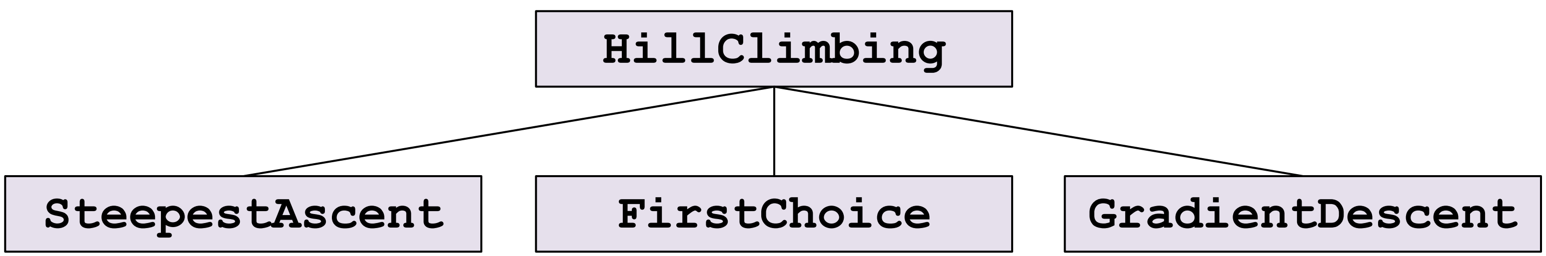
Search Algorithms: Object-Oriented Implementation (Part D)

Contents

Defining ‘HillClimbing’ Class

- We define `HillClimbing` class to put together all the search algorithms and unite all the programs into a single main program
  - Search algorithms become subclasses under `HillClimbing`
  - The class hierarchy of `HillClimbing` is stored in a separate file named ‘optimizer.py’



- The names of the search algorithms are now the names of the subclasses under the `HillClimbing` class
  - The body of each search algorithm becomes the body of the `run` method of the corresponding subclass

Defining 'HillClimbing' Class

- To have a single main program, we need a new user interface to ask the user the type of problem to be solved (**`pType`**) and the type of algorithm to be used (**`aType`**)
  - These information will be used to create the **`Problem`** and **`HillClimbing`** objects of the right types
  - **`pType`** will also be used when printing out messages about the settings of the search algorithm used

- **`displaySetting`** that was previously part of the main program is moved to the **`HillClimbing`** class because what it displays are the information about the settings of the search algorithms that are now the methods of **`HillClimbing`**

Defining ‘HillClimbing’ Class

- To store information necessary for **`displaySetting`** and the search algorithms, two variables are defined in **`HillClimbing`**
  - **`pType`**: integer indicating the type of problem to be solved
  - **`limitStuck`**: maximum evaluations allowed without improvement
    - It takes the role of the previous named constant **`LIMIT_STUCK`**
    - Currently, only **`firstChoice`** is under the control of this variable
    - Later, the stochastic hill-climbing algorithm to be added to the search tool will be controlled by this variable

- **`displaySetting`** in **`HillClimbing`** prints out the mutation step size (**`delta`**) when the type of problem is numerical optimization
  - This method is inherited to **`SteepestAscent`** and **`FirstChoice`**, but not to **`GradientDescent`**

Defining 'HillClimbing' Class

- **`displaySetting`** in each class of search algorithm prints out the algorithm name and additional setting information specific to that algorithm
  - **`displaySetting`** of **`FirstChoice`** prints out the maximum evaluations allowed without improvement (**`limitStuck`**)
  - **`displaySetting`** of **`GradientDescent`** prints out the values of **`alpha`** and **`dx`** (using accessor methods **`getAlpha`** and **`getDx`**?)
- Notice that **`delta`**, **`alpha`**, **`dx`** were variables of **`Numeric`** subclass (under **`Problem`**), and **`getDelta`**, **`getAlpha`**, **`getDx`** were methods of **`Numeric`**
  - For **`displaySetting`** to refer to **`alpha`** and **`dx`**, it needs as its argument the problem instance being solved
    (so that it can use the statement such as **`p.getAlpha()`**)

