# Manual Código TSP-Framework

# Índice

1 Distribución del código y los paquetes ..... 4
1.1 Paquete General ..... 4
1.2 Sub-Paquete de Algoritmos ..... 5
1.3 Sub-Paquete de Herramientas ..... 5
2 Como agregar un nuevo método de búsqueda ..... 6
2.1 Agregar las opciones ..... 6
2.2 Crear el módulo ..... 8
2.3 Agregar a modulo principal ..... 12
2.4 Implementar en la Interfaz Gráfica (Bonus) ..... 14
3 Ubicación métodos de interés ..... 18
3.1 Métodos para manejar la población de Algoritmo Genético ..... 18

# Figuras

**Figura 1.1: Distribución de los paquetes** ..... 4
**Figura 2.1: enum metaheurísticas** ..... 6
**Figura 2.2: Definir Argumentos** ..... 7
**Figura 2.3: Arg. nueva metaheurística** ..... 7
**Figura 2.4: Procesar Argumentos** ..... 8
**Figura 2.5: Constructor de clase** ..... 9
**Figura 2.6: Guardar Trayectoria** ..... 9
**Figura 2.7: métodos de archivos** ..... 10
**Figura 2.8: método para visualizar** ..... 10
**Figura 2.9: Actualizar log** ..... 11
**Figura 2.10: Mostrar mejor solución** ..... 11
**Figura 2.11: Imports __init__.py** ..... 12
**Figura 2.12: importar método en modulo principal** ..... 12
**Figura 2.13: Crear instancia del algoritmo** ..... 13
**Figura 2.14: Nombre de la instancia** ..... 13

# 1 Distribución del código y los paquetes

La distribución interna del código fuente está compuesto por diferentes paquetes como se muestra en la siguiente imagen:

