Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

MASTER THESIS

Evolutionary Hyper-Heuristics for Heuristic Selection

Bc. Jakub Weberschinke

Advisor: Ing. Jiří Kubalík, Ph.D.

Study Programme: Open Informatics

Field of Study: Artificial Intelligence

2nd of January, 2012

Prohlášení

Já, níže podepsaný Jakub Weberschinke, prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 3.1.2012 .....Weberschinke.....

podpis

Acknowledgements

I would like to thank my supervisor Ing. Jiří Kubalík, PhD. for his guidance and all the advice he gave me.
I would also like to thank my family for their enormous support.

Abstrakt

Hyper-heuristiky jsou trend, kterému se v posledních letech věnuje značná pozornost. Jakožto black box optimalizační metody pracující na vyšší úrovni abstrakce metody mají široké uplatnění v praxi. Tato práce má za cíl prozkoumat možnosti použití evolučních algoritmů a s nimi souvisejících metod v oblasti hyper-heuristik. Jejich vlastnosti umožňující prohledávání relativně rozsáhlé prostory řešení, navíc relativní přímočarost evolučních algoritmů znamená větší kontrolu nad řídícími parametry a lepší rozšiřitelnost. Je představen postup založený na prohledávání okolí v prostoru řešení pomocí evoluce sekvencí low-level heuristik. Je navrhnuto celkem 5 verzí algoritmu: základní verze a 4 komplexnější rozšíření. To má za cíl snahu o odhalení těch nepřínosnějších rozšíření. Všechny tyto verze algoritmu jsou testovány pomocí benchmarkovacího frameworku HyFlex poskytovaného organizátory soutěže CHeSC 2011, jejímž cílem bylo nalezení algoritmu schopného řešit co nejširší škálu optimalizačních problémů. Testování proběhlo na problémech z 6 domén, kde každá obsahovala 5 instancí problémů. Všechny 4 rozšířené verze algoritmu vykazují velmi dobré výsledky, kdy se všechny umístily na 6. až 4. místě z 21 soutěžních algoritmů. Tím se potvrdilo, že tato metoda je schopná nalézat řešení kvality srovnatelné či lepší v porovnání se v současnosti předními metodami. Ačkoli nebyla nalezena konkrétní implementace, která by jasně překonávala ostatní, prokázalo se, že každé rozšíření má svůj přínos na jiném typu problému.

Abstract

Hyper-heuristics are an emerging that has received increasing attention in the last years. As they are black box optimization techniques that work on higher level of abstraction, they have many real world application. This work aims to explore the possibilities of application of evolutionary algorithms and related methods in the field of hyper-heuristics. Their properties make them a particularly promising candidates. They can explore large solution spaces and at the same time are very straightforward, which gives the user greater control and easier extensibility. An approach based on exploration of neighborhoods in solution spaces using evolution of low level heuristics. Five version in total were proposed: a baseline version and four more complex extensions. This aims to pin-point the most contributing extensions. All of these algorithm versions were evaluated using a benchmark framework HyFlex supplied by the organizers of CHeSC 2011 challenge, which aimed at finding an algorithm capable of solving the widest spectrum of optimization problems. The benchmark consisted of solving problems from 6 different domains, each containing 5 problem instances. All of the 4 extended version performed very well, placing between 6th and 4th place out of 21 competing algorithms. This confirmed that the proposed method is capable of finding results of quality comparable to current state-of-the-art methods. Even though no implementation capable of outperforming all other was found, it was proven that each extension contributed on different types of problems.

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student: **Jakub Weberschinke**

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: **Evolutionary hyper-heuristics for heuristic selection**

Guidelines:

1. Study a field of hyper-heuristic optimization approaches. Focus on the category of selection hyper-heuristics for selecting and applying an appropriate low-level heuristic at each decision point and review the state-of-the-art selection hyper-heuristic implementations.

2. Study evolutionary algorithms and review their potential for realization of hyper-heuristics.

3. Design and implement problem-independent evolutionary-based hyper-heuristic.

4. Carry out experiments with the proposed hyper-heuristic on standard test problems. Statistically evaluate performance of your hyper-heuristic and compare it with other existing hyper-heuristics. You may consider materials freely available at Cross-domain Heuristic Search Challenge website (http://www.asap.cs.nott.ac.uk/chesc2011/index.html) for the experimental and analysis purposes.

Bibliography/Sources:

Will be provided by the supervisor

Diploma Thesis Supervisor: Ing. Jiří Kubalík, Ph.D.

Valid until the end of the summer semester of academic year 2012/2013

prof. Ing. Vladimír Mařík, DrSc.
Head of Department

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
L.S.

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, November 3, 2011

Contents

List of figures

List of tables

Chapter 1

1 Introduction

Hyper-heuristic (HH) is a term that was coined by Peter Cowling et al. in [22]. It is one of black box optimization techniques that have received much attention in the past years. The main difference from meta-heuristic approaches like local search (LS), particle swarm optimization (PSO) or ant colony optimization (ACO), hyper-heuristic are searching the space of actions to alter the solution, particularly low-level heuristics (LLHs), rather than space of solutions themselves. However, hyper-heuristics have often the form of meta-heuristics working on space of another meta-heuristics. Like other optimization methods', HH's aim is not to find the optimal solution of one problem, but rather be able to find solution to large variety of different ones of reasonable quality in reasonable time [3]. However, HHs offer greater level of generality because they are not domain specific. Some of HH algorithms were successfully applied on various real-world optimization problem instances, e.g. the traveling salesman problem [2], vehicle routing problem [4], Boolean satisfiability problem [6], university timetabling [5] and many others. There are many different ways of dividing HHs, however the most basic division is into two categories, selective and generative. The former uses LLHs that are already prepared for the problem, while the latter identifies components of existing LLHs to use them to create new LLHs [1].

It appears that evolutionary algorithm (EA) would be a good candidate for a basis of a HH algorithm. The main reasons is that EAs are general meta-heuristics that are suitable for exploration of large search spaces and well described method of black box optimization. They use a set of solution candidates that allows to make use of more information compared to single-point approaches. Also, evolutionary operators that are used by EAs are easy to adapt to efficiently work with different population. Finally, chromosomes, which form the population of EA, can contain any type of data organized in any structure, allowing to work with LLHs directly in form of sequences. All these properties suggest that EAs would provide good balance between exploration and exploitation of the search space and could be easily adapted to search the neighborhoods of solutions by sequential application of LLHs.

This work deals with designing an evolutionary selective hyper-heuristic. Our goal is to propose an evolutionary algorithm working with population of fixed-length sequences of LLHs. These sequences will be evolved with the aim to improve the current solution of a

corresponding problem. Thus, the fitness of a given sequence of LLHs will be determined based on the quality of the current solution and the quality of the solution obtained by applying the sequence on the current solution. Standard EA mechanisms such as reproduction, crossover and mutation will be used to evolve the population. Other mechanisms used in machine learning and black box optimization will be used to improve a performance of the evolutionary hyper-heuristic.

The proposed Hyperheuristics Using Evolutionary Algorithm to Drive Search (HEADS) algorithm will be implemented within a java benchmarking framework HyFlex [32], which provides interface for working with LLHs such as heuristic application and solution quality retrieval. The algorithm will be evaluated using the CHeSC 2011[8] competition data, which contain 6 domains of problems, each with 5 different instances. Results obtained with HEADS will be analyzed and the algorithm's performance will be compared with 20 other hyper-heuristic approaches that were submitted to the competition, the results being kindly provided by the organizers of the competition. The suitability of our evolutionary-based approach will be discussed based on these results.

Chapter 2

2 Hyper-heuristics

The term hyper-heuristics has been broadly defined as "heuristics to select heuristics" by Burke et al. in [3] and was later redefined by the same authors as "an automated methodology for selecting or generating heuristics to solve hard computational search problems" in [1]. The former definition refers to the fact that hyper-heuristics operate on a space of actions that modify a solution of a given problem rather than on space of solutions themselves, as is the case of meta-heuristics. One reason for introduction of this new approach was that meta-heuristics were usually problem specific and required fine tuning of their parameters [9]. This meant that domain knowledge was required for a correct and effective application as well as good knowledge of the applied meta-heuristic [20]. Therefore, the reusability of such sophisticated algorithms was often limited. Domain specific control parameters required meant that these algorithms could not have been used on other domain than they were designed for and even different set of constraints in the same domain would mean suboptimal performance and would require very different parameter settings [3].

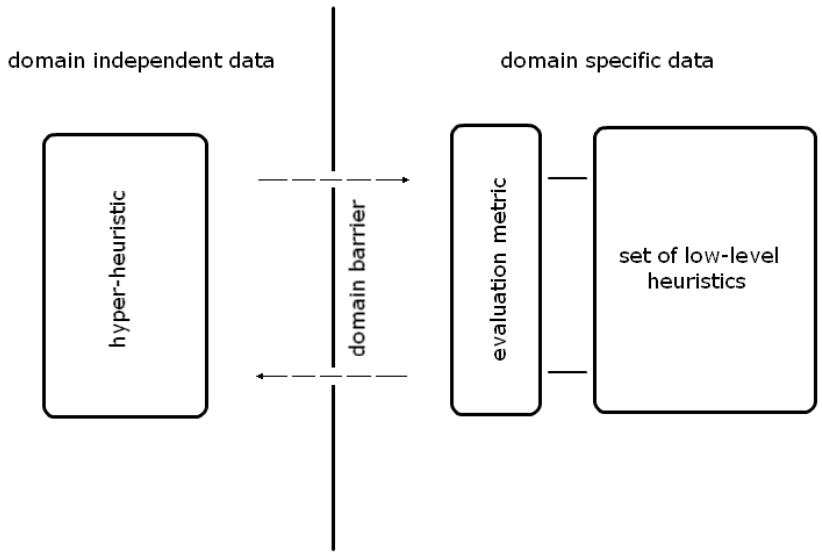

Figure 1: A schematic of separation of high level HH and LLHs

The idea of developing one universal effective heuristic that would be cheap and easy to use was soon dropped after the publication of "No Free Lunch" theorem [10], which states that if one heuristic performs significantly better on one problem than other heuristics, it will be

compensated by worse performance on other problems, causing all heuristics in general to deliver results of same quality on the whole universe of optimization problems. Therefore, hyper-heuristic were introduced as a high level approach to select a LLH or combination of LLHs that would be most suitable for the problem at hand. The Figure 1 shows the communication of hyper-heuristic high level through the domain barrier by using domain specific evaluation metrics.

Various different classifications of hyper-heuristics have been presented in literature. Majority of these classifications were summarized in [1], where new classification was introduced that reflects the most recent approaches in the field. In it hyper-heuristics are classified both by their learning mechanisms and by the nature of the heuristics search space.

From the learning point of view, three categories were introduced:

- No learning - these algorithms without learning simply follow predetermined sequence of actions that do not reflect the performance on the current problem nor any preceding problems. An example of such technique would be VNS as described in [11].
- Offline learning - these algorithms use information gathered through training on few problem instances to select the most suitable heuristics for the given problem domain as a whole. Example of this approach is the XCS framework described in [12].
- Online learning - these methods use feedback obtained during the execution on a given problem to alter the heuristic selection on the run. Algorithms that use reinforced learning for heuristic selection like [13] or [16] are a good example of this group as well as those using EA to select heuristics like [14], [17] or [15].

Classification by the nature of the heuristic search space introduces two categories:

- Selective - these hyper-heuristics take existing heuristics for a given domain and try to arrange them in a way that yields better results than obtained by using individual heuristics separately. These hyper-heuristics will be discussed in more detail in the next paragraph because the concept of selective hyper-heuristic will be adopted in the proposed evolutionary hyper-heuristic.
- Generative - this is an emerging trend that does not work on complete heuristics, but rather tries to decompose the existing ones into building blocks and then use those blocks to construct new heuristics that would fit the problem solved.

Note that this classification should not be confused with classification of heuristics used, which are:

- Construction heuristics - they start with empty solution and they gradually build it until complete solution is found.
- Perturbation heuristics - they can only modify existing solution and include mutational heuristics and hill-climbers or local searchers. These will typically use a randomly generated solution or solution created by construction heuristic as a starting point. Construction heuristics can be only applied when the solution is incomplete, whilst the perturbation heuristics can be applied any time [1].

2.1 Selective hyper-heuristics

Hyper-heuristics based on selection are trying to find the optimal sequence of low-level heuristics. Most methods used are considered single-state algorithms as they maintain only one (partial) solution in memory. This is true for methods like tabu search [18] or simulated annealing [20], however in other like genetic algorithms [14], [17], [15] the statement is does not hold completely as is often auxiliary solutions maintained with the population.

Selective methods based on construction heuristics will not be discussed in greater detail as they are not the subject of this work. They use similar approaches as their perturbation heuristics using counterparts, but there are still differences, one of main being that the number of applied heuristics is predetermined by the number of variables in the solutions, which is the consequence of properties mentioned in previous paragraphs.

The following description was introduced in [1]. In general, selective hyper-heuristics based on perturbation heuristics are iteratively applying these heuristics on existing complete solution until a stopping condition is met, effectively acting as heuristic scheduler. In every cycle, there are two phases (where the process can be deterministic or non-deterministic [20]):

- Heuristic selection - there are many selection methods which were categorized in [1] as either non-adaptive, which do not use online learning, and adaptive, which do. The following examples were summarized in [21]: greedy hyper-heuristics that use the best heuristic available at any given moment, choice function that weights heuristics based on their previous applications, tabu search that disfavors heuristics that performed poorly etc.

- Move acceptance - it can take into account several factors, typically including quality of the resulting solution, but also processing time required for heuristic application, and can be partially or completely randomized.

The stopping criterion is typically time limit. Another stopping criterion could be the quality of the best-so-far solution. However, because of the nature of black box optimization problems, it is usually hard to tell if the solution found is optimal or how significantly the solution can be improved. Therefore, using quality as stopping criterion would be impractical in these cases.

2.2 CHeSC 2011 Challenge

The CHeSC 2011 Challenge was issued by Automated Scheduling, Optimisation and Planning (ASAP) group at the University of Nottingham, Nottingham. Its aim was to bring together experts from different fields of research where optimization is used to devise more general methodologies usable on different types of problems using sets of domain specific LLHs. For the purposes of this challenge a benchmarking java framework HyFlex [32] was implemented to provide unified tools for solution creation and evaluation as well as LLH application and handling that would work with different problem definitions and data.

