Using HyFlex and irace for Automatic Configuration of Hyper-heuristics: Detailed Instructions

Weiyao Meng

In this challenge, your goal is to enhance the performance of provided hyper-heuristics for solving travelling salesman problems (TSP) by leveraging the flexibility and capabilities of HyFlex and irace.

HyFlex (Hyper-heuristics Flexible framework) [BCH+09] is a Java object-oriented benchmark framework for the design and analysis of hyper-heuristics. Hyper-heuristics can be seamlessly designed and implemented within HyFlex using its provided heuristic algorithms and functionalities, and rigorously tested across the predefined problem domains within HyFlex.

Some heuristic algorithms provided in HyFlex involve two parameters, i.e., '*intensity of mutation*' and the '*depth of search*' (denoted as 'i' and 'd' in the range of $[0, 1]$. These two parameters control the behaviour of the related search operators.

Notably, the configuration settings of these parameters can be intelligently managed through irace [LIDLC+16], a tool specialised in automatic parameter configuration.

To initiate this challenge, we provide irace configuration scenario files tailored for the automatic configuration of 'i' and 'd' parameters within the provided hyper-heuristics. The following resources are provided

- project-hyflex: the source files of the provided example hyper-heuristics and the HyFlex framework (`chesc-fixed-no-ps.jar`).
- project-irace: the files of an example configuration scenario of using irace.

This instruction provides step-by-step guidance on how to utilise the provided resources effectively. You will learn how to:

- Set up HyFlex and irace.
- Execute the provided hyper-heuristics within HyFlex.
- Use irace to automatically configure the parameters of the provided hyper-heuristics.

Contents

Setup

Setting up HyFlex

HyFlex can either be run using Java from the command line or by importing the files into a Java IDE. For simplicity, we use the Eclipse in this instruction.

1. Installing Java: `https://www.java.com/en/download/help/download_options.html`

* Ensure that you add Java to the PATH environment variable.

2. Installing Eclipse: `https://www.eclipse.org/downloads/`

3. Importing resources in 'project-hyflex' into Eclipse

- In Eclipse, create a new project named 'project-hyflex'.
- Import all the source files into this project.

4. Adding HyFlex JAR file to classpath

- In Eclipse, right-click the project folder in the **Project Explorer** on the left side of the screen.
- Select **Build Path** → **Add External Archives**.
- Navigate to the location of the HyFlex JAR file and select it.
- Click **Open** to add the JAR file to your project's classpath.

5. Verifying HyFlex is set up correctly

- In Eclipse, find the example program `examples.example.ExampleRun1.java` in your project.
- Compile and run `ExampleRun1`.

If HyFlex is setup properly, you will observe the expected output in the console:

```
Algorithm: Example Hyper Heuristic One
Problem instance: class travelingSalesmanProblem.TSP
Time limit set to: 60 seconds
Search ...
```

Setting up irace

Follow the **Setup** instructions in `https://lopez-ibanez.eu/2024-natcor-aac/#setup`.
For simplicity, we use the Rstudio console for the rest of the instruction.

Part 1: Executing Hyper-heuristics

When setting up HyFlex, you have attempted to run `ExampleRun1`. This class demonstrates the minimum that must be done to test a hyper-heuristic on a problem domain. It is advisable to study this class. Comments in the code explain each section in detail. A summary of the steps is given here.

1. Create a Problem Domain object
2. Create a Hyper-Heuristic object
3. Load a problem instance
4. Set a time limit
5. Tell the Hyper-Heuristic which problem it is to solve, by giving it the reference to the Problem Domain object
6. Run the Hyper-Heuristic
7. Print the best solution found within the time limit

In `ExampleRun1`, you will see that we create a Hyper-Heuristic object of the `ExampleHyperHeuristic1` type. The `Example Hyper Heuristic One` algorithm is executed on a specific TSP instance for 60 seconds. Once the termination time is reached, the console will display the best solution value found.

It is advisable to study `ExampleHyperHeuristic1` next, to learn how to develop a hyper-heuristic strategy[1]. Pseudo-code is provided in Algorithm 1.

Part 2: Preparing Hyper-heuristics Runner for irace

To enable irace to execute the hyper-heuristics within HyFlex and configure parameters effectively, the Runner must be executable from the command line and capable of receiving parameters as arguments.

Note: Several heuristics in HyFlex feature configurable parameters 'i' and 'd'. The effects of these parameters are problem and heuristic dependent. For more details of these parameters' behaviour, you can decompile the HyFlex JAR file and have a look. For simplicity, we only consider configuring these parameters for heuristics with such settings in the provided example. In the TSP domain within HyFlex, three heuristics have configurable 'i' and three have configurable 'd' parameters.

