Intelligent Systems Practical 1
Abstracting and Specifying Search Problems

Introduction

This practical illustrates the issues involved in specifying search problems. The practical is accompanied with a code archive (available for download from the course Web site) that contains a breadth-first search solver for the n-puzzle problem. Your task is to modify this solver into a generic Java library that can solve arbitrary search problems. In addition, you should apply this generic library to two concrete search problems: the n-puzzle and the problem of finding a tour through all the cities in Romania (optional).

You should ensure that the code of your solution is well documented. The deadline for submitting solutions to this practical is week 4.

A Description of the Code Archive

Before starting your work on this practical, download the code archive from the course Web site and import its contents into an Integrated Development Environment. (Use of an IDE is not mandatory; however, you are likely to find working on this practical significant easier if you use an IDE, such as Eclipse.) The code in the archive is organized in two packages, which are described next.

The *npuzzle* Package

The *npuzzle* package contains the breadth-first search solver for the n-puzzle problem. You should familiarize yourself thoroughly with the classes in this packages before you start working on this practical. A brief description of the classes in the package is given next.

- The *Tiles* class represents a configuration of tiles. The *width* member specifies the width of the puzzle; thus, each puzzle consists of tiles arranged in an $width \times width$ grid. The layout of the tiles is stored in the *tiles* array, where $tiles[i \times width + j]$ determines the number of the tile located in row i and column j of the grid; the empty tile is identified by the tile number 0. Finally, *emptyTileRow* and *emptyTileColumn* contain the row and the column of the empty tile, respectively; although these values can be obtained from the *tiles* array, they are stored explicitly for convenience. The *Tiles* class contains several methods whose meaning should be evident from each method's name.

- The *Movement* class enumerates the four directions that the empty tile can be moved in.

- The *Node* class represents a search node. It stores a reference to the parent node; a reference to the tile configuration; and a reference to the movement that, when applied to the tile configuration stored in the parent node, produces the tile configuration stored in this node. For root nodes, parent and movement are set to *null*.

- The *BreadthFirstTreeSearch* class implements the breadth first tree search algorithm. Its *findSolution()* method is given a root node, and it returns a node that contains a solution, or *null* if no solution can be found.
- The *TilesPrinting* class implements solution printing. It contains two *print()* methods that print a tile configuration and a movement, respectively. Furthermore, it contains the *printSolution()* method that, given a solution node, prints the path to the solution by calling the two *print()* methods mentioned earlier.
- The *BFTS_Demo* class demonstrates the breadth-first tree search (this is the meaning of the *BFTS* prefix). It contains a *main()* method and can thus be executed. In order to ensure that the Java Virtual Machine has sufficient heap space available, the *-Xmx700M* option should be specified on the command line when starting the class; in Eclipse, this option can be placed into the "VM Arguments" field of the "Arguments" tab of the "Run Configurations" menu option. When started, the *main()* method creates an initial tile configuration, invokes the search process, and prints a solution (if one was found).

The *tour* Package

The *tour* package should be used when solving the optional task of this practical. The classes in the *tour* package encode the map of Romania used in the AIMA book.

- The *City* class represents a city on the map. Each city is uniquely identified by its name, and one can retrieve for each city the set of roads leaving the city. Furthermore, for each city, one can also retrieve the shortest distance to an arbitrary other city on the map; if the cities are not connected, *Integer.MAX_VALUE* is returned. (The distances will be used only in the optional part of Practical 3.)
- The *Road* class represents a (directed) road between two cities. Each road consists of a source city, a target city, and the road length.
- The *Cities* class contains a set of cities indexed by their name.
- The *SetUpRomania* class contains two static methods that instantiate maps of Romania. The *getRomaniaMap()* method returns the entire map of Romania as given in the AIMA book, and the *getRomaniaMapSmall()* method returns a smaller version of the map.