LLHs in this framework are categorized into 4 different groups:

- Local search LLHs make small changes to the solution that are guaranteed to produce a solution that is not worse than the one it is applied on. The iterative improvement process is stopped when a condition is met or when local optimum is found.
- Mutation LLHs also make small changes to the solution by means of swapping, changing, removing, adding or deleting parts of the solution. These changes are not guaranteed to be improving.
- Ruin-recreate LLHs destroy parts of solutions rather than modifying the information it contains and then use problem specific methods to restore it.
- Crossover simply takes two existing solutions and combines them into new one.

The heuristics are parametrized by corresponding real valued parameters from interval <0,1>. For local search it is the *depth of search* (DoS), which governs the number of steps that will be used to improve the solution. For mutation and ruin-recreate heuristics it is the *intensity of mutation* (IoM) which specifies the degree of modification of the solution. However, the same value of parameter can have different impact on the newly generated solution if used with

different LLH of the same type.

LLHs are identified by indexes and apart from their type and information about which one (if any) of the parameters uses, a user does not have any additional information about the nature of the LLH.

2.2.1 CHeSC 2011 competing algorithms

The following algorithms were competing in the 2011 challenge [8]. These algorithms are listed in order of placement in the challenge. Please note that at the time of writing this thesis, some of the participating algorithms didn't have their description available, while description of some others were very brief.

1. The Adaptive Hyper-Heuristic (AdapHH) by Misir et al. [33] applies LLHs iteratively. Each step a LLH is selected either using the adaptive dynamic heuristic set, which favors LLHs finding improving solutions in short time, or using relay hybridization, which favors LLHs that work well with previously used LLH. Move acceptance uses adaptive iteration limited list-based threshold accepting to determine if a worsening solution should be accepted or if the search should be restarted using new randomly generated solution. The IoM and DoS parameters are adapted using reward-penalty mechanisms.
2. The Variable Neighborhood Search-Based Hyperheuristic (VNS-TW) by Ping-Che Hsiao et al. [34] is an iterative method that maintains a base of solutions. In first part, one solution is selected and mutation and ruin-recreate LLHs are applied on it. In next step, local search LLHs are applied until no improvement is found for given number of steps or all LLHs produce non-improving solution. Finally, the solution is inserted if it is better than at least one solution in the base; otherwise, it is discarded and the mutation or ruin-recretate LLH that was used in the first part is put into a tabu list. Parameters of the algorithm are periodically updated during the search.
3. The ML algorithm by Mathieu Larose [35] is building on foundations of Coalition-based metaheuristic system (CBM) by David Meignan et al. [36], particularly its reinforced learning mechanisms. The ML algorithm works in diversification-intensification cycle, where diversification is done by application of a single mutation or ruin-recreate LLH and intensification by application of several LLHs. The selection process is governed by a matrix that contains probabilities of selection, where rows are conditions like last used LLH or last solutions acceptance, and columns LLHs. Each

successful LLH application increases the probability under a given condition.

4. The Pearl Hunter (PHunter) by Fan Xue et al. [37] also uses diversification and intensification cycle. However, rehearsal run is first executed to determine the properties of the problem solved to determine the set of LLHs that will be used for the problem using a decision tree. Set of promising solutions is maintained. Firstly, their neighborhood is searched using local search with low DoS values, promising solutions are then selected and explored using local search with high DoS value. In the first half of the remaining time running LLHs are selected randomly, in the second based on their performance so far.
5. The Evolutionary Programming Hyper-heuristic (EPH) by David Meignan [38] uses evolutionary algorithms and co-evolution. Two populations are maintained; population of solutions and population of LLH sequences. These sequences, with head containing one or two perturbation LLHs and tail local search LLHs or variable neighborhood search, are sequentially applied on a solution randomly selected from population. The new solution replaces the worst one if it is better and unique. The population of sequences is updated using various mutation techniques, new population is created by using tournament selection, where the winner is decided by applying the two sequences on a solution from the solution population, where a number of new solutions successfully added to their population is the most important factor.
6. The HAHA algorithm by Andreas Lehrbaum didn't have the documentation available.
7. The NAHH algorithm by Franco Mascia didn't have the documentation available.
8. The Iterated Search Driven by Evolutionary Algorithm Hyper-Heuristic (ISEA) by Jiri Kubalik uses evolutionary algorithms working on space of actions modifying sequence of LLHs. A sequence of LLHs with a defined structure is used to improve a solution while population of action sequences is evolved to improve the prototype LLH sequence. The structure of prototype is the following: the head contains one LLH of any type excluding crossover, the body of arbitrary length contains only local search and mutation LLHs and the tail contains local search only. The actions can add a LLH, remove a LLH or modify an existing LLH in the prototype. Fitness of these action sequences is determined by the quality of solution the modified prototype produces. Current solution are replaced every time a better solution is found, prolonged stagnation triggers solution reinitialized.
9. The Simulated Annealing Hyper-Heuristic with Reinforcement Learning and Tabu-Search (KSATS-HH) by Kevin Sim [13] uses iterated local search methods. In every

iteration, two LLHs not in a tabu list are selected randomly and then the one with higher rank is applied on the solution. If the new solution is improving, the rank of the LLH is increased, otherwise the rank of the LLH is decreased and it is placed in tabu list for fixed number of rounds. Modified simulated annealing is used to determine whether to accept the new solution which takes into account normalized solution quality to reflect different problems and average running time of LLHs for the problem. Annealing parameters are reset if solution stagnation exceeds fixed threshold.

10. The Hybrid Adaptive Evolutionary Algorithm Hyper Heuristic by Jonatan Gómez [41] uses, as many algorithms previously mentioned, iterated local search techniques. Two subsets of LLHs are maintained throughout the run, first containing all the LLHs available, the second only local search LLHs. Every iteration a LLH is selected from both subsets and sequentially applied on the current solution, where crossover also uses the best solution found. Parameters DoS and IoM are dynamically changed during the run, whilst IoM is set as an inverse value of DoS. When a improving solution is found, both LLHs have the probability of being select from the current subsets increased. If no improving solution is found, IoM is increased. If IoM reaches a certain threshold, solution is reinitialized using the best solution found and new subsets are selected.
11. The general purpose Hyper – Heuristic based on Ant colony optimization (ACO-HH) by Nunez et al. [42] uses ant colony optimization to construct LLH sequences. A grid is constructed where columns represent positions in the generated sequence and rows LLHs. Ants are then traversing the grid from left to right, their path defining a LLH sequence, which is sequentially applied on the current solution. The probability of selecting a particular LLH is governed by pheromone trails defined by an average of step improvements caused by applying the given LLH at the given sequence position and by an average of total improvements caused by the whole sequence the LLH at the given position was a part of.
12. The Genetic Hive HyperHeuristic (GenHive) by a team from Institute of Computing Science from Poznan University of Technology [43] combine multi-agent techniques and evolutionary algorithms. It maintains a search location set containing solutions and set of LLH sequences called agents. There are active agents that are applied to modify their solution from search location set that are ranked by the improvement they made. The rest of agents is dormant. Each iteration step best agents are kept at their locations while the rest along with the dormant agents are used to create new population using mutation and crossover. Agents from this new population are randomly selected to fill

the vacant search locations.

13. A Hyperheuristic Based on Dynamic Iterated Local Search by Mark Johnston et al. [44] uses iterated local search with dynamic diversification. Each iteration a diversification LLH called kick is selected with the IoM selected using weight vector. After that local search LLHs are applied until local optimum is reached. Weights are updated based on the quality of the solution in the local optimum. If there is no improvement for a longer period of time, the probability is tilted towards selecting higher IoM values.
14. The SA-ILS algorithm by He Jiang didn't have the documentation available.
15. The XCJ algorithm by Kamran Shafi didn't have the documentation available.
16. The Reinforcement Learning approach (AVEG-Nep) by Tommaso Urli and Luca Di Gaspero [45] uses multi-agent local search using reinforced learning. A set of agents is maintained each searching the neighborhoods of different solution. Actions, which can be LLHs or solution restart, are iteratively selected based on the state of the agent. The state reflects the last trend of the solution at hand, i.e. improving, stagnating or worsening, based on the recent agent rewards. For each state, a mapping of actions to selection probabilities is maintained, which are increased when an action is rewarded. In a given state, action is selected either randomly or using these probabilities. Both the state and selection probabilities are deteriorating, meaning that recent actions play more important role than old ones.
17. Generic Iterative Simulated-Annealing Search (GISS) by Alberto Acuña et al. [40] uses simple simulated annealing approach. LLHs are selected randomly while the move acceptance is governed by exponentially decreasing probability to accept non-improving solutions. When a stagnation threshold is exceeded, the solution is restarted.
18. The SelfSearch by Jawad Elomari [31] iteratively applies LLHs on population of solutions. The algorithm works in two modes where each select LLHs using different performance statistics, but both of these use Adaptive Pursuit to assign specific probability values. One is predominantly exploitative for problems with LLHs that take long time to compute, where the other is more explorative. Exploitative uses solution quality to select LLH, working with higher DoS and lower IoM, increasing DoS on stagnation. Explorative uses number of new solutions identical to the original created by the LLHs, working with lower DoS and increasing IoM, but switching to higher DoS near the end of algorithm.
19. The Single Objective Variant of the Online Selective Markov chain Hyper-heuristic (MCHH-S) by Kent McClymont and Ed Keedwell [30] uses fully connected Markov

chain, where nodes of the chain are LLHs and labeled oriented edges are probabilities. A set of solutions is maintained, which is updated by applying one LLH on one solution each iteration. The LLH is selected by traversing the Markov chain, making one step each iteration. If the new solution is better, it replaces the old one, worse solution can replace the old with probability that increases with stagnation. Improvement caused is used to change probability on the edge. New LLH is selected every time, next solution is picked only when improvement is achieved.

20. The Ant-Q algorithm by Imen Khamassi [28] uses ant colony optimization. Ants are traversing a fully connected oriented graph where nodes are LLHs. At each iteration, every ant moves given number of times, causing the application of heuristics in the node it passes through on a solution from population. Selection is based on product of pheromone on a given edge and LLH information, where either roulette selection is used or the best variant is chosen. LLH information reflect past improvements caused by applying these two LLH in sequence. Pheromone trails of the most successful are updated using quality of solution they created and future rewards.

Chapter 3

3 Evolutionary algorithms

Evolutionary algorithms were inspired by principles from the theory of evolution like survival of the fittest, mutation and crossover of genetic information and competing of different individuals that are part of population of similar members [23]. They are used in many fields for black box search or optimization.

The solution in EA is stored inside an object called the chromosome, which is composed of genes. It is either the solution itself or a sequence of actions that can modify some existing solution. This information can be encoded into some simplified representation such as a string or an array of numbers, but it can also be complex graph or tree structures as in evolution strategies [25]. This is influenced by the characteristics of the problem solved.

Each individual has a value of a quality metric called the fitness function assigned based on how good the solution represented by the member is compared to others. Together with the encoding used to represent solutions this metric is the only domain specific part of EA. Note that definition of fitness functions may vary not only between problem domains but also between problem instances as it can reflect different restrictions imposed on the solution. A schematic of relation between EA population and solution space can be seen in figure 2.

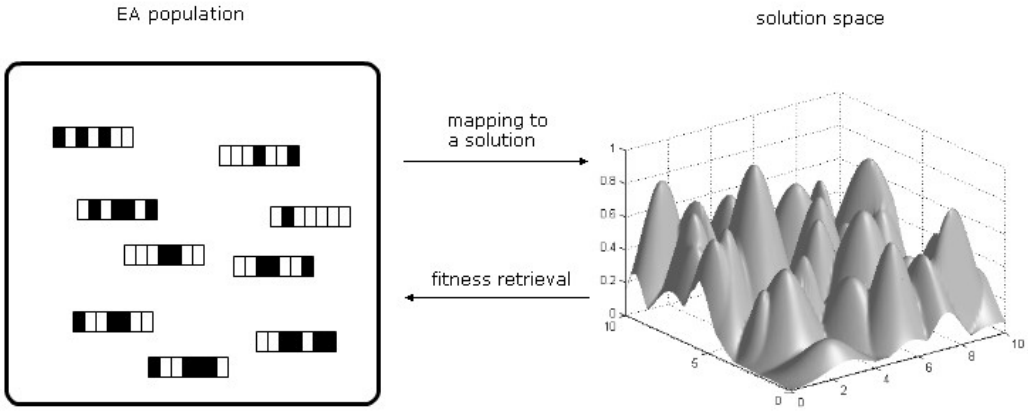

Figure 2: A schematic of the EA population and solution space relation.

The distinct chromosomes are kept in a set called the population. This set is iteratively modified in steps called generations. In each generation the members of the previous population

are used to create new population with members that are derived from their predecessors.

The process of creating new population includes selection of parent individuals followed by crossover and mutation of their genes. The resulting individuals are called offspring and are inserted into the new population. There are many different methods of parent selection, however they all share the characteristic that the use the fitness function to decide which population members should be selected and that they employ greater or smaller degree of randomization. These parents are then used to create new members of the next population by combining their genes using a crossover method and then perturbing the new genes by the means of mutation. Replacement of the old population by new one is referred to as generation.

The execution of algorithm is stopped when a condition usually in form of time limit or number of executed generations is met.

3.1 Example of EA hyper-heuristic

Ross et al. [29] have proposed a messy genetic algorithm concept to learn universal rules for the bin packing problem. In their work, they used a steady state evolutionary algorithm to evolve population of rule sets. Each of these set contain variable number of building block, each of which represent a rule in form of mapping from a problem state to a heuristic. When applied to a problem, problem state is iteratively inspected and a heuristic with the most similar state is applied. Fitness of the rule set is obtained by applying it to a training problem and comparing the result obtained by using single heuristic. Because of the time complexity associated with the fitness calculation, only a small subset of training instances were used in each fitness calculation. However, if the individual survived, its fitness was updated by adding results from new testing instances, making the fitness of older instances more accurate and preferred. At the time of the publication, this method outperformed the best algorithms in 98% of problem instances and finding optimal solution in more than 80%. However, this hyper-heuristic is not black box method as it exploits the current state of problem solution and correct state representation is essential for efficiency.

