)} \phi_i \tag{2.2}$$

En esta nueva función de aptitud, $f(s)$ es la función objetivo del problema, $\Phi(s)$ denota el conjunto de características de la solución s, ϕ es es la penalización asociada a la característica i y λ es un peso para la penalización.

La estrategia GLS incorpora información del problema en la función de aptitud $\hat{f}(s)$, lo que le permite guiar la exploración del espacio de búsqueda al castigar la presencia de ciertas características.

Empleando la función $\hat{f}(s)$ se aplica una búsqueda local hasta alcanzar un óptimo local mediante una heurística de trayectoria, una vez encontrado el óptimo local, se procede a actualizar las penalizaciones de las características presentes de la siguiente forma:

$$\phi_i = \phi_i + 1 \ \mid \ \arg\max_{i \in \Phi(s)} \frac{c_i}{1 + \phi_i} \tag{2.3}$$

Se incrementa la penalización de la característica i que maximiza el cociente entre el costo c_i (de la característica) y su penalización ϕ_i, presente en el óptimo local s actual. El objetivo de esta estrategia es que la frecuencia de penalización de las características sea proporcional a su costo (dirigir la búsqueda a zonas prometedoras) e inversamente proporcional a las veces que ha sido penalizada, evitando así un sesgo excesivo hacia penalizar características de alto costo.

En el Algoritmo {6} se esquematiza la búsqueda local guiada. En primer lugar, se genera una solución inicial, se inicializa la estructura que contendrá las penalizaciones y se genera una nueva función penalizada. Después hasta que se cumpla un criterio de paro, se realiza la búsqueda local a la solución s, actualizándose la mejor solución encontrada utilizando la función objetivo original. Finalmente se incrementa la penalización empleando las características del óptimo local encontrado.

Algoritmo 6: Búsqueda Local Guiada

```
def búsqueda-local-guiada (s = solución inicial)
    s* = s
    Φ = inicializar las penalizaciones en 0
    f̂ = -> (s){ f(s) + λ Σ_{i∈Φ(s)} φ_i }
    hasta que se cumpla el criterio de paro
        hasta que no se acepte una nueva solución
            s = seleccionar nueva solución candidata empleando la función f̂
            s* = s si f(s) ≺ f(s*)
        fin
        i = calcular la característica en s a penalizar
        φ_i += 1
    fin
    regresa la mejor solución encontrada s*
fin
```

En el TSP, la penalización sobre las aristas se representa como una matriz $P = [p_{i,j}]$ en donde cada posición indica la cantidad de veces que la arista ha sido penalizada. Se recomienda que el peso de penalización λ sea proporcional a la razón de la longitud promedio de una arista en un óptimo local

(2.4).

$$\lambda = \alpha \left(\frac{D(\Xi^*)}{|\Xi^*|} \right) \tag{2.4}$$

En el cálculo del peso de la penalización el valor α pertenece al rango $[0, 1]$, dependiendo de la instancia y la heurística a utilizar. Como regla general cuanto mejor desempeño tenga la búsqueda local menor el valor requerido de α y viceversa.

2.2.4. Búsqueda local iterada (ILS)

En las estrategias que iteran con diferentes soluciones, cuando las soluciones son generadas de forma aleatoria, cada búsqueda tiene la misma probabilidad de encontrar el óptimo. Esto provoca que se requieran muchos reinicios para realizar la exploración del espacio de búsqueda, disminuyendo drásticamente la probabilidad de que el nuevo reinicio mejore la solución actual. Para superar ésta limitación, la búsqueda local iterada propone un método de reinicio que explora la vecindad del óptimo local de forma aleatoria, con el objetivo de encontrar una solución que mueva la solución a un nuevo óptimo local.

La búsqueda local iterada está conformada por tres componentes principales:

- **Perturbación**

  Mutación al óptimo actual, con el fin de generar una nueva solución que sea suficientemente diferente para encontrarse en un nuevo atractor, pero bastante cercano para mantener algunas de las características benéficas con las que cuenta el óptimo local encontrado anteriormente.

- **Búsqueda Local**

  Heurística utilizada para mover la solución actual a una posiblemente mejor solución. Su selección depende del problema, de la restricción de tiempo y su interacción con la perturbación. Como regla general cuanto mejor sea la búsqueda local, mejores resultados se obtendrán.

- **Criterio de aceptación**

  Determina la trayectoria realizada por la búsqueda y permite controlar la intensificación y exploración. Pueden utilizarse diferentes criterios, desde un criterio de Markov en donde solo se aceptan las soluciones que mejoran, hasta una caminata aleatoria en donde se visita el nuevo óptimo independiente de su costo.

En el Algoritmo {7} se muestra la búsqueda local iterada. En donde primero se realiza una perturbación de la solución actual s'. Después se prosigue a realizar la búsqueda local sobre la solución modificada s hasta encontrar un nuevo óptimo local. Si el óptimo local mejora la mejor solución conocida s^*, se actualiza s^*. Finalmente se decide la dirección de la solución al sustituir s' con la nueva solución encontrada s si se cumple el criterio de aceptación.

Por ejemplo, al aplicar la búsqueda local iterada al TSP, la opción de perturbación más utilizada es el k-intercambio [29]. El cual se define como una transformación de la concatenación de los k-segmentación $\Xi = \{s_1, s_2, \ldots, s_k\}$ que conforman el recorrido Ξ en $\Xi' = \{s_1, s_k, s_{k-1}, \ldots, s_2\}$. En la Figura 2.1 se muestra el k-intercambio más utilizado en la literatura, el cual emplea cuatro segmentos y es conocido como movimiento doble puente.

Existen diversas opciones a utilizar como estrategia de búsqueda, entre ellas las búsquedas locales, como puede ser aquellas que emplean una vecindad k-opt. También pueden emplearse otros mecanismos de mejora como lo es la estrategia LK, la cual es actualmente la más utilizada pues obtiene mejores resultados con un coste computacional aceptable. Al estudiarse el criterio de aceptación se ha observado que solo aceptar soluciones que mejoran presenta un estancamiento de la búsqueda en ejecuciones

Algoritmo 7: Búsqueda Local Iterada

def *búsqueda-local-iterada (s = solución inicial)*
  $s^* = s' = s$
  hasta que *se cumpla el criterio de paro*
    s = perturbación de la solución s'
    s = búsqueda local sobre s
    $s^* = s$ **si** s *es mejor que* s^*
    s' = criterio de aceptación de s
  fin
  regresa *la mejor solución encontrada* s^*
fin

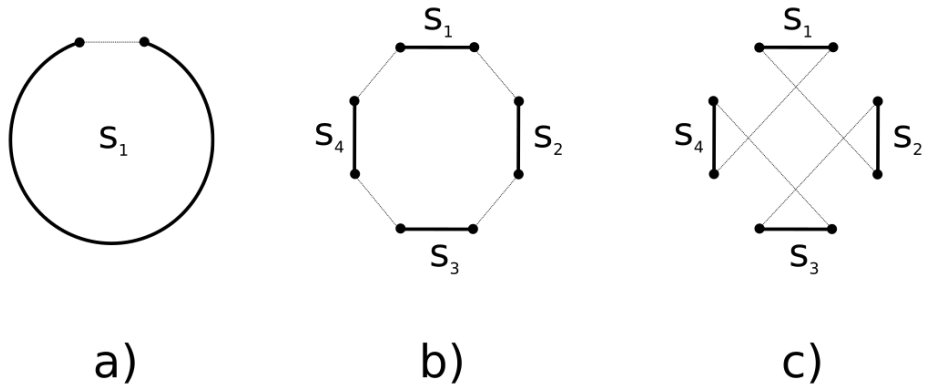


Figura 2.1: Movimiento de doble puente (4-intercambio) consiste en realizar una 4-segmentación (b) del recorrido inicial (a) para generar un nuevo recorrido (c)

largas, por lo que cambiar el criterio de aceptación para forzar la exploración (e.g. $s^* = s$ si $d(s^*, s) > d_{\text{mín}}$) ha logrado mejorar la calidad de solución [30].

2.3. Métodos poblacionales

Los métodos poblacionales deben su nombre a que manejan un conjunto de soluciones, que son consideradas de forma simultánea al realizar decisiones en la búsqueda. El Algoritmo {8} muestra el esquema general de los algoritmos poblacionales, primero se genera una nueva población, la cual en cada iteración intenta moverse hacia mejores soluciones del problema mediante la generación de nuevos individuos y alguna estrategia de selección.

Algoritmo 8: Estructura de un algoritmo poblacional

```
def metaheuristica-poblacional
    P = generación de la población inicial
    hasta que se cumpla el criterio de paro
        C = generar una nueva población
        P = seleccionar una nueva población de P ∪ C
    fin
    regresa la mejor solución encontrada
fin
```

Los algoritmos poblacionales emplean las soluciones de la población para realizar la exploración del espacio de búsqueda, por ello, en la generación de la población inicial es de gran importancia generar soluciones diversas. Siendo usual que las soluciones se generen de forma aleatoria, siendo deseable que las soluciones se encuentren distribuidas sobre todo el espacio de búsqueda.

Entre los algoritmos poblacionales, dos de los grupos más populares son: algoritmos evolutivos (*Evolutionary Algorithm* — EA) y algoritmos de inteligencia de enjambre (*Swarm Intellingece* — SI) [31].

Algoritmos evolutivos

En esta categoría se clasifican aquellos algoritmos inspirados en la selección natural y herencia genética.

Las clases más representativas de esta categoría son:

- **Algoritmo Genético [32] (*Genetic Algorithm* — GA)**

  Quizá el método más conocido y utilizado, basado en los principios de selección natural esta conformado por las fases de: cruza, mutación y selección.

- **Programación Evolutiva [33] (*Evolutionary Programming* — EP)**

  A diferencia de otros algoritmos evolutivos, la codificación se enfoca a nivel de fenotipo (evolución a nivel de especies) y no emplea los conceptos de evolución genética por lo que carece de fase de cruza, siendo la mutación el mecanismo de diversificación.

- **Estrategia Evolutiva [34] (*Evolutionary Strategy* — ES)**

  Se enfocan en problemas de optimización continua, y tienen una estrategia de selección elitista, algunas veces careciendo de fase de cruza. La mutación se realiza mediante una distribución normal.

- **Programación Genética [35] (*Genetic Programming* — GP)**

  En este tipo de algoritmos los individuos de la población son en sí programas. El objetivo es permitir la generación automática de un programa que resuelva la tarea especificada.

Inteligencia de enjambre

Esta segunda categoría contiene aquellos algoritmos inspirados en el comportamiento colectivo de colonias de insectos o sociedades de animales.

Entre las clases de algoritmos más representativas se encuentran:

- **Optimización por enjambre de partículas [36] (*Particle Swarm Optimization* — PSO)**

  Basado en el comportamiento de congregación de las aves, es un método enfocado en optimización continua. Distribuye en el espacio de búsqueda un conjunto de partículas. En función de una topología de comunicación actualiza la posición de cada partícula utilizando la posición de la mejor partícula en su red, la mejor posición conocida por la partícula y su posición actual.

- **Colonia de abejas artificiales [37] (*Artificial Bee Colony* — ABC)**

  Simula el comportamiento de recolección de nectar de las abejas y fue diseñado con un enfoque en optimización continua. Emplea tres tipos de abejas (individuos), trabajadoras, desempleadas y exploradoras. Las abejas trabajadoras desempeñan el papel de intensificación, explorando la vecindad sobre una solución candidata. Las abejas desempleadas realizan intensificación en las zonas más prometedoras. Finalmente las abejas exploradoras se emplean en la fase de exploración del espacio de búsqueda.

- **Optimización de colonia de hormigas [38] (*Ant Colony Optimization* — ACO)**

  Este algoritmo de optimización combinatoria se basa en el comportamiento alimenticio de algunas especies de hormigas. Consiste de dos pasos: generar soluciones candidatas usando un modelo probabilístico del espacio de soluciones y utilizar las soluciones para modificar el modelo. En este sentido una hormiga genera una solución empleando el modelo probabilístico (feromonas) y se actualiza el modelo en función de la calidad de la solución encontrada.

2.3.1. Algoritmos evolutivos

Este trabajo se enfocará en utilizar la estrategia de los algoritmos evolutivos. El Algoritmo {9} esquematiza de forma general un EA, identificándose distintas fases, cada una con diferentes aspectos a considerar [11, 39].

Algoritmo 9: Estructura de un algoritmo evolutivo

```
def algoritmo-evolutivo
    P = generar la nueva población
    hasta que se cumpla el criterio de paro
        P_a = seleccionar los padres de la población P
        C = generar la descendencia mediante la cruza de los padres
        mutar la descendencia C
        P = seleccionar las soluciones que sobreviven de P ∪ C
    fin
    regresa el mejor individuo encontrado
fin
```

Inicialización

El punto de inicio para cualquier algoritmo poblacional, consiste en generar la primera generación de individuos. Esta población inicial generalmente consiste de N individuos creados de forma aleatoria,

de forma que esta primer generación esté distribuida uniformemente en el espacio de búsqueda. El que los individuos de la población estén dispersos en el espacio de búsqueda aumenta probabilidad de detectar una zona prometedora.

Selección de padres

En cada generación, el primer paso consiste en seleccionar las soluciones de la población que serán utilizadas en la fase de cruza. Para aumentar la probabilidad de preservar las característica que definen a las mejores soluciones, los algoritmos de selección deben tener una preferencia hacia los individuos de alta calidad.

Algunos algoritmos de selección son:

- **Selección Proporcional**

  Consiste en asignar una probabilidad de seleccionar a cada individuo en función de su valor de aptitud.

- **Clasificación**

  Se asigna una clasificación a cada individuo y en base a ésta se asigna su probabilidad de ser seleccionado. Éste tipo de selección generalmente es utilizado cuando no se puede obtener eficientemente la aptitud de las soluciones o cuando se desea modificar de forma directa la presión de selección.

- **Torneo**

  En esta selección se eligen de forma aleatoria k elementos de la población, de estos elementos se selecciona al mejor individuo. La selección por torneo es bastante utilizada pues emplea solo información local, es fácil de implementar y su complejidad en tiempo es baja.

Cruza

En la etapa de cruza o recombinación, se combinan las características presentes en los padres para generar un nuevo conjunto de individuos (hijos). Para preservar las características de las mejores soluciones, en general la cruza de dos padres iguales, produce un hijo idéntico a ellos, mientras que dos padres cercanos en el espacio de búsqueda producen hijos cercanos a ellos. La cruza es el mecanismo de exploración del algoritmo evolutivo, al combinar las características de padres lejanos, los nuevos individuos estarán en una nueva zona del espacio de búsqueda.

Los operadores de cruza son específicos de la representación de las soluciones. Como ejemplo, utilizando una representación binaria (un individuo consiste de un arreglo cuyos elementos pueden ser verdadero V o falseo F), algunos operadores ampliamente utilizados son:

- **Aritmética**

  El nuevo individuo se genera aplicando una operación aritmética sobre los valores de ambos padres. Por ejemplo si se emplea el operador lógico $\wedge$, el valor del hijo en la posición i es: $h_i = p_{1,i} \wedge p_{2,i}$.

- **Multi punto**

  Para generar el nuevo individuo, se seleccionan k posiciones en el arreglo. De los $k+1$ segmentos generados por los puntos de cruza se intercambian los segmentos impares de los padres.

- **Uniforme**

  La descendencia se genera recorriendo cada posición del arreglo, intercambiando con una probabilidad fija los valores de los padres. Usualmente se emplea una probabilidad de $1/2$.

Mutación

Una vez que se tiene una población de hijos (descendencia), se aplica el operador de mutación con cierta probabilidad a los individuos.

El proceso de mutación consiste en modificar un individuo para generar uno nuevo y su intensidad determina que tan diferente es el individuo generado del original (cantidad de características modificadas). Aunque el proceso de mutación ha tenido diferentes aplicaciones en los algoritmos poblacionales, su aplicación típica es para promover la intensificación al generar soluciones cercanas mediante una intensidad de mutación baja.

Por ejemplo en la representación binaria, un operador de mutación clásico consiste en invertir los valores. Esto es si el valor es F se cambia a V y si es V se cambia a F.

Reemplazamiento

Finalmente se realiza el reemplazamiento, esto es la selección de los individuos que pasarán a la siguiente generación. El reemplazamiento en combinación con la selección de padres, determina la convergencia del algoritmo evolutivo. Algunos ejemplos de categorías de estrategias son:

- **Elitista**

  Asegura la convergencia al elegir el mejor individuo en cada generación. En muchas implementaciones se opta por descartar siempre la peor solución entre la descendencia y la población actual.

- **Aleatorio**

  Se selecciona de forma aleatoria los individuos que pasan a la siguiente generación. En función de las probabilidades de selección, existe la posibilidad de que se pierdan las mejores soluciones.

- **Competición**

  Se compara la descendencia frente a los padres y se selecciona la mejor solución. De esta forma los mejores individuos siempre se mantienen en la población.

Criterio de paro

La selección, cruza, mutación y reemplazamiento conforman una generación del algoritmo evolutivo. Este proceso se realiza iterativamente, hasta que uno o más criterios de paro se cumplan. Existen tres categorías ampliamente utilizadas:

- **Convergencia**

  Se define una medida de que tan diferentes son los individuos, generalmente se emplea la aptitud y se termina el proceso iterativo cuando la mayoría de los individuos son similares.

- **Cambio en la aptitud**

  Se detiene el algoritmo cuando el cambio en el valor de aptitud entre las generaciones, sea menor a un umbral predefinido.

- **Tiempo**

  Se termina el algoritmo cuando se cumple un número de iteraciones, evaluación de función objetivo o tiempo previamente establecido.

Una propiedad fundamental de los algoritmos evolutivos es el balance entre exploración y explotación. La exploración (distribuir las soluciones en el espacio de búsqueda) permite encontrar las zonas más prometedoras de contener el óptimo global. La explotación (concentrar las soluciones en una zona del

espacio de búsqueda) aumenta la probabilidad de encontrar los mejores óptimos locales de la región en la que se enfoca el algoritmo. La dificultad de encontrar un balance adecuado entre explotación y exploración, se ve reflejada en múltiples problemas como lo son la convergencia lenta (se explora pero no se intensifica) o convergencia prematura (se intensifica pero no se explora). Por lo anterior el encontrar un balance adecuado es un paso importante en el diseño de los algoritmos evolutivos [40].

Para tener un mejor control en este balance, se han propuesto múltiples modificaciones a los algoritmos evolutivos. Por ejemplo, los algoritmos meméticos suelen aumentar la explotación, mientras que los algoritmos que incluyen técnicas de diversidad son utilizados para evitar la convergencia prematura.

2.3.2. Algoritmos meméticos

En contraste a los EA, se denomina como algoritmos meméticos (*Memetic Algorithms* — MA) a los algoritmos híbridos que incorporan información del dominio del problema al integrar en la metaheurística poblacional una metaheurística de trayectoria [11].

El éxito de los algoritmos MA se debe a la capacidad de combinar la exploración (mediante el algoritmo genético) y la explotación (mediante una heurística de búsqueda) de una manera eficiente [41]. La incorporación de la búsqueda local, aumenta la velocidad de convergencia y evita la pérdida de zonas prometedoras del espacio de búsqueda (deriva genética). La estructura de un algoritmo memético se muestra en el Algoritmo {10}. Note que la fase de mutación ha sido sustituida por la búsqueda local, debido a que ambas fases están relacionadas con la intensificación es usual que los algoritmos meméticos obvien la mutación.

Algoritmo 10: Estructura de un algoritmo memético

```
def algoritmo-memético
    P = generar la población inicial
    hasta que se cumpla el criterio de paro
        P_a = seleccionar los padres de la población P
        C = cruzar la población P_a para generar la descendencia
        aplicar una heurística de búsqueda en la descendencia
        P = seleccionar las soluciones que sobreviven de P ∪ C
    fin
    regresa el mejor individuo encontrado
fin
```

Para ciertos problemas de optimización los algoritmos meméticos han mostrado ser órdenes de magnitud más precisos que los algoritmos evolutivos tradicionales. Sin embargo, aumentan el costo computacional al incrementar el número de evaluaciones de la función objetivo y son más propensos a perder la diversidad de la población. Por ello es típico que integren mecanismos de control de diversidad [42].

2.3.3. Control de diversidad

Un problema fundamental de los algoritmos evolutivos es la convergencia prematura, tendencia de la población a enfocarse en una zona, debida a una rápida homogeneización de los individuos, lo que evita la exploración del espacio de búsqueda.

Para evitar la convergencia prematura se requiere controlar la variedad de individuos que conforman la población (diversidad). El estudio del control de diversidad ha derivado en una gran cantidad de propuestas para lidiar con la convergencia prematura.

En [43, 44], los autores realizan una revisión de métodos para promover la diversidad, y se presenta las siguientes clasificaciones para las estrategias de control de diversidad:

- **En función del nivel en el que se enfoca la estrategia**
  - ***Linaje***

    Operan sobre las condiciones en las cuales se generan los individuos, un ejemplo es el modelo de islas [45], en donde se limita la interacción entre subconjuntos de la población.
  - ***Genotipo***

    Se basan en la representación de la solución en el problema. Por ejemplo la división de aptitud [46], consiste en penalizar las zonas más pobladas empleando la distancia entre soluciones, escalando la aptitud de la solución proporcionalmente a la cantidad de soluciones cercanas.
  - ***Fenotipo***

    Trabajan directamente sobre el espacio de aptitud (el comportamiento de los individuos). El operador de comparación de apiñamiento [47] pertenece a esta categoría. Este operador surge en la optimización multi-objetivo, emplea una distancia entre soluciones en el espacio de objetivos (aptitud).
- **En función de la fase modificada por la estrategia**
  - ***Selección de padres***

    La mayoría de los primeros algoritmos para el control de diversidad pertenecen a esta categoría, consisten en controlar la presión de selección en la selección de padres. Este tipo de esquemas son incapaces de mantener la diversidad a largo plazo.
  - ***Reemplazamiento***

    Propician la exploración en la siguiente generación al pluralizar los individuos seleccionados. Un ejemplo es la selección restringida por torneo [48], en donde por cada hijo se selecciona el mejor individuo entre el hijo y el individuo más cercano al hijo de un conjunto de k individuos seleccionados aleatoriamente de la población.
  - ***Cruza***

    Se imponen restricciones en la cruza entre los individuos de una parte o la totalidad de la población. Las restricciones toman en cuenta las características de los individuos. Por ejemplo el esquema de recombinación sin incesto del algoritmo CHC [49] donde se evita que soluciones cercanas sean cruzadas.
  - ***Evaluación***

    En estas estrategias, se modifica la forma de calcular la aptitud de cada individuo. El esquema más conocido de esta categoría es la división de aptitud [46].
  - ***Población***

    Estos esquemas afectan la estructura de la población, restringiendo la interacción entre los individuos, lo que retrasa la pérdida de diversidad. El modelo de islas [45] es un ejemplo de estrategia que modifica la población.
- **Si es dependiente del contexto**

  Una estrategia de preservación de diversidad es dependiente del contexto si la probabilidad de selección de un individuo es condicionado por los elementos previamente seleccionados.
- **Cantidad de modificaciones que aplica la estrategia**

  Cuando la estrategia modifica una sola fase del algoritmo (uni proceso) o modifica dos o más fases (multi proceso).

La implementación de un control de diversidad requiere de una medida de la diferencia entre dos individuos. La métrica utilizada depende de la fase que se esté modificando y de la representación de la solución, por ejemplo una métrica ampliamente utilizada en la representación binaria es la distancia de Hamming [50] (número de posiciones en las que los arreglos son diferentes). Existen dos formas principales de medir la diversidad de una población: a nivel individual (a qué distancia se encuentra el individuo de la población) y poblacional (cuál es la distancia individual promedio).

Uno de los algoritmos más recientes para el manejo de diversidad es el algoritmo RMDDC [12], el cual emplea una estrategia dependiente del contexto, centrada en el genotipo y dirigida a la etapa de reemplazamiento. En contraste con los algoritmos clásicos, RMDDC, combina la idea de transformar un problema de un solo objetivo a varios objetivos para controlar el balance entre exploración y explotación, incorporando el criterio de paro en su diseño.

El Algoritmo {11} describe el funcionamiento de la estrategia de reemplazamiento RMDDC. Primero se genera un conjunto M de soluciones candidatas empleando la población y la descendencia. Después se genera una nueva población P_n con la mejor solución de M, a continuación a cada solución se le asigna una dupla {aptitud,distancia}, la distancia utilizada es la distancia a la solución más cercana en la población P_n (*Distance to Closest Neighbor* — DCN). Con la intención de promover la diversidad, se penaliza la aptitud de la soluciones que se encuentren cerca de los elementos previamente seleccionados P_n ($DCN < D$). La penalización fuerza al algoritmo a elegir los elementos cercanos a P_n solamente en función de su diversidad, ignorando su valor de aptitud. La distancia D es una función del criterio de paro, la cual va decreciendo de un valor inicial D_0 hasta llegar a 0 (ninguna solución es penalizada). A continuación se obtienen el conjunto N de soluciones no dominadas ($\{s \in M \mid \nexists\ s' \preceq s \in M \setminus s\}$), es decir, aquellas soluciones para las cuales no existe otro elemento en M que tenga mejor aptitud y mejor DCN. Finalmente del conjunto de soluciones N se elige aleatoriamente la solución que se agrega a P_n.

Algoritmo 11: RMDDC: Esquema de reemplazamiento

def *rmddc (P = población, C = descendencia)*
- $M = P \cup C$
- $P_n = \varnothing \cup \{\text{mejor individuo} \in M\}$
- $M = M \setminus P$
- $D = \left(1 - \frac{\text{tiempo transcurrido}}{\text{tiempo total}}\right) D_0$
- **hasta que** *se tenga $|P|$ individuos en P_n*
  - calcular DCN de cada elemento de M usando de referencia P_n
  - penalizar la aptitud de M empleando la distancia D
  - obtener el conjunto N de los elementos no dominados (sin duplicados)
  - seleccionar $s \in N$ de manera aleatoria de N
  - $P_n = P_n \cup s$
  - $M = M \setminus s$
- **fin**
- **regresa** *la población P_n*

fin

Capítulo 3

El problema del ladrón viajero

Este capítulo describe el problema del ladrón viajero, se realiza una breve descripción de sus componentes, así como las técnicas clásicas para resolverlo. Después se define formalmente el TTP, y se realiza una revisión bibliográfica de las principales metaheurísticas diseñadas para el problema. Finalmente se propone una clasificación de las heurísticas en función de su enfoque al resolver el TTP.