1. Prepare the Runner to Accept Command-Line Arguments

- Navigating to the 'examples.rn' package to locate `RN`, which extends `ExampleHyperHeuristic1` and allows for the passage of 'i' and 'd' parameters.
- Executing `RNRunner`, which runs `RN` and accepts arguments from the command line or run configuration. If executed without the required arguments, it will prompt an error message:

```
Usage: java -jar <runner.jar> <id.configuration> <id.instance> <seed> <instance>
-d <configuration> -i <configuration> -t <time>
```

- Refer to the comments within the code for the required arguments and test the example run configuration. The Runner will now return the objective function value of the best-found solution.

2. Export the Runner as a Runnable JAR

To utilise `RNRunner` with irace, export it as a runnable JAR using the provided guide.

Note: To mitigate potential compatibility issues arising from different Java versions across machines, it's advisable to specify a lower target Java version (e.g., 1.7) when exporting the JAR file. This ensures that the resulting JAR file remains compatible with older Java runtime environments.

[1]Examples of more complex setup is available at HyFlex website.

3. Test the JAR File in Command Line

Before proceeding further, test the execution of `RNRunner.jar` from the command line to ensure it functions correctly with the required arguments.

To execute `RNRunner.jar`, navigate to the directory where it is located and run the following command:

```
java -jar RNRunner.jar 1 2024 1234 0 -d 0.1 0.2 0.3 -i 0.4 0.5 0.6 -t 10000
```

If the objective function value was successfully returned, move `RNRunner.jar` to the `"project-irace/irace-hyflex-example"` folder.

Part 3: Preparing Configuration Scenario in irace

The irace User Guide provides a guide for setting up a basic execution of irace in Section 4.1. The template files provided within the irace package serve as the starting point for creating new configuration scenarios.

In this challenge, example configuration files are provided to configure the heuristics of the hyper-heuristics `RN`. These configuration files, along with `RNRunner`, are located in the ”project-irace/irace-hyflex-rn” folder, showing the minimum that must be done.

1. ”parameters.txt”: Target algorithm parameters

The ”parameters.txt” file describes the parameters, one per line, specifying the name, type, and range.

In this example, there are 6 parameters $(d1, d2, d3, i1, i2, i3)$ to configure. These parameters are real-valued and fall within the range of [0, 1].

The `time "-t"` parameter is set to 10,000 milliseconds, indicating that each run of `RNRunner` terminates after 10 seconds. Note that this is an argument specification rather than a configurable parameter. Adapt this value as needed for your configuration scenario.

2. ”configurations.txt”: Initial configurations

The ”configurations.txt” file includes the initial parameter values. Currently, the values are all set to 0.2, consistent with the default setting in HyFlex.

3. ”instance-train.txt”: Training instances

The ”instance-train.txt” file specifies training instances, where each line contains the ID of a TSP instance.

Currently, we utilise the TSP instances 0, 1, 2, 3, 4, 5 as training instances. Adapt the list of instances as needed for your configuration. Table 1 shows the TSP instances provided in HyFlex.

4. ”scenario.txt”: Configuration scenario

The 'scenario.txt' file contains essential settings for the parameter configuration process. Some key parameters you may need to adjust include directory paths, instance files, and the targetRunner path etc.

An important setting is the tuning budget. In this example, `maxExperiments` is specified to limit the number of targetRunner executions by irace. Specifically, it is set to 180, meaning that irace will terminate after completing 180 runs. Adapt this value as needed for your configuration.

Part 4: Executing irace

The configuration scenario for the provided `RN` is now ready to use by irace. To execute the configuration scenario in the RStudio console, follow these steps:

1. Change the working directory to the 'irace-hyflex-rn' folder. If the folder's location is '/path/to/project-irace/irace-hyflex-rn', use the command:

