Hyper-Heuristic Based on Iterated Local Search Driven by Evolutionary Algorithm

Jiří Kubalík

Department of Cybernetics
Czech Technical University in Prague
Technická 2, 166 27 Prague 6, Czech Republic
kubalik@labe.felk.cvut.cz

Abstract. *This paper proposes an evolutionary-based iterative local search hyper-heuristic approach called Iterated Search Driven by Evolutionary Algorithm Hyper-Heuristic (ISEA). Two versions of this algorithm, ISEA-chesc and ISEA-adaptive, that differ in the re-initialization scheme are presented. The performance of the two algorithms was experimentally evaluated on six hard optimization problems using the HyFlex experimental framework [4] and the algorithms were compared with algorithms that took part in the CHeSC 2011 challenge [10]. Achieved results are very promising, the ISEA-adaptive would take the second place in the competition. It shows how important for good performance of this iterated local search hyper-heuristic is the re-initialization strategy.*

Keywords: hyper-heuristic, optimization, evolutionary algorithm

1 Introduction

Hard optimization problems cannot be solved to optimality for large instance sizes due to extreme computational demands. Hence, either approximation or metaheuristic algorithms [9] are often used to find if not optimal solutions then at least solutions of good quality in reasonable time. Recently, hyper-heuristics[1], described as "heuristics to choose heuristics", that are search methods or learning mechanisms for selecting or generating heuristics to solve computational search problems were proposed [2].

1.1 Hyper-Heuristics

There are two main classes of hyper-heuristic approaches – *heuristic selection* and *heuristic generation*. Heuristic selection approaches, for given particular problem instance and a set of problem-specific low-level heuristics (LLHs), select and apply the most suitable LLH at each problem solving state. Heuristic generation methods are used to automatically generate new heuristics, suited for the given problem, from components of pre-existing heuristics [1].

[1] Comprehensive bibliography of Hyper-heuristics:
http://www.cs.nott.ac.uk/~gxo/hhbibliography.html

The heuristic selection methods can be divided into *constructive* and *local search* hyper-heuristics based on the type of used LLHs. Constructive hyper-heuristics start with an empty solution and gradually build a complete solution, by iteratively selecting appropriate low-level construction heuristic (chosen from a pool of LLHs proposed for the problem at hand) and using it to enhance the developed solution. Local search heuristics start from a complete solution, then they try to iteratively improve the current solution by selecting appropriate neighborhood structure and/or simple local searcher and applying it to the current solution.

Typically, the heuristic selection methods operate on sequences of LLHs where an optimal sequence that produces the best solution (in case of constructive methods) or improves the most the initial solution (in case of local search methods) is sought. Since the construction of the optimal sequence of LLHs is not a trivial task, high-level heuristic or meta-heuristic techniques such as ILS, VNS, simulated annealing, tabu search or EAs are often used as search strategies across the search space of heuristics [2]. Some methods make use of information learned through the problem solving process, such as performance of individual LLHs, for selecting particular heuristic at given decision point. An association of LLHs to characteristic problem solving states can also be used. For example, at some point local search and mutation heuristics can be very effective while in other problem solving state more disturbing ruin-recreate heuristics can be the only way to escape from current local optimum solution [3].

Recent analysis show that the heuristic search space is likely to contain large plateaus, i.e. regions of many heuristic sequences producing solutions of the same quality. On the other hand, the heuristics search space have in a great picture a globally convex structure with the optimal solution surrounded by many local optima [2]. In this work we propose hyper-heuristic approach based on iterative local search algorithm called POEMS [6] that might be well suited for searching this kind of search space since it can make use of structured improving moves when trying to improve the current solution.

According to the classification provided in Section 1.1, the proposed hyper-heuristic belongs to the heuristic selection approaches. Specifically, to the local search hyper-heuristics as the goal of the hyper-heuristic is to iteratively improve starting complete solution by iteratively selecting and applying local search, mutation and ruin-recreate low level heuristics to it.