3.1. Sub-problemas

Los problemas reales generalmente están conformados de subproblemas interdependientes. Mientras que los problemas académicos en la mayoría de los casos consisten de un solo problema o no modelan una interdependencia entre los componentes. Para reducir esta brecha entre los problemas reales y académicos, en el 2013 se propuso el problema del ladrón viajero.

El problema del ladrón viajero es un problema de optimización, que considera dos componentes NP-Complejos, uno tiene un espacio de búsqueda basado en variables binarias y el otro basado en permutaciones. Concretamente el TTP combina el problema del vendedor viajero o TSP (el recorrido realizado por el ladrón) y el problema de la mochila o KP (selección de objetos a robar). Ambos problemas se acoplan de una forma no lineal, con el objetivo de estudiar el desempeño de los algoritmos frente a la interacción de sus componentes interdependientes.

Estos dos problemas, han sido estudiados por la comunidad científica por más de medio siglo [51, 52]. Debido a la larga trayectoria de ambos problemas, cada uno de ellos cuenta con un amplio repertorio de estrategias para aproximar soluciones de alta calidad. A continuación se describen brevemente ambos problemas y las principales estrategias para aproximarlos. Para una descripción más detallada se refiere al lector a [53] y [54] para el TSP y KP respectivamente.

3.1.1. El problema del vendedor viajero (TSP)

Es uno de los problemas más conocidos de optimización combinatoria, el cual consiste en: Dadas N ciudades (vértices) y un costo d (distancia) entre cada par de ciudades (aristas), determinar el ciclo que visita una sola vez cada ciudad (ciclo Hamiltoniano) de menor costo. El problema tiene dos variantes principales, el problema simétrico en donde la distancia de la arista $d(i, j) = d(j, i)$ y el no simétrico en el cual para algunos valores de i y j, se cumple que $d(i, j) \neq d(j, i)$. Existen diferentes variantes en función de la definición de distancia, siendo la distancia euclideana una de las más utilizadas.

El TSP como problema de minimización se expresa como:

$$\underset{\Xi \in S}{\arg} \left[T(\Xi) = \sum_{i=1}^{n} d_{\xi_i} \right] \mid T(\Xi) \leq T(s) \ , \ \forall s \in S \tag{3.1}$$

$S = \{\Xi_0, \ldots, \Xi_k\}$ es el conjunto de ciclos Hamiltonianos, cada recorrido $\Xi = (\xi_1, \xi_2, \ldots, \xi_n)$ se puede expresar como una permutación de las ciudades y la función $T(\Xi)$ es el costo del recorrido Ξ el cual consiste en la suma de la distancia d_ξ de la ciudad ξ a la siguiente.

Las principales estrategias para resolver el TSP son:

- **Métodos exactos** [55]

  Algoritmos de enumeración que emplean estrategias como ramificación y poda, programación lineal, y programación dinámica. La aplicación de estas estrategias es costosa y para instancias de gran tamaño resulta ser inviable. Una de las implementaciones más conocidas es el software CONCORDE [56].

- **Métodos de trayectoria** [57]

  Algoritmos basados en la búsqueda local, así como algunas extensiones de los mismos como ILS o (*Ejection Chain* — EC), siendo esta última posiblemente la familia de métodos la más exitosa. Esta familia genera vecindades conformadas por una secuencia de sub-vecindades. Los ejemplos más representativos son las variantes algoritmo estándar para encontrar recorridos (*Lin Kernighan* — LK). La implementación más extendida de este algoritmo (LK) es el programa LKH [58].

- **Métodos poblacionales** [59]

  Son los métodos utilizados para resolver el TSP cuando se requiere de una alta calidad de solución y se tiene disponibilidad de tiempo. Los operadores de cruza más utilizados tienen como característica el preservar aristas en común de los padres, algunos ejemplos de estos operadores son [60]: preservación máxima (*Maximal Preservation Crossover* — MPX), ensamblaje de aristas (*Edge Assembly Crossover* — EAX) y recombinación de aristas (*Edge Recombination Crossover* — ERX). En muchos casos son algoritmos híbridos, que pueden utilizar la metaheurística de trayectoria LKH.

Un concepto fundamental en la aproximación del TSP es el de un recorrido λ óptimo. Se define que un recorrido es λ óptimo si no existe un camino más corto al remplazar cualquier conjunto de λ aristas por cualquier otro conjunto de λ aristas. Entonces la vecindad λ-opt se define como el conjunto de posibles remplazos de λ aristas. Debido a que la complejidad de probar λ-intercambios es $O(n^\lambda)$, se restringe el valor de λ, siendo los valores más utilizados $\lambda = 2$ y $\lambda = 3$.

En el Algoritmo {12} se bosqueja la implementación directa de la vecindad 2-opt.

Note que un recorrido de N ciudades es óptimo si es N-opt. Cuanto más grande sea λ más se puede acercar la solución al óptimo. Empleando técnicas de geometría computacional, se puede mejorar la eficiencia del algoritmo, llegando a valores en torno a $\lambda = 5$. La estrategia LK consiste entonces en definir una vecindad λ-opt variable, iniciando de $\lambda = 2$ y realizando en cada iteración una serie de pruebas para determinar si se debe considerar la vecindad $\lambda + 1$ [6]. Al considerar vecindades de mayor complejidad, la metaheurística LK considera los movimientos más promisorios, otorgando el buen balance que presenta esta familia de estrategias.

En el contexto de algoritmos poblacionales se ha encontrado que para obtener un buen rendimiento de los algoritmos se requiere de un operador que preserve las aristas de los padres, un ejemplo de operador es el operador ERX [61].

El Algoritmo {13} bosqueja el funcionamiento de este operador, donde se genera un conjunto M conformado por $|\Xi|$ conjuntos. Cada ciudad ξ tiene asociado un conjunto $M_\xi \in M$, el cual está

Algoritmo 12: Vecindad 2-Opt para el TSP

def *vecindad-2opt (Ξ = recorrido inicial)*
  $V = \varnothing$
  para cada $k \in (\ 0\ \ldots\ (|\Xi| - 1)\ )$ **hacer**
    para cada $l \in (\ (k+1)\ \ldots\ (|\Xi| - 1)\ )$ **hacer**
      $\Xi^* = \Xi$
      invertir el segmento $\Xi^*_k \ldots \Xi^*_l$
      $V = V \cup \Xi^*$
    fin
  fin
  regresa *la vecindad V*
fin

conformado por las aristas $e \in \hat{E}$ en las que la ciudad ξ esta presente ($\xi \in e$). Empleando los conjuntos de aristas asociados a cada ciudad se genera el nuevo recorrido. En cada iteración se agrega la ciudad seleccionada ξ al nuevo recorrido Ξ y se elimina de las posibles opciones de selección. Finalmente se elige la próxima ciudad empleando dos criterios. Primero si no existe una arista en el conjunto M_ξ se selecciona una nueva ciudad aleatoriamente. En caso de existir una arista se selecciona aquella ciudad ξ^* que tenga asociado el conjunto con menor cantidad de aristas (ξ^* tal que $|M_{\xi^*}| \leq |M_s|\ \forall s \in \Xi^*$), en caso de que existan una o más ciudades que satisfagan este criterio se selecciona aleatoriamente entre ellas.

Note que una vez que se cuenta con las aristas de cada ciudad, el siguiente paso en la generación del recorrido es seleccionar la ciudad inicial. Para ello se selecciona de forma aleatoria de entre las posibles ciudades. En el TTP se tiene la restricción de iniciar el recorrido siempre en la misma ciudad, para facilitar su implementación y mejorar la eficiencia del algoritmo, siempre se inicia la generación en la ciudad inicial 0.

3.1.2. El problema de la mochila (KP)

El problema de la mochila consiste en: Dado un conjunto de objetos M, cada objeto π con una ganancia entera π_p y un peso entero π_w. Se debe encontrar el subconjunto $\Pi \subseteq M$ tal que se maximice la suma de las ganancias sin exceder una capacidad máxima de peso W establecida.

El KP como problema de maximización se modela de la siguiente manera:

$$\underset{\Pi \subseteq S}{\arg} \left[K(\Pi) = \sum_{\pi \in \Pi} \pi_p \right] \ \mid\ K(\Pi) \geq K(s),\ \forall s \in S \tag{3.2}$$

En el KP, $S = \{s \subseteq M \mid \sum_{\pi \in s} \pi_w \leq W\}$ es el conjunto de soluciones factibles (subconjuntos de M que cumplen con la restricción de peso). El conjunto $M = \{(\pi)_1 \ldots (\pi)_m\}$ es el conjunto de posibles objetos a seleccionar y $\Pi = \{(\pi)_i \ldots (\pi)_k\}$ son los objetos seleccionados. La función $K(\Pi)$ es la ganancia total de llevar los objetos en Π y consiste en la suma de las ganancias individuales de cada objeto π seleccionado.

Al igual que el TSP, el problema de la mochila ha sido ampliamente estudiado y se cuentan con estrategias que ofrecen una alta calidad en las soluciones. Actualmente se considera que se han resuelto las instancias de referencia clásicas y el enfoque se ha dirigido a aumentar su complejidad [62], la generalización del problema [63] u optimización multi-objetivo [64].

Los principales métodos para resolver el KP son:

Algoritmo 13: Operador de cruza ERX

def *erx* (Ξ_1 = *primer padre*,Ξ_2 = *segundo padre*)
  $M = \{\varnothing_0 \dots \varnothing_{|\Xi_1|-1}\}$
  $\hat{E} = E(\Xi_1) \cup E(\Xi_2)$
  para cada $\xi \in \Xi$ **hacer**
    $M_\xi = M_\xi \cup \{\xi^* \mid (e \in \hat{E}) \wedge (\xi^* \in e) \wedge (\xi \in e)\}$
  fin
  $\Xi^* = \{0 \dots (|\Xi_1| - 1)\}$
  $\Xi = \varnothing$
  ξ = selección de la ciudad inicial
  bucle hacer
    $\Xi = \Xi \cup \xi$
    romper si $\Xi^* = \varnothing$
    $\Xi^* = \Xi^* \setminus \xi$
    para cada $\xi' \in \Xi^*$ **hacer** $M_{\xi'} = M_{\xi'} \setminus \xi$
    $\xi =$ **caso** M_ξ
      cuando $\varnothing$
        elegir aleatoriamente de Ξ^*
      en otro caso
        elegir de M_ξ la ciudad ξ^* tal que $|M_{\xi^*}| \leq M_s, \forall s \in M_\xi$
    fin
  fin
  regresa Ξ
fin

- **Exactos** [65]

  Obtienen el óptimo del problema, incluyen estrategias de ramificación y poda, programación lineal y programación dinámica.

- **Métodos Aproximados** [66]

  Estiman una solución de alta calidad en un margen de tiempo corto, entre los más utilizados se encuentran las metaheurísticas de trayectoria y algoritmos genéticos, aunque en la última década se han aplicado algoritmos de inteligencia de enjambre con bastante éxito.

Una de las principales características del problema de la mochila, es que se han podido desarrollar algoritmos exactos, que lo resuelven en tiempo pseudo-polinomial. Es decir que los algoritmos tienen un tiempo de ejecución que depende no solo de la cantidad de objetos, sino de la magnitud de los valores de éstos. Entre los algoritmos exactos, la mejor estrategia conocida es el algoritmo COMBO [67]. Mientras que la primer implementación que fue capaz de resolverlo en tiempo pseudo-polinomial es la aplicación directa de programación dinámica.

El algoritmo {14} muestra una implementación directa de programación dinámica para resolver el problema de la mochila con un tiempo de ejecución pseudo-polinomial $O(MW)$.

El algoritmo de programación dinámica se divide en dos partes, la primera (Algoritmo {14}) genera la matriz de programación dinámica, en la cual cada celda dp_{iw} representa la ganancia máxima que se puede obtener al considerar los primeros i objetos con una capacidad máxima w. El problema de selección consiste en agregar o no un objeto a la mochila, si no es posible añadirlo se descarta ($dp_{i,w} = dp_{i-1,w}$), mientras que si es posible llevarlo se decide mediante el valor máximo entre llevarlo v_p o descartarlo v_d.

La segunda parte (Algoritmo {15}) consiste en reconstruir los objetos seleccionados, una vez que se

Algoritmo 14: Solución del problema de la mochila (KP) mediante programación dinámica

```
def kp-dp (M = posibles objetos,W = capacidad máxima)
    dp = matriz de |M| + 1 filas y W + 1 columnas inicializada en 0
    para cada i ∈ ( 1 ... |M| ) hacer
        para cada w ∈ ( 1 ... W ) hacer
            si (π_w)_i < w entonces
                v_d = dp_{i−1,w}
                v_p = dp_{i−1,w−(π_w)_i} + (π_p)_i
                dp_{i,w} = v_d > v_p ? v_d : v_p
            en otro caso
                dp_{i,w} = dp_{i−1,w}
            fin
        fin
    fin
    regresa dp
fin
```

Algoritmo 15: Reconstrucción del recorrido a partir de la matriz DP

```
def dp-plan (dp = matriz de programación dinámica)
    Π = ∅
    w = W
    para cada i ∈ ( |M| ... 1 ) hacer
        si el valor dp_{i,w} ≠ dp_{i−1,w} entonces
            Π = Π ∪ (π)_i
            w = w − (π_w)_i
        fin
    fin
    regresa Π
fin
```

cuenta con la matriz dp. El valor de interés para el KP es considerar todos los objetos $|M|$ empleando la máxima capacidad de la mochila W. Este valor puede ocurrir por dos situaciones, haber ignorado el objeto en cuyo caso $dp_{m,w} = dp_{m-1,w}$ o haber seleccionado el objeto en cuyo caso proviene de agregar el valor del objeto π_p al máximo valor en la posición $dp_{m-1,w-\pi_w}$. Se va recorriendo objeto por objeto agregando a la solución Π los objetos que modificaron el valor de ganancia en la matriz dp.

Por otra parte el concepto fundamental al aproximar el KP es el estado de cada objeto, se lleva o se ignora, naturalmente surge el concepto de vecindad Bit-Flip, la cual consiste del conjunto de soluciones factibles generadas al modificar (invertir) en Π el estado de cada elemento del conjunto M. En el Algoritmo {16} se bosqueja la implementación de la vecindad Bit-Flip. Primero se genera un conjunto vacío $V = \varnothing$ que contendrá los vecinos de Π. La variable w contiene el peso acumulado por los objetos presentes en la selección. Para cada objeto, se elimina si esta presente en la selección actual y se agrega en caso contrario. Al agregarse un objeto se debe comprobar la restricción de peso, esto es verificar que la suma del peso actual w y el peso del objeto π_w no exceda el peso máximo W.

Algoritmo 16: Vecindad Bit-Flip para el KP

def *bit-flip (Π : selección inicial)*
  $V = \varnothing$
  $w =$ peso total de los objetos en Π
  para cada $\pi \in M$ **hacer**
    si $\pi \in \Pi$
      $V = V \cup \{\Pi \setminus \pi\}$
    si no, si $\pi_w + w \leq W$
      $V = V \cup \{\Pi \cup \pi\}$
    fin
  fin
  regresa V
fin

3.2. Definición formal del TTP

El problema del ladrón viajero es un problema relativamente nuevo, propuesto en el 2013 [3], presentando la novedad de contar con una interdependencia no lineal entre los subproblemas que lo componen. El TTP es planteado de la siguiente manera:

> Dado un conjunto $N = \{(\xi)_1 \dots (\xi)_n\}$ de ciudades y un conjunto $M = \{(\pi)_1 \dots (\pi)_m\}$ de objetos, cada uno con una ganancia π_p y un peso π_w, repartidos entre las ciudades. Un ladrón parte de una ciudad establecida, visita una vez y solo una vez cada una de las $|N|$ ciudades y regresa a la ciudad de origen. Si el transporte de los objetos tiene un costo ρ por unidad de tiempo, su velocidad es función del peso $v(w)$ y tiene una capacidad máxima de peso W. ¿Cuál es la mejor ruta Ξ y selección de objetos Π para obtener el máximo beneficio?.

Obsérvese la similitud del TTP con los el problema del vendedor viajero (3.1) y el del problema de la mochila (3.2). La solución $\tau = \{\Xi, \Pi\}$ del TTP es una dupla de un ciclo Hamiltoniano Ξ y un conjunto de objetos $\Pi \subseteq M$, siendo su principal objetivo maximizar la ganancia total.

El TTP como problema de maximización se modela de la siguiente manera:

$$\underset{\tau \in S}{\arg} \left[F(\tau) = \sum_{\pi \in \Pi} \pi_p - \rho \sum_{\xi \in \Xi} \frac{d_\xi}{v_\xi} \right] \mid F(\tau) \geq F(s) \ \forall s \in S \tag{3.3}$$

Donde S es el conjunto de soluciones factibles, conformado por las soluciones de selección de objetos Π (que cumplan la restricción de peso $\sum \pi_w \leq W$) por cada permutación posible del recorrido Ξ. La función objetivo $F(\tau)$ consiste de dos partes, la primera corresponde a la ganancia total de los objetos seleccionados, mientras la segunda parte es el costo asociado a la duración del recorrido, donde los parámetros d_ξ y v_ξ son la distancia y velocidad en el tramo de la ciudad ξ a la siguiente en el recorrido Ξ. El costo ρ es el parámetro que determina la importancia entre la selección de los objetos y la distancia del recorrido.

La interdependencia en el problema del ladrón viajero se encuentra en la velocidad con la que se recorre cada segmento, la cual es dependiente del peso que se lleve. De esta manera cuanto mayor sea el peso de los objetos que se llevan, mayor será la penalización en el tiempo que toma realizar el recorrido. Mientras que al llevar menos objetos, el costo por realizar el recorrido disminuirá pero la ganancia por los objetos será menor. La velocidad con la que se recorre cada segmento es función del peso que se

lleva hasta la ciudad actual del recorrido y está definida como:

$$v_\xi = \left(v_{\text{máx}} - \frac{v_{\text{máx}} - v_{\text{mín}}}{W} w_\xi \right) \tag{3.4}$$

Donde $v_{\text{máx}}$ y $v_{\text{mín}}$ son las velocidades máxima y mínima en la que se realiza el recorrido, el valor W es el peso máximo que se puede llevar y w_ξ es el peso acumulado con el que se recorrerá el segmento entre la ciudad ξ y la siguiente.

3.2.1. Ejemplo

Para ilustrar el TTP se propone el ejemplo definido en la Tabla 3.1, el cual consiste de 6 ciudades, 5 objetos (uno por ciudad sin contar la inicial), un costo de transporte $\rho = 2.5$ y una velocidad en el rango $[0.1, 1.0]$. La matriz de adyacencia mostrada en la Tabla 3.2, indica la distancia entre las ciudades. Para calcular la distancia entre ciudades se emplea el esquema más aceptado para el TTP, esto es aplicar la función cielo a la distancia euclideana. La Figura 3.1 muestra la disposición de las ciudades en el plano, así como la distancia entre ellas.

Tabla 3.1: Problema ejemplo para el ladrón viajero

$v_{\text{máx}}$	:	1.0			
$v_{\text{mín}}$	:	0.1			
ρ	:	2.5			
W	:	500			
N	:	**id**	**x**	**y**	
		1	0.0	0.0	
		2	6.0	-5.0	
		3	14.0	5.0	
		4	11.0	13.0	
		5	7.0	5.0	
		6	2.0	5.0	
M	:	**id**	***p***	***w***	$\boldsymbol{\xi}$
		1	300	400	3
		2	50	200	4
		3	22	10	5
		4	30	40	6
		5	40	2	2

Tabla 3.2: Distancias entre ciudades del problema ejemplo para el TTP

id	**1**	**2**	**3**	**4**	**5**	**6**
1	—	8	15	18	9	6
2	8	—	13	19	11	11
3	15	13	—	9	7	12
4	18	19	9	—	9	13
5	9	11	7	9	—	5
6	6	11	12	13	5	—

Para tener una mejor idea de cómo se realiza el cálculo de la función objetivo del problema del ladrón viajero, se procede a realizar el cálculo para el recorrido $\Xi = (1, 2, 3, 4, 5, 6)$ y selección de objetos $\Pi = \{3, 5\}$.

1. El recorrido se inicia en la ciudad $\xi = 1$ con un peso $w_\xi = 0$ y una velocidad $v_\xi = 1$
2. Se recorre el segmento $\{1, 2\}$ con la máxima velocidad pues no se llevan objetos, con lo que la ganancia de los objetos es 0 y el tiempo transcurrido es 8/1.0.
3. En la ciudad 2 se selecciona el objeto disponible, el cual tiene un peso $\pi_w = 2$ y una ganancia $\pi_p = 40$. De manera que se tiene una ganancia de 40, un peso acumulado $w_\xi = 2$ y una velocidad $v_\xi = 0.9964$.

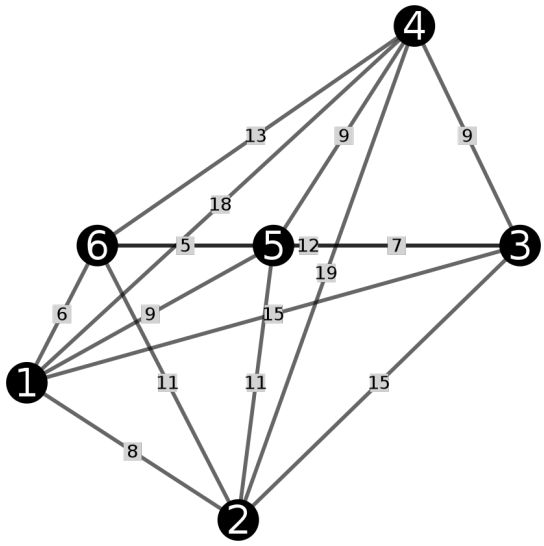


Figura 3.1: Disposición de las ciudades las ciudades en el problema de ejemplo.

4. Se recorren los segmentos $\{2,3\}$,$\{3,4\}$,$\{4,5\}$, de manera que se tiene un nuevo tiempo $8+(13+9+9)/0.9964 = 39.112$.

5. En la ciudad 5 se selecciona el objeto disponible, el cual tiene un peso $\pi_w = 10$ y una ganancia $\pi_p = 22$, por lo que la nueva ganancia es 62, el peso acumulado es $w_\xi = 12$, y la velocidad ahora es $v_\xi = 0.9784$.

6. Para finalizar se recorren los segmentos $\{5,6\}$ y $\{6,1\}$, siendo el tiempo transcurrido $39.112 + (5+6)/0.9784 = 50.3548$

7. Al finalizar el recorrido (regresar a la ciudad 1) la ganancia por los objetos es 62 y el tiempo utilizado es 50.3548. Considerando el costo por unidad de tiempo $\rho = 2.5$ obtenemos el valor objetivo de -63.887

De esta manera se obtiene el de función objetivo (aptitud) de una de las soluciones al problema de ejemplo. Aunque el problema es pequeño se puede observar la explosión del tamaño del espacio de búsqueda, el cual consiste de $O(2^m(n-1)!) = 2^5 * 5! = 3840$.

El TTP puede ser visto como un problema multi-objetivo, en el cual uno de los objetivos es maximizar la ganancia de los objetos y minimizar el tiempo del recorrido. Note que la solución óptima vive en el frente de Pareto (conjunto de soluciones no dominadas) del problema multi-objetivo con vector $\{(-\sum \pi_p), (\sum t_\xi)\}$. La Tabla 3.3 muestra el frente de Pareto, obtenido mediante una búsqueda exhaustiva, se observa que efectivamente la solución óptima se encuentra presente. En él se puede apreciar la principal característica del TTP, la mejor solución no es la solución de menor distancia, ni la solución con mayor ganancia. Sino aquella que tiene el balance (definido por el costo por unidad de tiempo ρ) entre el tiempo y la ganancia.

La Figura 3.2 muestra gráficamente el frente de Pareto, se grafican los valores de ganancia de la selección de objetos (maximizar) frente al tiempo del recorrido (minimizar). La escala de color indica el valor de la aptitud de cada solución y el mejor valor encontrado se indica por el círculo discontinuo. El impacto del valor ρ se verá reflejado en la posición del óptimo dentro del frente, cuando el valor sea pequeño el óptimo tenderá a estar en el extremo con ganancia máxima y cuando sea grande tenderá a estar en el extremo con menor tiempo.

Tabla 3.3: Frente de Pareto para el ejemplo de TTP

Ξ	Π	$F(.)$	$\sum t_\xi$	$\sum \pi_p$	$\sum d_\xi$	w
1,2,6,5,4,3	**1,3,5**	**110.738**	**100.504**	**362**	**57**	**412**
1,2,4,3,5,6	1,3,4,5	109.586	112.965	392	54	452
1,2,6,5,4,3	1,5	99.019	96.3920	340	57	402
1,2,4,3,5,6	1,4,5	97.823	108.870	370	54	442
1,2,6,5,4,3	1	61.071	95.571	300	57	400
1,2,3,4,5,6	2,3,4,5	-16.219	63.287	142	50	252
1,2,3,4,5,6	3,4,5	-35.104	50.841	92	50	52
1,2,3,4,5,6	2,4,5	-36.772	62.708	120	50	242
1,2,3,4,5,6	2,3,5	-43.023	62.009	112	50	212
1,2,3,4,5,6	4,5	-56.551	50.620	70	50	42
1,2,3,4,5,6	3,5	-63.887	50.354	62	50	12
1,6,5,4,3,2	5	-85.072	50.028	40	50	2
1,2,3,4,5,6	$\varnothing$	-125.0	50	0	50	0
1,6,5,4,3,2	$\varnothing$	-125.0	50	0	50	0

En la Figura 3.3 se visualiza la interdependencia, se muestra una solución con distancia mínima con una distancia de 50 y la solución óptima para el TTP con distancia 57. Se aprecia que las soluciones tienden a seleccionar los objetos al final del recorrido, con la intención de recorrer la máxima distancia posible a alta velocidad. Obsérvese que el objeto de la ciudad 3 modifica el recorrido, pues tiene un costo y ganancia altos, mostrando que existen objetos que dependiendo de la posición en el recorrido permiten que un recorrido más largo obtenga un mejor valor objetivo.

3.2.2. Interdependencia

Como se ha mencionado y se mostró con el ejemplo, la complejidad de resolver el problema proviene, no solo de estar conformado por dos problemas NP-Complejos, ni en el gran tamaño de su espacio de búsqueda ($O(2^M(N-1)!)$), sino en la interdependencia entre los subproblemas, esto hace que las técnicas ya establecidas para cada uno de los subproblemas no sean adecuadas para resolver el nuevo problema.

