Exercise: Primal Heuristic for the TSP

Implement a primal heuristic for the symmetric traveling salesman problem (TSP) on a complete graph $G = (V, E)$. This heuristic should be a greedy start heuristic, which is based on the current LP solution. It starts with an arbitrary node $s \in V$ and iteratively constructs an $[s, u]$-path T, as long as there are any nodes left not contained in T. It then closes the path to a tour. In each iteration, the $[s, u]$-path is extended to an $[s, v]$-path where node $v \in V$ is selected in a greedy fashion, such that

$$x^*_{uv} = \max\{x^*_{uw} : w \in V \backslash V(T),\ x_{uw} \text{ not fixed to be zero}\}.$$

Here, x^*_{uv} denotes the value of the variable x_{uv} in the current LP solution. A binary variable is fixed to be zero if its global upper bound has changed to zero.

In the doxygen documentation of SCIP, you will find the entry "How to add primal heuristics", which explains all steps of implementing a primal heuristic in detail. Since not all of these steps are needed for this exercise, again, the following instruction will guide you through this documentation.

Getting started

(a) Go to the directory `COatWork-TSP/src` and copy the template files `HeurXxx.cpp` and `HeurXxx.h` to files named `HeurLpgreedy.cpp` and `HeurLpgreedy.h`, respectively. Open the new files with a text editor and replace all occurrences of `xxx`, `Xxx`, and `XXX` by `lpgreedy`, `Lpgreedy`, and `LPGREEDY`, respectively.

(b) Adjust the name, description, and display character parameters in the default constructor `HeurLpgreedy()`, which you find in `HeurLpgreedy.h`.

(c) Include your header file `HeurLpgreedy.h` into the main file `cppmain.cpp`. Then, add `SCIPincludeObjHeur()` with a correct argument list to the `runSCIP()` method of `cppmain.cpp` and include the line "`HeurLpgreedy.o`" into the `MAINOBJ` list of the Makefile. Compile and test.

(d) *[src/HeurLpgreedy.cpp:104] ERROR: method of lpgreedy primal heuristic not implemented yet* in the SCIP output means that everything works (for now).

(e) Unless otherwise noted, all SCIP methods you need in this exercise are defined in `scip.h`, all graph methods are defined in `GomoryHuTree.h`.

Defining the primal heuristic data and implementing two additional callback methods

(a) Add two object variables to your class: one for the graph corresponding to the TSP instance (`GRAPH* graph`) and one for the current LP solution (`SCIP_SOL* lpsol`).

(b) Implement the `scip_init` method. It should initialize the object variables. Therefore, you have to create the solution `lpsol`, using `SCIPcreateSol()` and get the problem data structure by the following line:
`ProbDataTSP* probdata = dynamic_cast<ProbDataTSP*>(SCIPgetObjProbData(scip));`
Then access the graph from the `probdata` (struct defined in `ProbDataTSP.h`) and capture it, using `capture_graph()`.

(c) Implement the `scip_exit` method. It should free all of the memory that has been allocated in the `scip_init` method and release objects that have been captured in `scip_init`, hence call `SCIPfreeSol()` and `release_graph()`, respectively.

Implementing the fundamental callback method

(a) Implement the `scip_exec` callback. It should realize the actual heuristic outlined above. Here are some implementation hints:

- The following lines of code ensure that the current LP has been solved to optimality, copy the current LP solution into the object variable `lpsol`, and use `lpsol` for accessing the LP solution value `lpsolval` of the variable `var`:

```
if( SCIPgetLPSolstat(scip) != SCIP_LPSOLSTAT_OPTIMAL )
     return SCIP_OKAY;

SCIP_CALL( SCIPlinkLPSol(scip, lpsol) );
SCIP_Real lpsolval = SCIPgetSolVal(scip, lpsol, var);
```

- For the primal solution to be constructed in `scip_exec`, you need a `SCIP_SOL* newsol`, which you create by `SCIPcreateSol()`, free by `SCIPfreeSol()`, and which you can pass to SCIP by `SCIPtrySol()`. The value of a variable in this solution can be set by `SCIPsetSolVal()`. All variables for which no value is set explicitly, are treated as zero.
- `SCIPvarGetUbGlobal()` (defined in `pub_var.h`) returns the global upper bound of a variable, which you need to check that a variable is not globally fixed to 0.
- Use the methods `SCIPallocBufferArray()` and `SCIPfreeBufferArray()` for allocating and freeing memory for arrays in the `scip_exec` method, e.g., for a `SCIP_Bool` array that stores (at position `node->id`) whether `node` is already contained in $V(T)$.
- The `result` pointer should be set to `SCIP_DIDNOTFIND` or `SCIP_FOUNDSOL`.

(b) You can adjust the priority, frequency, frequency offset, maximal depth, and timing parameters of the default constructor in `HeurLpgreedy.h` to change the performance.

Final test

(a) Compile your project and test it on the provided TSP instances (which you find in `tspdata`). You should see your chosen display character as first character in an output line, whenever your heuristic found a solution.

Go for it!