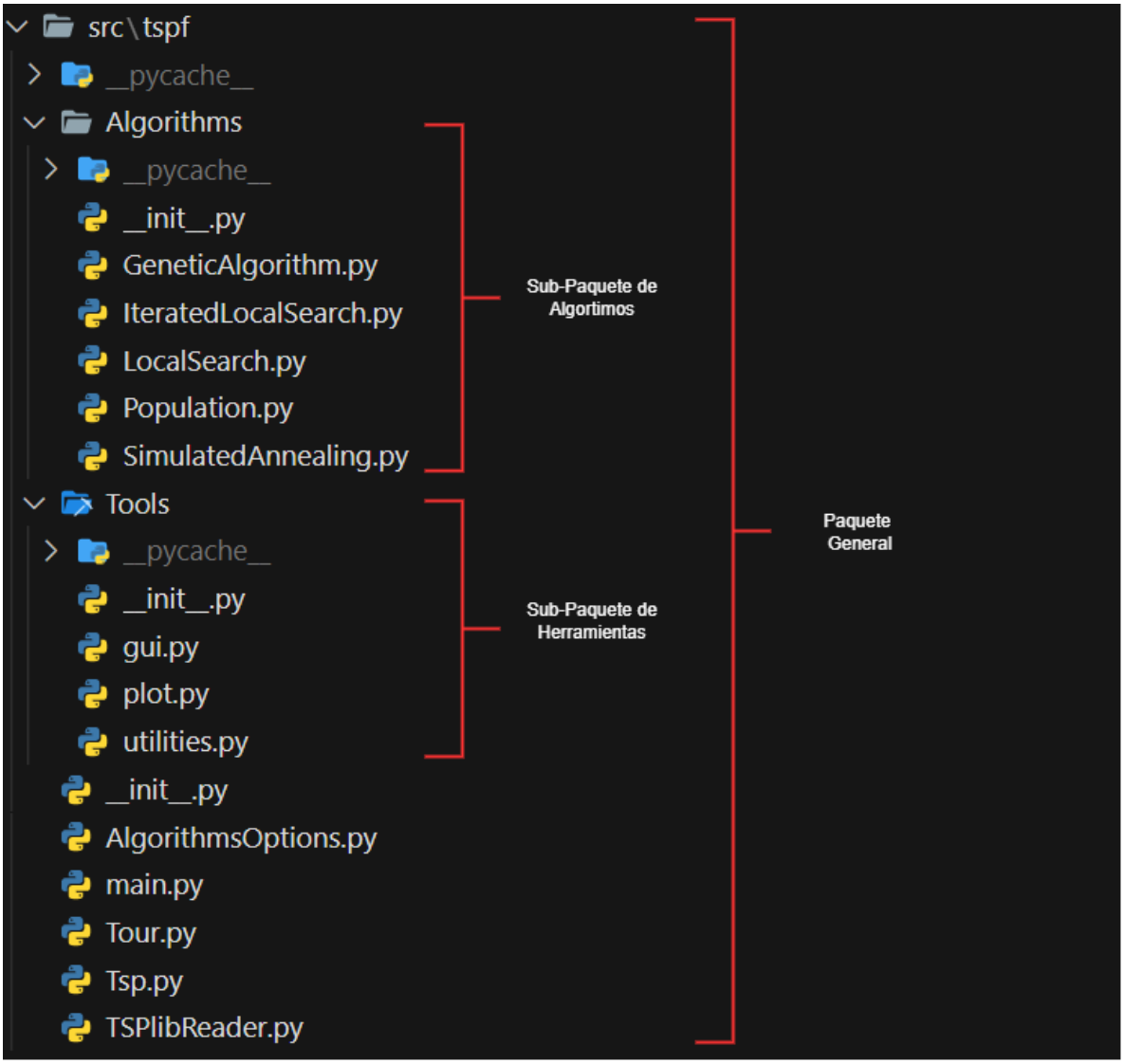


*Figura 1.1: Distribución de los paquetes*

## 1.1 Paquete General

Este paquete contiene los módulos generales para gestionar el problema TSP y sus métodos de búsqueda:

- **__init__.py:** Indicador del paquete que contiene todos los imports compartidos por este.
- **AlgorithmsOptions.py:** Modulo que contiene la clase que administra todas las opciones del framework, así como también la de los algoritmos de métodos de búsqueda.

- **main.py:** Modulo que contiene la función que crea las instancias de las clases y ejecuta los métodos de búsqueda.
- **Tour.py:** Modulo que contiene la clase que representa un recorrido del problema.
- **Tsp.py:** Modulo que contiene la clase que representa el problema TSP.
- **TSPlibReader.py**: Modulo que contiene la clase que ejecuta la lectura de una instancia de problema TSP y genera las matrices de distancias.

## 1.2 Sub-Paquete de Algoritmos

Este paquete contiene todos los algoritmos de métodos de búsqueda implementados y sus complementos:

- **__init__.py:** Indicador del paquete que contiene todos los imports compartidos por este.
- **GeneticAlgorithm.py:** Modulo con la clase que implementa el método de búsqueda de Algoritmo Genético.
- **IteratedLocalSearch.py:** Modulo con la clase que implementa el método de búsqueda local iterativo.
- **LocalSearch.py:** Modulo con la clase que implementa el método de búsqueda local.
- **Population.py:** Modulo con la clase que implementa todos los métodos para manipular las poblaciones generadas por algoritmo genético.
- **SimulatedAnnealing.py:** Modulo con la clase que implementa el método de búsqueda de Simulated Annealing.

## 1.3 Sub-Paquete de Herramientas

Este paquete contiene todas las herramientas y utilidades utilizadas por el framework.

- **__init__.py:** Indicador del paquete que contiene todos los imports compartidos por este.
- **utilities.py:** Modulo distintas utilidades utilizadas por los demás módulos.
- **plot.py:** Modulo encargado de generar y mostrar los distintos gráficos utilizados como los generados en la trayectoria de las soluciones.
- **gui.py:** Modulo encargado de generar y mostrar la interfaz gráfica de usuario.