Una de las características de muchos problemas sin interdependencia y de un solo componente como lo son el TSP y el KP, es la facilidad para explorar la vecindad de las búsquedas locales en forma eficiente, esto se debe a que dada una solución s el valor objetivo de los vecinos $N(s)$ es fácilmente calculado en tiempo constante (independiente del tamaño del problema) a partir del valor objetivo de s.

La Ecuación (3.5) muestra el cálculo en tiempo constante de la vecindad 2-Opt para el TSP, se considera una solución inicial $\Xi = \{a, \ldots, i, j, \ldots, k, l, \ldots, n\}$, se puede observar que el cambio en la función objetivo por la inversión del segmento $\{j, \ldots, k\}$ puede calcularse por la diferencia de las aristas nuevas y anteriores.

$$\begin{aligned} f(s) &= d_{a,b} + \cdots + d_{i,j} + \cdots + d_{k,l} + \cdots + d_{n,a} \\ f(s') &= d_{a,b} + \cdots + d_{i,k} + \cdots + d_{j,l} + \cdots + d_{n,a} \end{aligned} \tag{3.5}$$

$$\begin{aligned} f(s') &= f(s) + \Delta \\ &= f(s) + (d_{i,k} + d_{j,l}) - (d_{i,j} + d_{k,l}) \end{aligned}$$

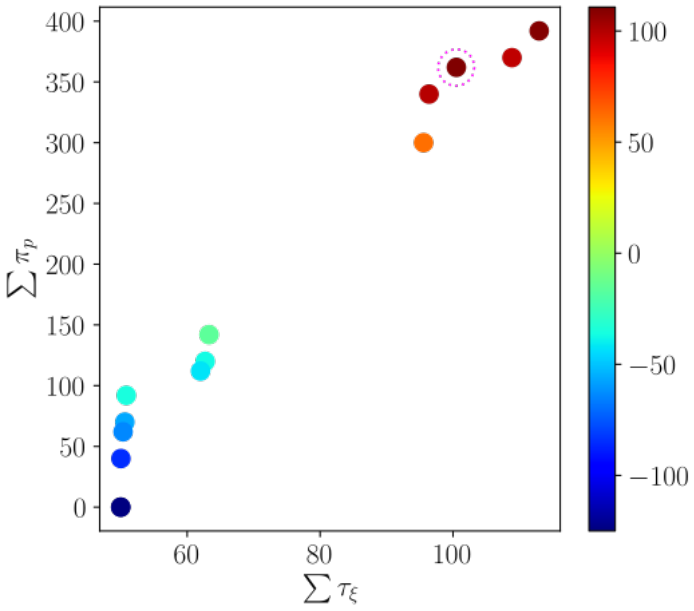


Figura 3.2: Visualización del frente de Pareto, se muestra la ganancia total de los objetos (maximizar) frente al tiempo del recorrido (minimizar). La escala de color indica el valor objetivo de cada solución, el circulo color (rosa) encierra el óptimo (valor objetivo máximo).

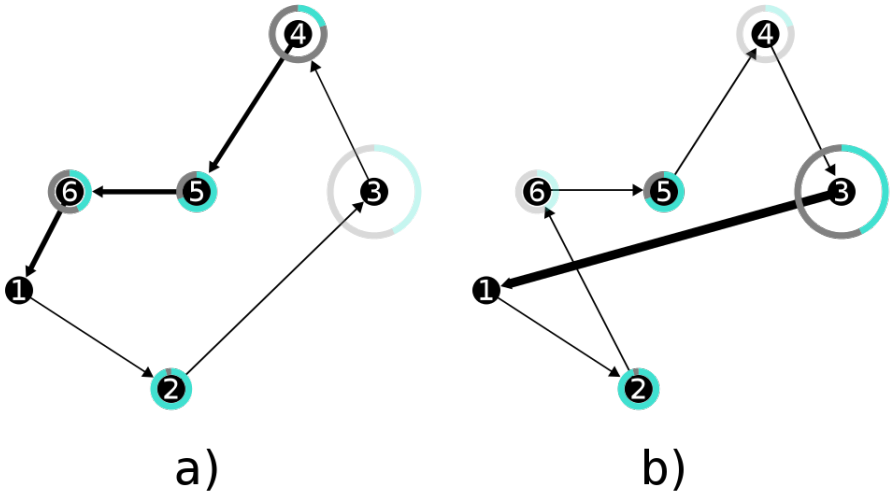


Figura 3.3: Visualización de la mejor solución con distancia mínima (a) frente a la solución óptima (b) al problema de ejemplo. El grosor de la línea es proporcional al peso acumulado, el radio del circulo es proporcional al valor $(\pi_p + \pi_w)$ y su proporción se muestra en color turquesa y gris respectivamente

La Ecuación (3.6) describe el cálculo de la función objetivo para la vecindad Bit-Flip en el KP. Se considera la solución inicial $\Pi = \{o, \ldots, p\}$ y el cambio en la función objetivo debido a la inclusión o eliminación del objeto $k \in M$ de la selección Π.

$$\begin{aligned} f(\Pi) &= o_p + \cdots + p_p \\ f(\Pi') &= o_p + \cdots + p_p \pm k_p \\ \\ f(\Pi') &= f(\Pi) + \Delta \\ &= f(\Pi) \pm k_p \end{aligned} \tag{3.6}$$

En problemas con componentes interdependientes como el TTP, la interdependencia de los problemas conlleva un cambio en la función objetivo que en general, no puede ser calculada de manera constante [68]. En la Ecuación (3.7) se detalla el cálculo de la función objetivo considerando solución $\{\Xi = \{a, \ldots, i, j, \ldots, x, y, \ldots, k, l, \ldots, n\}, \Pi = \{o, \ldots, p\}\}$.

$$\begin{aligned} f(\Xi, \Pi) = (o_p + \cdots + p_p) - \Bigg(& \frac{d_{1,a}}{v_{\text{máx}}} + \cdots + \\ & \frac{d_{i,j}}{v_{\text{máx}} - \nu w_i} + \cdots + \frac{d_{x,y}}{v_{\text{máx}} - \nu w_x} + \cdots + \frac{d_{k,l}}{v_{\text{máx}} - \nu w_k} + \cdots + \\ & \frac{d_{n,1}}{v_{\text{máx}} - \nu w_n} \Bigg) \end{aligned} \tag{3.7}$$

La Ecuación (3.8) expande la modificación realizada por un movimiento 2-Opt. Se puede apreciar que al invertir el segmento $\{k, l\}$, no es posible calcular de forma constante el cambio en el valor objetivo, como lo fue para el TSP, pues se requiere calcular nuevamente la velocidad con la que se recorre cada arista del recorrido.

$$\begin{aligned} f(\Xi', \Pi) = (o_p + \cdots + p_p) - \rho \Bigg(& \frac{d_{1,a}}{v_{\text{máx}}} + \cdots + \frac{d_{i,k}}{v_{\text{máx}} - \nu w_i} \\ & + \cdots + \frac{d_{y,x}}{v_{\text{máx}} - \nu(w_i + w_k - w_x)} + \cdots + \\ & \frac{d_{j,l}}{v_{\text{máx}} - \nu w_k} + \cdots + \frac{d_{n,1}}{v_{\text{máx}} - \nu w_n} \Bigg) \end{aligned} \tag{3.8}$$

En la Ecuación (3.9) se muestra el efecto de agregar o eliminar un elemento $k \in M$ de la selección Π, se puede apreciar que al igual que para el movimiento 2-Opt, el modificar la selección de objetos no es posible predecir de forma simple el cambio en la función objetivo. En este caso se debe recalcular la velocidad para cada segmento a partir de la ciudad en la cual se selecciona.

$$\begin{aligned} f(\Xi, \Pi') = (o_p + \cdots + p_p \pm k_p) - \\ \rho \Bigg(\frac{d_{1,a}}{v_{\text{máx}}} + \cdots + \frac{d_{i,j}}{v_{\text{máx}} - \nu(w_i \pm k_w)} + \cdots + \frac{d_{n,1}}{v_{\text{máx}} - \nu(w_n \pm k_w)} \Bigg) \end{aligned} \tag{3.9}$$

La interdependencia de los componentes del TTP, imposibilita la separación de la optimización, afecta el desempeño de las estrategias diseñadas para cada subproblema y presenta una evaluación de función objetivo con complejidad $O(N)$, haciendo del TTP un problema complejo que requiere de estrategias

con pocas evaluaciones de función objetivo, que tomen en cuenta desde su concepción la interacción entre la selección de objetos y el recorrido.

3.3. Estado del arte

Desde su publicación en el 2013 la investigación en el problema del ladrón viajero ha ido en aumento. Al ser un problema nuevo, se hizo un esfuerzo por encontrar todos los algoritmos publicados hasta la fecha, en la Tabla 3.4 se enlistan los algoritmos localizados ordenados por fecha de publicación.

En el año 2014 se desarrollan las primeras aproximaciones al problema, estos algoritmos son búsquedas locales simples como (*Simple Heuristic* — SH) que no profundizan en la interdependencia. En el mismo año se estandarizan los problemas de referencia mediante la librería TTPLib. Debido al tamaño de este conjunto de problemas existen solo dos artículos [4,69], que han considerado los 9720 problemas. En [69] Wagner et al, compararon 21 algoritmos con un tiempo de ejecución de 10 minutos por problema, los resultados que obtuvieron se encuentran en [70]. Posteriormente en el año 2015, se empiezan a aplicar los algoritmos poblacionales (EA), y se aplican fases de optimización entre recorrido y selección de objeto tanto en algoritmos poblacionales (*Two Stage Memetic Algorithm* — TSMA), como en metaheurísticas de trayectoria (COSOLVER2B). En el 2016 las estrategias fueron especializándose incluyendo búsquedas locales más complejas (CS2SA) y explorando combinaciones de estrategias más simples para generar nuevas heurísticas (S1-S6). Después en el 2017 se generan búsquedas locales especificas para el problema (J2B), las cuales exploran de manera directa la interdependencia entre el KP y TSP. A su vez se comienza a tratar el problema con técnicas multi-objetivo (NSGAII-TTP) y se encuentran estudios de técnicas exactas (*Branch and Bound Strategy* — BNBS). Finalmente en el 2018 se emplean heurísticas para escapar de óptimos locales.

Tabla 3.4: Algoritmos propuestos hasta la fecha para resolver el problema del ladrón viajero

Año	Algoritmos	Publicación
2014	DH, CO-SOLVER	[71]
	MATLS	[8]
	SH, RLS, (1+1) EA	[4]
2015	COSOLVER2B	[72]
	EA	[73]
	G1, G2	[74]
	TSMA	[75]
2016	ACO-TTP, M3, M4, M3B, M4B	[76]
	CC, MA	[77]
	MA2B, CS2SA	[7]
	S1, S2, S3, S4, S5, C1, C2, C3, C4, C5, C6	[78]
2017	CS2SA*, CS2SA-R	[79]
	DP, BNBS	[80]
	GPHS	[81]
	HSEDA	[82]
	J2B, JNB	[68]
	LF, SO, SASEGASA	[83]
	MCGA	[84]
	NSGAII-TTP	[85]
2018	HIBEA	[86]
	RES, RES_GLS	[87]

3.3.1. Clasificación

Al revisar las diferentes estrategias que se han propuesto para resolver el problema del ladrón viajero, se detectó que algunos algoritmos comparten características entre sí. Del estudio de las diferentes estrategias se propone la siguiente clasificación, la cual esta conformada por los siguientes cinco grupos:

- **Centrados en TSP**

  Generan un recorrido Ξ de alta calidad empleando alguna de las estrategias conocidas para el TSP (usualmente de la familia Lin Kernighan) y se enfocan en la optimización de la selección de objetos Π, realizando pocas o ninguna modificación al recorrido.

- **Sistema CoSolver**

  Basados en el algoritmo COSOLVER [71], alternan entre la optimización del recorrido Ξ y la optimización de la selección Π. La principal característica de los métodos en este grupo es que al optimizar Ξ, se mantiene fijo Π y viceversa. Diferentes algoritmos emplean diversas estrategias, sin embargo, todos los algoritmos conocidos emplean una búsqueda local simple para optimizar el recorrido Ξ.

- **Codificación completa**

  Tratan los dos espacios de búsqueda como si fueran uno, es decir, dada una solución candidata realizan transformaciones que modifican de forma simultánea el recorrido y la selección de objetos. Muchos de los esquemas que pertenecen a este grupo son algoritmos poblacionales, que manipulan ambas partes del problema de manera simultánea. También la mayoría de los algoritmos incorporan algunos de los principios del sistema CoSolver para la intensificación.

- **Exactos**

  Aplicación de estrategias como programación dinámica o ramificación y poda, garantizando encontrar el óptimo. Debido al tamaño del espacio de búsqueda estos algoritmos se han concentrado en instancias pequeñas, siendo los de mayor tamaño del orden de veinte ciudades con mil objetos. Su estudio se enfoca más a proporcionar resultados óptimos con los cuales poder estudiar el desempeño de las aproximaciones de otros algoritmos.

- **Mixtos**

  Estrategias que intentan automatizar el proceso de selección o combinación de heurísticas simples para resolver el problema.

En la Tabla 3.4 se muestra la pertenencia de los algoritmos a cada clase.

Tabla 3.5: Clasificación de los algoritmos del estado del arte

Clasificación	Algoritmos
Centrados en TSP	DH, SH, RLS, (1+1) EA, G1, G2, S1, S2, S3, S4, S5, C1, C2, C3, C4, C5, C6
Codificación completa	MATLS, EA, TSMA, ACO-TTP, M3, M4, M3B, M4B, MA, MA2B, J2B, JNB, MCGA, NSGAII-TTP
Sistema CoSolver	CO-SOLVER, COSOLVER2B, CC, CS2SA, CS2SA*, CS2SA-R, RES, RES_GLS
Exactos	DP,BNSBS
Mixtos	GPHS, HSEDA, LF, SO, SASEGASA, HIBEA

3.3.2. Problemas de referencia

En [4] se propuso un conjunto de problemas de referencia TTPLib, basado en la librería TSPLib [88]. El conjunto consta de 9720 problemas y tiene las siguientes características:

- Utiliza los problemas definidos en la librería TSPLib, contando con un rango $[50, 85900]$ ciudades.
- Para cada problema de TSP se cuenta con las siguientes variantes para el KP.
  - No correlacionado

    El peso π_w y ganancia π_p de cada objeto se generó con una distribución uniforme en el rango $[1, 10^3]$.
  - No correlacionado con pesos similares

    El peso π_w se genera en el intervalo $[10^3, 10^3 + 10]$ y la ganancia en $[1, 10^3]$.
  - Fuertemente correlacionado

    El peso $\pi_w \in [1, 10^3]$ mientras la ganancia se calcula como $\pi_p = \pi_w + 100$.

  Estas tres variantes son una selección de entre las quince propuestas presentadas en [67], trabajo que realiza un estudio profundo del KP. En ese estudio se encuentra que la dificultad para resolver los problemas del algoritmo COMBO es en orden decreciente, correlacionada, pesos similares y no correlacionados.
- Por cada combinación de TSP y KP se genera una cantidad de objetos por ciudad (factor de objetos) $\in \{1, 3, 5, 10\}$. Todas las ciudades cuentan con el mismo número de objetos a excepción de la primer ciudad, la cual no contiene objetos.
- Para cada combinación de TSP, KP y factor de objetos, se tienen diez categorías de capacidad $\in 1, \ldots, 10$, siendo la capacidad máxima función de la categoría como $(\frac{C}{11}) \sum \pi_w$.
- Cada problema generado cuenta con su propio costo ρ, que enlaza los dos subproblemas de tal forma que ninguno domine sobre el otro. El costo se calcula como $\rho = K(\Pi^*)/T(\Xi^{LK}, \Pi^*)$, donde Π^* es la selección de objetos (calculado mediante el algoritmo COMBO) con ganancia óptima para el KP y $T(\Xi^{LK}, \Pi^*)$ es el tiempo del recorrido Ξ^{LK} generado mediante la heurística LK.
- Finalmente todos los problemas cuentan con un rango de velocidad $[0.1, 1]$.

La librería TTPLib se encuentra disponible en [89] o [90], los 9720 problemas que la conforman tienen el formato que se muestra en la Tabla 3.6. Las primeras nueve líneas contiene en orden: el nombre de la instancia, el tipo de KP, la cantidad de ciudades $|N|$, la cantidad de objetos $|M|$, el peso máximo del problema W, la velocidad mínima $v_{\text{mín}}$, la velocidad máxima $v_{\text{máx}}$, el costo por unidad de tiempo ρ y el tipo de distancia a utilizar (remanente de los problemas TSP). En la línea 10 se encuentra el separador de la sección del conjunto de ciudades N, de la 11 a la $(10+N)$ se encuentran los datos de cada ciudad ξ separados por un espacio en el orden: etiqueta, primer coordenada x, segunda coordenada y. Mientras el separador de la sección del conjunto objetos M se encuentra en la línea $(11 + N)$, siendo que de la línea $(12 + N)$ a la línea $(12 + N + M)$ se describen las propiedades de cada objeto π separadas por un espacio en el orden: etiqueta, ganancia p, peso w y etiqueta de la ciudad ξ en la que se encuentra el objeto.

Para los problemas presentes en la librería la distancia estándar siempre se calcula mediante la Ecuación (3.10) que es la distancia euclideana redondeada al siguiente entero.

$$D(a, b) = \left\lceil \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2} \right\rceil \tag{3.10}$$

Tabla 3.6: Formato de los problemas en la librería TTPLib

1	PROBLEM NAME: berlin52-ttp
2	KNAPSACK DATA TYPE: uncorr
3	DIMENSION: 52
4	NUMBER OF ITEMS: 51
5	CAPACITY OF KNAPSACK: 22450
6	MIN SPEED: 0.1
7	MAX SPEED: 1
8	RENTING RATIO: 1.25
9	EDGE_WEIGHT_TYPE: CEIL_2D
10	NODE_COORD_SECTION (INDEX, X, Y):
11	1 565.0 575.0
	...
62	52 1740.0 245.0
63	ITEMS SECTION (INDEX, PROFIT, WEIGHT,NODE):
64	1 119 1 2
	...
114	51 989 204 52

3.3.3. Algoritmos

En esta sección se describen los algoritmos del estado del arte que se utilizarán para comparar el desempeño de nuestra propuesta. Se consideran las siguientes categorías: centrados en TSP, codificación completa y sistema CoSolver. Se descartan las otras dos categoría pues escapan del alcance de este trabajo, los algoritmo exactos no son aplicables a los tamaño de instancia que se trabajan y los algoritmos mixtos entran en conflicto con el análisis del beneficio del enfoque propuesto.

De las categorías seleccionadas se describe el algoritmo que mejor desempeño presenta en el estudio realizado en [69].

S5

La heurística S5 pertenece a la familia de estrategias TSP centradas, pues primero genera un buen recorrido Ξ y después se enfoca en la generación de una buena selección de objetos Π. El Algoritmo {17} esquematiza el procedimiento, en esencia es un esquema basado en reinicio, en donde la generación del recorrido se realiza mediante la estrategia de encadenamiento de Lin Kernighan [91] (*Chained Lin Kernighan* — CLK). El algoritmo S5 se encuentra disponible para su descarga en [92]

Algoritmo 17: Heurística S5

def *s5*
  $\tau = \{\Xi$ aleatorio, $\varnothing\}$
  hasta que *se cumpla el criterio de paro*
    $\Xi =$ `clk(`N`)`
    $\Pi =$ `packiterative(`Ξ`,`α`)`
    $\tau = \{\Xi, \Pi\}$ **si** $\{\Xi, \Pi\}$ *es mejor que* τ
  fin
  regresa τ
fin

El algoritmo CLK, es un algoritmo basado en reinicio que ha demostrado ser altamente competitivo

para resolver el TSP. La heurística consiste en que mientras no se cumpla el criterio de paro (usualmente por tiempo), se aplica la heurística LK hasta encontrar un óptimo local, después se perturba el óptimo local (usualmente el movimiento doble puente) y así sucesivamente.

Una vez que se tiene un buen recorrido (con bajo costo en distancia), se emplea la rutina PACKITERATIVE para generar la selección de objetos Π. Esta heurística codiciosa ejecuta iterativamente la rutina PACK, la cual se muestra en el Algoritmo {18}.

Algoritmo 18: Heurística codiciosa PACK para generar una selección de objetos

```
def pack ( Ξ : recorrido, α : exponente )
    ordenar de forma decreciente empleando la ecuación (3.11)
    Π* = Π = ∅
    k* = k = 1
    μ = frecuencia de evaluación de F(.)
    mientras existan objetos a agregar y μ > 1
        si el objeto (π)_k cabe en la mochila
            Π = Π ∪ {(π)}
            si (k mód μ) = 0
                si Π es mejor que Π*
                    Π*,k* = Π,k
                en otro caso
                    Π,k,μ = Π*,k*,⌊μ/2⌋
                fin
            fin
        fin
        k += 1
    fin
    regresa la mejor selección Π*
fin
```

Al quererse maximizar la ganancia, a la vez que se minimiza el tiempo dado por la interacción del peso y la distancia a recorrer, la heurística PACK asigna un puntaje a los objetos en M que incorpore un balance entre su ganancia π_p, su peso π_w y la distancia π_d que el objeto π debe cargarse en el recorrido Ξ.

$$\pi_s = \frac{\pi_p^{\alpha}}{\pi_w^{\alpha}\ \pi_d} \tag{3.11}$$

Empleando este puntaje como criterio se intenta generar un plan Π agregando los objetos. Para disminuir el costo de evaluación, la función objetivo se evalúa con una frecuencia de μ objetos, si la selección actual mejora la mejor solución conocida Π^*, esta última se actualiza, en caso contrario se reduce la frecuencia y se reinicia a la mejor solución conocida.

El Algoritmo {19} bosqueja la rutina PACKITERATIVE al aplicar iterativamente la subrutina PACK para diferentes valores de α, mejora la calidad de la solución y disminuye la dependencia al valor seleccionado. Para esto realiza una búsqueda en el rango $[\alpha - \delta, \alpha + \delta]$, donde δ define el intervalo del rango. Se emplean tres soluciones para controlar la búsqueda: Π_l para acotar el valor inferior, Π_m para el valor actual de la búsqueda y Π_r para acotar el valor superior. La búsqueda se enfoca en valores pequeños de α si $(\Pi_l \preceq \Pi_m) \wedge (\Pi_l \preceq \Pi_r)$ o en valores grandes si $(\Pi_r \preceq \Pi_m) \wedge (\Pi_r \preceq \Pi_l)$. El valor de α se actualiza en cada iteración y corresponde al valor utilizado para generar la mejor solución, a su vez el valor δ se actualiza como $\delta/2$ disminuyendo el rango a explorar. Finalmente el algoritmo regresa la mejor solución encontrada.

Algoritmo 19: Heurística codiciosa iterativa PACKITERATIVE

def *packiterative (Ξ : recorrido, α : exponente, δ : intervalo,$i_{\text{máx}}$: límite de iteraciones)*
  Π_l, Π_m, Π_r = `pack(`Ξ`,`$\alpha - \delta$`)`, `pack(`Ξ`,`α`)`,`pack(`Ξ`,`$\alpha + \delta$`)`
  para cada $k \in (\ 1\ \ldots\ i_{\text{máx}}\ )$ **hacer**
    si Π_l *es mejor que* Π_m *y* Π_r *es mejor que* Π_m
      α, Π_m = (Π_l es mejor que Π_r) **?** $\{\alpha - \delta,\ \Pi_l\}$ **:** $\{\alpha + \delta,\ \Pi_r\}$
    si no, si Π_l *es mejor que* Π_m
      α, $\Pi_m = \alpha - \delta$, Π_l
    si no, si Π_r *es mejor que* Π_m
      α, $\Pi_m = \alpha + \delta$, Π_r
    fin
    δ /= 2
    $\Pi^* = \Pi_m$
    Π_l, Π_r = `pack(`Ξ`,`$\alpha - \delta$`)`, `pack(`Ξ`,`$\alpha + \delta$`)`
    romper si *la mejora es menor que* ϵ
  fin
  regresa Π^*
fin

CS2SA

Este algoritmo pertenece a la categoría sistema CoSolver, pues alterna entre dos fases de optimización, la primera es la optimización del recorrido Ξ (búsqueda local con vecindad 2-Opt) y la segunda emplea el recocido simulado (*Simulated Annealing* — SA) para mejorar la selección de objetos Π.

El Algoritmo {20} describe la estrategia CS2SA, la inicialización consiste en generar una solución inicial empleando la heurística CLK, una vez que se encuentra un óptimo local se fija el recorrido y se genera una solución para la selección de objetos empleando una heurística codiciosa. Después, hasta que se cumpla el criterio de paro se optimiza el recorrido (sin modificar los objetos) empleando una búsqueda local con vecindad 2-Opt, en la que se emplea la triangulación Delaunay [93] para reducir el espacio de búsqueda. Una vez se encuentra un óptimo local, aplica una variante adaptada al TTP de la heurística SA, para optimizar la selección de objetos (sin modificar el recorrido), donde la temperatura absoluta (final) se fija a 1, la temperatura inicial en 100, el factor de enfriamiento en 0.95 y se utiliza un ciclo extra para explorar más soluciones en cada temperatura. La vecindad en el recocido simulado se genera de forma aleatoria empleando el operador Bit-Flip.

MA2B

El Algoritmo {21} bosqueja las ideas principales de la heurística MA2B, la cual pertenece a la familia de representación completa y es un algoritmo memético que utiliza búsquedas locales con vecindades 2-Opt y Bit-Flip para la fase de intensificación.

Para cada individuo de la población inicial se genera un recorrido aleatorio Ξ al cual se le aplica una estrategia LK por un número de iteraciones dependiente del tamaño del problema. Después se genera una selección de objetos Π empleando una heurística codiciosa y finalmente se realiza la intensificación empleando las búsquedas locales.

