Guía básica de funcionamiento

Intención

Esta guía está destinada a programadores con conocimiento general de Java y el paradigma orientado a objetos que desean modificar o extender el diseño del programa.

Funcionamiento general

El sistema recibe de entrada un archivo de configuración y una lista de problemas (uno o más). El formato en que deben ser recibidos estos archivos se especifica en los archivos FormatoXMLEntrada y FormatoXMLConfiguración. Estos archivos son leídos por el objeto Readr y transformados en objetos de tipo Problem y ConfigurationParams. Para ver el detalle de cómo son guardados los datos dentro de estos objetos revisar el diagrama de clases en los diagramas de diseño.

El funcionamiento del programa es el siguiente: (comandado por el objeto Controller)

1. Lee el archivo de configuración y aplica las configuraciones. Detalle en el diagrama de secuencia de Setup en los diagramas de diseño.
2. Lee el archivo xml del problema y crea el objeto Problem.
3. Resuelve un problema llamando a Solver.solveOneInstance(Problem) . Los detalles están en los diagramas de secuencia de las últimas dos páginas (uno para el algoritmo constructivo y el otro para el algoritmo de mejoramiento).
4. Escribe o muestra la solución al problema. El escritor default crea un archivo xml donde escribe la solución de acuerdo al formato especificado en FormatoXMLSalida
5. Regresa al paso 2 si hay más problemas por resolver.

En el programa se hace uso del patrón de diseño Strategy para poder seleccionar dinámicamente que tipo de algoritmo se desea utilizar, que tipo de función objetivo se desea utilizar, el tipo de escritor y el tipo de heurística. Todas estas configuraciones son pasadas a través del archivo de configuración.

Descripción especifica de las partes del programa

Lectores

El objeto Readr es una clase que tiene dos tipos de lectores: lector de problema y lector de configuración; y sirve como un adaptador de ambos para que el controlador utilice solo un objeto lector (Readr) para varios tipos de lecturas.

Escritores

El escritor usa el patrón de diseño Strategy para seleccionar que tipo de escritor se desea antes de utilizarlo. Esta selección mediante la llamada set Strategy se hace en el Controlador.

Solver

La intención de este objeto es ser la interface entre la parte de lectura/escritura y la parte de resolución de problemas. Ofrece funciones para resolver un problema.

Algoritmos

Los algoritmos también usan el patrón de diseño Strategy y se pueden subdividir en algoritmos de construcción y algoritmos de mejoramiento. Los algoritmos concretos (ConstructiveAlgorithm, HIllClimbing y SimulatedAnnealing) heredan varias funciones de la superclase Algorithm pero la implementación de ciertas funciones es específica para cada algoritmo (notablemente la función de resolución de problemas executeAlgorithm). Cada algoritmo posee una función objetivo (mediante la superclase Algorithm) y una heurística (en las subclases concretas). Los algoritmos de construcción usan heurísticas específicas de construcción y el caso es el mismo para los algoritmos de mejoramiento (para ver el detalle del funcionamiento revisar los diagramas de clase). La clase Algorithm es la encargada de orquestar la configuración del algoritmo en particular, la heurística que será utilizada y la función objetivo utilizada. Una vez que el sistema está configurado se llama a executeAlgorithm para resolver cada problema y cada algoritmo usa la función objetivo o la heurística según su criterio. (Revisar el diagrama de secuencia de configuración para ver el detalle de cómo se realiza la configuración)

Funciones Objetivo

La clase ObjectiveFunction hace de adaptador y default para las funciones objetivo específicas y provee varias firmas de llamadas a funciones objetivo. Las funciones objetivo concretas sobreescriben computeAptitude de acuerdo a su necesidad. Por ejemplo, algunas funciones objetivos permitirán calcular la aptitud de una ruta y otras solo permitirán calcular la aptitud de una solución completa.

Heurísticas

Las heurísticas están divididas en dos grupos: heurísticas constructivas (ConstructiveHeuristic) y heurísticas de mejoramiento (ImprovementHeuristic). Las heurísticas de construcción generan una nueva ruta cada vez que son llamadas con una lista de clientes, mientras que las heurísticas de mejoramiento generan una solución alternativa cuando les pasan una solución existente. El funcionamiento codificado en estas heurísticas es, desde luego, específico para la heurística.

Objetos modelo

Estos objetos modelan datos del programa y realizan funciones referentes a estos datos

ConfigurationParams

Contiene todos los parámetros de configuración recibidos mediante el archivo de configuración. Este objeto es pasado al algoritmo y a todas las clases que necesiten configuración en la primera etapa del programa (la etapa de configuración). Ofrece métodos de acceso y escritura. Este objeto permanece sin cambios luego de ser creado cuando se lee la configuración desde el archivo xml de configuración.