# 2 Como agregar un nuevo método de búsqueda

Para agregar un nuevo método de búsqueda se deben seguir los siguientes pasos:

## 2.1 Agregar las opciones

Para agregar las opciones del nuevo método de búsqueda nos dirigimos a el módulo que contiene las contiene: **AlgorithmsOptions.py**, dentro de este agregamos las clases enum que utilizará el método:

1. Primero agregamos la identificación del método de búsqueda como tipo de metaheurística y los otros tipos de clases enum que estimemos necesarios para facilitar las opciones:

```
class MHType(Enum):
    """Tipos de Metaheristicas disponibles
    SA: Simulated Annealing
    GA: Genetic Algorithm
    LS: Local Search
    ILS: Iterated Local Search
    """
    SA = 'SA'
    GA = 'GA'
    LS = 'LS'
    ILS = 'ILS'
```

**Figura 2.1: enum metaheurísticas**

2. Luego dentro e la clase principal: **AlgorithmsOptions,** agregamos los atributos de clase con las opciones del algoritmo.
3. Luego en el método **readOptions (aprox. Línea 241)** agregamos los argumentos que utilizará este método:

Definir los argumentos que utilizará en la terminal

```
        # Definir argumentos de Algoritmo Genetico
        parser.add_argument("-p", "--psize", help="Cantidad de individuos
        parser.add_argument("-o", "--osize", help="Cantidad de hijos a ger
        parser.add_argument("-ps", "--pselection", help="Operador de selec
        parser.add_argument("-cr", "--crossover", help="Operador de crossc
        parser.add_argument("-mu", "--mutation", help="Operador de mutacic
        parser.add_argument("-mp", "--mprobability", help="Probabilidad de
        parser.add_argument("-gs", "--gselection", help="Operador de selec
        parser.add_argument("-g", "--gstrategy", help="Estrategia de selec

        # Definir argumentos de Local Search e Iterated Local Search
        parser.add_argument("-b", "--best", help="Ejecuta Local Search en
        parser.add_argument("-per", "--perturbation", help="Tipo de pertur
        parser.add_argument("-np", "--nperturbations", help="Cantidad de p

        # Procesar argumentos
```

**Figura 2.2: Definir Argumentos**

Procesar nuevo tipo de metaheurística en el método **argsGeneral (aprox. Línea 364):**

```
# Seleccion de Metaheuristica
if (args.metaheuristic or 'metaheuristic' in kwargs):
    val = args.metaheuristic.upper() if args.metaheuristic else kwargs['metaheuristic'].upper()
    if (val == 'SA'):
        self.metaheuristic = MHType.SA
    elif (val == 'GA'):
        self.metaheuristic = MHType.GA
    elif (val == 'LS'):
        self.metaheuristic = MHType.LS
    elif (val == 'ILS'):
        self.metaheuristic = MHType.ILS
    else: print(f"{bcolors.FAIL}Error: Metaheuristica no reconocida (-mh | --metaheristic) {bcolors.ENDC}")
```

**Figura 2.3: Arg. nueva metaheurística**

Agregar nuevo método que procese los argumentos y valide que no hay errores:

```
# Procesar argumentos
args = parser.parse_args()

# Procesar argumentos generales
self.argsGeneral(args, kwargs)

if self.metaheuristic == MHType.SA:
    # Procesar argumentos de Simulated Annealing
    self.argsSA(args, kwargs)
    # Validar logica de opciones
    if self.errorsSA():
        exit()
elif self.metaheuristic == MHType.GA:
    # Procesar argumentos de Algoritmo Genetico
    self.argsGA(args, kwargs)
    # Validar logica de opciones
    if self.errorsGA():
        exit()
elif self.metaheuristic == MHType.LS or self.metaheuristic == MHType.ILS:
    # Procesar argumentos de Local Search e Iterated Local Search
    self.argsLS(args, kwargs)
```

**Figura 2.4: Procesar Argumentos**

4. Agregar las opciones para mostrar las opciones en el método **printOptions (aprox. Línea 606)**

## 2.2 Crear el módulo

Para agregar el nuevo método de búsqueda nos dirigimos a el paquete que contiene los algoritmos, y dentro de este:

1. Creamos el nuevo módulo que contenga la **clase** en el **paquete de algoritmos.** Esta clase debe tener un constructor que reciba una instancia de un objeto de opciones y otra instancia de un objeto con el problema TSP como se ve en este ejemplo:

```python
def __init__(self, options: AlgorithmsOptions = None, problem: Tsp = None) -> None:

    # Atributos de instancia
    self.problem: Tsp # Problema TSP

    self.cooling: CoolingType # Esquema de enfriamiento

    self.move_type: TSPMove # Tipo de movimiento

    self.alpha = 0.0 # Parametro alfa para el esquema de enfriamiento geometrico

    self.best_tour: Tour # Mejor tour

    self.evaluations = 1 # numero de evaluaciones

    self.total_time = 0.0 # tiempo de ejecucion de Simulated Annealing

    self.options: AlgorithmsOptions # Opciones

    self.trajectory = [] # lista con la trayectoria de la solucion
```