Chapter 4

4 Description of HEADS algorithm

In this section the core concepts and traits of the proposed algorithm will be described along with methods that were used to optimize certain settings of the algorithm. First, the baseline version of the evolutionary hyper-heuristic will be introduced. Then, several extended versions will be described. The last subsection summarizes all control parameters of the proposed algorithm.

4.1 Baseline HEADS

The HEADS algorithm is a variation of an iterated local search algorithm in the sense that it works with one solution at a time and updates it by traversing its neighboring solutions, i.e. those that can be accessed by application of actions that modify the solution. The solution that is used to traverse the solution space is called an *evaluation solution.* In each iteration that aims to improve the evaluation solution, an evolutionary algorithm is run to evolve a sequences of LLHs and their corresponding parameters DoS and IoM that modify the current evaluation solution in most desirable way.

The pseudocode 1 shows the main loop of HEADS, while the pseudocode 2 shows the EA loop.

```
procedure:  Baseline HEADS
input:      the problem definition
            the control parameters
output:     the problem solution
1.  load the problem and the corresponding LLHs definitions
2.  mark the fastest parameter control set as current control parameter set
3.  initialize random solution
4.  reset the LLH running time counter
5.  run the testing EA algorithm
6.  calculate the optimal control set using the running times from the
          testing EA algorithm
7.  reset the EA related statistics but keep the solution
8.  while (the running time limit for the whole algorithm is not exceeded)
9.    run an EA*
10.   replace the current evaluation solution with the one generated by EA
11.   if (EA generated a best solution with better quality than current best)
12.     replace the best solution
13.   end
14. end while
15. return the best solution found
```

Pseudocode 1: The main loop of the HEADS algorithm

```
procedure:  run an EA
input:      the problem definition containing the fitness
            function formula and available LLHs
            the control parameters including current VCPS
            the current evaluation solution
output:     the best solution generated by the EA
            the evaluation solution generated by EA
1.   generate random EA population
2.   for (specified number of generations)
3.     select parents from population by tournament selection  using their
             fitness
4.     generate offspring by parent crossover or mutation where the number of
             offspring is equal to the size of old population
5.     create empty new population
6.       for (all the offspring created)
7.          create new solution by applying the offspring on the evaluation
                solution
8.          if (is this the TCT-EA)
9.            save the running times of the LLHs used to create the solution
10.         end if
11.         assign a fitness value to offspring*
12.         if (the fitness of the new solution is not worse than the fitness
                 of the evaluation solution)
13.             accept the new solution as the evaluation solution
14.         end if
15.       end for
16.  end for
17.  return the best solution found and the current evaluation solution
```

Pseudocode 2: The loop of EA.

4.1.1 Population of EA

The EA works with a *population of sequences unary LLHs* defined by their parameters. Unary in this context means that only LLHs which take one solution as a parameter are accepted, leaving out the crossover type. More specifically, population is a set of sequences of *LLH triplets* $\{I \times M \times L\}^N$, where

- *I* is an integer representing an index of unary LLH as defined by problem definition in the library, which also defines the type it belongs to
- *M* is the IoM parameter
- *L* is the DoS parameter
- *N* is the length of the sequence

Sequences in one population always have the same length. However, the size of population as well as the length of sequences can be different between the EA test run and all the subsequent

runs. Note that after the test run, the control parameter set is fixed and will not change. Also note that in the HyFlex framework, the parameters IoM and DoS are exclusive.

4.1.2 Fitness function and effective sequence

Each population member has a corresponding fitness value assigned, which is equal to the value of quality measure of solution which is created by application of the sequence on the evaluation solution. However, there is an exception to this rule as if the sequence creates the same solution, its fitness is set to have the worst value possible. Pseudocode 3 illustrates the fitness assignment. More specifically, it is derived from the quality of the best solution that is obtained by applying the sequence on the evaluation solution when we ignore arbitrary number of heuristics on the tail of the sequence. In other words, LLHs are applied sequentially and the best solution quality that is obtained at any point of this process is used as fitness of the sequence.

```
procedure:  assign a fitness value to offspring
input:      the evaluation solution
            the solution created by application of offspring
1. if (the new solution is equal to the evaluation solution)
2.    assign the worst fitness to offspring
3. else
4.    assign a fitness value equivalent to objective function of solution
            created
5. end if
```

Pseudocode 3: The baseline version fitness assignment.

Example: Let [A,B,C,D,E,F] be a sequence of LLH triplets and [6,5,4,3,4,5] be the quality values of the solutions obtained by sequential application of minimization task. Then if a fitness value on i-th place in the list corresponds to a fitness of a solution obtained by applying first i heuristics from the sequence on the evaluation solution. Then the fitness value of that heuristic sequence is 3 and it was obtained by applying sequence [A,B,C,D] on the evaluation solution.

This approach effectively makes the used length of sequences dynamic. These subsequences are called *effective sequences* and only these are used when applying the sequence on solution, whether the aim is to modify the solution or to calculate fitness. However, please note that during the evolution of population members the whole triplet sequences is always used.

4.1.3 Evaluation solution

Evaluation solution marks the current position in the iterated search through the solution domain. Its initialization at the beginning is handled by the HyFlex framework and is random. Its exact form is also defined by the framework. When trying to replace the evaluation solution with a newly generated solution, solutions with worse quality are always rejected, solution with better or equal quality are always accepted.

The replacement is attempted every time a new solution is created, which in our case is when the fitness of population members is evaluated. This means that in one population, members can use a different evaluation solution to calculate their fitness. Even though it means that population members have varying conditions, it saves time compared to traditional EA approach where the fitness function calculation typically remains the same throughout the whole algorithm. The other reason is that the application of the LLHs is nondeterministic, meaning that same LLHs with same parameters applied on the same solution can yield different results of different quality. This would mean that the fitness value of each sequence would be only an approximate of the solution that could be generated, making the whole evolution process inaccurate.

4.1.4 Initial sequence generation

When random initialization of sequences is required, LLHs are selected based on their type. This is to compensate for problems where there are only few heuristics of one type and many of another. This should promote diversity in functionality, where LLHs of the same type can differ only in implementation but have very similar functionality. Also, to keep volume of diversification (i.e. mutation and ruin recreate) and intensification (i.e. local search) LLHs balanced, the probability of selection local search is higher.

Heuristic parameters are also selected randomly using uniform distribution, typically using middle values of both the IoM and DoS interval.

4.1.5 Crossover operators

One crossover method is used throughout the whole run of the algorithm. It is called *merge crossover* and it combines parent sequences in a way that is designed to maintain relative pairwise positions of heuristics in the sequence. This means that for every pair of heuristics the

following statement holds: if there are two heuristic instances A,B in a parent sequence and A precedes B, than if both of these instances are transferred into the same offspring, then A precedes B in this offspring.

Example: Let [A,B,C,D] be first parent sequence of heuristics and [S,T,U,V] be the second parent sequence. Then sequences [A,S,D,U] and [B,T,C,V] would be a valid result of merge crossover as all relative pairwise positions were preserved. However, [B,A,S,T] would not be a valid result.

4.1.6 Mutation operators

There are 3 different mutation operators that are integrated into one. The first is swap mutation, which traverses the sequence and at each position it randomly decides whether to change positions of the current heuristic and the next one. The second is parameter mutation, which is a standard real-valued mutation that is used in evolutionary strategies. The modified value can be generated either using uniform or normal distribution. The last is a heuristic replacement mutation which iterates through the sequence and at a given probability replaces the current heuristic with a heuristic of the same type.

4.1.7 Parent selection

Member sequences are first chosen at random from the old population. These sequences enter tournament, from which the solution with best fitness is selected for evolution. If there is a tie, winning parent is selected randomly.

4.1.8 Replacement strategy

No members of the old population are transferred directly to the new one as the benefits of approaches as elitism are diminished by the dynamic change of fitness calculation given by changing of evaluation solution.

4.1.9 Time complexity handling

To address the fact that different problem may require different settings, a mechanism is introduced to allow to change the algorithm control parameters depending on what is the expected number of heuristics applied in before the time runs out. Control parameters are

divided into two categories:

- Fixed control parameters that are set to values that remain the same throughout the whole run of the algorithm.
- Variable control parameter sets (VCPS) that are changed based on an algorithm performance on the given problem.

These sets contain the values following parameters: population size, sequence length and number of generation per one EA. Each set contains a value for all these control parameters. The sets are exclusive meaning that if the algorithm uses value of one parameters defined by set A, it can not use a value defined by set B for another parameter.

To assign a problem to certain control parameter set, an extra parameter is defined that specifies the minimum number of runs that should be executed with these settings defined. It is only used to select the most appropriate control parameter set, but it does not influence the algorithm after the time complexity calculation are made. The particular values of the control parameter sets are described in the chapter describing algorithm evaluation.

When the HEADS algorithm is started, the following mechanism is used to select the most appropriate set of control parameter values. First, the settings for the slowest LLH execution is used to run a time complexity testing EA (TCT-EA). This EA has the same functionality as all the following, but also includes the recording of the execution time of LLHs throughout the EA run. After TCT-EA has finished, the recorded execution time is used to approximate the running time of LLHs through the whole run. This value is then used to select the most appropriate set, which is the most complex one (i.e. the one with greatest values of control parameters) that will be able to finish the number of desired runs of EA in the time limit. All the subsequent generations are to be run with these settings. Note that this approach does not try to assign the problem to the clusters mentioned in previous text, but rather to find the optimal control parameters derived from those clusters.

$$S_{opt} = \{ S \in Sets \mid expRuns(S, R) \geq minRuns(S) \wedge (\forall S' \in Sets : expRuns(S, R) \leq expRuns(S', R)) \} \quad (4.1)$$

The formula 4.1 describes the selection, where:

- S_{opt} is the optimal VCPS
- *Sets* is a set of all the VCPSs that were defined
- R is the average LLH running time approximation

- *expRuns(S,R)* is a function that returns the expected number of EA runs for given problem
- *minRuns(S)* is a function that returns the minimum number of EA runs required for the given VCPS

4.2 HEADS with tabu list

The first extended version employs a tabu list to improve algorithm performance on problems where there are larger plateaus. Plateau is a part of solution quality landscape where neighboring solutions, i.e. solutions where by applying single LLH to one the other one is obtained. It is possible to get stuck in these regions when smaller values of IoM for mutation and of DoS for local search LLHs is used, because returning to solutions that were already visited during search is not penalized in any way.

To overcome this problem, tabu list was introduced that modifies the fitness calculation to reflect not only the possible identity of the new solution and the evaluation solution, but also identity of the new solution and the set of recently visited solutions.

This list has a form of queue, where every new evaluation solution is saved. When calculating a fitness of a sequence, solution that is obtained is compared with this list. If the new solution is present in the list, meaning that the given solution has been recently visited during the search, the sequence that generated this solution is assigned the worst fitness. This results in favoring the sequences that create worse solutions but in section that have not been searched yet over those sequences that make only small changes, thus keeping the evaluation solution in the plateau.

```
procedure:  assign a fitness value to offspring
input:      the evaluation solution
            the solution created by application of offspring
            the tabu list
1.  if (the new solution is equal to the evaluation solution)
2.     assign the worst fitness to offspring
3.  else if (the new solution is in the tabu list)
4.     assign the worst fitness to offspring
5.  else
6.     assign a fitness value equivalent to objective function of
            solution created
7.  end if
```

Pseudocode 4: The tabu list fitness assignment.

The pseudocode 4 replaces the fitness function from the baseline variant of HEADS, described by pseudocode 3.

4.3 HEADS with evaluation solution restart

Evaluation solution stagnation is one of the biggest challenges that needs to be addressed to come up with a robust and effective algorithm. This is especially true when working with dynamic environment where the fitness function is variable, which can lead to removal of population members that could be useful in the following generations. With the previous versions of HEADS, stagnation can be a result of two factors combined: the evaluation solution is stuck in a local optimum and the population has degenerated and is no longer able to create heuristic sequences that would help us leave the local optimum.

Method presented in this version of HEADS is restarting of evaluation solution independently on the sequence population when stagnation threshold is passed.

4.3.1 Evaluation solution restart

One of the ways of fighting this is evaluation solution restart. This is being done by applying either mutation or ruin recreate heuristic, depending on the settings. This is done in hope that the evaluation solution will move far enough from the optimum that the local searchers will push it in different direction, hopefully finding better solution than the one found so far.

To take different problems' time complexities into account, fraction of the allowed running time is used to control the evaluation solution restart. The probability of restart is depending on the time difference between the current time and the time of the most recent *restart-relevant event*. Restart-relevant event is either the last restart or the last improvement of evaluation solution. If a last restart is the most recent, it means that restart was unsuccessful and the solution obtained can not be improved by any sequence present in the population. If a solution improvement is the most recent, either this will be the first restart or the evaluation solution improved since last restart but is again stagnating.

This probability is zero until the first time threshold is passed. The initial probability can be set to arbitrary value and then it linearly increases until the second threshold, where it is 100% and where the restart is inevitable.

After every execution of an EA generation, restart probability is tested. If the restart test

passes, i.e. random number generated is lower than the current restart probability, restart mechanism described are applied on the evaluation solution.