Como técnica de intensificación se emplea la rutina búsqueda local restringida, la cual aplica una búsqueda local con vecindad 2-Opt al recorrido Ξ considerando fija la selección de objetos Π, después aplica una búsqueda local con vecindad Bit-Flip a la selección de objetos considerando fijo el recorrido. Debido a que la evaluación de las vecindades tiene un alto coste computacional, se emplea un máximo

Algoritmo 20: Algoritmo CS2SA basado en el sistema CoSolver

```
def cs2sa
    Ξ = clk(N)
    Π* = packiterative(Ξ)
    τ = { Ξ, Π* }
    mientras exista mejora
        Ξ = 2-opt-ls(Ξ)
        T = 100
        mientras T > 1
            para i ∈ ( 1 ... #iteraciones ) hacer
                Π = generar vecino Bit-Flip aleatorio de Π*
                δ = F(Ξ, Π) − F(Ξ, Π*)
                Π* = Π si (δ > 0) ∨ (e^{δ/T} > U(0,1))
                τ = {Ξ, Π*} si {Ξ, Π*} es mejor que τ
            fin
            T *= 0.95
        fin
    fin
    regresa τ
fin
```

de iteraciones en las búsquedas locales para disminuir el tiempo de ejecución.

En la etapa de cruza, la selección de los dos padres a utilizar se realiza mediante un torneo de seis individuos seleccionados de forma aleatoria de la población. Se emplea el operador de preservación máxima o MPX [94] (*Maximal Preservative Crossover*) en el cual el hijo se produce por dos segmentos. La Figura 3.4 ejemplifica la generación del hijo, el primer segmento se elije de manera aleatoria del primer padre, su tamaño está entre $[l, |\Xi|/2]$ ciudades (l es el tamaño mínimo del segmento) y el segundo segmento se genera con la concatenación de las ciudades en el segundo padre una vez que se eliminan las ciudades ya presentes en el hijo. En este algoritmo el operador MPX se comporta de mejor forma que el ERX, los autores lo atribuyen al efecto de la diversificación por ser un operador más disruptivo. Para la selección de objetos Π se hereda el estado de los objetos del padre del que proviene la ciudad.

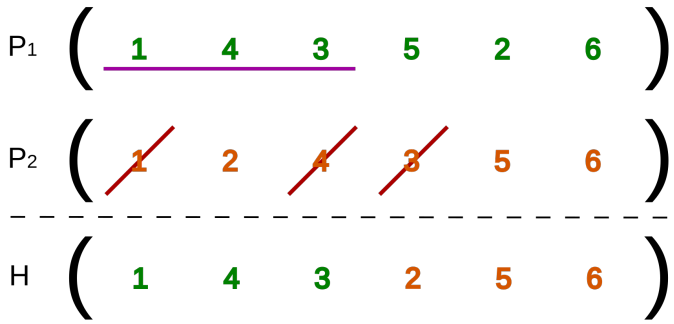


Figura 3.4: Operador de cruza MPX concatena dos segmentos, uno es generado a partir del primer padre P_1 (verde) eligiendo de forma aleatoria el tamaño. El segundo proviene del segundo padre P_2 (naranja), concatenando los valores que no están presentes en el primer segmento.

Cuando la cruza genera una hijo ya presente en la descendencia se realiza una mutación, la cual consiste

Algoritmo 21: Método memético MA2B

```
def ma2b (N: tamaño de la población = 40 )
    P = ∅
    para cada k ∈ ( 1 ... N ) hacer
        Ξ = aplicar LK a un recorrido aleatorio
        Π = pack(Ξ)
        Ξ = aplicar búsqueda local 2-Opt a Ξ
        Π = aplicar búsqueda local Bit-Flip a Π
        P = P ∪ {Ξ, Π}
    fin
    hasta que se cumpla el criterio de paro
        C = ∅
        para cada k ∈ ( 1 ... 0.75|P| ) hacer
            a, b = seleccionar padres por torneo de 6 individuos
            Ξ = mpx(a_Ξ, b_Ξ)
            Π = en cada ciudad el estado de los objetos es heredado del padre del que proviene la
              ciudad
            si τ = {Ξ, Π} ∈ C
                Ξ = double-puente(double-puente(Ξ))
                Π = pack(Ξ)
                Ξ = aplicar búsqueda local 2-Opt a Ξ
                Π = aplicar búsqueda local Bit-Flip a Π
                τ = {Ξ, Π}
            fin
            C = C ∪ τ
        fin
        P = Reemplazar los 0.75|P| peores elementos de P con C
    fin
    regresa el mejor individuo en P
fin
```

en perturbar la solución aplicando un movimiento doble puente para generar un nuevo recorrido y generar la selección de objetos para este nuevo recorrido. De manera similar a la inicialización se intensifica esta nueva solución empleando las búsquedas locales por un máximo de iteraciones.

Finalmente el criterio de paro es establecido en función del tiempo, el número de iteraciones y el número de generaciones sin mejora.

Capítulo 4

Propuestas para la solución del TTP

En este capítulo se describe el diseño e implementación de la propuesta de este trabajo para resolver el problema del ladrón viajero. Primero se plantea el enfoque del trabajo para la solución del TTP, después se dan detalles generales que comparten las implementaciones de los algoritmos. Las últimas dos secciones describen con detalle los algoritmos, primero se detallan los algoritmos de trayectoria propuestos, una búsqueda local simple (*Local Search* — LS), seguido de la búsqueda local iterada (ILS) y para finalizar con la búsqueda local guiada con reinicio (RESGLS). Finalmente en la última sección se explica la implementación del algoritmo memético (DMA).

4.1. Motivación

En el capítulo anterior se describió la clasificación de las principales estrategias para resolver el problema del ladrón viajero, entre ellas se encuentra la familia de heurísticas basadas en el sistema CoSolver. Este grupo de algoritmos presenta un buen balance entre complejidad computacional y calidad de solución, pues metaheurísticas específicas de cada uno de los subproblemas pueden ser utilizadas, requiriendo solamente pequeñas modificaciones.

Este trabajo hipotetiza que los dos factores limitantes en el desempeño de las estrategias CoSolver son:

- Carecen de una estrategia para escapar de óptimos locales de la interacción entre fases de optimización.
- Ofrecen una fuerte dependencia al recorrido inicial

Al no plantearse mecanismos explícitos para escapar de óptimos locales en el sistema CoSolver, el encadenamiento de las búsquedas locales que lo componen pueden ser un factor que propicie el estancamiento de exploración del espacio de búsqueda.

Para mejorar el desempeño del sistema CoSolver, se propone incluir un mecanismo para tratar de mejor forma los óptimos locales y mejorar la exploración del espacio de búsqueda. Durante el diseño de la propuesta, se plantearon dos algoritmos, una búsqueda local simple (LS) para estudiar el comportamiento del sistema CoSolver y una búsqueda local iterada (ILS) para comprobar el beneficio otorgado por el esquema basado en iterar con diferentes soluciones.

Las propuestas finales de este trabajo son:

- **RESGLS**

Una metaheurística de trayectoria basada en la búsqueda local guiada (GLS) que incluye un mecanismo de reinicio.

- **DMA**

  Una metaheurística poblacional híbrida (memético) que emplea la estrategia de trayectoria RESGLS en su fase de explotación y un mecanismo de control de diversidad basado en el esquema RMDDC para la exploración.

4.2. Representación de los componentes

La forma de representar los diferentes componentes del TTP, afecta los operadores, su desempeño y el espacio de búsqueda. Por ello definir la representación es el primer paso en el diseño de cualquier heurística, en específico para el problema del ladrón viajero debe decidirse la representación de la solución (recorrido y selección de objetos) y de las propiedades que definen el problema.

4.2.1. Representación del recorrido

Aunque para el TSP existen diferentes representaciones del recorrido, en el TTP la representación utilizada es la basada en permutación. En esta representación el recorrido se describe una permutación de las $N-1$ ciudades, que representa el orden en que se visitan las ciudades. Empleando esta representación se tiene acceso a una gran cantidad de operadores ya establecidos para el TSP que pueden ser utilizados de manera directa en el recorrido.

De esta forma el recorrido se representa como un vector que muestra la secuencia en que se visitan las ciudades:

$$\Xi = [0, 1, 5, 4, 3, 2] \tag{4.1}$$

Cada elemento es un entero que representa la etiqueta única que identifica a cada ciudad. El recorrido que representa Ξ consiste en iniciar en la ciudad 0, recorrer en orden las ciudades 1, 5, 4 y 3 hasta llegar a la ciudad 2 para finalmente regresar a la ciudad 0.

4.2.2. Representación de la selección de objetos

De manera similar para el problema de la mochila existen diferentes representaciones en la literatura, la más utilizada para el KP y la única empleada para el TTP es la representación binaria. Esta representación consiste de un vector que contiene un valor booleano por cada elemento, el cual es verdadero (**V**) si se incluye el objeto en la selección o falso (**F**) si se ignora. Al igual que la representación de trayectoria para el recorrido la representación binaria permite utilizar operadores clásicos para el problema. Su principal desventaja es la posibilidad de crear selecciones invalidas (se excede la capacidad máxima de peso), pues la representación no almacena la capacidad utilizada.

De esta forma la selección de objetos se representa como un vector que indica si se lleva o no el objeto:

$$\Pi = [V, F, V, F, V] \tag{4.2}$$

Cada posición en el vector representa la etiqueta única de cada elemento y el valor asociado representa si se lleva (**V**) o no (**F**) el objeto. De manera que la selección Π representa el incluir los objetos 0, 2 y 4

4.2.3. Representación de una solución del TTP

Finalmente la representación adoptada para el TTP, es una dupla que contiene el recorrido como permutación y la selección de objetos usando el vector binario. Hasta la fecha no se ha probado otro tipo de representación para el problema.

Una solución para el TTP se representa como en el Algoritmo {22}, como una estructura que contiene dos vectores, el primero consta de $|N|$ elementos que representan el recorrido y el segundo consiste de $|M|$ elementos que representan el estado de los objetos (si se encuentra seleccionado o no).

Algoritmo 22: Estructura *ladrón* que representa una solución del TTP

clase *ladrón*

Ξ	:	[ ] =	Arreglo con las etiquetas de las ciudades en el orden de visita
Π	:	[ ] =	Arreglo con el estado de selección de los objetos

fin

4.2.4. Representación de una instancia del TTP

Para la representación del problema se utiliza una estructura que contiene los diferentes componentes del ladrón viajero.

Representación de una ciudad

El Algoritmo {23} representa cada ciudad mediante la estructura *ciudad* que contiene una etiqueta única y su posición como un par de coordenadas, $\{x,y\}$.

Algoritmo 23: Estructura *Ciudad*

clase *ciudad*

id	:	**entero**	= etiqueta única que identifica a la ciudad
x	:	**real**	= coordenada x
y	:	**real**	= coordenada y

fin

Representación un objeto

Cada objeto se representa en el Algoritmo {24} mediante la estructura *objeto*, la cual se conforma de una etiqueta única, su peso, ganancia y la ciudad en la que se encuentra.

Algoritmo 24: Estructura *objeto*

clase *objeto*

id	:	**entero**	= etiqueta única que identifica al objeto
p	:	**entero**	= ganancia del objeto
w	:	**entero**	= peso del objeto
ξ	:	**entero**	= etiqueta de la ciudad en la que se encuentra el objeto

fin

Instancia

Como las estructuras *ladrón* y *objeto* no contienen información especifica a cada ciudad u objeto, se requiere de una estructura que contenga la información detallada de los parámetros del problema a la que se pueda hacer referencia cuando sea requerido.

El Algoritmo {25} muestra la estructura *ttp* que representa el problema del ladrón viajero, en donde se definen los parámetros de velocidad máxima $v_{\text{máx}}$, velocidad mínima $v_{\text{mín}}$, costo por unidad de tiempo ρ, peso máximo W y decremento de velocidad por unidad de peso ν. Así como el vector de ciudades N, el vector de objetos M.

Algoritmo 25: Estructura *ttp*

clase *ttp*

W	:	**entero**	= peso máximo
ρ	:	**real**	= costo por unidad de tiempo
$v_{\text{máx}}$	:	**real**	= velocidad máxima con la que se recorre Ξ
$v_{\text{mín}}$	:	**real**	= velocidad mínima con la que se recorre Ξ
ν	:	**real**	= $(v_{\text{máx}} - v_{\text{mín}})/W$
N	:	[ ]	= arreglo con los objetos *ciudad* del problema
M	:	[ ]	= arreglo con los objetos *objeto* del problema
D	:	[[ ]] | **fun**	= matriz o función de la distancia entre ciudades

fin

Opcionalmente se puede incluir una matriz de distancia D, en donde cada elemento $D_{f,c}$ determina la distancia entre la ciudad con etiqueta f y la ciudad con etiqueta c. Al igual que en el TSP la matriz D es una matriz simétrica, lo que permite ahorrar memoria si se representa solo la mitad superior. Dependiendo del tamaño del problema inclusive puede requerirse del cálculo sobre la marcha con algún tipo de memorización.

4.3. Evaluación de la función objetivo del TTP

4.3.1. Cálculo de la aptitud

Como se analizó en el capítulo anterior, la evaluación de la función objetivo es costosa, aunque existen diferentes estrategias para recuperar parcialmente el valor objetivo se omiten en este trabajo pues aumentan la complejidad de implementación y no existe conflicto en los objetivos el utilizar la implementación directa del problema.

Para determinar si una solución es mejor o peor que otra, se emplea como función objetivo la ganancia total definida en la Ecuación (4.3).

$$F(\tau) = \sum_{\pi \in \Pi} \pi_p - \rho \sum_{\xi \in \Xi} \left(\frac{d_\xi}{v_{\text{máx}} - \nu w_\xi} \right) \tag{4.3}$$

El Algoritmo {26} describe el procedimiento para calcular la aptitud de una solución $\tau = \{\Xi, \Pi\}$ factible. El primer paso consiste en obtener la ganancia de los objetos seleccionados, para esto se itera sobre los $|M|$ objetos. Si se ha seleccionado el objeto con etiqueta k, es decir $\Pi_{[k]} = \mathbf{V}$, se acumula en la variable p el valor π_p y en la posición π_ξ (ciudad asignada al objeto) del vector w se acumula el peso del objeto π_w. Después de obtener la ganancia p y el vector de peso en cada ciudad w se procede a calcular el tiempo del recorrido Ξ. Para esto se itera sobre cada elemento del vector Ξ, obteniendo la ciudad actual ξ y la siguiente ξ', en cada paso del recorrido se actualiza el peso acumulado w_ξ

debido a la selección de los objetos disponibles en ξ y se adiciona a la variable t el tiempo que toma recorrer la distancia $D_{[\xi][\xi']}$ utilizando la velocidad $v_{\text{máx}} - \nu w_\xi$. Finalmente la ganancia total se obtiene al substraer de p el costo debido al transporte de los objetos ρt.

Algoritmo 26: Evaluación de la función objetivo del TTP

```
clase ttp
    def fun-obj (τ : solución a evaluar)
        w = vector inicializado en ceros de tamaño |N|
        p = 0
        para cada k ∈ ( 0 ... |Π| − 1 ) hacer
            si Π_[k]
                π = M_[k]
                p += π_p
                w_[π_ξ] += π_w
            fin
        fin
        w_ξ = t = 0
        para cada k ∈ ( 0 ... |Ξ| − 1 ) hacer
            ξ, ξ' = Ξ_[k], Ξ_[(k+1) mód |Ξ|]
            w_ξ += w_[ξ]
            t += D_[ξ][ξ'] / (v_máx − ν w_ξ)
        fin
        regresa p − ρt
    fin
fin
```

4.3.2. Validación del cumplimiento de las restricciones

En este trabajo al aplicar los distintos operadores al problema del ladrón viajero, se tuvieron en consideración las restricciones del problema, de manera que los operadores utilizados en los algoritmos propuestos, generan siempre soluciones factibles. Aunque en general las estrategias diseñadas para el TTP generan soluciones factibles, existen operadores que pueden llevar a una solución a no cumplir con las restricciones. Para detectar el incumplimiento de las restricciones, el Algoritmo {27} determina si una solución es factible o no.

Cuando se detecta que una solución no es factible, se debe emplear algún algoritmo de reparación. La selección del mejor algoritmo de reparación dependerá de las propiedades y diseño del operador que generó la solución inválida, quedando a discreción del autor de la estrategia. Debido a que las restricciones del TTP son similares a las de los subproblemas, los algoritmos de reparación específicos a los estos últimos pueden ser aplicados. Un ejemplo de algoritmo de reparación para el recorrido es detectar las ciudades duplicadas o faltantes y reemplazarlas por las ciudades faltantes en el orden que aparecen en una plantilla (por ejemplo el padre con mejor aptitud). Como ejemplo de algoritmo de reparación en la selección de objetos, se pueden quitar los objetos con mayor peso que tengan menos aporte a la ganancia.

Algoritmo 27: Verificación del cumplimiento de las restricciones del TTP para la solución

def *verificación-de-restricciones (τ: solución, T: instancia)*
  regresa *error en cantidad objetos* **si** $|\tau_\Pi| \neq |T_M|$
  $w = 0$
  para cada $i \in (\ 0\ \ldots\ |\tau_\Pi|\ )$ **hacer**
    $\pi = T_{M_{[i]}}$
    regresa *etiqueta error en tipo de objeto* **si** $\tau_{\Pi_{[i]}} \notin \{V, F\}$
    $w\ +\!= \pi_w$ **si** $\tau_{\Pi_{[i]}}$
    regresa *error de capacidad máxima excedida* **si** $w > T_W$
  fin
  regresa *error en cantidad de ciudades* **si** $|\tau_\Xi| \neq |T_N|$
  regresa *error de ciudades repetidas* **si** $\nexists! c \in \tau_\Xi (c = \xi) \forall \xi \in T_N$
  regresa *error de ciudades faltantes* **si** $\exists \xi \in T_N \mid \nexists \in \tau_\Xi$
  regresa *error de ciudad origen* **si** $\tau_{\Xi_{[0]}} \neq 0$
  regresa *acierto en factibilidad*
fin

4.4. Construcción de la solución inicial

Debido al tamaño del espacio de búsqueda y forma del paisaje, la solución con la que se inicia tiene un fuerte impacto en la calidad de las soluciones. Para generar la solución inicial existen diferentes estrategias, siendo que todas parten de iniciar el recorrido Ξ para después generar las selección de objetos Π.

4.4.1. Generación del recorrido

Greedy randomized adaptive search (GRASP)

El Algoritmo {28} bosqueja la heurística codiciosa de búsqueda adaptativa aleatoria (*Greedy Randomized Adaptive Search Procesdure* — GRASP) que consiste en ir agregando las ciudades al vector $\Xi = [0]$, seleccionando cada ciudad de forma aleatoria de entre las k ciudades más cercanas a la última ciudad agregada a Ξ. Cuando $k = 1$ esta heurística se conoce como construcción por vecino más cercano y cuando $k = |N|$ se genera el recorrido de forma aleatoria.

Algoritmo 28: Heurística GRASP para generar el recorrido Ξ

def *grasp (k : vecinos a considerar)*
  $\Xi = [0]$
  $N' = \{1 \ldots |N| - 1\}$
  hasta que $N' = \varnothing$
    $N' =$ ordenar N' en función de la distancia a la última ciudad en Ξ
    $i =$ número entero aleatorio entre $[0, k]$
    $\Xi = \Xi \cup \{N'_{[i]}\}$
    $N' = N' \setminus N'_{[i]}$
  fin
  regresa Ξ
fin

K-Opt

Se emplea la búsqueda local con vecindad k-Opt, siendo la más utilizada la vecindad 2-Opt {12} en la que se modifican dos aristas del recorrido. El Algoritmo {29} bosqueja su implementación, donde para cada elemento de la vecindad de Ξ, considerado en un orden aleatorio, se compara frente a la mejor solución conocida Ξ^*, en caso que ser mejor se mueve la búsqueda a la nueva solución y se genera una nueva vecindad. La búsqueda termina si se ha enumerado toda la vecindad sin encontrar mejora, lo que indica que se ha encontrado el óptimo local.

Algoritmo 29: Búsqueda local con vecindad 2-Opt para el TTP

```
def optls ( Ξ : recorrido inicial)
    Ξ* = Ξ
    hasta que se cumpla el criterio de paro o no exista mejora
        V = ordenar de forma aleatoria la vecindad 2-opt(Ξ*)
        para cada Ξ ∈ V hacer
            si Ξ es mejor que Ξ*
                Ξ* = Ξ
                romper
            fin
        fin
    fin
    regresa Ξ
fin
```

Esta búsqueda local es extensamente utilizada en las fases de intensificación de los algoritmos que modifican el recorrido.

Lin-Kernighan

Familia de heurísticas que emplean encadenamiento de vecindades k-Opt, siendo la más utilizada la estrategia CLK y LKH.

La mayor parte de los algoritmos de la literatura emplean las estrategias Lin-Kernighan, pues se ha observado que cuanto menor sea la distancia del recorrido inicial, la probabilidad de encontrar soluciones de alta calidad aumenta. Esto se ha atribuído a la reducción del tiempo invertido en encontrar una solución aceptable para el recorrido en las etapas iniciales de las estrategias. Aunque varía la implementación utilizada, la más extendida es la implementación CLK.

En los algoritmos propuestos en este trabajo, se optó por utilizar la librería LKH, principalmente por su facilidad de integración dentro del flujo de trabajo. Se utiliza la versión 2.0.9 disponible en la página del autor [58]. Se generó un archivo ejecutable, el cual para generar un archivo con la solución, requiere como argumentos un archivo de configuración de parámetros y la instancia TSP a resolver. Debido a que la intención es generar una solución aceptable, no se detalló en la afinación de los parámetros, optándose por utilizar los recomendados en el manual de la librería.

4.4.2. Generación de la selección de objetos

Aleatoriamente

Para generar el plan Π se elige un peso máximo $\mathcal{U}(0, W)$ el cual será el nuevo límite, después se itera de manera aleatoria sobre los elementos de M y se intenta agregar cada objeto Π a la selección, hasta que se itere sobre todos los objetos o se alcanca el peso máximo.

Algoritmo 30: Generación de forma aleatoria de Π

```
def aleatorio
    Π = vector de tamaño |M| inicializado en F
    M' = ordenar de forma aleatoria los objetos en M
    W' = generar un peso máximo en el intervalo [0, W]
    w = 0
    para cada π ∈ M' aleatorio hacer
        si w + π_w ≤ W'
            Π_[π_id] = T
            w += π_w
        fin
    fin
    regresa Π
fin
```

Codicioso

Heurística que ordena los objetos en función de un puntaje, y procede a intentar agregarlos hasta cumplir la restricción de peso. En el Algoritmo {31} se muestra el procedimiento en donde se cambia el estado de cada objeto en función de su puntaje, solo se acepta el objeto si supone un incremento en la aptitud de la solución. Empleando este procedimiento se pueden generar diferentes estrategias cambiando la función del puntaje y el criterio de aceptación. Algunas variantes emplean un puntaje en función del incremento en la aptitud del objeto con respecto al recorrido sin seleccionar ningún objeto.

Algoritmo 31: Algoritmo codicioso para generar Π

```
def codicioso ( Ξ : recorrido)
    M' = ordenar los objetos M en función de su razón π_p/π_w
    para cada π ∈ M' hacer
        si w + π_w ≤ W
            Π_[π_id] = T
            si Π es mejor
                w += π_w
            en otro caso
                Π_[π_id] = F
            fin
        fin
    fin
    regresa Π
fin
```

Packiterative

En esta estrategia se genera Π empleando el Algoritmo {19}. El cual consiste en variar la importancia de la relación ganancia-peso de forma iterativa al evaluar el puntaje utilizado para ordenar los objetos. En esta estrategia una vez ordenados los objetos en función de su puntaje se van agregando de manera codiciosa hasta que se alcance el peso máximo.

Bit-Flip

Se emplea la búsqueda local con vecindad Bit-Flip (ver Algoritmo {16}) alternando el estado de cada objeto de una selección inicial Π aleatoria. En el Algoritmo {32} se muestra su implementación, se recorre la vecindad de Π en orden aleatorio y se compara frente a la mejor solución conocida Ξ^* en caso de ser mejor se mueve la búsqueda a la nueva solución y genera una nueva vecindad. La búsqueda termina si se enumera toda la vecindad sin mejorar (se ha encontrado el óptimo local).

Algoritmo 32: Generación de Π mediante la búsqueda local Bit-Flip

```
def bfls (Ξ : recorrido)
    Π* = selección objetos inicial
    hasta que se cumpla el criterio de paro o no exista mejora
        para cada vecino aleatorio Π de bit-flip(Π*) hacer
            si Π es mejor que Π*
                Π* = Π
                romper
            fin
        fin
    fin
    regresa Π*
fin
```

Exacto

Se emplea un esquema de programación dinámica y fue propuesto en [95], para el problema derivado de empacar al viajar (*Packing While Travelling* — PWT), en el cual se parte de un recorrido Ξ dado y se debe encontrar la selección de objetos óptima. De forma similar al KP se genera una matriz dp con $|M|+1$ filas y $W+1$ columnas, en donde cada posición $dp_{[i][w]}$ representa la máxima ganancia posible al considerar i objetos con un peso máximo w. La regla de recursividad es:

$$\text{máx}\begin{cases} dp_{[i-1][w]} \\ dp_{[i-1][w-\pi_w]} + \pi_p - \rho\, r_{d[\pi_{id}]}\left(\frac{1}{v_{\text{máx}}-\nu w} - \frac{1}{v_{\text{máx}}-\nu(w-\pi_w)}\right) \end{cases} \tag{4.4}$$

El método exacto para generar la selección de objetos se resume en el Algoritmo {33} y consiste en primero obtener la distancia restante a terminar el recorrido en cada ciudad r_d. Empleando esta distancia se ordenan los objetos de mayor a menor distancia y se genera la matriz dp. El caso base del problema es recorrer Ξ sin ningún objeto, por lo que la ganancia para los casos en que $i=0$ y $w=0$ consisten en el tiempo que toma realizar el recorrido $\frac{\rho r_d[0]}{v_{\text{máx}}}$. Después se calcula la matriz dp por fila, calculando el mejor valor posible mediante la Ecuación (4.4), si el objeto puede llevarse con el peso considerado o en caso contrario el mejor valor es el obtenido considerando el objeto anterior con el mismo peso. Finalmente la reconstrucción del recorrido se realiza empleando el Algoritmo {15} considerando como w el peso que tiene la máxima aptitud considerando los $|M|$ objetos.