1.2 Proposed Hyper-Heuristic

Iterated Search Driven by Evolutionary Algorithm Hyper-Heuristic (ISEA) presented in this paper is based on an evolutionary-based iterative local search algorithm called POEMS [6–8]. POEMS is an optimization algorithm that operates on a single candidate solution called a *prototype* and tries to improve it in an iterative process. In each iteration, it runs an evolutionary algorithm (EA) that seeks for the most valuable modification to the prototype. The modifications are represented as fixed length sequences of elementary actions, i.e.

sequences of problem-specific variation or mutation operators. Such action sequences, produced by the EA, can be viewed as evolved *structured mutations*. Action sequences are assessed based on how well/badly they modify the current prototype, which is passed as an input parameter to the evolutionary algorithm. Besides actions that truly modify the prototype, there is also a special type of action called *nop* (no operation). The nop actions are interpreted as void actions with no effect on the prototype. Action sequences can contain one or more nop actions. This way a variable effective length of action sequences is realized. After the EA finishes, the best evolved action sequence is checked for whether it worsens the current prototype or not. If an improvement is achieved or the modified prototype is at least as good as the current one, then the modified prototype is considered as a new prototype for the next iteration. Otherwise, the current prototype remains unchanged. The iterative process stops after a specified number of iterations.

POEMS takes the best of the two worlds of the *single-state* and *population-based* metaheuristics. It iteratively improves the current solution, but contrary to the single-state metaheuristics it searches much bigger neighborhood structure defined by fixed length sequence of elementary actions (not just a single variation operator). Such a neighborhood structure is effectively searched by means of the EA, where the EA is capable of finding both the local fine-tuning move as well as the perturbation move.

Therefore, the exploration capability of POEMS should be better than the standard single-state techniques. Moreover, the EA run in each iteration searches a limited space of the current prototype's modifications instead of the whole space of all candidate solutions to the given problem that is searched by the traditional EAs. Thus, minimal resources (i.e. small population size and small number of generations) can be sufficient to effectively search the space of structured mutations.

Two versions of the ISEA hyper-heuristic were implemented and tested with the use of the HyFlex framework [4]. The two ISEA versions were evaluated and compared with the algorithms that took place in the competition of the Cross-domain Heuristic Search Challenge CHeSC 2011[2]. The performance of the algorithms was assessed based on the point scoring system used for the competition. The competition results and the program for calculating the hyper-heuristics' score being kindly provided by the organizers of the CHeSC 2011. The presented analysis shows how important for good performance of the iterated local search ISEA hyper-heuristic is the re-initialization strategy.

Map of the paper: In Section 2 an original ISEA version is proposed in the form that took part in CHeSC 2011 challenge. Section 3 presents the version with an adaptive re-initialization rate. Section 4 presents experimental evaluation of the two ISEA algorithms. Last section concludes the paper and proposes future work directions.

[2] `http://www.asap.cs.nott.ac.uk/chesc2011/index.html`

2 Original ISEA Algorithm

The general-purpose ISEA hyper-heuristic is based on the POEMS approach, but differs in several aspects. In the following paragraphs the main structure of the ISEA algorithm and its key components will be described.

Memory solutions. Generally, the ISEA maintains a set of three solutions to the given problem via methods provided by the HyFlex abstract class `ProblemDomain`. The meaning of the three solutions is as follows:

- *Evaluation solution* is used for evaluating candidate sequences of low level heuristics (LLHs) defined for the problem at hand.
- *Working solution* stores intermediate solutions and the final solution that are successively generated from the starting evaluation solution by applying individual LLHs of the evaluated candidate sequence of LLHs. Thus, at the beginning of the candidate sequence of LLHs evaluation process the *working solution* is set to the *evaluation solution* and then it is modified step-by-step as the LLHs are applied to it one by one.
- *Best-so-far solution* is the best solution found so far in the whole ISEA run.

Low level heuristics and the prototype. Unlike other traditional applications of POEMS, ISEA operates on a prototype that does not directly represent a solution to the given problem. Instead, the prototype is a fixed-length sequence of LLHs that is to be modified by evolved action sequences. Despite the HyFlex framework provides four types of LLHs, the ISEA considers only local search, mutation and ruin-recreate heuristics for the generated sequences of LLHs. When initialized, LLHs are initially assigned to conform with the following schema:

- At the first position of the prototype, only local search LLH can be generated.
- All of the remaining positions but the last one can be initialized with any type of LLH, where all of the three types of LLH have equal probability to be chosen.
- Only local search LLH can be assigned to the last position of the prototype.

The idea behind this scheme is that when looking for better solution in the neighborhood of the *working solution*, one might try to locally optimize it first, then any type of LLH can be chosen multiple times and finally some local search heuristics can be used to finalize the new solution.