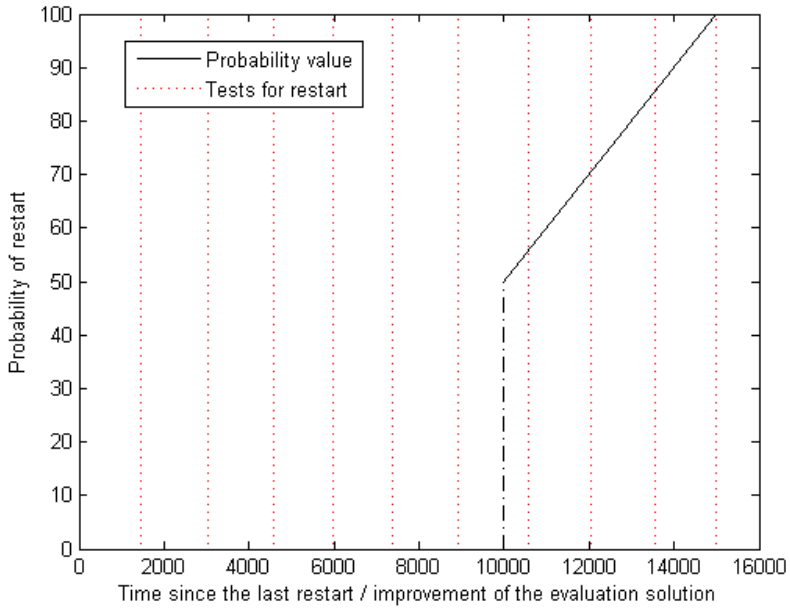

Figure 3: The change of probability of restart in time

Figure 3 shows an example of the progress of the probability function plotted in time. The dotted line represents the moment where restart probability was tested. In this example no improvements in solution were found in this time frame and only the last test for restart was successful. Otherwise, both these events would set the probability back to zero.

4.4 Buffered diversification HEADS

This version adds another method of prevention of evaluation solution stagnation. Apart from the use of evaluation solution restart, it is storing the sequences that appeared in the population in and using them later on independent of EA. This approach promotes diversity in the EA population by using these stored sequences when population degenerates by converging to one type of sequences. Typically, this happens when only LLHs with low destruction potential, such as mutation LLHs with low IoM parameter or local search LLHs, are present in the sequences.

4.4.1 Improvement buffer

The *improvement buffer* aims to record the sequences that contributed to improvement of the

evaluation solution. A diversity measure, which is described in the following one of the following chapters, is used to keep such sequences in the buffer that will be as unlike as possible.

Every time a evaluation solution is replaced, the sequence that caused the improvement is used to update the buffer by replacing a sequence that is most similar to it. When creating new population at the beginning of an EA, part of it is initialized by randomly selecting the desired number of sequences from the buffer without modifying them. The rest of the population is initialized randomly.

4.4.2 Double buffer

The second variant of the buffered approach uses two buffers instead of only one. The improvement buffer is used as described in the previous paragraphs, but *no-improvement buffer* is also added. This buffer stores only those sequences that didn't contribute to solution improving, so sequences are added exclusively to these buffers. The main goal of the no-improvement buffer is to maintain a set of sequences that would be as different as possible, the hope being that some of these representatives will have positive impact at a given time.

The buffer updating works the same as with the previous version, with the difference that non-improving solutions are added to to no-improvement buffer, but using the same replacing mechanism. However, to make the pressure towards diversity even stronger, the buffers are used every time a new population is created by evolution of the old one. This means that only part of the population is created by evolution of parents and the rest is filled with sequences randomly selected from both buffers.

4.4.3 Diversity measure

To fulfill the condition of diversity inside the buffers, metric called *diversity measure* that compares two sequences is introduced. It approximates how similar solutions created by these two sequences would be if applied on the same evaluation solution. Because we do not know what the behavior of individual heuristics are, only their type and parameter values are considered when calculating this metric.

$$DM(S_1, S_2) = \sum_{t \in heuristic\ types} (W_t \cdot (|ps(S_1, t) - ps(S_2, t)| + ex(S_1, S_2, t) \cdot E_t)) \quad (4.2)$$

$$ps(S,t)=\sum_{\{h\in S|isType(h,t)\}} pv(h) \quad (4.3)$$

Formulas (4.2) and (4.3) describes the metric calculation, where:

- t is a heuristic type as defined by HyFlex
- S_1 and S_2 are sequences of LLH triplets
- h is an individual LLH
- W_t is a weighting constant for a given type specified by control parameter, which defines how important the given LLH is when calculating diversity measure, where greater number means greater importance
- *excl(S,S',t)* is a function that returns 1 if one of the sequences contains the specified type and other does not, and 0 otherwise
- E_t is the exclusivity constant
- *pv(h)* is a mapping from a heuristic instance in sequence to its relevant parameter (i.e. IoM for mutation and ruin-recreate, DoS for local search)
- *isType(h,t)* is relation that is true if heuristic h is of type t

In other words, the diversity measure is equal to the absolute difference between sums of parameters of all the LLH types increased by a constant specified by a control parameter if one sequence does not have a LLH type while other has and multiplied by constant defining the importance of a type for diversity calculation, which is also specified by a control parameter.

Example: Let there be two sequences of LLHs with their corresponding parameters [A/0.2; B/0.5;II/0.5] and [III/0.3; B/0.7; c/0.3]. The presented sequences are recorded in LLH/parameter format, where upper case letters, lower case latter and roman numbers each form a separate LLH type. Let us set all the weighting constants to 2 and the exclusivity constant to 4. Then the diversity measure will have the following value for these sequences:

$$DM = 4\cdot(|0.2+0.5-0.7|+0)+4\cdot(|0.5-0.3|+0)+4\cdot(|0-0.3|+2)=0+0.8+9.2=10$$

When the buffer is not full, sequences are added without replacing, but dropping those sequences which have the same diversity index as any sequence already present in the buffer. If it is full, then the sequence that is being inserted will replace the one that has the lowest diversity metric value, i.e. is the most similar to the one being inserted.

4.5 Control parameters

This section explains the control parameters of HEADS algorithm. Please note that this is not an exhaustive description of all possible parameters in the implementation but rather description of parameters that can be used for tweaking of the algorithm. First, general parameters are introduced, parameters specific for each algorithm version are discussed separately. Parameters in italics can not be changed for the given version.

4.5.1 General control parameters

- the time available for the algorithm execution - is fixed for the purposes of CHeSC
- the lengths of triplet sequences, the number of EA generations, the size of EA population for each speed setting
- the number of minimum EA runs to be executed for each speed setting; this parameter plays major role in selection of speed settings
- the probabilities of selection of each LLH type, random distribution type and its parameters used on sequence creation
- the tournament size on the parent selection
- the crossover probability
- the mutation probability of all the mutation components as well as distribution types and parameters associated with mutations

4.5.2 Tabu list parameters

- the tabu list size

4.5.3 Parameters shared by buffered versions

- the time threshold since last restart-relevant event as a fraction of allowed running time until which the probability is 0% and at which the probability is 100%, initial probability of restart when reaching the first threshold
- weights of LLH types for the purposes of calculation of the diversity measure

4.5.4 Single buffer

- the buffer size
- the number of sequences that are taken from the buffer on population initialization

4.5.5 Double buffer

- the size of both buffers
- the number of sequences that are taken from improvement and no-improvement buffer every EA generation

Chapter 5

5 Implementation

The main goal when designing HEADS was to create algorithm that would not only effectively solve black box optimization and search problems, but that would also be reusable and extendable. This is reflected in the structure and components of the algorithm implementation.

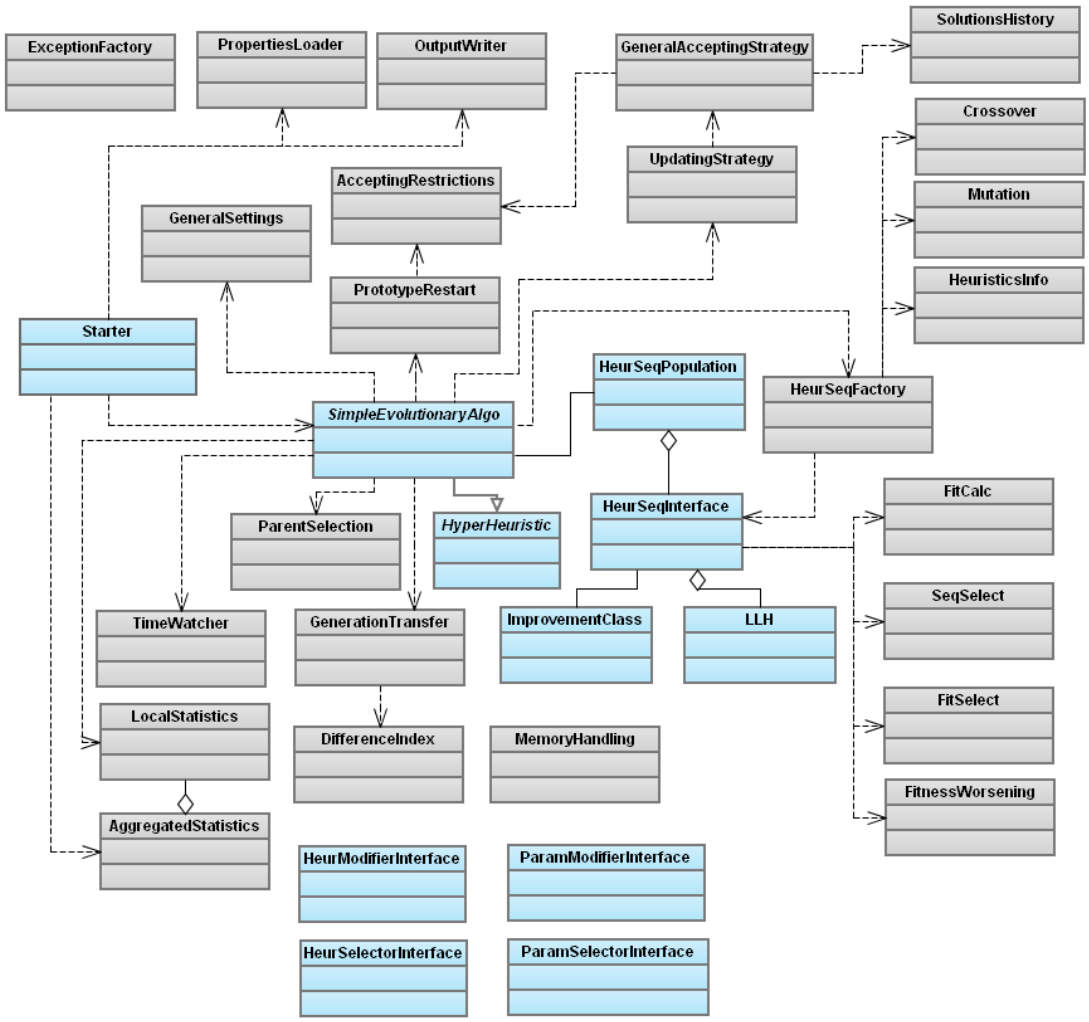

Figure 4: The UML class diagram of HEADS implementation

The Java programming language was selected because of its portability as it was developed on the Windows XP system but tested on Unix platform and because the HyFlex framework offers the data that is necessary for a comparative testing on wide range of problems with other

algorithms. NetBeans 7.0.1 IDE[1] was used to write the core of the algorithm. The processing scripts were written in Matlab R2010b[2].

The algorithm is composed of independent modules that can be replaced using configuration file changing the core functionality. The modularity of is achieved via reflection, where there is a singleton object that works as an interface for the retrieval of the corresponding object as well as for loading of the properties from the configuration file. This means that very few object are actually passed as an argument through method calling, which prevents the calling from being cluttered up. Of course, it somehow reduces ease with which the code can be read at the first glance, so the reader is kindly asked to refer to the documentation provided in the Java code for greater detail and explanation of all the methods.

Please note that the implementation, as presented on the CD that is a part of this thesis, contains extra classes containing additional functionality. Even though that these classes can be used instead of those used in the experiments, please note that they contain experimental features and should be treated as such.

5.1 UML overview

The figure 4 shows the most important classes of the HEADS implementation. The light blue classes are standard meaning that new instances can be created anywhere anytime, while the gray ones are singleton classes, which have only one instance that is used by the other objects. These task of these classes is to share information by objects in different parts of code or to provide interfaces that allow the calling of methods of objects that are dynamically loaded as governed by the configuration file.

5.2 HyFlex classes

There are two abstract classes that are directly used by HEADS algorithm that are part of the HyFlex framework. Please note that these are by no means the work of the author of this thesis. Their description here is only to illustrate the interface of the HyFlex framework that is used by HEADS algorithm..

1 http://netbeans.org/

2http://www.mathworks.com/products/matlab/

5.2.1 ProblemDomain

An abstract class that is the superclass of all the objects representing problems. If defines methods for LLH application, controls for handling solutions in memory, setting the IoM and DoS parameters, test for solution identity and converting solution to string, but also methods for obtaining LLH indexes of given type.

5.2.2 HyperHeuristic

Is the superclass of all the algorithms competing in the challenge. Its main role is to make sure that only the solutions that the algorithm searches within the time limit are used. Everything else including the algorithm termination after the time limit is exceeded must be implemented in the subclass.

5.3 Core classes description

By *core classes* we mean those that contain code related to HEADS main loop handling the EA execution and containing the inner EA data. These classes typically call on methods of *modular classes* by the means of their provided interface to use specific functionality. This functionality may include specific methods of fitness calculation, crossover, parent selection etc.

For each class, a brief definition of its function is provided. For details such as the methods available or variables used please consult the JavaDoc documentation.

5.3.1 Starter

This class serves as a facade for the whole algorithm. Its methods are called to start the algorithm while delegating the task of loading the control parameters from the properties file, prepare the output directory and start the algorithm core. This class is also used to handle all the exceptions that can be thrown during these preparatory phases. To start the algorithm, the user must specify the output directory, the location of the properties file to be used, the problem class name, the instances number and the number of times the algorithm should be executed. The optional parameters are seed for the problem and for the algorithm core.

5.3.2 PropertiesLoader

Loads the control parameters from the file specified by user prior to the algorithm execution.

The file must be in a standard format recognized by the Properties class, i.e. having each control parameter on a separate line, while using the format *"parameter_name = parameter_value"*.

5.3.3 OutputWriter

Is used to prepare the output directory. This means that it creates a directory having the same name as the properties file in the path specified by the user. Inside this directory another one is created having the name containing the problem class name and the problem instance in it. Here, the output text files for the runs are created when the algorithm core terminates the solution calculation process. After all the runs were executed, a global statistics file is created.

5.3.4 ExceptionFactory

Handles the creation of exceptions that can be thrown in different parts of code but have always the same structure. Typical example are exceptions that are thrown during the process of reading properties when a property has incorrect formatting, the values are missing etc.