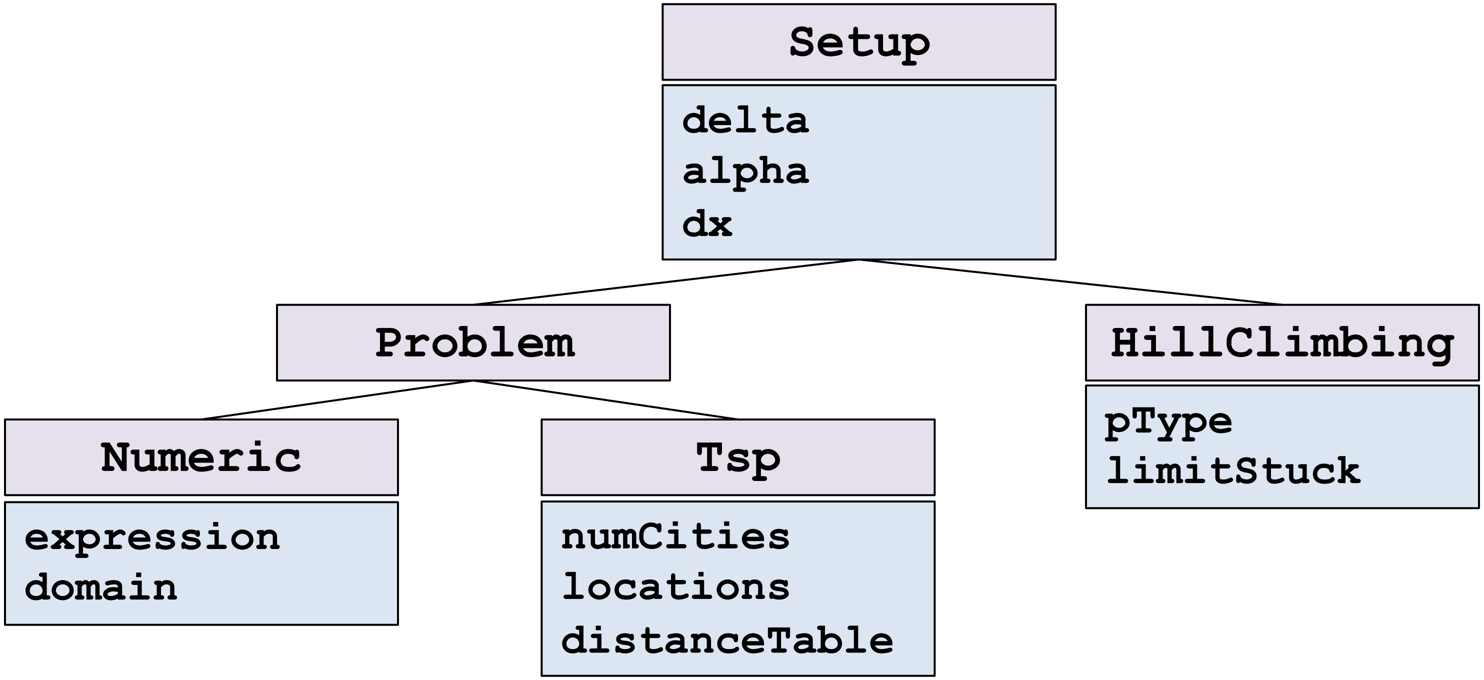
Defining 'HillClimbing' Class

- What if we make `alpha` and `dx` variables of `GradientDescent`, and `delta` a variable of `HillClimbing`?
  - Not a good idea because they are already variables of the subclass `Numeric`

- We better create a superclass of `Numeric` and `HillClimbing`, and have those variables belong to that superclass
  - Once this is done, the three accessors, `getDelta`, `getAlpha`, and `getDx` of the `Problem` class are no longer necessary because `displaySetting` that deals with these information will belong to `HillClimbing` and thus can access those variables directly

Adding a Superclass 'Setup'

- Since **`delta`**, **`alpha`**, and **`dx`** are needed by both the classes **`HillClimbing`** and **`Problem`**, we define a new class named **`Setup`** to hold those variables and make it a parent class of both
- We store **`Setup`** in a separate file named 'setup.py' and let the 'problem.py' and 'optimizer.py' files import it from that file



The Main Program

- The main program is stored in a file named ‘main.py’
  - The ‘main.py’ file should import everything from ‘problem.py’ and ‘optimizer.py’

- The main program includes a few functions for a new user interface to ask the user to choose the problem to be solved and the optimization algorithm to be used

The Main Program

- **`main()`**:
  - Creates a **`Problem`** object **`p`** of the right type by querying to the user (**`selectProblem`**)
  - Creates a **`HillClimbing`** object **`alg`** (search algorithm) by querying to the user (**`selectAlgorithm`**)
  - Runs the search algorithm by calling the **`run`** method of the **`HillClimbing`** class (**`alg.run`**)
  - Shows the specifics of the problem just solved (**`p.describe`**)
  - Shows the settings of the search algorithm (**`alg.displaySettings`**)
  - Displays the result of search (**`p.report`**)

The Main Program

- **`selectProblem()`**:
  - Asks the user to choose the type of problem to be solved
  - Creates a **`Problem`** object **`p`** of the right type
  - Sets the variables of the corresponding subclasses of **`Problem`**
  - Returns **`p`** and **`pType`** (an integer indicating the problem type)
- **`selectAlgorithm(pType)`**:
  - Asks the user to select a search algorithm
  - Asks the user to select again if gradient descent is chosen for a TSP (**`invalid`**)
  - Prepares a dictionary whose keys are integers corresponding to **`aType`**, and values are the names of the subclasses of **`HillClimbing`** (i.e., search algorithms)

The Main Program

- Creates an object **`alg`** of the targeted **`HillClimbing`** subclass using the dictionary (**`alg = eval(optimizers[aType])`**)
- Sets the variables of the **`HillClimbing`** class
- Returns **`alg`**

• **`invalid(pType, aType)`**:

- If gradient descent is chosen for a TSP, informs the fact to the user and returns **`True`**
- Otherwise, returns **`False`**