Actions. Note, that the prototype is to be modified by evolved action sequences. Following three simple actions are proposed for the ISEA:

- `addLLH(position, type, parameters)` – adds a new LLH of certain type with specified parameters to the prototype at given position.
- `removeLLH(position)` – removes the LLH that is at given position in the prototype.
- `changeLLH(position, parameters)` – modifies the LLH at given position in the prototype according to the new parameters. It can change either the type of the LLH (mutation / local search / ruin-recreate) and/or its parameters.

Obviously, the length of the prototype can vary as some LLH can be added to or removed from it. At the beginning of the EA, the prototype's length is set to p (that is one of the user defined parameters of the algorithm).

The task for the EA is to permanently evolve modifications to the current prototype that will, if possible, produce sequences of LLHs capable of improving the *evaluation solution* at any moment of the optimization process. Thus, it is assumed that at some stages the evolved action sequences will be able to adjust the prototype to realize rather disturbing moves (in order to escape a local optimum of the solved problem instance represented by the current *evaluation solution*) while at other stages the action sequences can transform the prototype so as to realize fine-tuning moves (in case there is still some room for improvement of the *evaluation solution* that can be effectively attained just by using the local search heuristics). In other words, the EA realizes the adaptation of the optimization process to the current status of the *evaluation solution*.

Assessing quality of action sequences. In order to assess the performance of an action sequence, the resulting sequence of LLHs, obtained by modifying the prototype with the action sequence, is applied to the *evaluation solution* and the quality of the best solution generated through successive applying the LLHs of the sequence is taken as the quality of the candidate action sequence.

Perturbation. The *evaluation solution* is re-initialized from time to time so that it is assigned a solution that is created by perturbing the *best-so-far solution*. The perturbation is realized by mutation and ruin-recreate heuristics. This re-initialization operation is invoked as soon as the following two conditions are fulfilled

1. more than T_1 seconds has elapsed since the last perturbation action, and
2. there has been no improvement of the *evaluation solution* observed for T_2 seconds,

indicating that there was no progress observed for considerable amount of time. The parameters T_1 and T_2 are static in a sense that they stay constant during the whole ISEA run. The question is how to set the parameters in order to provide the ISEA procedure with an efficient re-initialization scheme.

2.1 ISEA Pseudo code

At the beginning of the ISEA run, the *prototype*, *evaluation solution*, *best-so-far solution*, and starting population of action sequences are initialized. Also the counter of calculated fitness evaluations and the variables storing the time of the last improvement of the *evaluation solution* and the time of the last perturbation action are reset.

Then the algorithm repeats steps of the main loop until the maximal specified time has expired (steps 8-18 in Fig. 1). In the first step of this block one generation of the EA is carried out. This means that a pair of new action sequences is generated either by means of crossover and/or mutation. The newly generated action sequences are evaluated on the *evaluation solution*. Whenever the new

```
input:  set of local search, mutation and ruin-recreate heuristics
output: best solution found
1       lastPerturbationTime ← currentTime
2       lastImprovementTime ← currentTime
3       evaluations ← 0   // the number of evaluations calculated in one EA
4       initialize(evaluation_solution)
5       best_so_far_solution ← evaluation_solution
6       initialize(prototype)
7       initialize(populationAS)      // initialize population of action sequences
8       while (!hasTimeExpired())
9           calculateGenerationEA()       // calculates one generation of the EA
10          evaluations ← evaluations + 2   // two new solutions are generated
11          if(((currentTime − lastPerturbationTime) > T1) and
              ((currentTime − lastImprovementTime) > T2))
12              evaluation_solution ← perturb(best_so_far_solution)
13              lastPerturbationTime ← currentTime
14          if((evaluations) > N)          // start new EA
15              evaluations ← 0
16              initialize(prototype)
17              initialize(populationAS)
18      end while
19      return best_so_far_solution          // return best-so-far solution
```

Fig. 1. Pseudo code of ISEA hyper-heuristic.

solution is at least as good as the *evaluation solution* the new solution replaces the *evaluation solution*. Similarly, when the new solution outperforms the *best-so-far solution*, the *best-so-far solution* is updated as well (all these operations are done in step 9 in Fig. 1). If the conditions for perturbation are fulfilled, the perturbation is carried out and the $lastPerturbationTime$ variable is updated accordingly (steps 11-13 in Fig. 1).

If the number of fitness evaluations exceeds the specified number of fitness evaluations allocated for one EA, the prototype and the population of action sequences are re-initialized (steps 14-17 in Fig. 1). Once the time allocated for the whole run has expired the *best so far solution* is returned as the final solution.