5.3.5 SimpleEvolutionaryAlgo

The core class which extends the HyperHeuristic abstract class that is supplied by the HyFlex framework. As such it implements all the methods that are called by the framework. It handles the execution of EAs and all their inner workings such population handling. Also, it checks if the time limit has been exceeded and in such case terminates the algorithm. Also, all the output messages are printed from this class.

Because the abstract class does not allow exceptions to be thrown from it, they are all handled by this class.

Note that the name does not refer to any EA type but to the fact that the vast majority of functionality is performed by modular classes described in further in the text.

5.3.6 HeurSeqPopulation

It contains the array of sequences in form of subclasses of the HeurSeqInterface. This class also provides methods for extracting best sequences in terms of the current fitness schema, extracting the sequences that improved the solution etc.

5.3.7 HeurSeqInterface

An abstract class that defines methods for sequence application, fitness calculation and

improvement analysis obtained by comparing the evaluation solution and the newly created one. Note that the exact approaches used for sequence selection and fitness calculation and retrieval are governed by methods of abstract classes SeqSelectInterface, FitCalcInterface and FitSelectInterface respectively. However, fitness for both the full sequence and effective sequence is always stored in this class.

5.3.8 ImprovementClass

A simple class that contains a status-specifying enumeration which specifies the improvement achieved by the application of the corresponding sequence and whether the new solution actually replaced the evaluation solution. Also, it provides static methods for changing and querying this status and modifying it.

5.3.9 LLH

A class representing the LLH. It contains the LLH index as defined by the HyFlex framework and both the IoM and DoS parameter. Note that it is possible to have a LLH does not use any of these parameters and that no LLH will ever use both of them. However, they are always present for convenience.

5.3.10 MemoryHandling

Provides a facade for the memory handling supplied by the HyFlex framework in the ProblemDomain class. It uses all of its solution related methods described, i.e. the manipulation with existing solution, creation of new ones by LLH application and loading the appropriate IoM and DoS for each LLH.

5.3.11 HeuristicsInfo

The class that is used for queries about LLHs defined for a given problem. Contains a mapping from heuristic indexes to their respective types.

5.3.12 SolutionHistory

The class that handles the history and the tabu list, if used. This means that it maintains a list of copies of the last N solution in the memory and offers methods for testing if the specified solution is already present in this list.

5.3.13 TimeWatcher

This class handles all the time complexity calculation. It is responsible for selection of the most appropriate VCPS.

5.3.14 GeneralSettings

Contains general settings of the EA, such as population size and number of EA generations.

5.3.15 LocalStatistics & AggregatedStatistics

These classes store all the statistics from a HEADS run and from batch of runs, respectively.

5.4 Module classes

This chapter describes the modular classes, i.e. those that are defined by an abstract class, whose specific subclasses are selected via the properties file. Typically, only one subclass can be active and cannot be switched for another one at run time, unless stated otherwise. Each of these classes has a singleton *mother class* that is used to store the subclass of the abstract class. The names of the singletons are present in the headings, while the names of the abstract classes can be obtained by adding Interface at the end.

5.4.1 FitCalc

Provides an interface for fitness calculation. The final implementation includes fitness based solely on the quality of the slution created by the sequence, although version based partly on time or using relative improvement are also present.

5.4.2 FitSelect

Provides an interface for fitness selection. This can be either the fitness of the full sequence or the fitness of the effective sequence. Only the fitness of effective sequence is used in the final implementation.

5.4.3 FitWorsening

Defines the fitness penalization of sequences. There are two strategies: one penalizing sequences that did not modify the evaluation solution at all, other also the sequences that were refused as candidates for a new evaluation solution, which is the result of using a tabu list. Both of these strategies were used.

5.4.4 SeqSelect

Defines what part of a sequence is returned when requested by another class, typically when calculating the diversity measure. Its functionality is paired with FitSelect in the sense that if a fitness of the effective sequence is used, then effective sequences is returned on request. This relation is enforced by the algorithm.

5.4.5 HeurSeqFactory

Contains the methods for a creation of LLH sequence both by random initialization and by mutation and crossover using already existing sequences. This class decides how the MutationInterface and CrossoverInterface classes will be utilized exactly as there is no mandatory internal structure of the sequences created by this class. Only one version is available which uses a sequence without any restrictions.

5.4.6 ParentSelection

Handles the parent selection during a creation of a next generation. The only currently available version is a tournament based on the fitness of parents.

5.4.7 Crossover & Mutation

Handles the crossover and mutation of parents during the creation of new population. They are called from the HeurSeqFactoryInterface class and work on arrays of LLH objects rather than on HeurSeqInterface objects directly.

5.4.8 GenerationTransfer

Defines how the new EA population will be created from the previous one. This also includes the creation of a new population in the first generation of a new EA. Subclasses of this mother class contain wide range of strategies from a simple elitism to the diversity buffers.

5.4.9 EvalSolutionRestart

Specifies when, if ever, should the evaluation solution be restarted. How the restart should be executed is handled by the RestarterInterface class and its subclasses.

5.4.10 GeneralAcceptingStrategy

Defines under what conditions will the new evaluation solution be accepted. The final version

uses the strategy that accepts only improving solutions. However, accepting of non-improving with certain probability is also implemented.

5.4.11 AcceptingRestrictions

Imposes additional restrictions on evaluation solution accepting strategy. Rules defined in this class override any defined by the GeneralAcceptingStrategy class.

5.4.12 UpdatingStrategy

Defines when the algorithm should attempt to replace the current evaluation solution with a new one. The only strategy used in the final version attempts update every time a fitness of a sequences has its fitness calculated, which requires a creation of the new solution by applying the sequence on the evaluation solution.

5.4.13 HeurSelectorInterface & HeurModifierInterface

Provide strategies for selecting a new LLH and modification of an existing one, respectively. These classes are not singletons and different instances are used in various parts of the algorithm.

5.4.14 ParamSelectorInterface & ParamModifierInterface

Define ways of selecting new LLH parameters and modification of an existing ones, respectively. These classes are not singletons and different instances are used in various parts of the algorithm.

Chapter 6

6 Experiments

6.1 Setup

The algorithms are each tested on 6 different problem classes, each class containing 5 instances with different parameters. HyFlex java framework provided for the CHeSC 2011 challenge is used to provide problem definitions and evaluate the quality of the solutions. The problems are described in the following paragraphs.

Note that the descriptions of problems provided on the CHeSC web page sometimes use the term fitness to describe the quality of the solutions. In this work, the term quality measure is used to prevent confusion with the fitness value of population members in EA.

Also note that the last two problems described here, Vehicle Routing Problem and Traveling Salesman Problem do not have an official documentation provided on CHeSC as these were not part of the problem set originally provided for algorithm testing. However, the documentation was not added even after these problems were released after the challenge was closed. Therefore the description is based general definitions of the problems and on observation of the behavior of the problem classes in HyFlex framework.

6.1.1 The one dimensional bin packing problem

In this problem, as defined in [27], there is a number of objects that need to be fitted into bins. Any number of bins can be used. Each object has a weight value w_i assigned and all the bin have the same capacity C defined. The goal is to put all the objects into bins in such a way that minimizes the number of bins used but respects their capacity.

The quality measure is usually equal to the number of bins used. However, to avoid large plateaus, HyFlex uses a measure that depends on the fullness of the container, which is the fraction of room taken. It uses the formula 6.1.

$$Q = 1 - \left(\frac{\sum_{i=1}^{n} \left(fullness_i / C \right)^2}{n} \right) \quad (6.1)$$

The problem definition comes with 2 local search, 3 mutation and 2 ruin-recreate LLHs.

The 1 crossover LLH is not used by the proposed algorithm.

The real world applications of this problem include cutting material of fixed length with the aim to minimize waste.

6.1.2 The maximum satisfiability problem (MAX-SAT)

In this problem, as defined in [26], the goal is to find out whether it is possible to assign values to Boolean variables in a given formula, or rather find such an assignment to variables that would result in minimization of numbers of clauses in the formula that will not be satisfied. The quality measure is equal to this number.

The problem definition comes with 2 local search, 6 mutation and 1 ruin-recreate LLHs. The 2 crossover LLH are not used by the proposed algorithm.

Even though that there is not a direct translation to a real world problem, MAX-SAT is a popular benchmarking problem.

6.1.3 Permutation Flow Shop

In this problem as defined in [24], the task is to schedule *n* jobs on *m* consecutive machines. Each job must be processes on machines in order from machine 1 to machine M, while the order in which they are processed is the same for all the machines in the system. Also, one machine can work on one job at a time. If a machine is free and the job that is scheduled next is available, it must process it immidiately. All the machines are available at the start.

The goal is to find such a schedule that will minimize the time of completion of the last job on the last machine while respecting these rules and addtional constraints. The could be a length of the job, minimal job start time etc. The quality measure is equal to the of completion.

The problem definition comes with 4 local search, 5 mutation and 2 ruin-recreate LLHs. The 4 crossover LLH are not used by the proposed algorithm.

6.1.4 Personnel Scheduling

As described in [19], personnel scheduling is a broad class of problems with different objectives and different constraints. However, in this case, a more specific problem of nurse rostering is used, which comes with coverage objectives and employee constraints. This means the goal is to get as close as possible to preferred number of nurses working each shift and at the same time respecting various limitations such as maximum number of working hours per month, maximum

consecutive days off, shift rotation etc.

The quality measure is equal to a sum of weighted goals each representing an objective or constraint. In this case, there are 16 formulas describing the goals.

The problem definition comes with 5 local search, 1 mutation and 3 ruin-recreate LLHs. The 3 crossover LLH are not used by the proposed algorithm.

6.1.5 Traveling Salesman Problem

In this problem, a set of cities, is given that must be visited in arbitrary order while returning to the city of origin, thus forming a cycle. The minimal distances between all the cities are provided. The task is to minimize the distance traveled. [7]

The problem definition comes with 3 local search, 5 mutation and 1 ruin-recreate LLHs. The 4 crossover LLH are not used by the proposed algorithm.

6.1.6 Vehicle Routing Problem

The basic version can be described as follows: a fleet of vehicles, each starting from the same depot, need to cover all the stations on map and deliver goods there, whilst each station is demanding different goods and each vehicle has a limited capacity. Usually, it is assumed that the vehicle cannot return to depot to restock. The shortest distance between all the stations and between stations and depot is provided. The goal is to minimize the sum of distances traveled by all the vehicles. As such, this problem is a generalized version of the TSP. [8]

The problem definition comes with 3 local search, 3 mutation and 2 ruin-recreate LLHs. The 2 crossover LLH are not used by the proposed algorithm.

6.2 Algorithm evaluation

The algorithms are individually compared with the algorithms that competed in CHeSC 2011. For this purpose, the scoring system used in the competition is adopted as only median and best results are publicly available. Then, a pairwise statistical test is performed on each problem instance to attempt to reject the hypothesis that the algorithms are in fact equal. Both of these evaluation methods used results from 6 domains specified in previous text, while from each domain 5 instances were selected and the algorithm was run on each 31 times.

As there are problems where the quality measure is continuous, the results will used with precision of 10^{-6} only, ignoring negligible differences between real-value quality measure

results.

6.2.1 CHeSC scoring

The CHeSC 2011 used a specific scoring system for the purposes of ranking participating algorithms inspired by Formula 1 scoring system. For each problem instance, the median values for each algorithm are used to rank the algorithms. Then, points are distributed from the 1st place to 8th in the following order: [10, 8, 6, 5, 4, 3, 2, 1]. The ties are handled by dividing the points equally between the tied algorithms. This ensures that each instances has the same amount of points divided between competitors. Points from all problem instances are summed and the algorithm with most points wins. If there is a tie in the final ranking, the winner is the algorithm with most wins.

6.2.2 Wilcoxon rank-sum test

However, the algorithm evaluation described in previous paragraph is ensufficient as it does not investigate the possibility that the destributions that are generating the results of algorithms are equal and the observed better or worse behaviour is just a result of lucky or unlucky draws from that distribution.

Therefore, Wilcoxon rank-sum test was selected to test the null hypothesis that some of the pairs of algorithms presented in this thesis return results that are drawn from distributions with the same median. This test was used because for two reasons. One being that the test works with medians of distributions, which are also used by the CHeSC scoring. The second reason is that Wilcoxon rank-sum test makes no assumptions whatsoever about the distribution it tests. Although it could be tempting to assume that the results are drawn from a normal distribution, there is no guarantee that this assumption is even close to being correct.

6.3 HEADS configuration

The following paragraphs describe the experiments carried out in order to find out an efficient configuration of certain control parameters. The rest of the configuration was selected using a rule of thumb as a reliable method of parameter tuning would be too computationally demanding.

6.3.1 Time complexity analysis

Because the algorithm aims to achieve maximum generality and run efficiently on different

problems, an analysis of time complexity of problems used to test this algorithm was performed to come up with the most appropriate settings of the VCPSs that were introduced in the previous chapter.

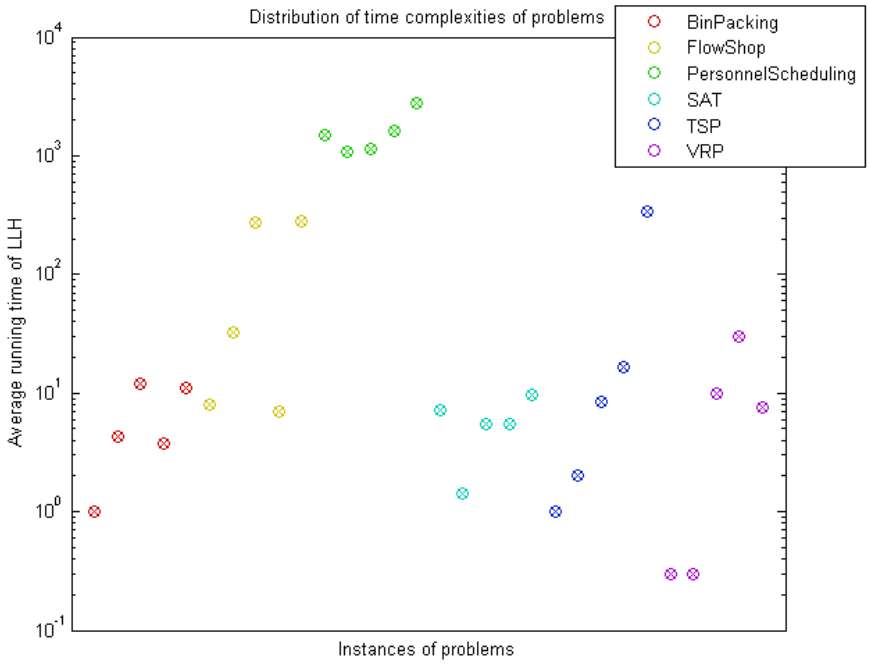

Figure 5: The average running times of a single LLH by problem instnaces. The x axis logarithmic and in milliseconds.