   ```
   setwd("/path/to/project-irace/irace-hyflex-rn/")
   ```

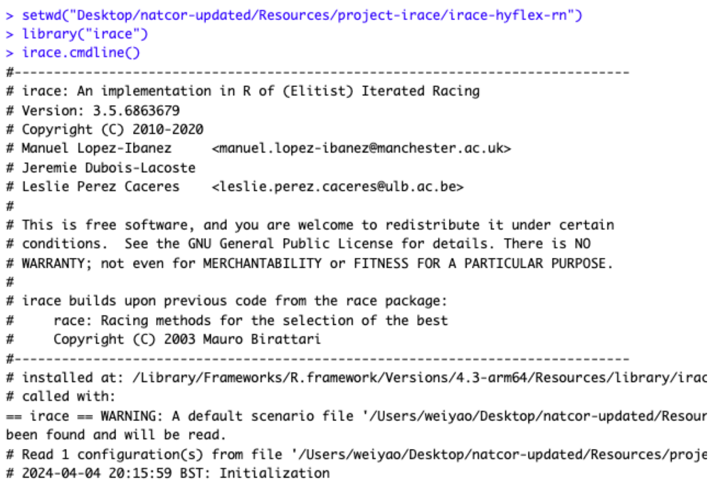
2. Load irace:

   ```
   library("irace")
   ```

3. Run irace:

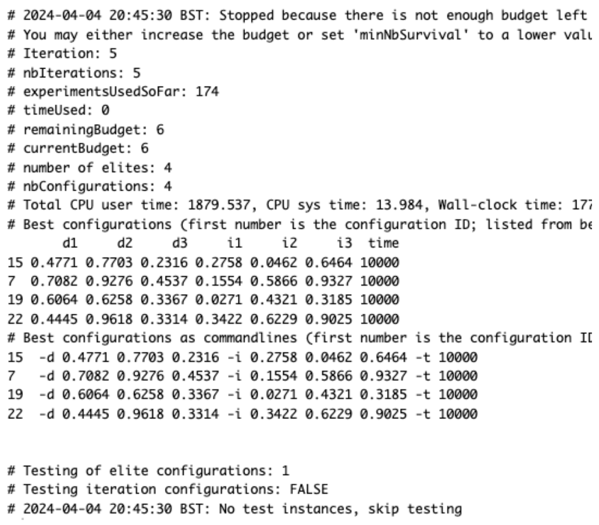
   ```
   irace.cmdline()
   ```

You'll see text output similar to the figure below.



```
> setwd("Desktop/natcor-updated/Resources/project-irace/irace-hyflex-rn")
> library("irace")
> irace.cmdline()
#------------------------------------------------------------------------------
# irace: An implementation in R of (Elitist) Iterated Racing
# Version: 3.5.6863679
# Copyright (C) 2010-2020
# Manuel Lopez-Ibanez     <manuel.lopez-ibanez@manchester.ac.uk>
# Jeremie Dubois-Lacoste
# Leslie Perez Caceres    <leslie.perez.caceres@ulb.ac.be>
#
# This is free software, and you are welcome to redistribute it under certain
# conditions.  See the GNU General Public License for details. There is NO
# WARRANTY; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
#
# irace builds upon previous code from the race package:
#     race: Racing methods for the selection of the best
#     Copyright (C) 2003 Mauro Birattari
#------------------------------------------------------------------------------
# installed at: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/irac
# called with:
== irace == WARNING: A default scenario file '/Users/weiyao/Desktop/natcor-updated/Resour
been found and will be read.
# Read 1 configuration(s) from file '/Users/weiyao/Desktop/natcor-updated/Resources/proje
# 2024-04-04 20:15:59 BST: Initialization
```

Upon completion, you will see an output as below.



```
# 2024-04-04 20:45:30 BST: Stopped because there is not enough budget left
# You may either increase the budget or set 'minNbSurvival' to a lower valu
# Iteration: 5
# nbIterations: 5
# experimentsUsedSoFar: 174
# timeUsed: 0
# remainingBudget: 6
# currentBudget: 6
# number of elites: 4
# nbConfigurations: 4
# Total CPU user time: 1879.537, CPU sys time: 13.984, Wall-clock time: 177
# Best configurations (first number is the configuration ID; listed from be
       d1     d2     d3     i1     i2     i3  time
15 0.4771 0.7703 0.2316 0.2758 0.0462 0.6464 10000
7  0.7082 0.9276 0.4537 0.1554 0.5866 0.9327 10000
19 0.6064 0.6258 0.3367 0.0271 0.4321 0.3185 10000
22 0.4445 0.9618 0.3314 0.3422 0.6229 0.9025 10000
# Best configurations as commandlines (first number is the configuration ID
15  -d 0.4771 0.7703 0.2316 -i 0.2758 0.0462 0.6464 -t 10000
7   -d 0.7082 0.9276 0.4537 -i 0.1554 0.5866 0.9327 -t 10000
19  -d 0.6064 0.6258 0.3367 -i 0.0271 0.4321 0.3185 -t 10000
22  -d 0.4445 0.9618 0.3314 -i 0.3422 0.6229 0.9025 -t 10000


# Testing of elite configurations: 1
# Testing iteration configurations: FALSE
# 2024-04-04 20:45:30 BST: No test instances, skip testing
```

Congratulations! You've successfully configured parameters for the hyper-heuristic `RN`. You can then try deploying these parameters in `RN`.

