USING HYPER-HEURISTICS TO SOLVE THE JOB SHOP SCHEDULING PROBLEM

Benjamin Valpey, Adam Case (advisor)
Drake University, Des Moines, IA
benjamin.valpey@drake.edu, adam.case@drake.edu

The problem of determining how to allocate limited resources to complete a certain number of jobs is a difficult problem in computer science. While there have been several successful solutions to this type of problem, they are often tailored to a very specific kind of scheduling problem each with unique constraints. A constraint is any rule that a solution must meet in order to be considered valid. For example, one type of scheduling problem might be assigning employees to work shifts. A constraint for this problem might be a limit to the number of total shifts that an employee can work, and any schedule that assigns a particular employee to work more than the limit would not be considered valid. Another variation of the scheduling problem might have no limit on the number of jobs a resource might be assigned to. For instance, creating a schedule that assigns space shuttles to payload missions. In this case, there may not be a limit to the maximum number of missions that a shuttle can be assigned. Algorithms that are designed to find solutions to the scheduling problem may have to be dramatically altered if there are slight changes to the constraints. Thus, there is a need to create a single algorithm capable of solving any instance of the scheduling problem, regardless of the kinds of variations that are placed on its constraints or even the number or types of resources that must be assigned to jobs.

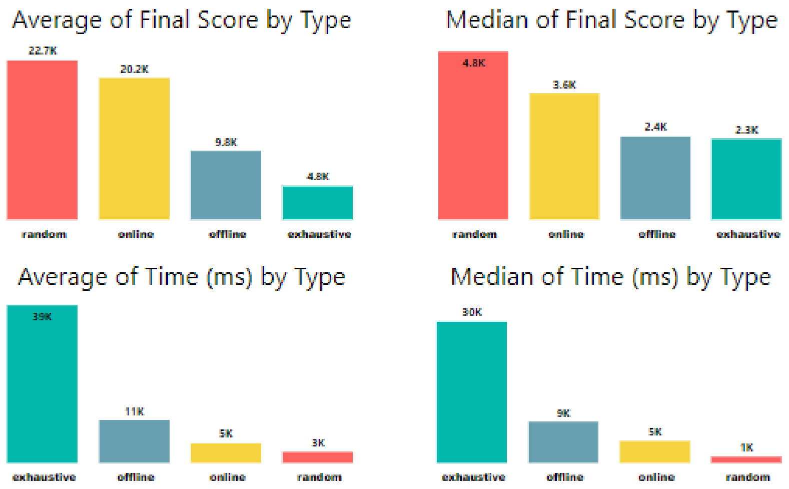
The goal of this project is to propose a method for efficiently computing solutions to a variety of these problems. Current approaches employ the use of heuristics, techniques used to search through the state space of a problem [3]. Different heuristics aid in the search for a solution to a problem in a variety of ways. For example, a heuristic designed to solve a tile-sliding puzzle game, where tiles in a grid must be moved to form a picture, could be designed to take several different approaches. An algorithm designed to use a heuristic to solve the puzzle will do so by simulating moves, one after another, until a solution is found. The heuristic is the tool that is used to determine which move to simulate next. Usually, algorithms that employ the use of heuristics must be designed with a specific problem in mind in order to function properly. This means that extensive work must be done to develop a program which will solve a specific problem. If a new constraint is added which affects that problem, the algorithm might have to be drastically altered. NASA states that they are especially interested in solutions which can adapt to unforeseen difficulties [2]. Simple low-level heuristics are ill-suited for problems such as theirs. A better solution is required.

Instead of using specific heuristics which only work for the problems they were designed for, this paper will focus on hyper-heuristics. A hyper-heuristic works by trying to discover a sequence of simple heuristics to use rather than focus on solving the problem directly [4]. Hyper-heuristics are incredibly versatile as they are designed to improve the way in which a problem is solved. Because of their ability to use different kinds of heuristics, the hyper-heuristic is able to adapt to a multitude of problems. The way in which a hyper-heuristic is used to solve a problem is key. It is not the hyper-heuristic itself that solves the problem, instead it is used as a tool to choose a simple heuristic that modifies an existing solution. This makes hyper-heuristics an incredibly powerful tool, especially for problems where any improvement goes a long way.

For this project, four different hyper-heuristics were developed. The first two are unintelligent algorithms used as a baseline to compare the other two hyper-heuristics. One of these algorithms is the Random Hyper-Heuristic. As its name implies, the algorithm will choose a random low-level heuristic at each stage to modify the current state. The other unintelligent algorithm is referred to as the Exhaustive Hyper-Heuristic. After every step, this hyper-heuristic will try every available low-level heuristic, choosing the heuristic which improved the score the most. The two methods described are naïve approaches for selecting the next heuristic. In order to improve upon these methods, two algorithms were designed which select heuristics in a more intelligent way. The first of these methods, the Online Hyper-Heuristic, is designed to learn about the problem as it develops a solution. When it first begins, the online approach mirrors the Random Hyper-Heuristic. It chooses a random low-level heuristic and applies it to the current solution. If the solution was improved by the chosen heuristic, that heuristic is given “points” that make it more likely to be chosen in the future. If the solution was not improved, the heuristic’s points are unaffected. However, the program has a 50% chance of overwriting the current solution with the new solution, even if the new solution is worse.

The final hyper-heuristic designed is one which uses an “offline” approach to choose the next heuristic. This approach uses a data file containing information about which heuristics are most likely to perform well based on previously selected heuristics. The data file records sequences of heuristic used on a specific problem instance and their final score. The offline approach starts by randomly applying three heuristics. It will then search through the data file for observations which used the same first three heuristics. It will select the top three observations with the highest score, and will choose its next heuristics in the same order as the observations, exploring three separate solution paths.

HyFlex is a Java framework that provides an environment to develop hyper-heuristic [1]. This software includes packages which allow for the testing of the hyper-heuristics on instances of the scheduling problem. It also provides a benchmark to evaluate and compare different hyper-heuristics against one another by including a way to score the best solution found by the hyper-heuristic as well as the length of time that the algorithm ran. To test the different methods, each hyper-heuristic was allowed to apply a sequence of five low-level heuristics. The score of the best solution for each method was recorded, along with the total amount of time (in milliseconds) that it took for the algorithm to complete. Each hyper-heuristic was run on all of the 12 problem instances provided by HyFlex. The figure displays the average and median scores and time for each hyper-heuristic (note that a lower score is better). The observed results show that both the online and offline learning methods performed better than just choosing heuristics at random. The median score for the offline-approach is competitive with the exhaustive approach, despite only being trained on one instance. Notice the significant reduction in time that the online and offline hyper-heuristics had compared to the exhaustive hyper-heuristic. The results confirm that there are practical algorithms capable of solving any type of scheduling problem.



[1] T. Curtois, M. Hyde, G. Ochoa and J. A. Vázquez-Rodríguez, "A HyFlex Module for the Personnel Scheduling Problem," School of Computer Science, University of Nottingham, Nottingham, 2012.

[2] T. G. Dietterisch and W. Zhang, "A Reinforcement Learning Approach to Job-Shop Scheduling," in *IJCAI'95 Proceedings of the 14th international joint conference on Artificial intelligence*, Montreal, 1995.

[3] G. F. Luger, Artificial Intelligence: Structures and Strategies for Complex Problem Solving, Boston: Pearson Education, Inc, 2009.

[4] E. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan and R. Qu, "Hyper-heuristics: a survey of the state of the art," *Journal of the Operational Research Society,* vol. 64, no. 12, pp. 1695-1724, 2013.