Algoritmo 33: Generación de Π mediante programación dinámica

def *dp (Ξ : recorrido inicial)*

r_d = vector de distancia restante desde cada ciudad
M' = ordenar los objetos en función de la distancia a finalizar el recorrido
dp = matriz de $|M|+1$ filas y $W+1$ columnas inicializadas en $-\infty$
$dp_{[0][0\ldots W]}$ = ganancia de recorrer Ξ sin ningún objeto para todos los pesos máximos -$\rho\left(\frac{r_{d[0]}}{v_{\text{máx}}}\right)$
para cada $i \in (\ 1\ \ldots\ W\ )$ **hacer**
  para cada $w \in (\ 1\ \ldots\ W\ )$ **hacer**
    $\pi = M'_{[i]}$
    si $\pi_w \leq w$
      $\Delta_{1/v} = \frac{1}{v_{\text{máx}}-\nu w} - \frac{1}{v_{\text{máx}}-\nu(w-\pi_w)}$
      $p = \pi_p - \rho\Delta_{1/v} r_{d[\pi_\xi]}$
      $p_d = dp_{[i-1][w]}$
      $p_p = dp_{[i-1][w-\pi_w]} + p$
      $dp_{[i][w]} = (p_d > p_p)$ **?** p_d **:** p_p
    en otro caso
      $dp_{[i][w]} = dp_{[i-1][wgt]}$
    fin
  fin
fin
w = peso con aptitud máxima al considerar los $|M|$ objetos
Π = reconstruir Π a partir de la matriz dp empleando el peso w
regresa Π

fin

4.5. Propuestas de estrategias de trayectoria

En esta sección se describe el diseño e implementación de las tres heurísticas diseñadas e implementadas en esta tesis: LS consiste en una búsqueda local simple que utiliza la estrategia CoSolver, ILS es la implementación de la búsqueda local iterada y RESGLS es la propuesta final de este trabajo de tesis para la heurística de trayectoria.

4.5.1. LS

Para entender el comportamiento del sistema CoSolver, se implementó la heurística más simple, la cual consiste en una búsqueda local que alterna entre una fase de optimización de la selección de objetos y una optimización del recorrido. El Algoritmo {34} muestra la implementación realizada, donde la función *bfls* corresponde al Algoritmo {32} y la función *optls* al Algoritmo {29}.

Este tipo de búsqueda (dos fases de optimización) generan un paisaje diferente por cada sub-problema en cada iteración de la heurística, dificultando el análisis del comportamiento de la búsqueda local [96]. Una vez que se ha optimizado en una fase el problema, la segunda optimización solo se enfoca en una región alrededor del óptimo local de la fase precedente. Lo anterior dificulta la exploración del espacio de búsqueda pues es de esperar, que debido a la dependencia existente, ninguna de las dos fases de optimización sea capaz de mover significativamente la solución. Note que al seleccionar los objetos en función de una ruta, se seleccionarán aquellos objetos que mejoren el tiempo y aporten mayor ganancia en la ruta definida. Cuando se regrese a la optimización del recorrido, los mejores movimientos serán aquellos que disminuyan el tiempo, lo que con una alta probabilidad reforzará la selección de objetos. Esta interacción entre las búsquedas locales dificulta la exploración del espacio de búsqueda, lo cuál

Algoritmo 34: Búsqueda local del sistema CoSolver

def *ls (τ : solución inicial)*
  $\tau^* = \tau$
  hasta que *se cumpla el criterio de paro o no exista mejora*
    $\tau_\Pi = \texttt{bfls}(\tau^*_\Pi, \tau^*_\Xi)$
    $\tau_\Xi = \texttt{optls}(\tau^*_\Xi, \tau_\Pi)$
    $\tau^* = \tau$ **si** τ *es mejor que* τ^*
  fin
  regresa τ^*
fin

debe considerarse al implementar el sistema CoSolver.

4.5.2. ILS

En el Algoritmo {35} se describe la implementación de la búsqueda local iterada para el TTP empleando como búsqueda local simple con el sistema CoSolver para TTP del Algoritmo {34} y como perturbación un 4-intercambio para el recorrido.

Algoritmo 35: Búsqueda Local Iterada para el TTP

def *ils (τ: solución inicial, k: cantidad segmentos = 4)*
  $\tau^* = \tau$
  hasta que *se cumpla el criterio de paro*
    hasta que *no exista mejora*
      $\Pi = \texttt{bfls}(\tau_\Pi, \tau_\Xi)$
      $\Xi = \texttt{optls}(\tau_\Xi, \Pi)$
      $\tau = \{\Xi, \Pi\}$ **si** $\{\Xi, \Pi\}$ *es mejor que* τ
    fin
    $\tau^* = \tau$ **si** τ *es mejor que* τ^*
    seleccionar de forma aleatoria k segmentos
    $s =$ obtener los k segmentos $[(0 \dots b)_0, \dots, (m \dots n)_k]$ del recorrido τ^*_Ξ
    $\tau_\Xi =$ reordenar los segmentos s como $[s_0, s_k, s_{k-1}, s_1]$
    $\tau_\Pi = \tau^*_\Pi$
  fin
  regresa τ^*
fin

El principal objetivo de la búsqueda local iterada es escapar del óptimo local τ^*. A diferencia de la búsqueda local simple, la *ils* se realiza hasta que se cumpla el criterio de paro, independiente de si existe mejora o no de la solución. Al igual que en la búsqueda local simple primero se realiza una optimización de la selección de objetos Π, seguida de la optimización del recorrido Ξ, empleando el óptimo encontrado para la selección de objetos, este proceso se realiza hasta que no se mejore la solución actual. Si el nuevo óptimo local τ es mejor que la mejor solución conocida τ^* se actualiza τ^*. Finalmente para mover la solución del óptimo local se realiza un 4-intercambio en el recorrido τ^*_Ξ, sin modificar la selección de objetos τ^*_Π. Note que el siguiente paso es optimizar la selección de objetos τ^*_Π al nuevo recorrido τ_Ξ.

La búsqueda local iterada mejora los resultados de la búsqueda local simple, pero no es suficiente para competir con los resultados obtenidos por los algoritmos de la literatura.

4.5.3. RESGLS

De los resultados obtenidos por la búsqueda loca iterada para el TTP se observó que a pesar de que la perturbación es capaz de mover la solución, el sistema CoSolver limita la exploración del espacio de búsqueda.

Para permitir el escape del óptimo local generado por el sistema CoSolver, se plantea emplear la estrategia de búsqueda local guiada. Para ello se utiliza un esquema similar al Algoritmo {6}, empleando la penalización al momento de optimizar el recorrido Ξ. La intención es que la búsqueda local considere movimientos previamente ignorados por el sistema CoSolver, "guiando" la solución mediante el recorrido. Hasta donde sabemos, este esquema es la primera estrategia que se concentra en el recorrido Ξ para escapar del óptimo local. Los resultados de esta nueva estrategia mejoraron los obtenidos previamente por la búsqueda local iterada, sin embargo se observó que existía la posibilidad de mejorar la exploración mediante un esquema de reinicio, pues la búsqueda es demasiado lenta para escapar del óptimo local. Debido a lo anterior la propuesta final incorpora un esquema de reinicio.

El Algoritmo {36} muestra el esquema propuesto. Como se describió anteriormente, GLS está basado en utilizar una función modificada basándose en la idea de incluir penalizaciones a ciertas características que estén presentes en la solución. Para generar la nueva función de aptitud $\hat{f}$ se utiliza una penalización sobre las aristas del recorrido Ξ, la cual se almacena como una matriz Φ que contiene el número de veces ϕ que una arista ha estado presente y ha tenido la menor utilidad en el óptimo local. El factor λ controla el impacto de la penalización y es calculado como en [28].

De esta forma el primer paso consiste en inicializar las penalizaciones a 0 y definir la función objetivo penalizada $\hat{f}$. Esta función hace que el óptimo local sea menos atractivo para la búsqueda y permite la generación de nuevas aristas que de otra forma serían ignoradas debido a la selección de objetos Π.

Mientras no se cumpla el tiempo especificado de búsqueda, se realiza una búsqueda local, basada en la metaheurística de trayectoria GLS y el sistema CoSolver. Esta búsqueda intercala entre la fase de optimización del recorrido *optls*, empleando la función penalizada $\hat{f}$ y la fase de optimización de la selección de objetos *bfls* que emplea la función de aptitud original (3.3).

Finalmente, una vez que se ha encontrado un óptimo local en el sistema CoSolver, se actualiza la penalización Φ. Si ha transcurrido cierto tiempo sin mejora, se realiza un reinicio. Para ello, basado en los resultados obtenidos con la búsqueda local simple y búsqueda local iterada, se genera un recorrido empleando la meta-heurística Lin-Kernighan y la selección de objetos mediante una búsqueda local con vecindad BitFlip.

Algoritmo 36: Búsqueda local guiada para el TTP

def *gls (τ: solución inicial, $\alpha = 1/2$)*
- $\tau^* = \tau$
- $\lambda = {}^{F(\tau)}/_{|\tau_\Xi|}$
- Φ = inicializar las penalizaciones a 0
- $\hat{f}(\tau) = F(\tau) - \lambda \sum_{\phi_i \in \Phi(\tau)} \phi_i$
- iniciar la cuenta del tiempo de reinicio
- **hasta que** *se cumpla el criterio de paro*
  - **hasta que** *se cumpla el tiempo de reinicio o no exista mejora*
    - τ_Ξ = `optls(`τ`)` empleando la función penalizada $\hat{f}$`()`
    - τ_Π = `bfls(`τ`)` empleando la función original F`()`
    - $\tau^* = \tau$ **si** τ *es mejor que* τ^*
  - **fin**
  - Π = actualizar la penalización π_i empleando el óptimo local τ
  - **si** *se cumple el tiempo de reinicio*
    - reiniciar el tiempo de reinicio
    - **si no** *se mejoró*
      - τ = generar nueva solución mediante LKH y Bit-Flip
      - Φ = reiniciar las penalizaciones
    - **fin**
  - **fin**
- **fin**
- **regresa** τ^*

fin

4.6. Algoritmo memético basado en control de diversidad (DMA)

En esta sección se describe el diseño e implementación del algoritmo memético propuesto para resolver el TTP. El Algoritmo {37} muestra el esquema DMA, el cual es un algoritmo memético basado en la estrategia RESGLS. La idea central del algoritmo consiste en promover la intensificación de la solución mediante un sistema CoSolver basado en la metaheurística GLS y la exploración del espacio de búsqueda mediante la inclusión de un método explicito de control de diversidad que permita mantener una heterogeneidad suficiente en la población.

Algoritmo 37: Algoritmo memético para el TTP

def *dma (np: tamaño de la población, g: número mínimo de generaciones)*
- P = generar la población inicial con *np* individuos
- **hasta que** *se cumpla el criterio de paro*
  - C = descendencia mediante la cruza de P
  - C = aplicar procedimiento de mejora a los elementos en C
  - P = seleccionar la nueva población de $P \cup C$
- **fin**
- **regresa** *el mejor elemento en* P

fin

El algoritmo DMA esta conformado por las siguientes fases: inicializar la población, selección de padres, cruza de la población, mejora de la descendencia y reemplazar la nueva generación. A continuación se procede a detallar la implementación de cada fase.

4.6.1. Inicialización

El Algoritmo {38}, muestra la inicialización de la población. Genera cada individuo mediante la estrategia Lin-Kernighan en el recorrido y una búsqueda local con vecindad Bit-Flip en la selección de objetos. En la literatura se ha reportado que generar soluciones iniciales de alta calidad es importante, pues se requiere de una extensa cantidad de movimientos para mover las soluciones a configuraciones prometedoras. La inicialización se mantiene simple y sin considerar el promover la diversidad pues la aleatoriedad del algoritmo de mejora (RESGLS) ha mostrado ser capaz de modificar los individuos lo suficiente, para generar soluciones diferentes, en las instancias trabajadas.

Algoritmo 38: Inicialización de la población para el algoritmo DMA

```
def init ( np: tamaño de la población )
    P = ∅ para cada i ∈ ( 1 ... np ) hacer
        τ_Ξ = generar recorrido mediante estrategia Lin-Kernighan
        τ_Π = generar la selección de objetos aplicado la búsqueda local bitflip a Ξ
        P = P ∪ τ
    fin
    regresa la población P
fin
```

4.6.2. Proceso de cruza

En DMA la descendencia C esta conformada por $|P|$ individuos productos de la recombinación de los individuos de la población P. El algoritmo {39}, muestra el procedimiento para generar la descendencia.

Para generar cada hijo $\tau \in C$, primero se seleccionan los dos padres, mediante una selección por torneo binario. El torneo binario consiste en seleccionar de forma aleatoria dos individuo de la población P y compararlos por su aptitud, eligiendo al mejor como uno de los padres. En los operadores de cruza es habitual que la descendencia herede las características presentes en los padres. En el TTP se deben heredar las características del recorrido Ξ y de la selección de objetos Π.

Para el recorrido Ξ, la característica más importante es si una arista es recorrida o no, por ello con la intención de preservar la información presente en los padres sobre que aristas son más útiles, se emplea el operador de recombinación de aristas *erx* del Algoritmo {13}. Este operador emplea los recorridos del primer padre a_Ξ y del segundo padre b_Ξ para generar un recorrido, que fomenta el uso de aristas presentes en los padres y no utiliza las aristas que no están presentes en ninguno de los padres.

Siguiendo la misma línea de pensamiento se desea aquellos objetos que los padres compartan de forma que para la selección de objetos Π, optándose entonces por emplear la intersección de la selección del primer padre a_Π y la selección de objetos del segundo padre b_Π, de forma que el hijo solo incluye aquellos objetos presentes en ambos padres.

4.6.3. Intensificación

Una vez que se cuenta con la descendencia, se procede a realizar la fase de intensificación, para ello se emplea la propuesta de estrategia de mejora basada en GLS. Debido a que se emplea dentro de un algoritmo poblacional, no se considera indispensable la fase de reinicio, pues la población cubre el papel de exploración. El Algoritmo {40} muestra la implementación de la fase de intensificación.

Un criterio importante en la fase de intensificación es el tiempo dedicado a la búsqueda local. El algoritmo DMA tiene como criterio de paro el tiempo de ejecución, con la restricción de realizar al menos g generaciones. Para calcular el tiempo dedicado a cada iteración se considera que en un inicio

Algoritmo 39: Heurística de cruza

def *cx (P: población)*
  $C = \varnothing$
  para cada $i \in (\ 1\ \ldots\ |P|\ )$ **hacer**
    a, b= `torneo-binario(`P`)`, `torneo-binario(`P`)`
    $\tau = \{$ `erx(`a_Ξ, b_Ξ`)`, $a_\Pi \cap b_\Pi\ \}$
    $C = C \cup \{\tau\}$
  fin
  regresa *la descendencia* C
fin

Algoritmo 40: Búsqueda local utilizada en la estrategia DMA

def *gls (C: población)*
  $C' = \varnothing$
  para cada $\tau \in C$ **hacer**
    $\tau^* = \tau$
    $\lambda = {}^{F(\tau)}/_{|\tau_\Xi|}$
    Φ = inicializar las penalizaciones a 0
    $\hat{f}(\tau) = F(\tau) - \lambda \sum_{\phi_i \in \Phi(\tau)} \phi_i$
    hasta que *se cumpla el tiempo de intensificación*
      hasta que *no exista mejora*
        τ_Ξ = `optls(`τ`)` empleando la función penalizada $\hat{f}$`()`
        τ_Π = `bfls(`τ`)` empleando la función original F`()`
        $\tau^* = \tau$ **si** τ *es mejor que* τ^*
      fin
      Φ = actualizar la penalización empleando el óptimo local τ
    fin
    $C' = C' \cup \{\tau^*\}$
  fin
  regresa *la población* C'
fin

se debe favorecer la exploración, por ello la fase de intensificación tendrá menor importancia que hacia el final de la ejecución en donde se favorece la intensificación. Un ejemplo para el cálculo del tiempo es emplear una progresión aritmética como la que se muestra en la Ecuación (4.5). En esta progresión, g indica el número de iteraciones a realizar, t_m el tiempo mínimo dedicado a la búsqueda, t_f el tiempo total del algoritmo e i el número de iteración actual. Note que en la primer iteración $i = 1$ el valor es el tiempo mínimo $t_{ls} = t_m$, mientras que en la última generación $i = g$ el tiempo es el doble del tiempo promedio $\frac{2t_f}{g|P|} - t_m$ y la suma es el tiempo total $\sum t_{ls_i} = \frac{g(t_{ls_1} + t_{ls_g})}{2} = \frac{t_f}{|P|}$. De esta forma la Ecuación (4.5), permite estimar el tiempo dedicado a la búsqueda en función del número de iteraciones y el tiempo dedicado al algoritmo, aumentando gradualmente los recursos computacionales dedicados a la intensificación de las soluciones.

$$t_{ls} = t_m + \left(\frac{2(t_f - |P| g t_m)}{g(g-1)|P|} \right) (i-1) \tag{4.5}$$

4.6.4. Reemplazamiento

La selección de los objetos para la nueva generación se basa en la estrategia de reemplazamiento RMDDC del Algoritmo {11}, en la cual se penaliza aquellas soluciones que tengan una distancia cercana a la población actual, la cual se varía entre una distancia mínima D_0 y 0. También, el algoritmo favorece la diversidad al seleccionar de forma aleatoria entre aquellas soluciones para las cuales no exista una mejor solución tanto en aptitud como en distancia (dominancia de Pareto).

El Algoritmo propuesto en este trabajo emplea la idea de penalizar la aptitud de las soluciones en función de la distancia. Se observó que la convergencia de la población es lenta. Para aumentar la intensificación y mejorar la convergencia de la población, se optó por emplear un esquema elitista, donde la solución seleccionada es la que tenga mejor aptitud una vez se ha aplicado la penalización en función de la distancia DCN.

En el Algoritmo {41} se presenta la propuesta. Primero se agrega el mejor elemento en la union M, de la población actual P con la descendencia C. Después hasta que se hayan seleccionado $|P|$ elementos se calcula la distancia al vecino más cercano d, de cada solución en M usando como referencia la nueva población P_n. Utilizando la distancia d se penalizan aquellas soluciones que tengan una distancia menor que D, lo que indica que son muy similares a los elementos ya incluidos en la nueva población. Finalmente se selecciona la solución con mejor aptitud de las soluciones penalizadas $\hat{F}$, eliminándose de las soluciones candidatas M y agregándose a la nueva población P_n.

Algoritmo 41: Esquema de selección RMDDC en el algoritmo DMA

def *sel (M: población ∪ descendencia, D: distancia mínima)*
  F = aptitud de los elementos en M
  τ_i = posición del mejor individuo en M
  M, $F = M \setminus M_{[\tau_i]}$, $F \setminus F_{[\tau_i]}$
  $P_n = [M_{[\tau_i]}]$
  hasta que *se* $|P_n| = P$
    d = arreglo con el valor `dcn`$(M_{[i]}, P_n)$ para cada elemento en M
    $\hat{F}$ = penalizar F con el valor $-\infty$ si $(d_{[i]} < D)i$
    τ_i = elemento con mejor aptitud en $\hat{F}$
    M, $F = M \setminus M_{[\tau_i]}$, $F \setminus F_{[\tau_i]}$
    $P_n = P_n \cup M_{[\tau_i]}$
  fin
  regresa *la nueva población* P_n
fin

Distancia mínima al vecino más cercano

La estrategia de selección requiere de una medida de distancia entre los individuos de la población para mantener la diversidad. Al estar conformadas por componentes, las soluciones en el TTP pueden diferir en el recorrido Ξ o en la selección de objetos Π. Debido a que la búsqueda local se enfoca en la optimización del recorrido, como medida de distancia se utiliza la diferencia de aristas entre los recorridos Ξ. En la medida de la distancia se considera que el TTP no es simétrico, es decir arista $\{\xi_i, \xi_j\}$ es diferente a $\{\xi_j, \Xi\}$. El cálculo de la distancia se muestra en el Algoritmo {42}, donde se emplea un vector auxiliar *aux* para almacenar las aristas del recorrido τ_Ξ, después se procede a contar el número de veces que cada elemento de la nueva población es diferente, para finalmente regresar la menor distancia encontrada.

Algoritmo 42: Distancia al vecino más cercano

def *dcn (τ: solución, P_n: población de referencia)*
  $a = \tau_{\Xi}$
  $dcn = \infty$
  para cada $s \in P_n$ **hacer**
    $b = s_{\Xi}$
    aux = arreglo inicializado en 0
    para cada $i \in (\ 0\ \ldots\ |\Xi| - 1\ )$ **hacer**
      $aux_{[a_{[i]}]} = a_{[i+1 \mod |a|]}$
    fin
    $d = 0$
    para cada $i \in (\ 0\ \ldots\ |a| - 1\ )$ **hacer**
      $d \mathrel{+}= 1$ **si** $aux_{[b_{[i]}]} = b_{[i+1 \mod |b|]}$
    fin
    $dcn = d$ **si** $dcn > d$
  fin
  regresa *la distancia dcn*
fin

Penalización

Se emplea la distancia mínima D para controlar el balance entre calidad (aptitud) y diversidad (diferentes soluciones). Para ello se penaliza la aptitud de las soluciones F en la población M en función de su distancia mínima d a la selección actual P_n. De esta manera la nueva aptitud penalizada $\hat{F}$ disminuye la importancia de aquellas soluciones que sean demasiado similares ($d \leq D$) a los elementos previamente seleccionados.

El variar el parámetro D permite controlar la exploración e intensificación del algoritmo. Si la distancia mínima D es baja ninguna solución es penalizada y el algoritmo incluirá solo las soluciones de alta calidad con poca diversidad incrementando la intensificación. Por el contrario si el valor de D es alto, la selección incluirá aquellas soluciones que aporten mayor diversidad, independientemente de su calidad, promoviendo la exploración.

Usando el criterio de paro por tiempo, la manera más utilizada para controlar la diversidad es variar de forma dinámica el valor de D empleando el tiempo de ejecución como parámetro. La función más común para el cálculo de la distancia mínima para penalizar es un decremento lineal de la forma:

$$D = D_0 \left(1 - \frac{t}{t_f}\right) \tag{4.6}$$

Siendo D_0 la distancia mínima inicial, t_f es la duración del algoritmo DMA y t el tiempo transcurrido desde el inicio del algoritmo. De esta forma al inicio de la ejecución $t = 0$ el requisito de distancia es D_0, y al final de la ejecución $t = t_f$ el requisito es 0, permitiendo que las soluciones sean seleccionadas solo en base a su aptitud.

Capítulo 5

Resultados

En este capítulo se presentan los resultados del diseño y la evaluación del desempeño de las estrategias propuestas en este trabajo de tesis. Debido al tamaño del conjunto de problemas de referencia (TTPLIB), se trabajó con un subconjunto que abarcan los tamaños pequeños y medianos. Debido a que el tiempo de límite clásico de 600 segundos para el TTP fue eliminado en la tercera competencia de optimización de problemas con multiples componentes interdependientes [97], se utilizaron diferentes duraciones de ejecución.

Al contarse con procesos estocásticos en las propuestas de este trabajo, para la comparación del desempeño de los algoritmos se sigue un esquema similar al propuesto en [98]. En cada ejecución se realizan treinta repeticiones independientes y la comparación se realiza aplicando un conjunto de pruebas estadísticas [99], como se muestra en el Algoritmo {43}. Asumiendo un nivel de confianza del noventa y cinco por ciento, se realizan comparaciones por pares para determinar si existe diferencia entre las distribuciones. En específico, primero se verifica mediante la prueba *Shapiro-Wilk* si los dos conjuntos siguen una distribución normal, en caso de no ser así, se compara empleando la técnica *Kruskal-Wallis*. Si ambos conjuntos provienen de una distribución normal se comprueba la homogeneidad de las varianzas mediante la prueba *Levene* y al tener varianzas iguales se aplica la prueba *ANOVA*, en caso contrario, la prueba *Welch*.

Algoritmo 43: Comparación de dos distribuciones de resultados

```
def comparación (a: primer distribución, b: segunda distribución )
    si Shapiro-Wilk(a) ∧ Shapiro-Wilk(b) > 0.05
        si Levene(a,b) > 0.05
            regresa ANOVA(a,b) ≤ 0.05
        en otro caso
            regresa Welch(a,b) ≤ 0.05
        fin
    en otro caso
        regresa Kruskal-Wallis(a,b) ≤ 0.05
    fin
fin
```

Empleando el criterio anterior, en este trabajo la oración “el algoritmo A es mejor que el algoritmo B” significa que las diferencias de las medias entre ambos algoritmos son estadísticamente significativas y que la media obtenida por A es mejor o igual ($\preceq$) que la media obtenida por B.

5.1. Solución de la selección de objetos

En este trabajo de tesis, se propone un algoritmo que emplea el sistema CoSolver para resolver el TTP, que incluye una búsqueda con penalizaciones para la ruta Ξ de una solución. Por ello se requiere seleccionar un algoritmo para resolver la selección de objetos de entre los descritos en la sección anterior.

Para seleccionar el algoritmo a incorporar, se comparó el desempeño de las siguientes heurísticas: **bitflip** {32}, **exacto** {33}, **codicioso** {31}, **aleatorio** {30} **packiterative** {19}.

Debido a las restricciones de memoria impuestas por la búsqueda exacta, se seleccionaron problemas con pocas ciudades. Las instancias TTP utilizadas son: eil51, berlin52, eil76, kroA100, con la capacidad máxima (10) y factor de objetos 10, en sus variantes no correlacionadas (*uncorr* — unc), no correlacionadas con pesos similares (*uncorr-similar-weights* — usw) y fuertemente correlacionada (*bounded-strongly-corr* — bsc). Para cada instancia, se generó un conjunto de 100 recorridos diferentes de forma aleatoria empleando la heurística *GRASP* y como solución inicial se emplea una selección de objetos vacía.

En la Tabla 5.1 se muestran los resultados obtenidos al comparar las aptitudes obtenidas por los métodos al generar la selección de objetos Π para los 100 recorridos aleatorios de las 12 instancias de TTP. Se muestra la cantidad de veces que el algoritmo de la fila fue mejor ($\wedge$), igual ($=$) o peor ($\vee$) que el algoritmo de la columna. En los dos extremos principales se tienen a los métodos **aleatorio** (no es mejor que ningún otro algoritmo) y el **exacto** (no es peor que ningún otro algoritmo). Al comparar las dos heurísticas de construcción (**codicioso** y **packiterative**) se puede observar que presentan un comportamiento similar, frente a los otros algoritmos, siendo **packiterative** superior a la estrategia codiciosa. Finalmente el algoritmo que mejores resultados obtiene es la búsqueda local **bitflip**, el cual en más del cincuenta por ciento de los algoritmos obtiene la solución óptima, siendo superior a las estrategias restantes.