Figura 2.5: Constructor de clase

Como se ve en la figura, el método debe contar con sus opciones como **atributo de instancia,** además de la mejor solución, de tipo Tour que guardará la mejor solución y se utilizará como punto de comparación para obtener mejores soluciones a través de las iteraciones del algoritmo. También, para concretar la trayectoria se debe tener una lista que guarde la **trayectoria.** Esta trayectoria, definida en el módulo **utilities.py** del paquete de herramientas (Tools), se utilizará para generar la visualización de esta y debería guardarse por cada vez que el método de búsqueda implementado encuentre una mejor solución de la que ya tenemos. Para utilizarla debemos ir añadiendo a la lista instancias del objeto **Trajectory** como se ve a continuación:

```python
# Guardar Trayectoria
self.trajectory.append( Trajectory(
                        tour=tour.current.copy(),
                        cost=tour.cost,
                        iterations=self.evaluations,
                        evaluations=self.evaluations) )
```

Figura 2.6: Guardar Trayectoria

2. Implementar estos métodos para guardar los archivos de solución y trayectoria (se copiar desde otro algoritmo). Ver modulo **utilities.py** en el paquete de herramientas (Tools) si desea modificar.

```python
def printSolFile(self, outputSol: str) -> None:
    """ Guarda la solucion en archivo de texto"""
    utilities.printSolToFile(outputSol, self.best_tour.current)

def printTraFile(self, outputTra: str) -> None:
    """ Guarda la trayectoria de la solucion en archivo de texto"""
    utilities.printTraToFile(outputTra, self.trajectory)
```

Figura 2.7: métodos de archivos

3. Implementar método para la visualización de la trayectoria (se copiar desde otro algoritmo). Ver modulo **plot.py** en el paquete de herramientas (Tools) si desea modificar.

```python
def visualize(self) -> None:
    """ Visualiza la trayectoria de la solucion """
    plot.Graph.replit = self.options.replit
    plot.Graph.trajectory = self.trajectory

    plot.show(self.options.gui)
```

Figura 2.8: método para visualizar

4. (OPCIONAL) Implementar método **updateLog** para guardar log con las mejores soluciones utilizando este método de búsqueda. Se recomienda consultar uno ya implementado de otro algoritmo para guiarse.

```
def updateLog(self) -> None:
    """ Actualiza el registro de mejores soluciones con todas las caracteris
    # crea la carpeta en caso de que no exista (python 3.5+)
    Path("log/").mkdir(exist_ok=True)
    logFile = "log/SAlog.csv"
    # usar el archivo en modo append
    with open(logFile, "a", newline="\n") as csvfile:

        print(f"{bcolors.OKGREEN}\nActualizando log con mejores soluciones e
        # Headers
        fields = ["solution","cost","instance","date","alpha","t0","tmin",
                  "cooling","seed","move","max_evaluations","max_time","initi
        writer = csv.DictWriter(csvfile, delimiter=';', fieldnames=fields)
        # Si la posicion de el archivo es cero se escriben los headers
        if not csvfile.tell():
            writer.writeheader()
```

Figura 2.9: Actualizar log

5. Implementar método **print_best_solution** que muestre la mejor solución (eliminar **updateLog** en caso de no implementarlo).

```
def print_best_solution(self) -> None:
    """ Escribir la mejor solucion """
    self.updateLog()
    print()
    print(f"\t\t{bcolors.UNDERLINE}Mejor Solucion Encont
    self.best_tour.printSol(True)
    print(f"{bcolors.BOLD}Total de evaluaciones:{bcolors
    print(f"{bcolors.BOLD}Tiempo total de busqueda con
```