Problem

Contiene todos los datos necesarios para describir un problema de VRP con diferentes variables. Tiene una lista de clientes y una lista de vehículos. Luego de que un problema es resuelto el objeto problema contiene solo los clientes que no fueron asignados en la solución (normalmente queda vacío) y los vehículos que no fueron utilizados por la solución.

Client

Contiene los datos referentes a cada cliente en un problema VRP. Ofrece métodos de acceso y escritura. Este objeto permanece sin cambios luego de ser creado por primera vez (cuando se lee el problema desde el archivo xml de descripción del problema).

Vehicle

Contiene los datos de un tipo de vehículos que serán usados en el problema VRP. Notar que es una clase o conjunto de vehículos no un vehículo en sí.

Solution

Contiene los datos relevantes a la solución del problema y es el archivo que se le pasa al escritor o gráficador para que muestren la solución. Ofrece métodos de accesos y escritura y también métodos de actualización de los datos contenidos, por ejemplo actualiza la distancia y el tiempo total de la solución cada vez que una ruta es añadida o eliminada. Contiene una lista de rutas.

Route

Este objeto describe la información relevante a una sola ruta y los métodos necesarios para manejarla. Su primer cliente siempre es el almacén y tiene además un tipo de vehículo con una capacidad específica asignada a ella. Ofrece métodos de escritura y lectura, de actualización de datos (es decir que se actualiza automáticamente cuando se hace una inserción o eliminación de un cliente). Además, se encarga de mantener a la ruta como una ruta valida mediante el método isFeasible: por ejemplo, si se intenta hacer una inserción que haga a la ruta imposible entonces regresa falso y se conserva sin hacer cambios.

Agregar nuevas funcionalidades

El programa usa el patrón de diseño Strategy e interfaces/clases abstractas para facilitar la adhesión de nuevas funcionalidades. Generalmente para añadir una nueva funcionalidad (por ejemplo un nuevo escritor u otro algoritmo u otra función objetivo) se deben hacer dos cosas:

- Crear la subclase concreta que sobre escriba las funciones necesarias para obtener las definiciones específicas. Por ejemplo, un nuevo escritor debe sobre escribir la función write y ser una subclase de Writr, una función objetivo debe sobre escribir la función computeAptitude y ser una subclase de ObjectiveFunction , etc.
- Su nombre debe ser añadido en el código para que al ser recibido en el archivo de configuración el programa sepa que clase inicializar y establecer mediante setStrategy.

Para utilizar una nueva funcionalidad se debe nada más escribir su nombre en el archivo de configuración y enviarle los parámetros que vaya a necesitar. El programa hará que esos parámetros lleguen hasta la nueva clase como un Map <String, String>.

Que se va a agregar	**Algoritmos**	**Escritores**	**Funciones objetivo**	**Heurística constructiva**	**Heurística de mejoramiento**
Superclase	Algorithm	Writr	ObjectiveFunction	ConstructiveHeuristic	ImprovementHeuristic
Funciones a sobre escribir	executeAlgorithm setHeuristic De ser necesarias: applySpecificConfiguration applyBaseConfiguration	write	computeAptitude (una o varias)	createNewRoute De ser necesaria: setObjectiveFunction	generateAlternativeSolution
Donde agregar nombre	Solver.applyConfiguration	Controller.processProblem	Algorithm.applyConfiguration	ConstructiveAlgorithm.setHeuristic	HillCLimbing.setHeuristic (este depende del algoritmo usado)
Ejemplo	SimulatedAnnealing	GraphicalWriter	ObjectiveFunction3	NAH	Exchange

Programas alternos

Se generaron dos programas pequeños que pueden ser útiles para investigadores de VRP y usuarios de este programa

textTOxml

Este programa recibe una lista de problemas VRP definidos como en las instancias de Solomon y los transforma a xml del tipo definido en FormatoXMLEntrada para que puedan luego ser resueltos por el programa. Para calcular las distancias utiliza las coordenadas de los clientes y calcula la distancia euclidiana entre ellos y el tiempo es igual a la distancia euclidiana multiplicada por 60 y redondeada.

OutputGraph

Recibe un problema como archivo xml del tipo definido en FormatoXMLEntrada y un archivo xml de solución del tipo definido en FormatoXMLSalida y grafica las rutas de la solución.

Errores

Se tiene un catálogo de errores descrito en el archivo “Documentación de errores.doc”.

Para agregar un nuevo error se debe agregar el código en el constructor estático de la clase ErrorHandler de forma similar a como los otros errores están establecidos.