Preliminary testing was performed that consisted of running the baseline HEADS algorithm with ad hoc values of control parameters and the average running time of each LLH executed was recorded. Note that HEADS algorithm was used instead of a simple algorithm that would select LLHs randomly or sequentially based on their index in the system to come as close as possible to the distribution of LLHs in the final version of HEADS. This approach will for example reduce the impact of inefficient LLHs that would also get low attention in the final version.

The figure 5 shows the average running times of LLHs for all the problems that were used in the challenge. Not only that the values are very different for different problems, it should be also noted that even though some problems have consistent running times throughout all instances, some have great variance. An excellent example of a this is the traveling salesman problem (TSP) where the lowest average running time is 1 ms, while the highest is 594 ms.

These observations are indicating that there is not a single value of control parameters that influence the running time of the algorithm that would ensure optimal performance across

all the problems. This shows that indeed different control parameters must be supplied for different problems.

To identify the optimal settings, cluster analysis was performed using average linkage on logarithm of running times. Logarithms were used because otherwise the distribution of cluster would be uneven with more clusters for problem instances with slow running LLHs. However, with the increasing LLH running time, the impact on the difference in values is diminishing. In other words, if a problem instance has an average running time of LLH in 1000 ms, an improvement by 10 ms would cause that only 1% more LLHs would be executed. But the same change for 10 ms average running time would mean doubling of the LLHs executed.

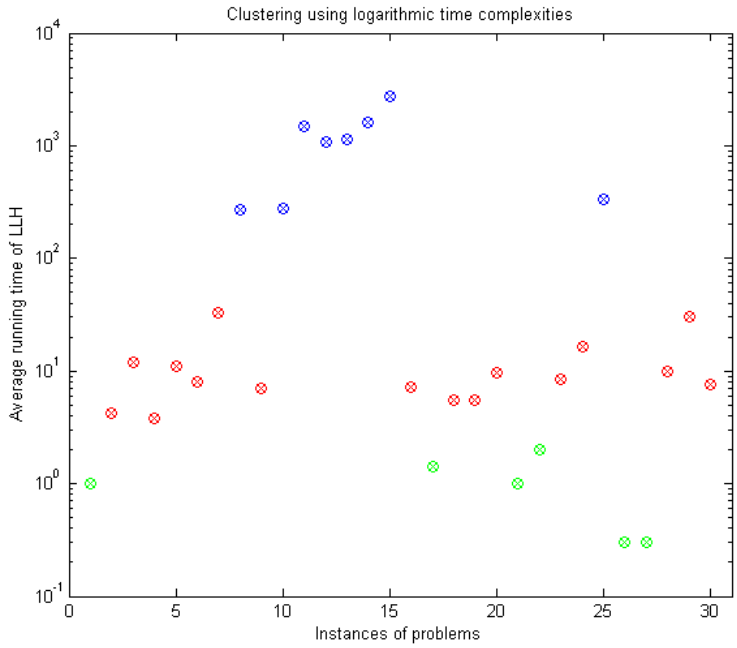

Figure 6: The final clustering of problem instances using average linkage clustering with their running times.

The number of clusters to be created was decided to be 3 as a tradeoff between complexity of control parameters and ability to handle different problems. The clustering can be seen in the figure 3.

For these clusters, optimal parameters were selected using a rule of thumb such as that for the most complex problem of the cluster, control parameters must be set in a way that at least 5 generations of EA will be executed. An exception is made with the group with most demanding LLHs, where the aim is not to guarantee a given number of EA runs but rather that the EA will behave reasonably, meaning that the control parameter values will not be too small,

causing a diminishing of positive EA traits and an increase of negative ones. In other words, rather than having many VCPS with parameter values linearly decreasing towards zero, a VCPS is set that is used for all the problems that are above a certain complexity threshold.

The VCPSs used are (given in the order [sequence size, population size, generations per EA run, minimal EA runs]):

- [5,40,15,5]
- [4,20,10,5]
- [3,20,10,5]*

Note that the VCPS marked with the asterisk is the threshold set. This also means that the desired EA runs value actually does not play any role as all the problem instances that would not meet the minimal number of runs condition in the other two will be automatically assigned to this VCPS.

6.3.2 Destructive potential analysis

To properly set the control parameters in the diversity measure, a basic analysis was performed to quantify the impact of different LLH types on a solution. The *destructive potential* of a LLH type is examine, which is a measure that defines how much a solution will probably change if an arbitrary LLH of the type is applied onto it. Because the solution state can not be inspected directly, the difference between the original solution and the solution obtained by an application of LLH is approximated by the difference in their quality measure. Even though this is not by far accurate approach, the framework leaves no other option.

The local search LLHs are by definition never destructive in a sense that they always improve solution by making small changes to it [32]. However, it is not known what the destructive potential of mutation and ruin-recreate LLHs is. To find out, a simple approach is used. For each problem group, and instances is randomly picked. For this instance, a random solution is created using the framework and then local search LLHs are applied 100 times, replacing the solution after each application. During this process the LLHs are selected randomly. This should cause the solution to reach the local optimum. Then, for both the ruin-recreate and mutation LLH types, a random LLH of

the type is randomly selected and applied on the local optima. The difference between the quality measure of the local optima and the newly created solution is recorded. This is performed 100 times for each type. To make the comparison of destructive potential between problem instances possible, the difference values are stored as a fraction of difference between the quality measures of the original randomly created solution and the local optima. This is repeated for 3 different IoM values.

Example: A random solution is created with a quality measure 250. After the application of 100 local search LLHs, the quality measure is 50. A mutation LLH is applied on the local optimum and a solution with quality measure 100 is created. The recorded value will then be 0.25.

Problem: BinPacking			**Problem: FlowShop**		
IOM parameter	*MUTATION*	*RUIN_RECREATE*	*IOM parameter*	*MUTATION*	*RUIN_RECREATE*
0.1	3.1482202561880253	**-4.483418721437669**	0.1	1.36	**0.355**
0.3	6.415018576721646	**-5.445011528223493**	0.3	1.36	**0.595**
0.5	6.066155172992538	**-5.149470881878271**	0.5	1.36	**0.645**
Problem: VRP			**Problem: SAT**		
IOM parameter	*MUTATION*	*RUIN_RECREATE*	*IOM parameter*	*MUTATION*	*RUIN_RECREATE*
0.1	**0.0**	0.01196240917730617	0.1	**0.0**	0.2977667493796526
0.3	**0.0**	0.05234625415478063 5	0.3	**0.004962779156327543**	0.5235732009925558
0.5	**0.0**	0.11050883091207331	0.5	**0.007444168734491315**	0.6997518610421837
Problem: TSP			**Problem: PersonnelScheduling**		
IOM parameter	*MUTATION*	*RUIN_RECREATE*	*IOM parameter*	*MUTATION*	*RUIN_RECREATE*
0.1	0.45660329120437615	**0.16188198857022146**	0.1	0.382911392405063 3	**0.002531645569620253**
0.3	0.550606237334237	**0.3734859392194525**	0.3	1.1537974683544303	**0.005063291139240506**
0.5	0.8329449365298114	**0.6590284514101655**	0.5	1.8018987341772152	**0.00949367088607595**

Table 1: The median values of changes caused by given LLH types recorded as a fraction of improvement of quality measure.

The tables 1 and 2 contain the median and variance values, respectively, of the sets of values recorded. Note that negative values in median table should be translated as improvement to the solution

Problem: SAT			**Problem: VRP**		
IOM parameter	*MUTATION*	*RUIN_RECREATE*	*IOM parameter*	*MUTATION*	*RUIN_RECREATE*
0.1	**0.07197355914426638**	0.3682946422957701	0.1	**0.06258685717910849**	0.8027689964296387
0.3	**0.07916527248634185**	0.4979606088460880 6	0.3	**0.1584808151592619**	2.08282224010887
0.5	**0.133466469662221**	0.5624326069429205	0.5	**0.22795290492181866**	2.584735138249198
Problem: PersonnelScheduling			**Problem: TSP**		
IOM parameter	*MUTATION*	*RUIN_RECREATE*	*IOM parameter*	*MUTATION*	*RUIN_RECREATE*
0.1	2.056236736719882	**0.32320252777711395**	0.1	514.5209104991629	**0.3001667978705557**
0.3	3.35694428617757	**0.5592584850989955**	0.3	511.82056840483386	**0.4492433267243448**
0.5	3.740044706608851	**0.49086412470072527**	0.5	522.8872693215886	**0.6473211890645539**
Problem: BinPacking			**Problem: FlowShop**		
IOM parameter	*MUTATION*	*RUIN_RECREATE*	*IOM parameter*	*MUTATION*	*RUIN_RECREATE*
0.1	34.9640458670379	**2.0350641528025593**	0.1	82.57344770043647	**4.291637432900029**
0.3	51.394930818720475	**1.5973778915333157**	0.3	72.79561383794334	**5.731272789962868**
0.5	65.15655373263205	**0.7137107001406788**	0.5	97.34649538413912	**7.548783402600382**

Table 2: The variance values of changes caused by given LLH types recorded as a fraction of improvement of quality measure.

The results show that there is no general pattern concerning the relation of

destructiveness between mutation and ruin-recreate LLHs. As can be seen from the tables, even though the ruin-recreate LLHs are more stable in the sense that the resulting quality measure has lower variance, no assumptions can be made about as in some cases the mutation LLHs provides better results, while in other ruin-recreate dominates.

Therefore, for the purposes of calculating the diversity measure, following values will be used (given in the order [local search, mutation, ruin-recreate]):

- the weighting constants W_t will have the following values [1,2,2] to reflect that local search LLHs make small changes and mutation and ruin-recreate LLHs greater
- the exclusivity constants E_t are set to [1,1,1], which should not make it the most important factor yet have an impact

6.3.3 Control parameters values

	Baseline	Tabu Only	Tabu Restart	Improvement Buffer	Double Buffer
max running time (ms) *	550000	550000	550000	550000	550000
sequence length **	[3,4,5]	[3,4,5]	[3,4,5]	[3,4,5]	[3,4,5]
population size **	[20,20,40]	[20,20,40]	[20,20,40]	[20,20,40]	[20,20,40]
EA generations **	[10,10,15]	[10,10,15]	[10,10,15]	[10,10,15]	[10,10,15]
desired number of EA runs **	[5,5,5]	[5,5,5]	[5,5,5]	[5,5,5]	[5,5,5]
LLH generation - type selection probabilities - local search	0.5	0.5	0.5	0.5	0.5
LLH generation - type selection probabilities - mutation	0.25	0.25	0.25	0.25	0.25
LLH generation - type selection probabilities - ruin-recreate	0.25	0.25	0.25	0.25	0.25
LLH parameter generation - uniform distribution	0.5 ± 0.1	0.5 ± 0.1	0.5 ± 0.1	0.5 ± 0.1	0.5 ± 0.1
tournament size	2	2	2	2	2
crosover probability	0.8	0.8	0.8	0.8	0.8
swap mutation probability	0.15	0.15	0.15	0.15	0.15
replace mutation probability	0.15	0.15	0.15	0.15	0.15
parameter mutation probability	0.15	0.15	0.15	0.15	0.15
param.mut. - uniform distribution	± 0.3	± 0.3	± 0.3	± 0.3	± 0.3
tabu list size	**X**	10	10	10	10
try restarting stagnating ** solution after time fraction ***	**X**	**X**	[1,0.075,0.05]	[1,0.075,0.05]	[1,0.075,0.05]
stagnating solution restart initial probability **	**X**	**X**	[0.25,0.25,0.25]	[0.25,0.25,0.25]	[0.25,0.25,0.25]
force restart of stagnating ** solution after time fraction ***	**X**	**X**	[1,0.1,0.1]	[1,0.1,0.1]	[1,0.1,0.1]
mutation or ruin-recreate LLHs generated for restart	**X**	**X**	1	1	1
restarting LLH parameters generated by uniform distribution	**X**	**X**	0.175 ± 0.075	0.175 ± 0.075	0.175 ± 0.075
diversity weighting constant LS/M/RR ****	**X**	**X**	1 / 2 / 2	1 / 2 / 2	1 / 2 / 2
improvement buffer size as a fraction of the population	**X**	**X**	**X**	1	0.5
fraction of the population replaced by improvement buffer (first EA generation only)	**X**	**X**	**X**	0.5	**X**
population fraction replaced by improvement buffer (every EA generation)	**X**	**X**	**X**	**X**	0.25
no-improvement buffer size as population fraction	**X**	**X**	**X**	**X**	0.5
population fraction replaced by no-improvement buffer (first EA generation only)	**X**	**X**	**X**	**X**	**X**
population fraction replaced by no-improvement buffer (every EA generation)	**X**	**X**	**X**	**X**	0.25

* note that the running time should always be machine specific
** parts of VCPS
*** when the parameter value is 1, no restart will take place
**** exclusivity constant has a fixed value =1

Table 3: The control parameters of all the versions of HEADS.

6.4 Experiments' results

6.4.1 CHeSC ranking

The figures 7-10 show the ranking for all the competition problem instances and the total points and final rank. Note that the ranking reflects new distribution of points as the introduction of new competitor changes it.

It can be seen that only the baseline is falls behind the other versions, placing 10th. The extensions of the baseline, namely Tabu Only, Tabu Restart, Improvement Buffer and Double Buffer, have produced results of comparable quality, placing 6th, 5th, 6th and 4th respectively.