Figura 2.10: Mostrar mejor solución

6. Importar en __init__.py dentro del paquete, agregar el import siguiendo el patrón:

```
import csv
import math
from os import path
from datetime import datetime
from pathlib import Path
import statistics as stats
from timeit import default_timer as timer
from prettytable import PrettyTable

from src.tspf.Algorithms.Population import Population
from src.tspf.Algorithms.GeneticAlgorithm import GeneticAlgorithm
from src.tspf.Algorithms.SimulatedAnnealing import SimulatedAnnealing
from src.tspf.Algorithms.LocalSearch import LocalSearch
from src.tspf.Algorithms.IteratedLocalSearch import IteratedLocalSearch
```

*Figura 2.11: Imports __init__.py*

## 2.3 Agregar a modulo principal

Una vez completados los pasos anteriores podemos agregar el método de búsqueda a el módulo principal **main.py** en su función **main** siguiendo los pasos:

1. Importar desde el paquete de algoritmos:

```
from .Algorithms import GeneticAlgorithm, SimulatedAnnealing, LocalSearch, IteratedLocalSearch, timer
from .Tools import bcolors, gui
from . import sys, os, AlgorithmsOptions, MHType, Tsp, Tour
```

**Figura 2.12: importar método en modulo principal**

2. Dentro de la condicional correspondiente crear una instancia de la clase con el método de búsqueda llamada **solver**:

```python
# leer e interpretar el problema TSP leido desde la instancia definida
problem = Tsp(filename=options.instance)

# Ejecutar Metaheuristica Simulated Annealing
if (options.metaheuristic == MHType.SA):

    # Solucion inicial
    first_solution = Tour(type_initial_sol=options.initial_solution, problem=problem)
    # Crear solver
    solver = SimulatedAnnealing(options=options, problem=problem)
    # Ejecutar la busqueda
    solver.search(first_solution)

# Ejecutar Metaheuristica Algoritmo Genetico
elif (options.metaheuristic == MHType.GA):
    # Crear solver
    solver = GeneticAlgorithm(options=options, problem=problem)
    # Ejecutar la busqueda
    solver.search()

elif (options.metaheuristic == MHType.LS):
    # Solucion inicial
    first_solution = Tour(type_initial_sol=options.initial_solution, problem=problem)
    # Crear solver
    solver = LocalSearch(options=options, problem=problem)
    # Ejecutar la busqueda
    solver.search(first_solution)
```

Figura 2.13: Crear instancia del algoritmo

Debe llamarse **solver** para ser compatible con los métodos en común con los demás algoritmos

```python
# Guardar la solucion y trayectoria en archivo
solver.printSolFile(options.solution)
solver.printTraFile(options.trajectory)
# Escribir la solucion por consola
solver.print_best_solution()

end = timer() # tiempo final de ejecucion
print(f"{bcolors.BOLD}Tiempo total de ejecucion: {bcolo

if options.visualize:
    solver.visualize()
```

Figura 2.14: Nombre de la instancia

3. El método de búsqueda debería estar funcionando, probar con “python tspf.py -mh <abreviación definida>”.

## 2.4 Implementar en la Interfaz Gráfica (Bonus)

Si desea implementar el nuevo método de búsqueda debe seguir los siguientes pasos:

1. Dirigirse al paquete de herramientas (Tools) e ir al módulo **gui.py**
2. Agregar selección de método de búsqueda en la clase **Gui** y el método **mainScreen** (aprox. Línea 145)

Crear el botón acomodándolo dentro del **grid** y creando el método que lo gestione

```
def mainScreen(self) -> None:

    self.frameL.destroy()
    self.root.geometry('960x540')
    self.frame.destroy()
    self.frame = LabelFrame(
                self.root,
                text='Metodo de Busqueda',
                bg='#f0f0f0',
                font=('consolas', 20)
            )

    self.frame.pack(anchor='center', pady=10)

    f = Frame(self.frame)
    f.grid(row=0, column=0)

    b = Button(f, text='Simulated Annealing', command=self.simulatedAnnealing)
    b.config(font=buttonFont)
    #b.pack(anchor='ne', side='left', padx=50, pady=50)
    b.grid(row=0, column=0, padx=25, pady=25)
```