Tabla 5.1: Comparativa por pares de la aptitud al generar la selección de objetos Π para 1200 recorridos aleatorios

	bitflip			exacto			codicioso			packiterative			aleatorio		
	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$
bitflip		——		0	819	381	1200	0	0	1175	15	10	1200	0	0
exacto	381	819	0		——		1200	0	0	1177	23	0	1200	0	0
codicioso	0	0	1200	0	0	1200		——		375	0	825	1200	0	0
packiterative	10	15	1175	0	23	1177	825	0	375		——		1200	0	0
aleatorio	0	0	1200	0	0	1200	0	0	1200	0	0	1200		——	

En la Tabla 5.2 se muestra la comparación del tiempo requerido por cada método para los 1200 recorridos, indicando que el método de la fila requirió menos tiempo ($\wedge$), el mismo tiempo ($=$) o más tiempo ($\vee$) que el método de la columna. Al analizar los resultados para cada método se encuentran resultados acorde a los esperado, el algoritmo que requiere la mayor cantidad de tiempo es el **exacto**, seguido por **bitflip**, los algoritmos de construcción y siendo el método **aleatorio** el que requiere el menor tiempo.

Tabla 5.2: Comparativa por pares del tiempo requerido para generar la selección de objetos, dado un recorrido

	bitflip			exacto			codicioso			packiterative			aleatorio		
	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$
bitflip		——		1200	0	0	0	0	1200	0	0	1200	0	0	1200
exacto	0	0	1200		——		0	0	1200	0	0	1200	0	0	1200
codicioso	1200	0	0	1200	0	0		——		33	18	1149	0	0	1200
packiterative	1200	0	0	1200	0	0	1149	18	33		——		0	0	1200
aleatorio	1200	0	0	1200	0	0	1200	0	0	1200	0	0		——	

Los resultados anteriores son más claros al observar los valores típicos del tiempo de ejecución de los diferentes algoritmos. Para ello en la Tabla 5.3 se muestra el tiempo de ejecución promedio de un recorrido aleatorio requerido por cada uno de los algoritmos para los tres tamaños de problemas manejados. Los resultados son congruentes con la complejidad de los algoritmos, el método **exacto** obtiene la solución pero es muy lento, los algoritmos de construcción son rápidos pero la calidad de su solución es baja, finalmente la heurística **bitflip** es más costosa pero su comportamiento es similar a los algoritmos de construcción. Note que el algoritmo **bitflip** requiere un orden de magnitud más de tiempo que los algoritmos de construcción, pero en calidad de solución es superior.

Tabla 5.3: Estadísticas para 30 ejecuciones de los tiempos promedio en segundos requeridos para resolver un recorrido aleatorio de la instancia *kroA100* fuertemente correlacionada

	berlin52		eil72		kroA100	
	$\bar{t}$	σ	$\bar{t}$	σ	$\bar{t}$	σ
bitflip	0.02837	0.004516	0.04245	0.003504	0.08294	0.008643
exacto	4.77763	0.196371	11.06786	0.644300	16.69762	0.538159
codicioso	0.00287	0.000465	0.00485	0.000116	0.00859	0.000847
packiterative	0.00315	0.000659	0.00159	0.00006	0.00403	0.000618
aleatorio	0.00004	0.00001	0.00002	0.000005	0.00003	0.000006

El tiempo requerido por los algoritmos es dependiente del tamaño del problema. Los resultados muestran que el algoritmo **bitflip** presenta un buen balance entre capacidad de encontrar la mejor solución y el tiempo que requiere para realizar la búsqueda. A continuación se procede a mostrar la importancia de la inicialización en el desempeño del método. Para ello se inicializa la solución a mejorar mediante la búsqueda local, con las siguientes heurísticas de construcción de la selección de objetos: **vacío**, **codicioso** {31}, **aleatorio** {30} y **packiterative** {19}.

En la Tabla 5.4 se muestra la comparación por pares de los resultados obtenidos por la heurística **bitflip**. Muestra que la calidad de la solución mediante la búsqueda local no se ve afectada por la inicialización, para la mayoría de los recorridos; sin embargo existen casos particulares en los cuales es evidente que la inicialización juega un papel relevante.

Tabla 5.4: Comparativa por pares de las diferentes inicializaciones en la selección de objetos en 1200 recorridos aleatorios, al aplicarles la búsqueda local.

	vacío			codicioso			packiterative			aleatorio		
	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$
vacío		——		14	1178	8	19	1133	48	18	1173	9
codicioso	8	1178	14		——		18	1138	44	17	1166	17
packiterative	48	1133	19	44	1138	18		——		55	1128	17
aleatorio	9	1173	18	17	1166	17	17	1128	55		——	

Por otra parte, se espera que exista un impacto en el tiempo de ejecución de la búsqueda local, dependiendo de la inicialización. La Tabla 5.5 muestra esta dependencia al comparar por pares el tiempo requerido para obtener la selección de objetos, indicando cuántas veces el método de la fila fue más rápido ($\wedge$), igual ($=$) o más lento ($\vee$) que el método de la columna. Al analizar el tiempo, se observa que, es menor conforme se mejora la calidad de la solución inicial.

Sin embargo, como se observa en la Tabla 5.6, el tiempo se mantiene en el mismo orden de magnitud independientemente de la inicialización. Esto indica que, para los problemas utilizados, la búsqueda local es lo suficientemente robusta para mover la solución en el espacio de búsqueda.

Es importante destacar que en caso de utilizarse el método exacto, se podrían mejorar las soluciones para un porcentaje de las instancias, pero esto implicaría realizar muy pocas generaciones (debido al

Tabla 5.5: Comparativa por pares del tiempo requerido por la búsqueda local para obtener la solución del recorrido, al utilizar diferentes inicializaciones

	vacío			codicioso			packiterative			random		
	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$
vacío		——		0	5	1195	0	5	1195	265	802	133
greedy	1195	5	0		——		161	226	813	1197	3	0
packiterative	1195	5	0	813	226	161		——		1197	3	0
random	133	802	265	0	3	1197	0	3	1197		——	

Tabla 5.6: Estadísticas del tiempo de solución de un recorrido aleatorio por las diferentes inicializaciones para tres instancias

	berlin52		eil76		kroA100	
	$\bar{t}$	σ	$\bar{t}$	σ	$\bar{t}$	σ
vacío	0.02837	0.004516	0.04245	0.003504	0.08379	0.011812
codicioso	0.02116	0.003055	0.03203	0.004311	0.06651	0.014572
packiterative	0.00931	0.002401	0.03248	0.005321	0.05218	0.012149
aleatorio	0.02741	0.004224	0.04879	0.007486	0.09410	0.014660

alto coste computacional en tiempo y memoria del método exacto). Esto provocaría que las búsquedas estuvieran muy enfocadas en encontrar una buena selección de objetos y dejarían pocos recursos para optimizar el recorrido. Por otra parte la heurística de construcción sufre de inconvenientes similares, pues aunque requiere de pocos recursos computacionales, la calidad de las soluciones es baja en comparación a los otros métodos. Provocando que la búsqueda se centre demasiado en la optimización de la ruta al utilizar pocos recursos para la selección de objetos.

En base a los resultados anteriores se justifica la implementación de la búsqueda local con vecindad Bit-Flip como la heurística a utilizar para la solución de la selección de objetos. Debido a que los resultados indican que la selección de objetos para la clase de problemas utilizados, pueden ser eficientemente aproximada mediante esta búsqueda local simple.

5.2. Sistema CoSolver

La idea principal del sistema CoSolver consiste en alternar entre la optimización del recorrido Ξ y la optimización de la selección de objetos Π. Con la intención de establecer una referencia del comportamiento del sistema CoSolver se implementó la búsqueda local simple **ls** {34}, la cual intercala dos búsquedas locales, una con vecindad 2-Opt que optimiza el recorrido y otra con vecindad Bit-Flip que optimiza la selección de objetos. En esta sección se presentan los resultados obtenidos de la estrategia CoSolver en el rango de tamaño de problemas pequeño a mediano. Para esto se seleccionan los siguientes problemas: eil51, berlin52, kroA100, a280, u574, dsj1000, fnl4461. Se utilizan las tres variantes: no correlacionadas, no correlacionadas con pesos similares y fuertemente correlacionadas, empleando la máxima capacidad para la mochila y contando con diez objetos a seleccionar en cada ciudad.

Primero se estudió la importancia de la inicialización en la búsqueda local, para ello se realizaron ejecuciones empleando diferentes combinaciones de inicialización. Se observa el desempeño de la búsqueda al variar la inicialización del recorrido, para ello se comparan las siguientes tres estrategias para inicializar Ξ: **lkh** (emplear la librería LKH [58]), **vecino cercano** y **aleatorio** (ambas mediante el Algoritmo {28}).

La Tabla 5.7 muestra la comparación por pares de la aptitud obtenida por **ls**, iniciando con una

solución generada con alguna de las estrategias de construcción para el recorrido y generando la selección de objetos mediante **bitflip** partiendo de una selección de objetos vacía. Se observa que existe una ventaja clara al iniciar con un buen recorrido, siendo superior el partir la búsqueda local de un recorrido generado mediante **lkh**. Se prosigue con comparar el desempeño en la aptitud obtenida por la búsqueda local al inicializar con **lkh** y construir la selección de objetos con diferentes estrategias.

Tabla 5.7: Comparación de la aptitud obtenida por **ls** al variar las heurística de construcción de Ξ, generando Π mediante bitflip

	lkh			vecino cercano			aleatorio		
	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$
lkh		——		19	2	0	21	0	0
vecino cercano	0	2	19		——		12	7	2
aleatorio	0	0	21	2	7	12		——	

La Tabla 5.8 muestra la comparación por pares de aplicar **ls** a la solución inicial generada mediante cada método de construcción para la selección de objetos. Cada método genera una solución inicial a partir de una selección de objetos vacía y un recorrido generado mediante la librería LKH. Se puede apreciar que existe cierto balance entre las inicializaciones, pero la inicialización aleatoria de la selección de objetos Π obtiene mejores resultados en ocho instancias.

Tabla 5.8: Comparación de las diferentes heurísticas de construcción de la selección de objetos, empleando un recorrido generado mediante la librería LKH y aplicar la búsqueda local **ls**.

	bitflip			codicioso			packiterative			aleatorio		
	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$
bitflip		——		0	19	2	2	16	3	4	10	7
codicioso	2	19	0		——		1	19	1	4	9	8
packiterative	3	16	2	1	19	1		——		4	10	7
aleatorio	7	10	4	8	9	4	7	10	4		——	

Esto indica que es importante un recorrido corto, pero la solución encontrada mediante la estrategia **lkh** no es el recorrido óptimo para el TTP. El utilizar una mala inicialización de la selección permite a la heurística **ls** modificar el recorrido.

Para mostrar lo anterior, se compara la distancia entre el recorrido inicial y el resultado obtenido por la búsqueda **ls** (óptimo local). Se utiliza una métrica en función de la diferencia en estructura [100], donde la distancia $D(s, s) = k$ entre dos soluciones indica que la solución s' difiere en k aristas con s', considerando que el TTP no es simétrico. En la Tabla 5.9 se presenta la comparación por pares de la distancia entre el recorrido inicial y el resultado de la búsqueda **ls**. Utilizando distintas meta-heurísticas de inicialización para la selección de objetos, las cuáles parten de una selección vacía y un recorrido generado mediante LKH. Se indica que la distancia de la inicialización obtenida por la heurística de la fila es mayor ($\wedge$), igual ($=$) o menor ($\vee$) que la obtenida por la columna. Observe que la inicialización con la estrategia aleatoria es por mucho la estrategia que modifica más el recorrido.

Los resultados anteriores ejemplifican las siguientes características importantes del sistema CoSolver.

- La capacidad de exploración del sistema CoSolver es poca, pues requiere de una buena inicialización para obtener resultados de alta calidad.
- El sistema CoSolver carece de una estrategia para escapar de óptimos locales, lo que se observa al encontrar mejores soluciones con inicializaciones sub-optimas.

Tabla 5.9: Comparación de la distancia entre la solución inicial y el resultado de la búsqueda local. Se varia la heurística de construcción de la selección de objetos y se genera el recorrido mediante LKH

	bitflip			codicioso			packiterative			aleatorio		
	∧	=	∨	∧	=	∨	∧	=	∨	∧	=	∨
bitflip		——		0	18	3	0	8	13	0	0	21
codicioso	3	18	0		——		0	11	10	0	0	21
packiterative	13	8	0	10	11	0		——		0	8	13
aleatorio	21	0	0	21	0	0	13	8	0		——	

5.3. Métodos de trayectoria

Para mejorar el desempeño del sistema CoSolver observado en la sección anterior se implementaron las siguientes estrategias: la búsqueda local simple (Algoritmo {34}) (**ls**), la búsqueda local iterada (Algoritmo {35}) (**ils**) y la búsqueda local guiada (Algoritmo {36}) con reinicio (**resgls**) y sin reinicio (**gs**). Con la intención de abarcar el rango de tamaño de instancia pequeño a mediano se seleccionaron las siguientes instancias: berlin52, kroA100, a280, u574, dsj1000, rl1889 y fnl4461. De cada instancia se utilizan dos dos variantes, correlacionadas y no correlacionadas, empleando la maxima capacidad y diez objetos disponibles por ciudad.

En la Tabla 5.10 se muestra la comparación por pares de la aptitud obtenida por los diferentes algoritmos propuestos. Se realizaron treinta repeticiones independientemente con una duración de tiempo máxima de seis horas y la inicialización fue mediante **bitflip** y **lkh**. Es importante destacar que la búsqueda local simple no emplea la totalidad del tiempo establecido, pues detiene la búsqueda al encontrar un óptimo local. Se realiza la comparación de esta forma pues la intención es mostrar el comportamiento de la búsqueda frente a estrategias que permiten el escape del óptimo. Se observa que los algoritmos propuestos son claramente superiores a la búsqueda local simple, siendo el algoritmo con mejor desempeño la estrategia de búsqueda local guiada en sus dos variantes.

Tabla 5.10: Comparación por pares de los algoritmos propuestos

	gls			ils			ls			resgls		
	∧	=	∨	∧	=	∨	∧	=	∨	∧	=	∨
gls		——		5	6	3	8	6	0	0	8	6
ils	3	6	5		——		5	9	0	3	0	11
ls	0	6	8	0	9	5		——		0	0	14
resgls	6	8	0	11	0	3	14	0	0		——	

En la Tabla 5.11 se muestran los valores objetivos máximos obtenidos por cada algoritmo en las treinta ejecuciones y el mejor valor entre los algoritmos se muestra resaltado en negritas. El algoritmo **resgls** no solo es superior estadísticamente sino que obtiene, en la mayoría de las instancias, el valor máximo de las tres estrategias. Note que la estrategia basada en GLS obtiene (entre **resgls** y **gsl**) la mejor solución en todas las instancias, salvo en una. Mostrando que existen claros beneficios de esta estrategia al incluirla en el sistema CoSolver.

La estrategia **ils** realiza una perturbación al óptimo local para encontrar una nueva solución candidata que permita escapar del óptimo, por otra parte la heurística **gls** fuerza a la solución a alejarse del óptimo al penalizar las características presentes en el óptimo. Esto hace que el algoritmo **gls** tenga una mayor capacidad de exploración, ya que el algoritmo **ils** podría caer una y otra vez en el mismo óptimo local, mientras que si esto ocurre en **gls**, cada vez se penalizará más la solución y eventualmente escapará de esa región.

Tabla 5.11: Valor objetivo máximo obtenido por los diferentes algoritmos con ejecuciones de seis horas

	gls	ils	ls	resgls
berlin52–unc	**96519.12**	**96519.12**	92950.29	**96519.12**
berlin52–bsc	**180307.93**	**180307.93**	162213.33	**180307.93**
kroA100–unc	**159135.22**	**159135.22**	155585.14	**159135.22**
kroA100–bsc	**215684.85**	**215684.85**	206779.33	**215684.85**
a280–unc	**425617.67**	418614.07	413702.25	423735.85
a280–bsc	579299.18	580595.32	578982.35	**583523.12**
u574–unc	944546.90	929542.96	924034.54	**948416.77**
u574–bsc	**1353990.15**	1316935.70	1303124.00	1347731.11
dsj1000–unc	1475497.88	1474061.39	1482348.52	**1483661.13**
dsj1000–bsc	2031609.23	2017078.41	2009904.73	**2052018.26**
rl1889–unc	2871677.90	2877856.18	2905295.94	**2923563.36**
rl1889–bsc	3796458.80	4163815.16	3992097.49	**4271724.02**
fnl4461–unc	656059.21	659210.83	657442.88	**659375.02**
fnl4461–bsc	835102.33	**836388.78**	832660.53	836254.91

En la Tabla 5.12 se muestra la comparación de la distancia (en aristas) entre el recorrido inicial y el final, al resolver las distintas instancias con los algoritmos. Se indica cuántas veces el algoritmo de la fila obtiene una distancia promedio mayor ($\wedge$), igual ($=$) o inferior ($\vee$) con respecto al algoritmo de la columna. Se observa que las estrategias de búsqueda local guiada son estadísticamente superiores a la búsqueda local iterada, lo que indicaría que tienen un mayor potencial de exploración.

Tabla 5.12: Comparación de la distancia entre solución inicial y el resultado de la búsqueda con las diferentes estrategias

	gls			ils			ls			resgls		
	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$
gls		——		5	8	1	8	6	0	1	7	6
ils	1	8	5		——		6	8	0	0	3	11
ls	0	6	8	0	8	6		——		0	0	14
resgls	6	7	1	11	3	0	14	0	0		——	

Para entender el comportamiento de la exploración de los algoritmos se muestra la evolución del valor objetivo obtenido por las estrategias de trayectoria con ejecuciones de tres días. En la Figura 5.1 se muestra la evolución para seis instancias, donde nuevamente se aprecia la superioridad del algoritmo *resgls*. Se observa que el algoritmo tiene mejor desempeño en las instancias de mayor tamaño en comparación con la búsqueda local guiada. Este comportamiento se atribuye a que la búsqueda local guiada es incapaz de moverse a una distancia grande con respecto a la solución inicial, por lo que realizar reinicios esporádicos es beneficioso para explorar el espacio de búsqueda. Note que no se observa estancamiento al final de la ejecución, existiendo la posibilidad de mejorar la calidad de la solución al emplear estrategias para mejorar la eficiencia de la implementación.

En la Figura 5.2 se muestran los diagramas de caja para la aptitud obtenida por cada método, para seis instancias. Se aprecia como el método **resgls**, en general, obtiene mejor solución promedio con una distribución con poca varianza. Esta poca variabilidad de la distribución de las soluciones de la búsqueda local guiada **gs** muestra que la búsqueda es insuficiente para realizar una correcta exploración del espacio de búsqueda. Esto indica que aún se tiene una fuerte dependencia a la solución inicial, debido a esto la estrategia de reinicio en **resgls** funciona como mecanismo de exploración, al independizar la búsqueda de la solución inicial.

Figura 5.1: Evolución del valor objetivo de las estrategias de trayectoria

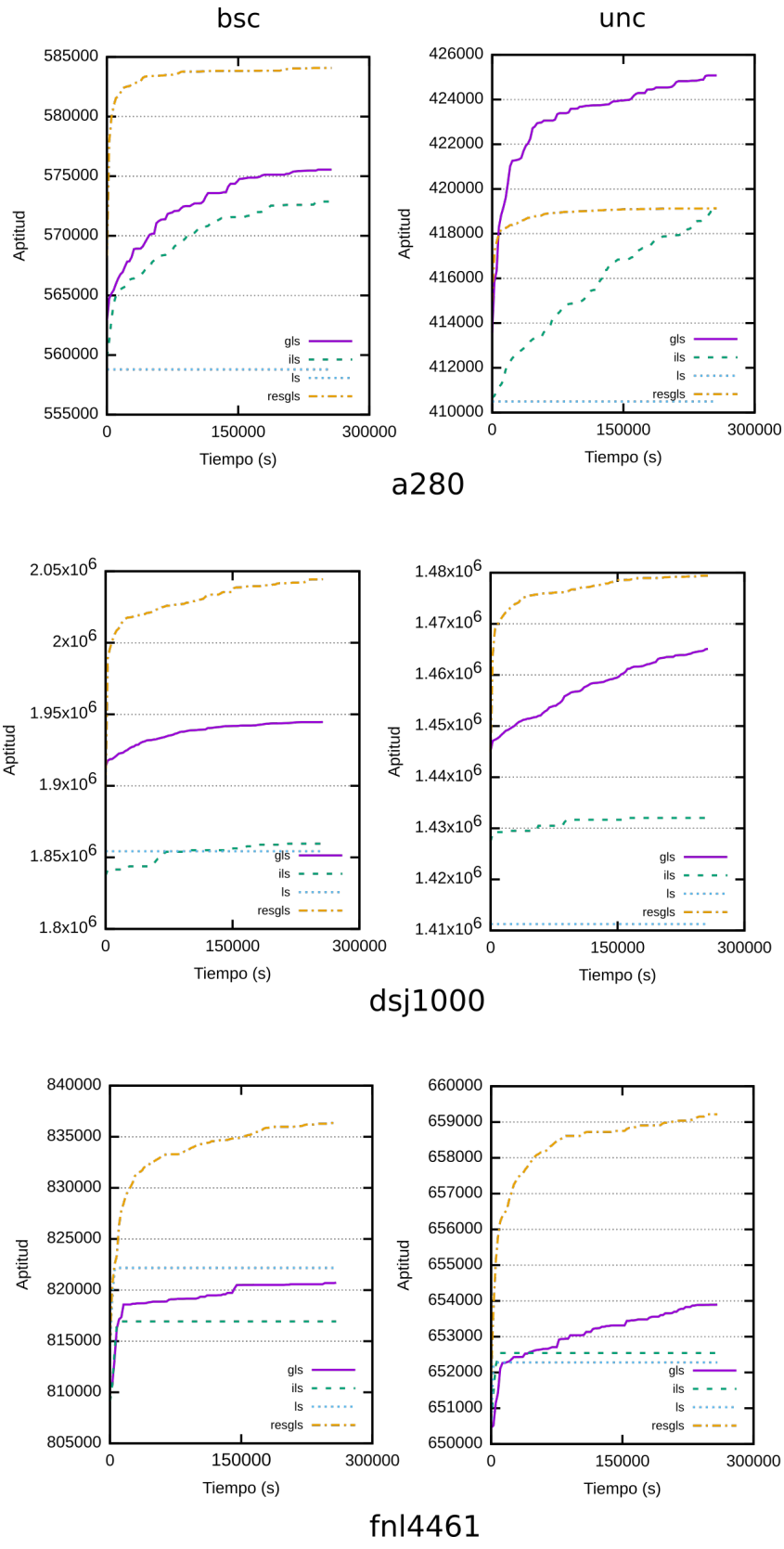
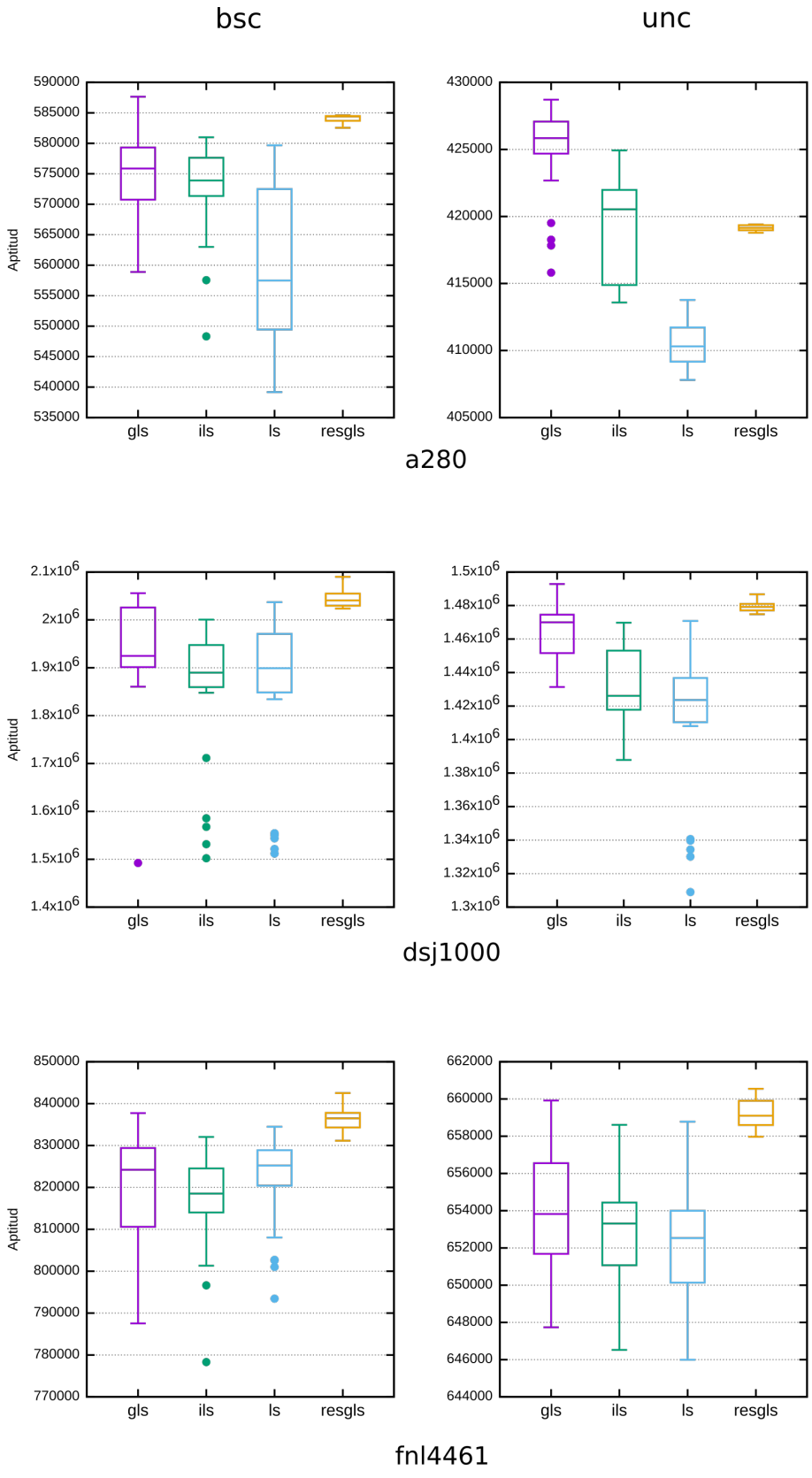


Figura 5.2: Diagrama de caja de la distribución de la aptitud de cada método



5.4. Algoritmo memético

Partiendo de los resultados obtenidos de los algoritmos de trayectoria propuestos, se implementó el Algoritmo {37} (**dma**) el cual emplea la estrategia **gls** como mecanismo de intensificación y un control de diversidad como mecanismo de exploración. Se realizaron ejecuciones a tres días con treinta repeticiones de las instancias: berlin52, kroA100, a280, u574, dsj1000, rl1889 fnl4461. De cada instancia se consideran dos variantes, correlacionadas y no correlacionadas, empleando la máxima capacidad y considerando diez objetos por ciudad. El algoritmo **dma** emplea un tamaño de población de cincuenta individuos, la distancia mínima de diversidad inicial entre soluciones se define como el 5 % del número de aristas. La ejecución del algoritmo tiene una duración de 3 días, con un mínimo de 200 iteraciones y el tiempo mínimo dedicado a cada búsqueda local es de cinco minutos.