However, the results suggest that the decision of using the EA as the main driving force in search is the most important factor influencing the quality of the results. This can be seen especially in the case of Bin Packing and MAX-SAT problems. In the former, all the algorithms delivered very good results, while on the latter all the algorithms provided rather poor results. This leads to an assumption that there is an inherent feature of EA that prevents the algorithm from being effective on problems while making it very effective on others.

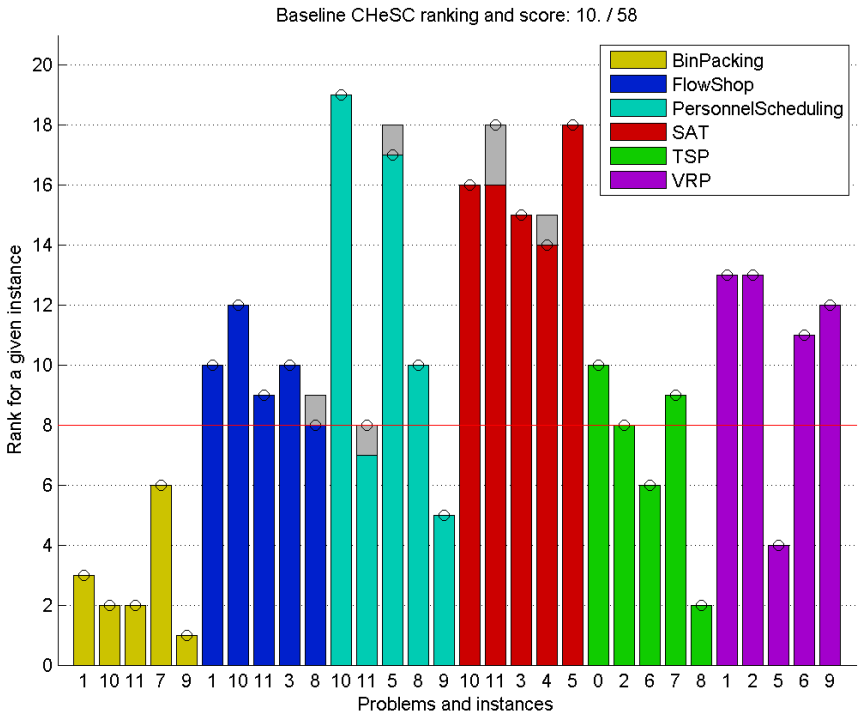

Figure 7: The ranking of the Baseline HEADS.

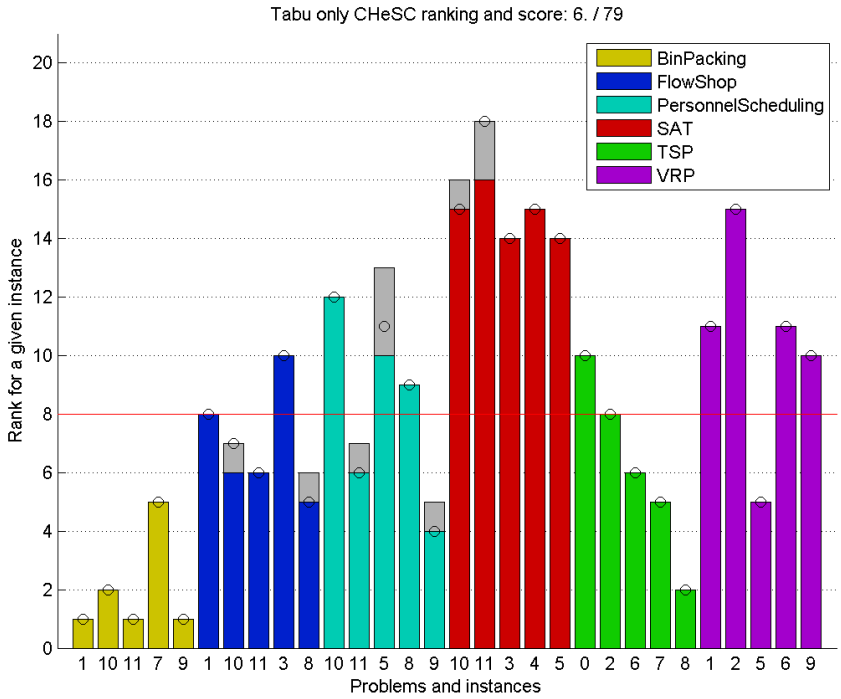

Figure 8: The ranking of the Tabu Only HEADS.

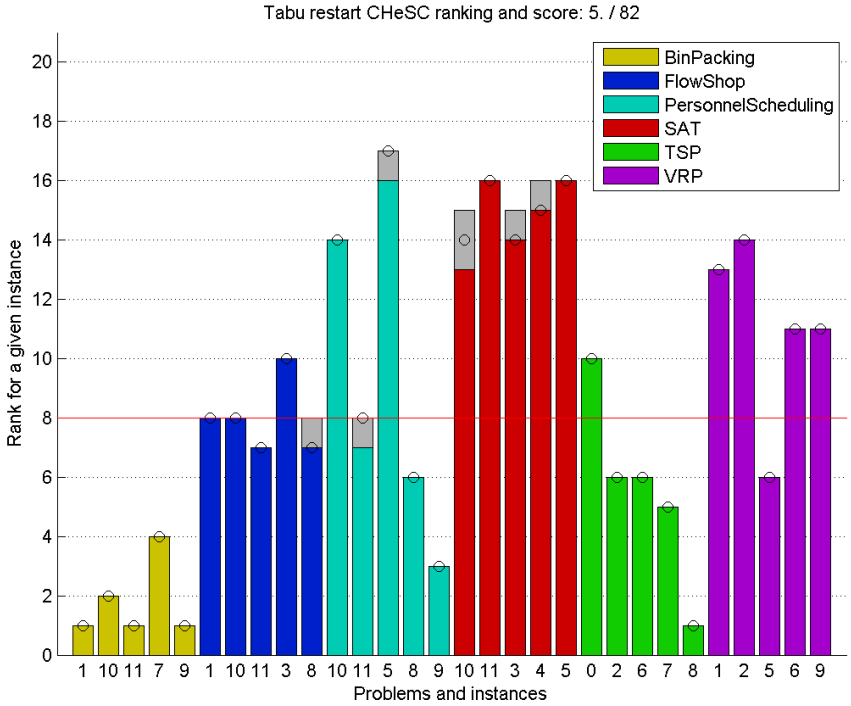

Figure 9: The ranking of the Tabu Restart HEADS.

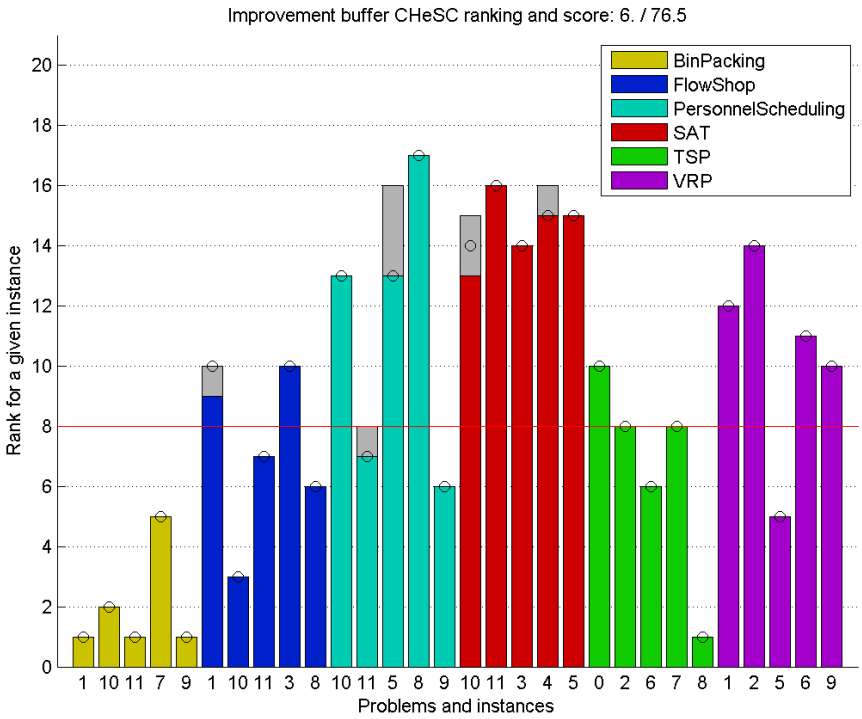

Figure 10: The ranking of the Improvement Buffer HEADS.

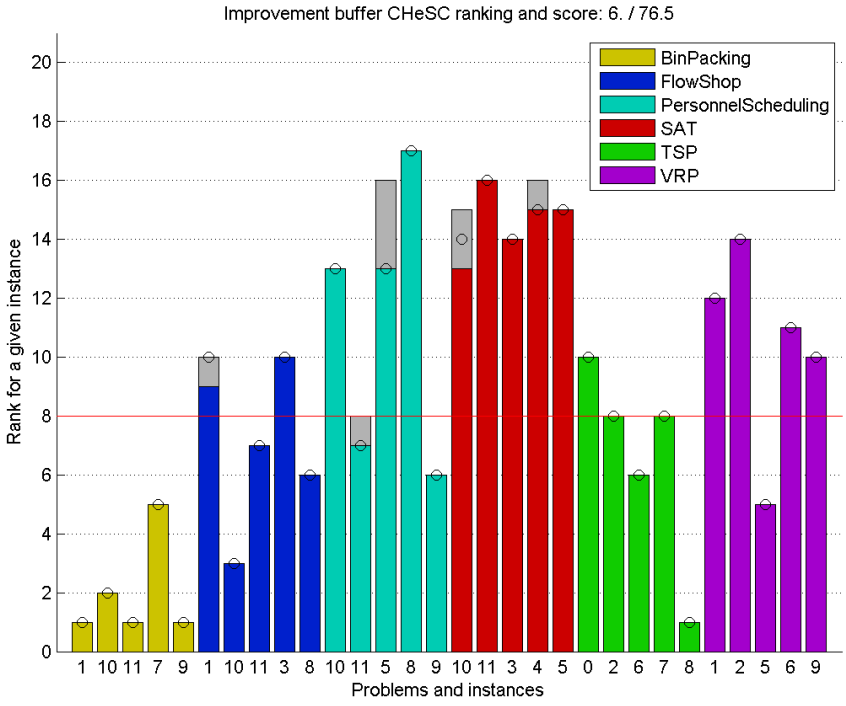

Figure 11: The ranking of the Double Buffer HEADS.

6.4.2 Significance test

In the table N, the median values of results given by the 31 runs of different algorithm variants can be seen. Note that the highlighted values are those that are nondominated, i.e. there is no algorithm for the given instance that would provide results that would be better with significance level 95%. The level of significance 80% is present to illustrate the possible trends if more data were added.

The table 4 shows the sum of problem instances where the algorithm variant was dominated, nondominated or solo dominating. The meaning is the following, with the corresponding level of significance :

- Solo dominating means that this variant outperformed all the other variants.
- Nondominated means that there was not a variant that would outperform this one, but there was at least one variant that was also not outperformed.
- Dominated means there was at least one variant that outperformed this one.

Significance **90%**	**Dominating**	**Nondominated**	**Dominated**
Baseline	0	11	19
Tabu only	0	26	4
Tabu restart	0	20	10
Improvement buffer	0	24	6
Double buffer	2	26	4

Significance **80%**	**Dominating**	**Nondominated**	**Dominated**
Baseline	0	7	23
Tabu only	0	20	10
Tabu restart	0	14	16
Improvement buffer	0	19	11
Double buffer	4	25	5

Table 4:The ratio of dominating, non-dominated and dominated problem instances.

	Baseline	Tabu only	Tabu restart	Improvement buffer	Double buffer	Dominance with significance 95,00%	80,00%
BinPacking_1	0,0033887	0,0029889	0,0031032	0,003006	0,0031188	2 4	2 4
BinPacking_10	0,10839	0,10835	0,10846	0,10834	0,10833	5	5
BinPacking_11	0,0053185	0,0024389	0,0034627	0,0024803	0,0034755	2 4	2 4
BinPacking_7	0,036598	0,031843	0,029431	0,031518	0,028744	3 5	5
BinPacking_9	0,0025776	0,0014917	0,002444	0,0014581	0,0024689	2 4	2 4
FlowShop_1	6268	6256	6259	6262	6250	2 5	2 5
FlowShop_10	11428	11393	11401	11381	11388	2 4 5	2 4 5
FlowShop_11	26664	26649	26654	26653	26650	2 3 4 5	2 3 4 5
FlowShop_3	6363	6363	6363	6356	6353	2 3 4 5	3 4 5
FlowShop_8	26850	26826	26844	26827	26828	2 3 4 5	2 4 5
PersonnelScheduling_10	1815	1679	1745	1720	1768	2 3 4 5	2 3 4 5
PersonnelScheduling_11	335	330	335	335	325	2 3 4 5	2 5
PersonnelScheduling_5	30	25	27	26	26	2 3 4 5	2 4 5
PersonnelScheduling_8	3260	3253	3237	3311	3263	1 2 3 4 5	1 2 3 4 5
PersonnelScheduling_9	9672	9667	9657	9736	9659	1 2 3 4 5	1 2 3 5
SAT_10	16	15	14	14	15	2 3 4 5	2 3 4 5
SAT_11	13	13	12	12	11	1 3 4 5	3 4 5
SAT_3	12	10	11	10	9	2 4 5	2 5
SAT_4	9	10	12	12	8	5	5
SAT_5	38	16	25	22	14	2 5	5
TSP_0	48194,9201	48194,9201	48194,9201	48194,9201	48194,9201	1 2 3 4 5	1 2 3 4 5
TSP_2	6828,6949	6827,6814	6823,0446	6825,9926	6829,2791	1 2 3 4 5	1 2 3 4 5
TSP_6	53615,9971	53657,2687	53562,277	53540,2638	53559,9501	1 2 3 4 5	3 4 5
TSP_7	67366,3496	67140,6083	67149,8006	67304,7777	67082,0217	2 3 5	2 3 5
TSP_8	20838186,4603	20823038,2245	20817627,4428	20814087,0115	20808821,3336	2 3 4 5	3 4 5
VRP_1	20660,401	20657,3289	20660,9316	20657,568	20660,4554	1 2 3 4 5	2 4
VRP_2	13358,2675	13370,1021	13363,6056	13366,8536	13364,131	1 2 3 4 5	1 3 5
VRP_5	147008,6627	148355,9778	148544,1016	148367,893	148523,2549	1 2 4	1 2
VRP_6	68119,0796	68843,7015	68974,2182	68005,9274	68157,3971	1 2 3 4 5	1 4 5
VRP_9	150607,2933	149999,171	150372,506	150015,0521	150448,2057	1 2 3 4 5	2 3 4 5

Table 5:The median results of all HEADS algorithms with tests of domination with two levels of significance.