Figura 2.15: Agregar nuevo algoritmo GUI

3. Crear y gestionar las opciones en el método que administre el algoritmo:

```
""" S I M U L A T E D   A N N E A L I N G """

def simulatedAnnealing(self) -> None:
    """ Configura las opciones de simulated annealing """

    self.options.metaheuristic = MHType.SA
    self.frameL.destroy()
    self.frame.destroy()
    self.optionsFrame()


    frameSA = LabelFrame(
                self.frameOptions,
                text='Simulated Annealing',
                bg='#f0f0f0',
                font=("consolas", 22)
            )

    #frameSA.pack(anchor='n', side='right', padx=25, pady=15)
    frameSA.grid(row=0, column=3, padx=15, pady=10)
```

Figura 2.16: Método del algoritmo GUI

Dentro de este seguir el ejemplo del patrón anterior, pero con los nombres del nuevo algoritmo y agregar las opciones de este posteriormente.

4. Agregar a la ejecución en el método **search** (aprox. Línea 814) de forma similar a lo hecho para la ejecución en la terminal:

```
# Mostrar Opciones
options.printOptions()

# leer e interpretar el problema TSP leido desde la instancia definida
problem = Tsp(filename=options.instance)

# Ejecutar Simulated Annealing
if (options.metaheuristic == MHType.SA):

    # Solucion inicial
    first_solution = Tour(type_initial_sol=options.initial_solution, problem
    # Crear solver
    self.solver = SimulatedAnnealing(options=options, problem=problem)
    # Ejecutar la busqueda
    self.solver.search(first_solution)

# Ejecutar Algoritmo Genetico
elif (options.metaheuristic == MHType.GA):
    # Crear solver
    self.solver = GeneticAlgorithm(options=options, problem=problem)
    # Ejecutar la busqueda
    self.solver.search()

# Ejecutar Local Search
```

Figura 2.17: Agregar ejecución GUI

5. Agregar cambio de algoritmo en la clase de **MenuBar** (aprox. Línea 940) siguiendo el patrón:

```
    # cambiar metodo
    self.editMenu = Menu(self.menuBar, tearoff = False)
    self.editMenu.add_command(label="Simulated Annealing", command=lambda: self.changeSearch(MHType.SA))
    self.editMenu.add_command(label="Algoritmo Genetico", command=lambda: self.changeSearch(MHType.GA))
    self.editMenu.add_command(label="Local Search", command=lambda: self.changeSearch(MHType.LS))
    self.editMenu.add_command(label="Iterated Local Search", command=lambda: self.changeSearch(MHType.ILS))

    self.menuBar.add_cascade(menu=self.editMenu, label="Cambiar de Algoritmo")
```

Figura 2.18: Cambio de algoritmo GUI

6. Gestionar el cambio de algoritmo siguiendo el patrón en el método **changeSearch**

```python
def changeSearch(self, searchType: MHType) -> None:
    """ Cambia de metodo de busqueda en la barra de herramientas """

    if self.gui.frameOptions != None:
        self.gui.frameOptions.destroy()
    if not self.gui.options.replit and self.gui.frameFeedback != None:
        self.gui.frameFeedback.destroy()

    if searchType == MHType.SA:
        self.gui.simulatedAnnealing()
    elif searchType == MHType.GA:
        self.gui.geneticAlgorithm()
    elif searchType == MHType.LS:
        self.gui.localSearch()
    elif searchType == MHType.ILS:
        self.gui.iteratedLocalSearch()
```

Figura 2.19: Gestionar cambio de algoritmo GUI

7. Probar con “python tspf.py --gui”

# 3 Ubicación métodos de interés

Algunos métodos de interés que podría resultar de utilidad saber su ubicación.

## 3.1 Métodos para manejar la población de Algoritmo Genético

A continuación, veremos algunas secciones con los métodos importantes utilizados para manipular las poblaciones de algoritmo genético. Estos métodos están ubicados en el módulo **Population.py** dentro del paquete **Algorithms**.

- **Métodos de selección de padres:** Línea 240 hasta 420
- **Métodos de cruzamiento:** Línea 431 hasta 632
- **Métodos de mutación:** Línea 654 hasta 720
- **Métodos de selección de población:** Línea 733 hasta 898