En la Tabla 5.13 se muestra la comparación entre el algoritmo memético y las estrategias de trayectoria. Se puede observar que el algoritmo memético es estadísticamente superior a la búsqueda local guiada y presenta un comportamiento similar al algoritmo con reinicio.

Tabla 5.13: Comparación por pares de los métodos de trayectoria y el algoritmo memético propuesto.

	gls			dma			resgls		
	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$
gls		——		1	3	10	3	2	9
dma	10	3	1		——		3	7	4
resgls	9	2	3	4	7	3		——	

Al comparar las mejores soluciones obtenidas por cada algoritmo en la Tabla 5.14, en trece de las catorce instancias el algoritmo **dma** obtiene la mejor solución. Por ello, se espera que la distribución de los resultados tenga una varianza grande.

Tabla 5.14: Mejores soluciones obtenidas por los algoritmos propuestos

	gls	dma	resgls
berlin52–unc	**96519.12**	**96519.12**	**96519.12**
berlin52–bsc	**180307.93**	**180307.93**	**180307.93**
kroA100–unc	**159135.22**	**159135.22**	**159135.22**
kroA100–bsc	**215684.85**	**215684.85**	**215684.85**
a280–unc	428711.09	**429258.04**	419405.75
a280–bsc	587645.97	**592262.44**	584595.94
u574–unc	950998.64	**966297.45**	946365.36
u574–bsc	1346688.59	**1384748.10**	1359403.11
dsj1000–unc	**1492841.65**	1484265.26	1486720.54
dsj1000–bsc	2055726.45	**2110641.62**	2090017.15
rl1889–unc	2812592.55	**3016141.82**	2984448.30
rl1889–bsc	4059234.01	**4322449.89**	4264072.24
fnl4461–unc	659918.34	**661418.65**	660548.80
fnl4461–bsc	837738.96	**850078.47**	842517.63

En la Figura 5.3 se muestra el diagrama de caja de la distribución de aptitud de los tres algoritmos para tres instancias representativas. Observe cómo efectivamente el algoritmo tiene una distribución amplia, pero llega a tener muy buenas ejecuciones, esta variabilidad de los resultados puede deberse a la inicialización utilizada, pues es de esperarse que las soluciones iniciales tengan poca diversidad limitando la capacidad de exploración.

El algoritmo **dma** incorpora los mejores aspectos de la búsqueda local guiada y la estrategia de reinicio implementadas en los algoritmos de trayectoria, lo que se ve reflejado en un aumento en la exactitud en comparación con **gls** y una mejora en la calidad de solución frente a **resgls**.

La Figura 5.4 presenta la evolución en el tiempo del promedio de la mejor aptitud del algoritmo propuesto, para seis instancias. De forma similar que en los métodos de trayectoria, se observa una mejora continua a lo largo de las generaciones, dejando abierta la posibilidad de mejora al alargar el tiempo de ejecución o al mejorar la definición de vecindad.

Para analizar la evolución de la diversidad de la población en el tiempo, en la Figura 5.5 se muestra la diversidad del recorrido Ξ como la diferencia en aristas, para seis problemas representativos.

El algoritmo **dma** implementa un decremento lineal de la diversidad, lo cual se confirma en la evolución de la diversidad, tanto para el recorrido como para la selección de objetos. En un inicio las soluciones son muy similares, pero conforme la búsqueda local guiada logra mover la solución fuera del óptimo local, el control de diversidad favorece diferentes soluciones aumentando la diversidad de las soluciones. Finalmente al disminuir el requisito de distancia, la población va convergiendo a la mejor solución. Se confirma que existe un decremento en el desempeño del control de diversidad conforme se aumenta el tamaño de las instancias. Éste es un comportamiento esperado pues al crecer el espacio de búsqueda, las restricciones de diversidad requieren de un mayor tiempo para la convergencia. Cabe destacar que se utilizó una distancia mínima pequeña y se obtienen valores de diversidad altos, por lo que es necesario un estudio más detallado de la selección de la distancia mínima entre soluciones para el TTP.

Figura 5.3: Diagrama de caja de la aptitud obtenida en las treinta ejecuciones independientes por los tres algoritmos

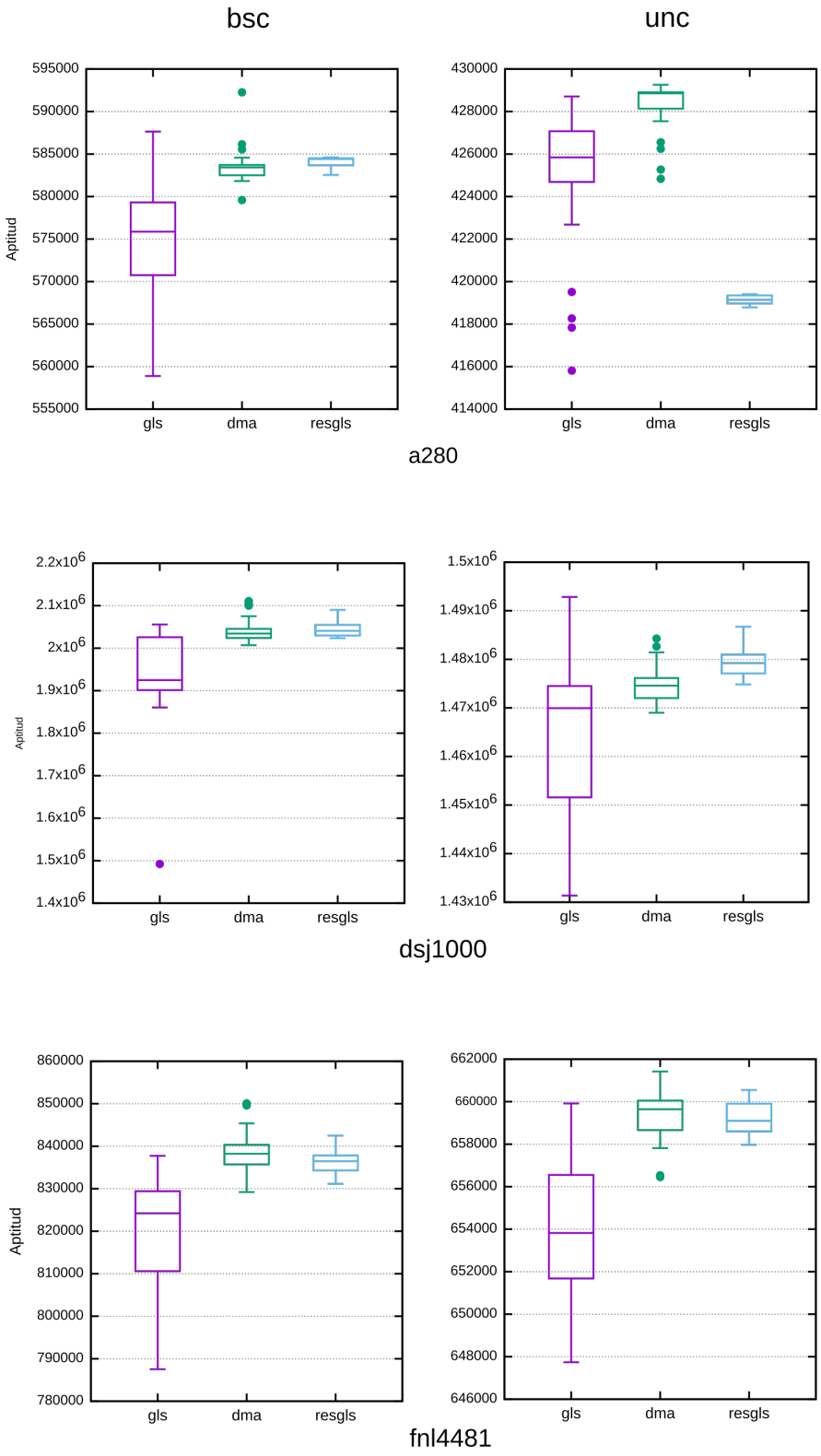


Figura 5.4: Evolución de la mejor aptitud promedio obtenida por el algoritmo *dma* en seis instancias

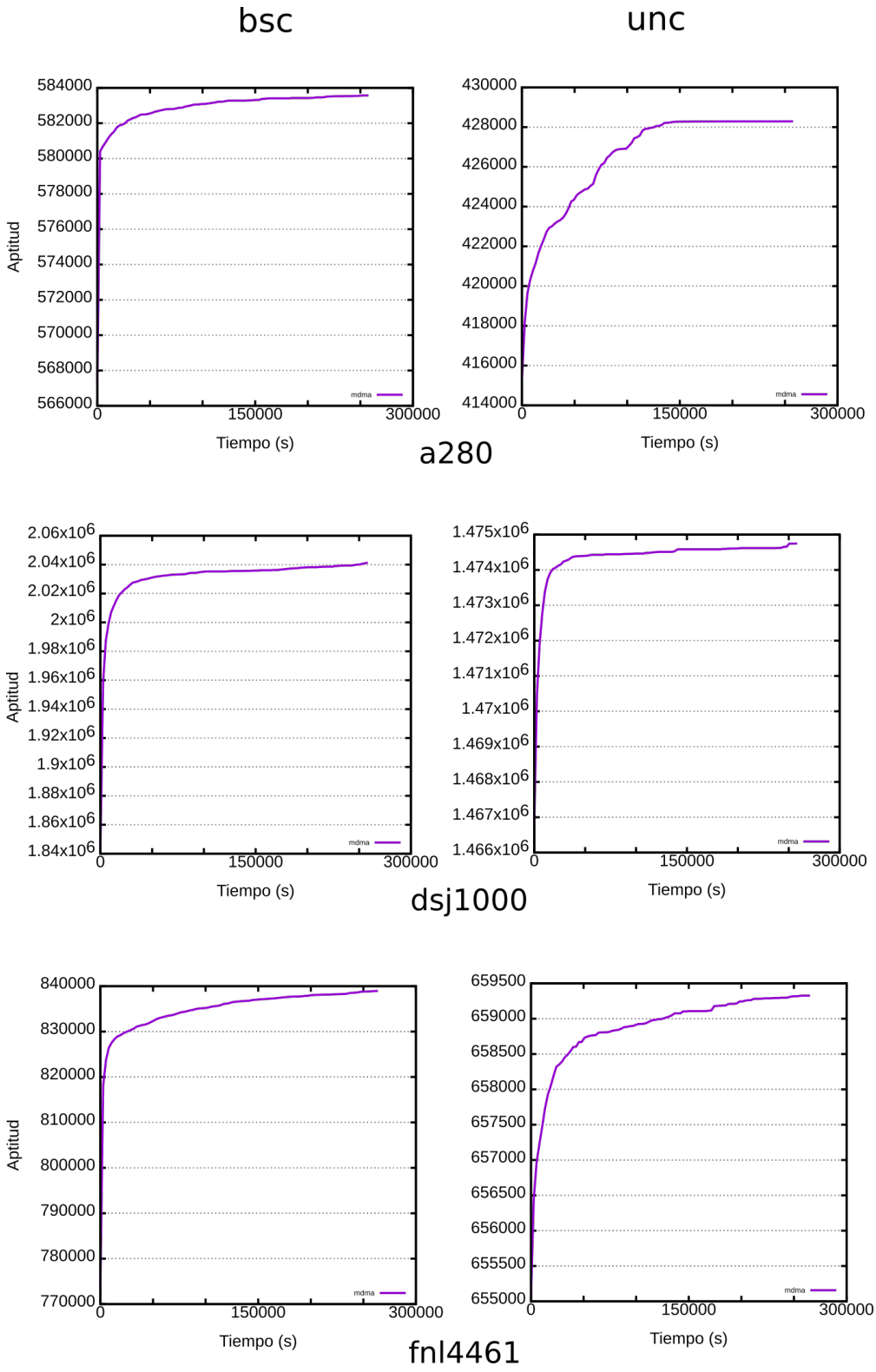
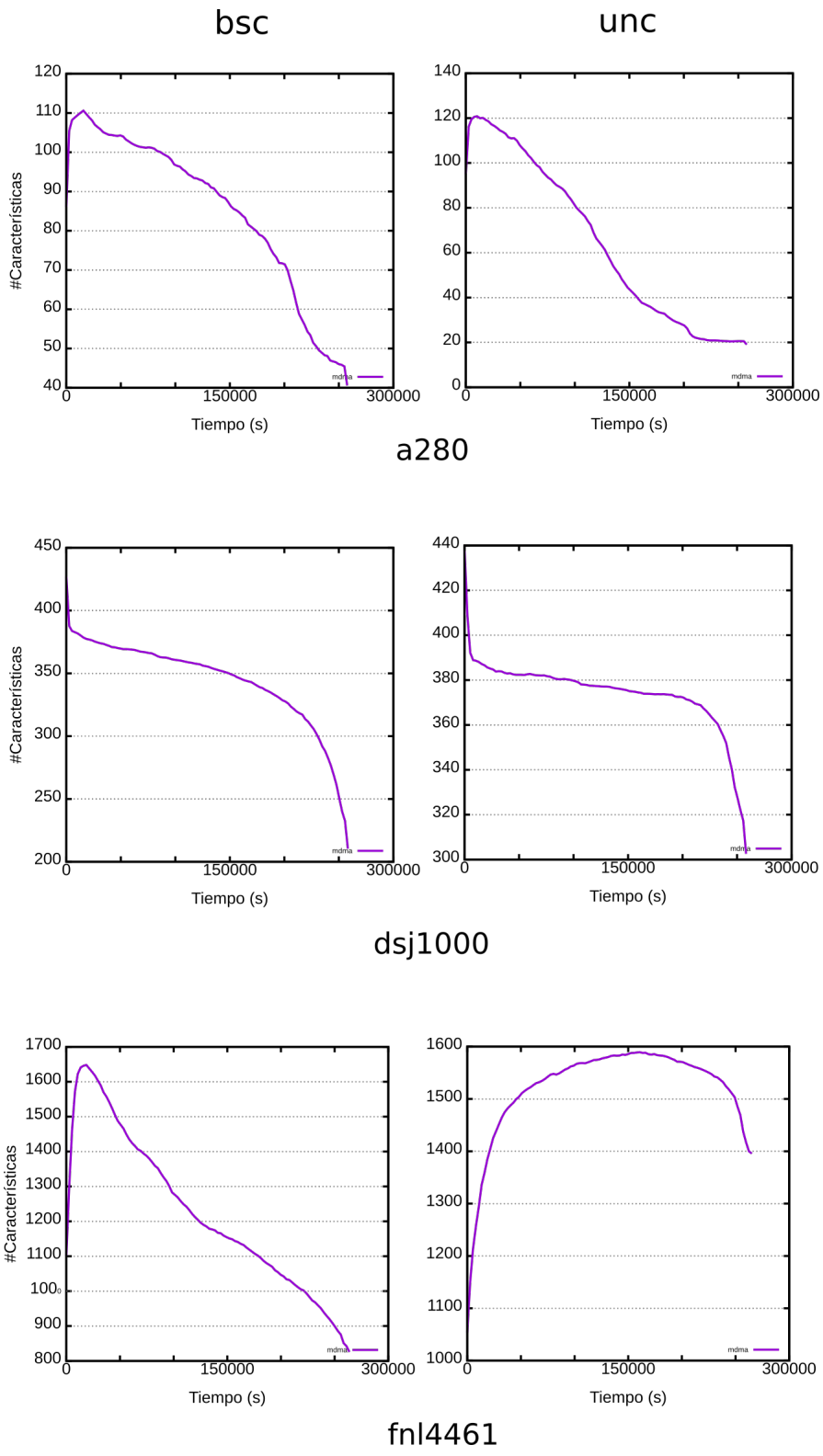


Figura 5.5: Evolución en el tiempo de la distancia promedio entre soluciones de la población del algoritmo *dma* en seis instancias



5.5. Comparación frente al estado del arte

Finalmente se comparan las dos estrategias propuestas frente a dos algoritmos de la literatura. Para su comparación se utilizaron ejecuciones de tres días con treinta repeticiones independientes, para resolver las instancias utilizadas en la sección anterior.

En este análisis solo se consideran los algoritmos CS2SA y MA2B. Inicialmente, se consideró también el algoritmo S5 ya que se han reportado buenos resultados [69]. Sin embargo, tras analizar los resultados obtenidos por la implementación oficial, se encontró un error en la evaluación de la función objetivo, que resultaba en el cálculo de la aptitud de forma incorrecta.

Los algoritmos considerados para realizar la comparación, han mostrado tener un buen desempeño al resolver las diferentes instancias del TTP y ambos pertenecen a las familias de heurísticas implementadas en este trabajo de tesis. El algoritmo CS2SA es un representante de la estrategia CoSolver, mientras MA2B es un algoritmo memético. Los algoritmos fueron descargados de la página web [101]. En específico el algoritmo CS2SA consideraba criterios de paro adicionales al tiempo especificado de ejecución, por ello se modificó ligeramente el código fuente descargado para ignorar estos criterios adicionales.

En la Tabla 5.15 se muestran la comparación de la aptitud promedio obtenido por los algoritmos. También se indica el número de instancias en las que el algoritmo de la fila es estadísticamente superior ($\wedge$), igual ($=$) o inferior ($\vee$) al algoritmo de la columna. Al comparar entre algoritmos de trayectoria (**cs2sa**,**gls** y **resgls**), se observa que la heurística con peor comportamiento es el algoritmo **cs2sa**. Claramente existe un beneficio al incluir estrategias para escapar de óptimos locales, mostrando la incapacidad de exploración generada por el sistema CoSolver. Al comparar los dos algoritmos meméticos **ma2b** y **dma** se observa una clara superioridad estadística al incluir la estrategia **gls**. De manera global los mejores algoritmos son **dma** y **resgls**. Lo que se muestra mediante el puntaje Δ de cada algoritmo, que consiste en la diferencia entre la cantidad total de veces que fue superior e inferior. En base al puntaje Δ se observa que el mejor algoritmo fue **dma** seguido por **resgls**.

Tabla 5.15: Comparación por pares del valor objetivo promedio obtenido en las distintas instancias por los algoritmos

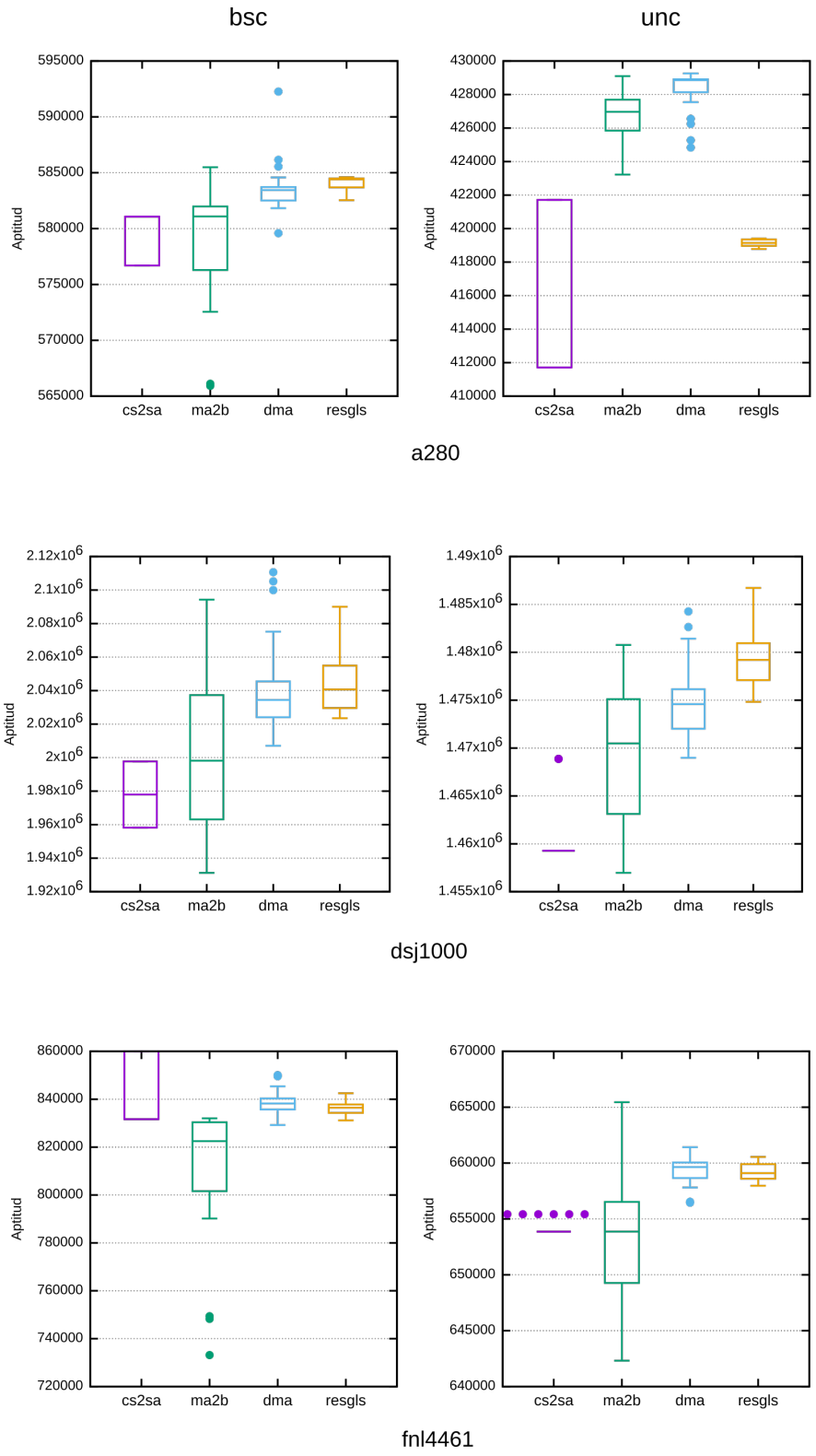
	cs2sa			gls			ma2b			gls			resgls			Δ
	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\wedge$	$=$	$\vee$	$\sum\wedge - \sum\vee$
cs2sa		——		4	3	7	1	4	9	1	3	10	1	3	10	-29
gls	7	3	4		——		4	3	7	1	3	10	3	2	9	-15
ma2b	9	4	1	7	3	4		——		1	3	10	3	2	9	-4
dma	10	3	1	10	3	1	10	3	1		——		3	7	4	26
resgls	10	3	1	9	2	3	9	2	3	4	7	3		——		22

Los diagrama de caja mostrados en la Figura 5.6 para seis instancias representativas muestra que la distribución de soluciones en los algoritmos propuestos tiene menor varianza que los algoritmos de la literatura, atribuído a la función de intensificación de la búsqueda local guiada y la exploración del espacio de búsqueda para escapar de óptimos locales.

Recientemente se ha detectado que existieron errores en las funciones de evaluación de algunos algoritmos. Por ello, los mejores valores conocidos, obtenidos empleando los algoritmos S5, MA2B y CS2SA no son confiables. Por lo anterior se ignorarán en la comparación final. Los Algoritmos MA2B y CS2SA fueron corregidos posterior a su publicación [102], para confirmar que se esta utilizando la nueva versión se realizó una verificación de las soluciones proporcionadas por los algoritmos. De manera que los nuevos resultados reportados en este trabajo fueron calculados empleando la versión corregida de los algoritmos.

La Tabla 5.16 muestra la comparación de los mejores resultados obtenidos por cada algoritmo en las diferentes instancias. Se resaltan en negritas los mejores valores de aptitud obtenidos y se tachan las

Figura 5.6: Diagrama de caja del valor objetivo obtenido en las treinta ejecuciones independientes por los tres algoritmos



soluciones atribuidas a los algoritmos **S5**, **MA2B** y **CS2SA** los cuáles se han decidido ignorar. La columna **mejor** contiene el mejor valor reportado en [69] y disponible para su descarga en [89]. Se observa que los algoritmos propuestos son competitivos encontrando la mejor solución conocida para seis instancias. Se puede apreciar que para ocho instancias el algoritmo **dma** obtiene mejor solución que el algoritmo **ma2b** y en diez instancias supera las mejores soluciones del algoritmo **cs2sa**.

Tabla 5.16: Mejores valores objetivo conocidos para las catorce instancias comparadas

	Mejor	cs2sa	gls	ma2b	dma	resgls
berlin52–unc	**96519.12**	91009.26	**96519.12**	**96519.12**	**96519.12**	**96519.12**
berlin52–bsc	**180307.93**	159251.66	**180307.93**	**180307.93**	**180307.93**	**180307.93**
kroA100–unc	~~156378.00~~	155585.14	**159135.22**	155630.14	**159135.22**	**159135.22**
kroA100–bsc	~~208813.00~~	206779.33	**215684.85**	206779.33	**215684.85**	**215684.85**
a280–unc	~~429099.08~~	421713.65	428711.09	429099.08	**429258.04**	419405.75
a280–bsc	~~604279.00~~	581075.02	587645.97	585491.36	**592262.44**	584595.94
u574–unc	~~969132.43~~	954001.12	950998.64	**967965.05**	966297.45	946365.36
u574–bsc	~~1~~433877.48	1394554.74	1346688.59	**1435384.33**	1384748.10	1359403.11
dsj1000–unc	~~1493275.76~~	1468865.96	**1492841.65**	1480763.41	1484265.26	1486720.54
dsj1000–bsc	~~2093341.27~~	1997740.40	2055726.45	2094279.72	**2110641.62**	2090017.15
rl1889–unc	**3033282.46**	3019998.33	2812592.55	3008694.95	3016141.82	2984448.30
rl1889–bsc	**4498231.29**	4435487.24	4059234.01	4474995.87	4322449.89	4264072.24
fnl4461–unc	~~6744903.00~~	655423.39	659918.34	**665448.80**	661418.65	660548.80
fnl4461–bsc	~~9081826.00~~	**859952.35**	837738.96	832022.61	850078.47	842517.63

Note que las instancias mejoradas por la propuesta de esta tesis son en las instancias pequeñas. Esto indica que el esquema propuesto tiene problemas para escalar, pero deja ver que la calidad de las soluciones que han sido reportadas aún distan de los óptimos. La mejora en las instancias de menor tamaño sugiere que difícilmente se estén alcanzando soluciones de alta calidad en las instancias de mayor tamaño.