It is worth noting that the most notable differences in performance were observed in the Bin Packing problem domain. However, it is arguable whether this is caused by the nature of the problem, which could be causing some of the variants to perform worse than others, or if it is caused by the quality measure that is capable of distinguishing even the slightest change in the solutions.

No other problem domain demonstrated an intresting behaviors.

The results of these test suggests that the baseline version can be indeed called inferior to other variants as it was dominated in almost two thirds of the problem instances. However, the situation is not so clear with the rest of the variants.

It would be tempting to call the double buffered variant a winner as it is the only one that managed to achieve solo domination on both the 80% and 95% level of significance. Moreover, this variant was the only one to avoid reduction of number of nondominated results when reducing the level of significance from 95% to 80%. However, bear in mind that the number of experiments executed was relatively small and more data would be needed to be able to name the winner with certainty.

Therefore, if using the data currently at disposal, the only correct thing to say is that it is impossible to rank the performance of the following 4 variants: Tabu Only, Tabu Restart, Improvement Buffer and Double Buffer.

Chapter 7

7 Conclusion

This thesis proposes an evolutionary algorithm based hyper-heuristic approach to solving black box optimization problems that would be abble to efficiently tackle different problem domains. To measure the quality of the resulting algorithm, competition data and problem definitions from the Cross-domain Heuristic Search Challenge (CHeSC) 2011 were used. The java framework HyFlex, which was designed for the purposes of the challenge, was used as an implementation support.

Based on the research conducted, five different versions of the proposed algorithm of various complexity were introduced. The simplest *Baseline* version utilizes only evolutionary techniques. The EA population was formed by sequences of low-level heuristics(LLHs), which were used to improve a single solution. The fitness was equal to the quality measure of the solution created. The *Tabu Only* adds tabu list to restrict the new solutions created to unexplored regions of solution space. The *Tabu Restart* added solution restart on longer periods of stagnation by means of application of destructive LLHs on the solution. The last two version added sequence buffers to promote EA population diversity by partially filling new population from these buffer. The *Improvement Buffer* uses one buffer to store improving sequences only, while the *Double Buffer* stores bot non-improving and improving sequences.

The reason for having different versions was the aim to identify the contribution of each of these extensions and focus on those parts of the algorithm that provide the greatest improvement in problem solving.

All these versions were tested on the same problem instances and under same conditions as the algorithms that were competing in the CHeSC 2011 challenge and then compared to them using the scoring system designed for the competition. The algorithms independently placed 10th (Baseline), 6th (Tabu Only), 5th (Tabu Restart), 6th (Improvement Buffer) and 4th (Double Buffer) respectively, proving that the proposed method provides competitive results to the current state of the art approaches.

However, statistical testing did not managed to disprove the null hypothesis that on general the different versions are returning results drawn from the same distribution, the only exception being the simplest *Baseline* version, which was outperformed in almost two thirds of

problem instances. This means that this work failed to identify the most promising extensions of EA as different versions benefited from different structures of problem instances. However, it was proven that all of the extensions to the baseline version of the algorithm caused an improvement in the quality of returned solutions. This is a good result.

Future research in this field could focus on the following topics:

- A detailed research of concepts used in dynamic environment EAs and the possibilities of application of these concepts in the context of the proposed algorithm could improve its effectiveness. Since the algorithm works with a evaluation solution can be changing quite frequently and therefore also changing the fitness function calculation, incorporation of these concepts into the algorithm could yield significant improvement in effectiveness.
- Since different problems use different quality function, an improved method of overcoming plateaus should be introduced for problems where the solution quality is very sensitive to even small changes, causing that neighboring solutions never have the same fitness, but the difference is insignificant. This should improve the speed of reaching the local optima.

Bibliography:

[1] Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E. and Woodward, J. (2009), 'A classication of hyper-heuristic approaches'.

[2] Keller, R. E. and Poli, R. (2008), Subheuristic search and scalability in a hyperheuristic, in 'Proceedings of the 10th annual conference on Genetic and evolutionary computation', GECCO '08, ACM, New York, NY, USA, pp. 609-610.

[3] Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P. and Schulenburg, S. (2003), Hyper-heuristics: An emerging direction in modern search technology, in F. Glover and G. Kochenberger, eds, 'Handbook of Metaheuristics', Vol. 57 of International Series in Operations Research & Management Science, Springer New York, pp. 457-474.

[4] Bai, R. et. al. (2007), A simulated annealing hyper-heuristic:Adaptive heuristic selection for different vehicle routing problems, The 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2007).

[5] Qu, R. and Burke, E. (2009), 'Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems', Journal of the Operational Research Society 60, 1273-1285.

[6] Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Ozcan, E. aWoodward, J. R. (2009) 'Exploring hyper-heuristic methodologies with genetic programming', Computational Intelligence 1, 177-201.

[7] Schneider, J. J. and Kirkpatrick, S. (2006), The traveling salesman problem, in `Stochastic Optimization', Scientic Computation, Springer Berlin Heidelberg, pp. 211-231.

[8] Beck, J. C., Prosser, P. and Selensky, E. (2003), Vehicle routing and job shop scheduling: What's the difference?, in 'Proc. of the 13th International Conference on Automated Planning and Scheduling', pp. 267-276.

[9] Bilgin, B., Özcan, E. and Korkmaz, E. (2007), An experimental study on hyper-heuristics and exam timetabling, in E. Burke and H. Rudová, eds, 'Practice and Theory of Automated Timetabling VI', Vol. 3867 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, pp. 394-412.

[10] Wolpert, D. H. and Macready, W. G. (1997), 'No free lunch theorems for optimization', IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION 1(1), 67-82.

[11] Hansen, P. and Mladenovič, N. (2001), 'Variable neighborhood search: Principles and applications', European Journal of Operational Research 130(3), 449-467.

[12] Marín-Blázquez, J. and Schulenburg, S. (2007), A hyper-heuristic framework with XCS:

Learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients, in T. Kovacs, X. Llor#a , K. Takadama, P. Lanzi, W. Stolzmann and S. Wilson, eds, 'Learning Classier Systems', Vol. 4399 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, pp. 193-218.

[13] Sim, K. (2011), KSATS-HH: A Simulated Annealing Hyper-Heuristic with Reinforcement Learning and Tabu-Search[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/sim-chesc.pdf>.

[14] Han, L. and Kendall, G. (2003), Investigation of a tabu assisted hyper-heuristic genetic

algorithm, in 'In proceedings of Congress on Evolutionary Computation (CEC2003)', IEEE Press, pp. 2230-2237.

[15] Cowling, P., Kendall, G. and Han, L. (2002), An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem, in 'Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress - Volume 02', CEC '02, IEEE Computer Society, Washington, DC, USA, pp. 1185-1190.

[16] Burke, E. K., Kendall, G. and Soubeiga, E. (2003), 'A tabu-search hyperheuristic for timetabling and rostering', Journal of Heuristics 9, 451-470.

[17] Terashima-Marin, H., Ross, P. and Valenzuela-Rendon, M. (1999), Evolution of constraint satisfaction strategies in examination timetabling, in W. Banzhaf, J. Daida, A. E.Eiben, M. H. Garzon, V. Honavar, M. Jakiela and R. E. Smith, eds, 'Proceedings of the Genetic and Evolutionary Computation Conference', Vol. 1, Morgan Kaufmann, Orlando, Florida, USA, pp. 635-642.

[18] Kendall, G. and Mohd Hussin, N. (2004), Tabu search hyper-heuristic approach to the examination timetabling problem at university technology MARA, in 'In proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2004)', pp. 199-217.

[19] Tim Curtois, Gabriela Ochoa, M. H. J. A. V.-R. (2011), A HyFlex Module for the Personnel Scheduling Problem[online]. [cit. 2011-11-28], <http://www.asap.cs.nott.ac.uk/chesc2011/reports/PermutationFlowshopHyFlex.pdf>.

[20] Özcan, E., Bilgin, B. and Korkmaz, E. E. (2008), 'A comprehensive analysis of hyper-heuristics', Intelligent Data Analysis 12, 3-23.

[21] Özcan, E., Bilgin, B. and Korkmaz, E. (2006), Hill climbers and mutational heuristics in hyperheuristics, in T. Runarsson, H.-G. Beyer, E. Burke, J. Merelo-Guervás, L. Whitley and X. Yao, eds, 'Parallel Problem Solving from Nature - PPSN IX', Vol. 4193 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, pp. 202-211.

[22] Cowling, P. I., Kendall, G. and Soubeiga, E. (2001), A hyperheuristic approach to scheduling a sales summit, in 'Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III', PATAT '00, Springer-Verlag, London, UK, pp. 176-190.

[23] Ashlock, D. (2004), Evolutionary Computation for Modeling and Optimization, Springer New York. ISBN 978-0-387-22196-0.

[24] José Antonio Vázquez-Rodríguez, Gabriela Ochoa, T. C. M. H. (2011), A HyFlex Module for the Permutation Flow Shop Problem[online]. [cit. 2011-11-28], <http://www.asap.cs.nott.ac.uk/chesc2011/reports/PermutationFlowshopHyFlex.pdf>.

[25] Beyer, H.-G. and Schwefel, H.-P. (2002), 'Evolution strategies -a comprehensive introduction', 1, 3-52.

[26] Matthew Hyde, Gabriela Ochoa, J. A. V.-R. T. C. (2011), A HyFlex Module for the MAX-SAT Problem[online]. [cit. 2011-11-28], <http://www.asap.cs.nott.ac.uk/chesc2011/reports/BoolenSatisfiabilityHyflex.pdf>.

[27] Matthew Hyde, Gabriela Ochoa, J. A. V.-R. T. C. (2011), A HyFlex Module for the One Dimensional BinPacking Problem[online]. [cit. 2011-11-28], <http://www.asap.cs.nott.ac.uk/chesc2011/reports/BinPackingHyFlex.pdf>.

[28] Khamassi, I. (2011), Ant-Q Hyper Heuristic Approach applied to the Cross-domain

Heuristic Search Challenge problems[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/khamassi-chesc.pdf>.

[29] Ross, P., Marín-Blázquez, J., Schulenburg, S. and Hart, E. (2003), Learning a procedure that can solve hard bin-packing problems: A new GA-based approach to hyper-heuristics, in E. Cantú-Paz, J. Foster, K. Deb, L. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. Potter, A. Schultz, K. Dowsland, N. Jonoska and J. Miller, eds, 'Genetic and Evolutionary Computation GECCO 2003', Vol. 2724 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, pp. 215-215.

[30] McClymont, K. and Keedwell, E. (2011), A Single Objective Variant of the Online Selective Markov chain Hyper-heuristic (MCHH-S)[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/mcclymont-chesc.pdf>.

[31] Elomari, J. (2011), Self-Search (Extended Abstract)[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/elomari-chesc.pdf>.

[32] Burke, E., Curtois, T., Hyde, M., Ochoa, G. and Vazquez-Rodriguez, J. A. (2011), 'HyFlex: A benchmark framework for cross-domain heuristic search', ArXiv e-prints.

[33] Misir, M. et.al. (2011), An adaptive hyper-heuristic for CHeSC 2011[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/misir-chesc.pdf>.

[34] Ping-Che Hsiao, Tsung-Che Chiang, L.-C. F. (2011), A variable neighborhood search-based hyperheuristic for cross-domain optimization problems in CHeSC 2011 competition[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/hsiao-chesc.pdf>.

[35] Larose, M. (2011), A Hyper-heuristic for the CHeSC 2011[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/larose-chesc.pdf>.

[36] Meignan, D. (2011), An Evolutionary Programming Hyper-heuristic with Co-evolution for CHeSC'11[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/meignan-chesc.pdf>.

[37] Fan Xue, C.Y. Chan, W. I. and Cheung, C. (2011), Pearl Hunter: A Hyper-heuristic that Compiles Iterated Local Search Algorithms[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/xue-chesc.pdf>**.**

[38] Meignan, D. (2011), An Evolutionary Programming Hyper-heuristic with Co-evolutionfor CHeSC'11[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/meignan-chesc.pdf>.

[39] Kubalík, J. (2011), Iterated Search Driven by Evolutionary Algorithm Hyper-Heuristic[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/kubalic-chesc.pdf>.

[40] Alberto Acuña, V. P. and Gatica, G. (2011), Cross-domain Heuristic Search Challenge: GISS Algorithm presentation[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/acuna-chesc.pdf>.

[41] Gómez, J. (2011), Hybrid Adaptive Evolutionary Algorithm Hyper Heuristic[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/gomez-chesc.pdf>.

[42] JoséLuis Núnez, A. C. (2011), A general purpose Hyper-Heuristic based on Ant colony optimization[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/nunez-chesc.pdf>.

[43] Michal Frankiewicz, Tomasz Cichowicz et.al. (2011), Genetic Hive HyperHeuristic[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/csput-chesc.pdf>.

[44] Mark Johnston, Thomas Liddle, J. M. and Zhang, M. (2011), A Hyperheuristic Based on Dynamic Iterated Local Search[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/johnston-chesc.pdf>.

[45] Luca Di Gaspero, T. U. (2011), A Reinforcement Learning approach for the Cross-Domain Heuristic Search Challenge[online]. [cit. 2011-10-22], <http://www.asap.cs.nott.ac.uk/chesc2011/entries/urli-chesc.pdf>.

Appendix

Contents of the CD

- HEADS - source codes of the HEADS algorithm in NetBeans project
- matlab_scripts - scripts for Matlab that were used to process experiment data
- experiments_results - the results that were presented in the thesis
- HEADS Demo - short introduction to HEADS with example settings
- thesis.pdf - this thesis in pdf format