Next Steps

To complete this challenge, please utilise the flexibility and capabilities in HyFlex and irace to enhance the provided hyper-heuristics for obtaining better TSP solutions. Below are some guidance to assist your further exploration of these two packages.

More Example Hyper-heuristics

Alongside the basic random hyper-heuristics, we offer two additional hyper-heuristics and their corresponding Runner in the "project-hyflex" folder.

- Within the `examples.mcf` package, you'll find the source code[2] for a modified choice function hyper-heuristic [DÖB12], including `MCF` and its corresponding runner `MCFRunner`.
- For the purpose of this challenge, we present a simplified choice function hyper-heuristic in the `examples.scf` package. This includes the `SCF` and its corresponding runners.

Specifically, `SCFRunner` is designed to execute `SCF` and `SCFRunnerConfig` facilitates easy integration with irace, simplifying the preparation of the Runner for irace.

If you choose to automatically configure `MCF`, ensure a comprehensive understanding of the algorithm's implementation.

For those interested in enhancing the heuristic selection method and acceptance criteria, a thorough understanding of the algorithm's implementation is essential. You can draw inspiration from the implementation of `RN` or explore examples available on the HyFlex website.

Explore further in irace

The provided example scenario covers essential elements, but there are additional optional settings within the configuration scenario that have not been addressed. Feel free to delve deeper into the irace user guide, as it offers a wealth of features and can be applied to various configuration scenarios. Some of these include utilising testing instances, implementing parallelisation, and conducting multi-objective tuning. Additionally, the guide provides valuable insights into setting the tuning budget and more information on interpreting output and results.

Appendix

Algorithm 1 Pseudocode for the ExampleHyperheuristic1

```
1: $s \leftarrow$ generateInitialSolution();
2: while termination_criterion_not_satisfied do
3:     $h \leftarrow$ randomHeuristicSelection(heuristic_set);
4:     $s' \leftarrow$ applyHeuristic($h, s$);
5:     $accept \leftarrow$ naiveAcceptance($s, s'$);
6:     if accept then
7:         $s \leftarrow s'$;
8:     else
9:         $s \leftarrow s$;
10:    end if
11:    if $f(s') < f(s_{best})$ then
12:        $s_{best} \leftarrow s'$;
13:    end if
14: end while
15: return $s_{best}$;
```

[2]The original source code is available at here.

Table 1: TSP instances within HyFlex.

Instance ID	Names
0	pr299
1	pr439
2	rat575
3	u724
4	rat783
5	pcb1173
6	d1291
7	u2152
8	usa13509
9	d18512

Simplified choice function hyper-heuristics

Simplified Choice Function (SCF) in the provided `SCF` class is based on Modified Choice Function (MCF) hyper-heuristic [DÖB12] but is simplified for this practical challenge.

Equation 1 shows the choice function in `SCF`, which calculates the score of the heuristic h_j at iteration t. Once the scores of all heuristics have been calculated, the heuristic with the highest score is chosen.

$$F_t(h_j) = \phi_t \cdot f_1(h_j) + \delta_t \cdot f_3(h_j) \tag{1}$$

Table 2: Simplified choice function parameters.

Parameters	Meaning
F_t	The score of heuristic h_j at iteration t
ϕ_t	A parameter for adjusting the behaviour of the choice function, adaptive changing based on the solution quality change during the search.
δ_t	Mathematically calculated as $1 - \phi_t$
$f_1(h_j)$	A function to score the heuristic h_j based on the improvement and time taken from its previous application.
$f_3(h_j)$	A function to score the heuristic h_j based on the time since h_j was last chosen.

References

[BCH+09] Edmund K Burke, Tim Curtois, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Sanja Petrovic, José Antonio Vázquez-Rodríguez, et al. Hyflex: A flexible framework for the design and analysis of hyper-heuristics. In *Multidisciplinary International Scheduling Conference (MISTA 2009), Dublin, Ireland*, pages 790–797, 2009.

[DÖB12] John H Drake, Ender Özcan, and Edmund K Burke. An improved choice function heuristic selection for cross domain heuristic search. In *Parallel Problem Solving from Nature-PPSN XII: 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part II 12*, pages 307–316. Springer, 2012.

[LIDLC+16] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.