Capítulo 6

Conclusiones

El problema del ladrón viajero (TTP) combina dos problemas NP-Complejos en uno. La complejidad del problema se debe a la interacción entre los componentes, lo que limita la aplicación de estrategias clásicas para la solución de los mismos. El entrelazado de los problemas genera un espacio de búsqueda extenso y complejo que requiere de algoritmos específicos para su correcta exploración. En esta tesis se presentan los conceptos necesarios para la formulación del problema del ladrón viajero. Se describen sus componentes, analiza su interdependencia, formula matemáticamente y describen los problemas de referencia existentes, así como sus principales estrategias de solución. Se propone una clasificación de las heurísticas para resolver el TTP y después de una revisión bibliográfica se clasifican los algoritmos existentes. De las estrategias encontradas se seleccionó la familia CoSolver para su estudio, debido a que presenta un buen balance entre calidad de solución, facilidad de implementación y costo computacional.

Se detectó que los algoritmos de la familia CoSolver, carecen de mecanismos para escapar de óptimos locales, ignorando la posible generación de los mismos debido a la interacción entre las fases de optimización del recorrido y las fases de optimización de la selección de objetos. Para mostrar esta característica se implementó una búsqueda local simple y una búsqueda local iterada, ambas pertenecientes a la estrategia CoSolver. Los resultados de la búsqueda local simple permiten inferir que efectivamente se tiene un pobre desempeño, tanto en calidad de solución, como en exploración indicando que la búsqueda mueve poco la solución en el espacio de búsqueda. Por otro lado, los resultados de la búsqueda local iterada muestran una mejora en la exploración, pero la heurística es incapaz de mejorar significativamente la calidad de las soluciones. Esto se atribuye a que después de realizar la perturbación de la solución, rápidamente se encuentra un óptimo local, del cuál la estrategia CoSolver es incapaz de escapar. Al analizarse los diferentes métodos para generar la solución al sub-problema de selección de objetos, se observó que una estrategia simple como la búsqueda local con vecindad Bit-Flip es capaz de encontrar soluciones de alta calidad. Lo que indica que la complejidad de generar una solución para el sub-problema es más sencillo de lo esperado. Esto confirma que la complejidad del TTP se debe a la interdependencia de los sub-problemas.

La principal contribución de este trabajo de tesis es el desarrollo de dos nuevos algoritmos de la familia CoSolver y un algoritmo memético con control de diversidad. La novedad de la propuesta es el tratar de mejor forma los óptimos locales. Basado en los algoritmos de la literatura se propone la estrategia de trayectoria GLS (basada en la búsqueda local guiada) y RESGLS (algoritmo basado en GLS que incorpora reinicios). Esta heurística es única en su clase al enfocarse en el recorrido para mover la solución en el espacio de búsqueda. Los resultados obtenidos muestran que las penalizaciones aplicadas en la etapa de optimización del recorrido dentro del sistema CoSolver, mejoran la calidad de las soluciones, indicando que se permite el escape de los óptimos locales del sistema. Se obtuvieron mejores resultados en las instancias de menor tamaño, pues el alto coste computacional de la búsqueda local guiada disminuye drásticamente la capacidad de exploración al limitar la cantidad de reinicios realizados.

En función de los resultados obtenidos se implementó el algoritmo memético DMA, el cual emplea una control de diversidad como mecanismo de exploración y una búsqueda local guiada para la fase de intensificación. La novedad propuesta en esta heurística es la inclusión de un mecanismo de diversidad en un algoritmo para el TTP, con la intención de mejorar la exploración del espacio de búsqueda y la implementación de un sistema CoSolver que trate de manera explicita los óptimos locales. Los resultados obtenidos son competitivos y estadísticamente similares a la estrategia RESGLS, mejorando la calidad y confiabilidad de las soluciones de las estrategias poblacionales. Al comparar las propuestas frente a algoritmos de la literatura, los resultados muestran los claros beneficios de incorporar la búsqueda local guiada para tratar los óptimos locales e incorporar mecanismos explícitos de control de diversidad. Específicamente se lograron generar nuevas mejores soluciones conocidas para ocho problemas de referencia, siendo un logro importante en un campo donde, debido a la incorporación del TTP en las competencias del congreso GECCO, se han propuesto diversas estrategias. La diversificación de los recorridos explorados proporciona beneficios importantes, contrastando con el objetivo de las técnicas existentes en de la literatura que están enfocadas principalmente a la selección de objetos. De igual forma, los resultados muestran que para los problemas de mayor tamaño, las restricciones de tiempo clásicas de diez minutos son demasiado restrictivas. De hecho, incluso para ejecuciones de tres días, no se aprecia un estancamiento claro, por lo que es posible que al aumentar el tiempo, aún se sigan mejorando los resultados.

Finalmente, a partir de las conclusiones pueden enumerarse a continuación diferentes líneas de investigación futuras que pueden ser exploradas. Primeramente con la intención de reducir los requerimientos computacionales de las propuestas, se planea implementar algunas técnicas de reducción de complejidad como las propuestas en [7,68], lo que a su vez permitirá trabajar con instancias de mayor tamaño. Segundo, el estudio a detalle del comportamiento de la búsqueda local guiada puede permitir encontrar funciones de penalizaciones que guien de mejor forma la solución en el espacio de búsqueda. Tercero, un estudio detallado del impacto del control de diversidad en el TTP, permitirá encontrar las métricas de distancia más adecuadas para explorar el espacio de búsqueda.

El código de las propuestas aquí presentadas se encuentran en [103] y una lista de las mejores soluciones conocidas en [89].

Bibliografía

[1] Z. Michalewicz and D. B. Fogel, *How to solve it: modern heuristics*. Springer Science & Business Media, 2013.

[2] Z. Michalewicz, "Quo vadis, evolutionary computation?" *Advances in Computational Intelligence*, pp. 98–121, 2012.

[3] M. R. Bonyadi, Z. Michalewicz, and L. Barone, "The travelling thief problem: the first step in the transition from theoretical problems to realistic problems," in *Evolutionary Computation (CEC), 2013 IEEE Congress on*. IEEE, 2013, pp. 1037–1044.

[4] S. Polyakovskiy, M. R. Bonyadi, M. Wagner, Z. Michalewicz, and F. Neumann, "A comprehensive benchmark set and heuristics for the traveling thief problem," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '14. New York, NY, USA: ACM, 2014, pp. 477–484. [Online]. Available: http://doi.acm.org/10.1145/2576768.2598249

[5] M. R. Garey and D. S. Johnson, *Computers and intractability, A guide to the Theory of NP-Completeness*. wh freeman New York, 2002, vol. 29.

[6] K. Helsgaun, "An effective implementation of the lin–kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106 – 130, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221799002842

[7] M. El Yafrani and B. Ahiod, "Population-based vs. single-solution heuristics for the travelling thief problem," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ser. GECCO '16. New York, NY, USA: ACM, 2016, pp. 317–324. [Online]. Available: http://doi.acm.org/10.1145/2908812.2908847

[8] Y. Mei, X. Li, and X. Yao, "Improving efficiency of heuristics for the large scale traveling thief problem," in *Simulated Evolution and Learning*. Cham: Springer International Publishing, 2014, pp. 631–643.

[9] C. Voudouris and E. Tsang, "Guided local search," *Handbook of metaheuristics*, pp. 185–218, 2003.

[10] R. Martí, M. G. Resende, and C. C. Ribeiro, "Multi-start methods for combinatorial optimization," *European Journal of Operational Research*, vol. 226, no. 1, pp. 1–8, 2013.

[11] P. Moscato and C. Cotta, "A gentle introduction to memetic algorithms," in *Handbook of metaheuristics*. Springer, 2003, pp. 105–144.

[12] C. Segura, A. Hernández-Aguirre, F. Luna, and E. Alba, "Improving diversity in evolutionary algorithms: New best solutions for frequency assignment," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 4, pp. 539–553, Aug 2017.

[13] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM computing surveys (CSUR)*, vol. 35, no. 3, pp. 268–308, 2003.

[14] E. K. Chong and S. H. Zak, *An introduction to optimization*. John Wiley & Sons, 2013, vol. 76.

[15] B. Korte, J. Vygen, B. Korte, and J. Vygen, *Combinatorial optimization*. Springer, 2012, vol. 2.

[16] J. Hromkovič, *Algorithmics for hard problems: introduction to combinatorial optimization, randomization, approximation, and heuristics*. Springer Science & Business Media, 2013.

[17] P. Siarry, *Metaheuristics*. Springer, 2017.

[18] E.-G. Talbi, *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009, vol. 74.

[19] R. Martí, J. A. Lozano, A. Mendiburu, and L. Hernando, "Multi-start methods," *Handbook of Heuristics*, pp. 1–21, 2016.

[20] H. R. Lourenço, O. C. Martin, and T. Stützle, *Iterated Local Search: Framework and Applications*. Boston, MA: Springer US, 2010, pp. 363–397. [Online]. Available: https://doi.org/10.1007/978-1-4419-1665-5_12

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

[22] D. Bertsimas, J. Tsitsiklis *et al.*, "Simulated annealing," *Statistical science*, vol. 8, no. 1, pp. 10–15, 1993.

[23] F. Glover and M. Laguna, "Tabu search," in *Handbook of combinatorial optimization*. Springer, 2013, pp. 3261–3362.

[24] J.-P. Watson, *An Introduction to Fitness Landscape Analysis and Cost Models for Local Search*. Boston, MA: Springer US, 2010, pp. 599–623. [Online]. Available: https://doi.org/10.1007/978-1-4419-1665-5_20

[25] J. Gu and X. Huang, "Efficient local search with search space smoothing: A case study of the traveling salesman problem (tsp)," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 5, pp. 728–735, 1994.

[26] P. Hansen, N. Mladenović, and J. A. M. Pérez, "Variable neighbourhood search: methods and applications," *Annals of Operations Research*, vol. 175, no. 1, pp. 367–407, 2010.

[27] A. Alsheddy, C. Voudouris, E. P. K. Tsang, and A. Alhindi, *Guided Local Search*. Cham: Springer International Publishing, 2016, pp. 1–37. [Online]. Available: https://doi.org/10.1007/978-3-319-07153-4_2-1

[28] C. Voudouris and E. Tsang, "Guided local search and its application to the traveling salesman problem," *European Journal of Operational Research*, vol. 113, no. 2, pp. 469 – 499, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S037722179800099X

[29] D. Richter, B. Goldengorin, G. Jäger, and P. Molitor, "Improving the efficiency of helsgaun's lin-kernighan heuristic for the symmetric tsp," in *Combinatorial and Algorithmic Aspects of Networking*, J. Janssen and P. Prałat, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 99–111.

[30] H. R. Lourenço, O. C. Martin, and T. Stützle, *Iterated Local Search*. Boston, MA: Springer US, 2003, pp. 320–353. [Online]. Available: https://doi.org/10.1007/0-306-48056-5_11

[31] I. BoussaïD, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Information Sciences*, vol. 237, pp. 82–117, 2013.

[32] D. Beasley, D. R. Bull, and R. R. Martin, "An overview of genetic algorithms: Part 1, fundamentals," *University computing*, vol. 15, no. 2, pp. 56–69, 1993.

[33] G. B. Fogel, *Evolutionary Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 699–708. [Online]. Available: https://doi.org/10.1007/978-3-540-92910-9_23

[34] N. Hansen, *The CMA Evolution Strategy: A Comparing Review*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 75–102. [Online]. Available: https://doi.org/10.1007/3-540-32494-1_4

[35] R. Poli and J. Koza, *Genetic Programming*. Boston, MA: Springer US, 2014, pp. 143–185. [Online]. Available: https://doi.org/10.1007/978-1-4614-6940-7_6

[36] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[37] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," *Applied mathematics and computation*, vol. 214, no. 1, pp. 108–132, 2009.

[38] M. Dorigo and M. Birattari, "Ant colony optimization," in *Encyclopedia of machine learning*. Springer, 2011, pp. 36–39.

[39] X. Yu and M. Gen, *Introduction to evolutionary algorithms*. Springer Science & Business Media, 2010.

[40] J.-M. Renders and S. P. Flasse, "Hybrid methods using genetic algorithms for global optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 2, pp. 243–258, 1996.

[41] T. A. El-Mihoub, A. A. Hopgood, L. Nolle, and A. Battersby, "Hybrid genetic algorithms: A review." *Engineering Letters*, vol. 13, no. 2, pp. 124–137, 2006.

[42] N. Krasnogor and J. Smith, "A memetic algorithm with self-adaptive local search: Tsp as a case study," in *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 2000, pp. 987–994.

[43] G. Squillero and A. Tonda, "Divergence of character and premature convergence: A survey of methodologies for promoting diversity in evolutionary optimization," *Information Sciences*, vol. 329, pp. 782 – 799, 2016, special issue on Discovery Science. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002002551500729X

[44] C. Segura, A. H. Aguirre, S. I. V. Peña, and S. B. Rionda, *The Importance of Proper Diversity Management in Evolutionary Algorithms for Combinatorial Optimization*. Cham: Springer International Publishing, 2017, pp. 121–148. [Online]. Available: https://doi.org/10.1007/978-3-319-44003-3_6

[45] D. Whitley, S. Rana, and R. B. Heckendorn, "The island model genetic algorithm: On separability, population size and convergence," *Journal of Computing and Information Technology*, vol. 7, no. 1, pp. 33–47, 1999.

[46] B. Sareni and L. Krahenbuhl, "Fitness sharing and niching methods revisited," *IEEE transactions on Evolutionary Computation*, vol. 2, no. 3, pp. 97–106, 1998.

[47] F.-A. Fortin and M. Parizeau, "Revisiting the nsga-ii crowding-distance computation," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013, pp. 623–630.

[48] G. Harik, "Finding multimodal solutions using restricted tournament selection," in *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1995, pp. 24–31.

[49] L. J. Eshelman, "The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination," in *Foundations of Genetic Algorithms*, G. J. RAWLINS, Ed. Elsevier, 1991, vol. 1, pp. 265 – 283. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780080506845500203

[50] R. W. Morrison, *Diversity Measurement*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 25–52. [Online]. Available: https://doi.org/10.1007/978-3-662-06560-0_4

[51] M. Bellmore and G. L. Nemhauser, "The traveling salesman problem: a survey," *Operations Research*, vol. 16, no. 3, pp. 538–558, 1968.

[52] H. M. Salkin and C. A. De Kluyver, "The knapsack problem: a survey," *Naval Research Logistics (NRL)*, vol. 22, no. 1, pp. 127–144, 1975.

[53] G. Gutin and A. P. Punnen, *The traveling salesman problem and its variations*. Springer Science & Business Media, 2006, vol. 12.

[54] H. Keller, P. Ulrich, and P. David, *Knapsack Problems*. Springer Science & Business Media, 2004.

[55] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[56] W. Cook. (2018, jun) Traveling salesman problem. [Online]. Available: http://www.math.uwaterloo.ca/tsp/concorde/index.html

[57] C. Rego, D. Gamboa, F. Glover, and C. Osterman, "Traveling salesman problem heuristics: Leading methods, implementations and latest advances," *European Journal of Operational Research*, vol. 211, no. 3, pp. 427 – 441, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221710006065

[58] K. Helsgaun. (2018, jun) Lkh version 2.0.7. [Online]. Available: http://www.akira.ruc.dk/~keld/research/LKH/

[59] C. Segura, S. Botello Rionda, A. Hernández Aguirre, and S. I. Valdez Peña, "A novel diversity-based evolutionary algorithm for the traveling salesman problem," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '15. New York, NY, USA: ACM, 2015, pp. 489–496. [Online]. Available: http://doi.acm.org/10.1145/2739480.2754802

[60] A. Umbarkar and P. Sheth, "Crossover operators in genetic algorithms: A review." *ICTACT journal on soft computing*, vol. 6, no. 1, 2015.

[61] L. D. Whitley, T. Starkweather, and D. Fuquay, "Scheduling problems and traveling salesmen: The genetic edge recombination operator," in *ICGA*, 1989, pp. 133–140.

[62] D. Pisinger, "Where are the hard knapsack problems?" *Computers & Operations Research*, vol. 32, no. 9, pp. 2271 – 2284, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S030505480400036X

[63] A. Fréville, "The multidimensional 0–1 knapsack problem: An overview," *European Journal of Operational Research*, vol. 155, no. 1, pp. 1 – 21, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221703002741

[64] L. Thibaut and T. Jacques, "The multiobjective multidimensional knapsack problem: a survey and a new approach," *International Transactions in Operational Research*, vol. 19, no. 4, pp. 495–520, 2012. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1475-3995.2011.00840.x

[65] R. E. Bellman and S. E. Dreyfus, *Applied dynamic programming*. Princeton university press, 2015.

[66] T. Y. Lim, M. A. Al-Betar, and A. T. Khader, "Taming the 0/1 knapsack problem with monogamous pairs genetic algorithm," *Expert Systems with Applications*, vol. 54, pp. 241 – 250, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417416300197

[67] S. Martello, D. Pisinger, and P. Toth, "Dynamic programming and strong bounds for the 0-1 knapsack problem," *Management Science*, vol. 45, no. 3, pp. 414–424, 1999.

[68] M. E. Yafrani and B. Ahiod, "A local search based approach for solving the travelling thief problem: The pros and cons," *Applied Soft Computing*, vol. 52, pp. 795 – 804, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494616305063

[69] M. Wagner, M. Lindauer, M. Mısır, S. Nallaperuma, and F. Hutter, "A case study of algorithm selection for the traveling thief problem," *Journal of Heuristics*, vol. 24, no. 3, pp. 295–320, Jun 2018. [Online]. Available: https://doi.org/10.1007/s10732-017-9328-y

[70] S. P. Frank Neumann. (2018, nov) Travelling thief problem: a transition from theoretical problems to realistic problems. [Online]. Available: https://cs.adelaide.edu.au/~optlog/research/combinatorial.php

[71] M. R. Bonyadi, Z. Michalewicz, M. R. Przybylek, and A. Wierzbicki, "Socially inspired algorithms for the travelling thief problem," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '14. New York, NY, USA: ACM, 2014, pp. 421–428. [Online]. Available: http://doi.acm.org/10.1145/2576768.2598367

[72] M. E. Yafrani and B. Ahiod, "Cosolver2b: An efficient local search heuristic for the travelling thief problem," in *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*, Nov 2015, pp. 1–5.

[73] N. Lourenço, F. B. Pereira, and E. Costa, "An evolutionary approach to the full optimization of the traveling thief problem," in *Evolutionary Computation in Combinatorial Optimization*. Springer International Publishing, 2016, pp. 34–45.

[74] B. C. Gupta and V. P. Prakash, "Greedy heuristics for the travelling thief problem," in *2015 39th National Systems Conference (NSC)*, Dec 2015, pp. 1–5.

[75] Y. Mei, X. Li, F. Salim, and X. Yao, "Heuristic evolution with genetic programming for traveling thief problem," in *2015 IEEE Congress on Evolutionary Computation (CEC)*, May 2015, pp. 2753–2760.

[76] M. Wagner, "Stealing items more efficiently with ants: A swarm intelligence approach to the travelling thief problem," in *Swarm Intelligence*. Cham: Springer International Publishing, 2016, pp. 273–281.

[77] Y. Mei, X. Li, and X. Yao, "On investigation of interdependence between sub-problems of the travelling thief problem," *Soft Computing*, vol. 20, no. 1, pp. 157–172, Jan 2016. [Online]. Available: https://doi.org/10.1007/s00500-014-1487-2

[78] H. Faulkner, S. Polyakovskiy, T. Schultz, and M. Wagner, "Approximate approaches to the traveling thief problem," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '15. New York, NY, USA: ACM, 2015, pp. 385–392. [Online]. Available: http://doi.acm.org/10.1145/2739480.2754716

[79] M. E. Yafrani and B. Ahiod, "Efficiently solving the traveling thief problem using hill climbing and simulated annealing," *Information Sciences*, vol. 432, pp. 231 – 244, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025517311325

[80] J. Wu, M. Wagner, S. Polyakovskiy, and F. Neumann, "Exact approaches for the travelling thief problem," in *Simulated Evolution and Learning*. Cham: Springer International Publishing, 2017, pp. 110–121.

[81] M. El Yafrani, M. Martins, M. Wagner, B. Ahiod, M. Delgado, and R. Lüders, "A hyperheuristic approach based on low-level heuristics for the travelling thief problem," *Genetic Programming and Evolvable Machines*, vol. 19, no. 1, pp. 121–150, Jun 2018. [Online]. Available: https://doi.org/10.1007/s10710-017-9308-x

[82] M. S. R. Martins, M. El Yafrani, M. R. B. S. Delgado, M. Wagner, B. Ahiod, and R. Lüders, "Hseda: A heuristic selection approach based on estimation of distribution algorithm for the travelling thief problem," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '17. New York, NY, USA: ACM, 2017, pp. 361–368. [Online]. Available: http://doi.acm.org/10.1145/3071178.3071235

[83] J. Karder, A. Beham, S. Wagner, and M. Affenzeller, "Solving the traveling thief problem using orchestration in optimization networks," in *Computer Aided Systems Theory – EUROCAST 2017*. Cham: Springer International Publishing, 2018, pp. 307–315.

[84] D. K. S. Vieira, G. L. Soares, J. A. Vasconcelos, and M. H. S. Mendes, "A genetic algorithm for multi-component optimization problems: The case of the travelling thief problem," in *Evolutionary Computation in Combinatorial Optimization*, B. Hu and M. López-Ibáñez, Eds. Cham: Springer International Publishing, 2017, pp. 18–29.

[85] J. Blank, K. Deb, and S. Mostaghim, "Solving the bi-objective traveling thief problem with multi-objective evolutionary algorithms," in *Evolutionary Multi-Criterion Optimization*. Cham: Springer International Publishing, 2017, pp. 46–60.

[86] J. Wu, S. Polyakovskiy, M. Wagner, and F. Neumann, "Evolutionary computation plus dynamic programming for the bi-objective travelling thief problem," *arXiv preprint arXiv:1802.02434*, 2018.

[87] R. Nieto-Fuentes, C. Segura, and S. I. Valdez, "A guided local search approach for the travelling thief problem," in *Evolutionary Computation (CEC), 2018 IEEE Congress on*, 2018.

[88] G. Reinelt, "Tsplib—a traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991. [Online]. Available: https://doi.org/10.1287/ijoc.3.4.376

[89] R. Nieto-Fuentes. (2018, apr) Travelling thief problem best known solutions. [Online]. Available: https://gitlab.com/NIFR91/Travelling-Thief-Problem

[90] S. Polyakovskiy, M. R. Bonyadi, M. Wagner, Z. Michalewicz, and F. Neumann. (2018, jul) Ttplib. [Online]. Available: https://cs.adelaide.edu.au/~optlog/CEC2014COMP_InstancesNew/

[91] D. Applegate, W. J. Cook, and A. Rohe, "Chained lin-kernighan for large traveling salesman problems. informs journal on computing, 15, 82-92," *INFORMS Journal on Computing*, vol. 15, pp. 82–92, 02 2003.

[92] T. S. Hayden Faulkner, Sergey Polyakovskiy and M. Wagner. (2018, jan) 2015geccocodefordistribution. [Online]. Available: https://cs.adelaide.edu.au/~optlog/research/ttp/2015gecco-codeForDistribution.zip

[93] D. T. Lee and B. J. Schachter, "Two algorithms for constructing a delaunay triangulation," *International Journal of Computer & Information Sciences*, vol. 9, no. 3, pp. 219–242, Jun 1980. [Online]. Available: https://doi.org/10.1007/BF00977785

[94] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer, "Evolution algorithms in combinatorial optimization," *Parallel Computing*, vol. 7, no. 1, pp. 65 – 85, 1988. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0167819188900981

[95] F. Neumann, S. Polyakovskiy, M. Skutella, L. Stougie, and J. Wu, "A fully polynomial time approximation scheme for packing while traveling," *CoRR*, vol. abs/1702.05217, 2017. [Online]. Available: http://arxiv.org/abs/1702.05217

[96] M. E. Yafrani, M. S. R. Martins, M. E. Krari, M. Wagner, M. R. B. S. Delgado, B. Ahiod, and R. Lüders, "A fitness landscape analysis of the travelling thief problem," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '18. New York, NY, USA: ACM, 2018, pp. 277–284. [Online]. Available: http://doi.acm.org/10.1145/3205455.3205537

[97] M. W. Wanru Gao, Sergey Polyakovskiy. (2018, jan) 3rd competition on the optimisation of problems with multiple interdependent components. [Online]. Available: https://cs.adelaide.edu.au/~optlog/TTP2017Comp/

[98] J. J. Durillo, A. J. Nebro, C. A. C. Coello, J. García-Nieto, F. Luna, and E. Alba, "A study of multiobjective metaheuristics when solving parameter scalable problems," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 4, pp. 618–635, 2010.

[99] M. Cortina-Borja, "Handbook of parametric and nonparametric statistical procedures," 2012.

[100] K.-T. Mak and A. J. Morton, "Distances between traveling salesman tours," *Discrete Applied Mathematics*, vol. 58, no. 3, pp. 281–291, Apr. 1995. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/0166218X93E0115F

[101] M. E. Yafrani. (2018, jan) Ttplab. [Online]. Available: https://github.com/yafrani/ttplab

[102] Yafrani. (2018, nov) Corrected metric issue (use ceil distance) commit. [Online]. Available: https://github.com/yafrani/ttplab/commit/2eb058f298ab335381113249e7d0c6664824e7db

[103] R. Nieto-Fuentes. (2018, apr) Ttp solver. [Online]. Available: https://gitlab.com/NIFR91/TTP-solver