Task 1: Develop a General Search Library

The n-puzzle solver from the *npuzzle* package fully implements the breadth-first search algorithm, but it is tied to a specific problem (i.e., the n-puzzle). Thus, if we wanted to apply the breadth-first search to a different problem, we would need to copy-paste the algorithm and then adapt it to the details of the problem in question. This is not a very elegant solution, and, from a software engineering point of view, it is quite problematic. For example, if we found an error in our

implementation of the n-puzzle solver, we would need to correct the same error in all places obtained by copying and pasting the original code.

In order to prevent such problems, you are required to develop the n-puzzle solver into a general search library that is applicable to an arbitrary search problem. That is, assuming one formalizes a general problem as discussed Lectures 2 and 3, it should be possible to solve the problem using the breadth-first tree search algorithm without touching the algorithm's code.

You should proceed as follows. Create a new Java package called *search* for your new library, and then add the following interfaces to the package.

- The *Action* interface should represent an abstract notion of an action. This interface does not need to have any methods.
- The *State* interface should represent a state in the search, and it should contain methods that implement the following functionality:
  - It should be possible to retrieve the set of actions (as a collection of type *Set<Action>*) that are applicable to a state s.
  - Given an action a that is applicable to a state s, it should be possible to retrieve the state obtained by applying a to s.
- The *GoalTest* interface should contain a method that can be used to test whether a state is a goal state.

Thus, to adapt the search library to a particular search problem, one should create classes that implement the *Action*, *State*, and *GoalTest* interfaces and thus provide the suitable notions of actions, states, and a goal test. In order words, the three interfaces from the *search* package should allow one to cast a formal specification of a search problem into code.

In addition to these interfaces, the *search* package should contain the following classes.

- Modify the *Node* class of the n-puzzle solver such that it is not tied to the n-puzzle—that is, so that it can be applied to an arbitrary implementation of the *State* and *Action* interfaces.
- Modify the *BreadthFirstTreeSearch* class of the n-puzzle solver such that it is not tied to the n-puzzle—that is, so that it can be applied to an arbitrary search problem.
- Modify the *TilePrinting* class of the n-puzzle solver such that it is not tied to the n-puzzle—that is, so that it can print solutions to an arbitrary search problem. The modified class should be called *Printing*, and it should contain two abstract *print()* methods: one that prints states, and one that prints actions. Furthermore, the class should contain the *printSolution()* method that can be used to print a path to a solution node.

Task 2: Apply the General Search Library to n-Puzzle

Apply the general search library developed within Task 1 to n-puzzle. To this end, you should ensure that the *npuzzle* package contains the classes listed below.

You should *modify* the relevant classes from supplied code archive such that they implement/use the classes/interfaces from Task 1 as appropriately, and you should *add* the missing classes.

- *Tiles*
- *Movement*
- *TilesGoalTest*
- *TilesPrinting*
- *BFTS_Demo*

It should be possible to start your *BFTS_Demo* class and obtain the same output as in the case of the n-puzzle solver from the code archive accompanying this practical.

Task 3: Apply the General Search Library to Romania Tour (Optional)

Apply the general search library to the problem of finding a tour in Romania—that is, a path that starts and ends at a given city and that visits all cities in Romania at least once. Note that this formulation allows a tour to visit a city several times; also, note that the goal in this practical is to find an arbitrary (i.e., not necessarily the shortest) tour. You should achieve this goal by adding the following classes to the *tour* package:

- The *TourState* class should represent a state in a tour searching process. Take into account that, in order to find a tour, we need to keep track not only of the current city, but also of all cities that were visited previously.
- The *TourGoalTest* class should implement the goal test for the problem of finding a tour through a given set of cities.
- The *TourPrinting* class should implement the methods necessary for printing solutions to the touring problem.

Furthermore, you should modify the *Road* class such that its instances can be used as actions. One might object to such a solution: a road is *not* an action; rather, an action consists of driving along a particular road. In order to keep matters simple, however, we will not be that pedantic and will simply identify the notion of a road with the notion of driving along a road.

Finally, you should create a class called *BFTS_Demo* that invokes the search in order to find a tour starting and ending in Bucharest. It is likely that your implementation will not be able to process the entire map of Romania, in which case you should use the smaller version of the map provided in the *SetUpRomania* class. Describe in a few sentences why breadth-first search cannot handle the larger version of the map.