2.2 ISEA Control Parameters' Setting

Before the main procedure of the ISEA starts, it is checked whether low time demand or high time demand executions of the LLHs are expected for the problem instance at hand. This is estimated by applying several LLHs (five LLHs were used in this work) on the initial *evaluation solution*. Based on the estimate one of two possible configurations is chosen. The configuration for instances with low time demand LLHs uses longer prototypes, longer action sequences, bigger

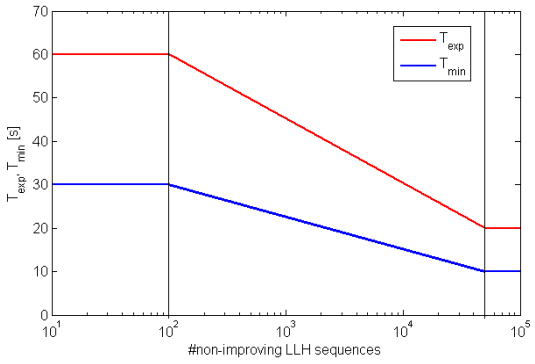

Fig. 2. Mapping the estimated total number of non-improving LLH sequences to the T_{exp} and T_{min}.

population of action sequences and larger number of fitness evaluations calculated in one EA. The configuration for instances with high time demand LLHs uses shorter prototype, shorter action sequences, smaller population of action sequences and smaller number of fitness evaluations in one EA. Note, that just the time complexity of the LLHs for the problem instance at hand is estimated. There is no attempt to disclose the type of the problem at hand.

3 ISEA with Adapted Re-initialization Rate

A variant of ISEA algorithm described in this section does not use the hard-wired re-initialization strategy based on the rule, where the perturbation action takes place if and only if the two crisp time-based conditions for the action are fulfilled. Here, a time interval, (T_{min}, T_{exp}), is determined within which the re-initialization of the *evaluation solution* can take place with certain probability, P_{reinit}. Both, the boundaries of the time interval and the probability P_{reinit} are recalculated anew after each re-initialization action. Then, if a new solution is generated such that it is not better than the current *evaluation solution* (i.e. the candidate sequence of LLHs did not improve the *evaluation solution*) and one of the two following conditions is fulfilled

1. at least T_{exp} seconds has elapsed since the last re-initialization action,
2. at least T_{min} seconds has elapsed since the last re-initialization action and a random number from the interval (0.0, 1.0) smaller than P_{reinit} has been generated,

the new *evaluation solution* is calculated by perturbing the *best-so-far solution*.

Clearly, the parameters T_{min}, T_{exp} and P_{reinit} are crucial for proper functioning of the re-initialization strategy. Their values are derived from the estimated total number of non-improving LLH sequences generated during the whole ISEA

run. The idea behind setting T_{min} and P_{reinit} is such that when solving less time-demanding problem instance, for which we can evaluate large number of candidate LLH sequences per time unit, the re-initialization frequency can be quite high since the algorithm can converge to some local optimum in short time-period. On the other hand, when solving heavy time-demanding problem instance, for which we can evaluate only very small number of candidate LLH sequences per time unit, the re-initialization frequency should be rather low since the algorithm might not be able to converge to a local optimum fast enough.

This idea is realized by an adaptation procedure, which is invoked repeatedly during the ISEA run, consisting of the following steps:

1. A total number of non-improving LLH sequences generated within the whole run, $N_{non-imp}$, is estimated based on the number of non-improving LLH sequences observed so far.
2. Log-linear transformation is used to map $N_{non-imp}$ to the *expected time-interval between two consecutive re-initialization actions*, T_{exp}, see the red line in Fig. 2. The figure shows that for "easy" instances (i.e. instances for which the heuristics are less time-demanding) the ISEA is allowed to re-initialize the *evaluation solution* in shorter time, T_{exp}, while for "hard" instances the expected time interval between two re-initializations gets longer. There are two boundary values of the $N_{non-imp}$ considered for the transformation – the lower boundary of 100 and the upper boundary of 5×10^5. If the value of $N_{non-imp}$ smaller than 100 is estimated then the T_{exp} is set to 60s. For any value of $N_{non-imp}$ greater than 5×10^5 the T_{exp} is set to 20s.
3. Given the values of T_{exp} and $N_{non-imp}$ determined in steps 1. and 2., a number of non-improving LLHs, N_{Texp}, expected within time interval of T_{exp} seconds is estimated.
4. The probability of realizing the re-initialization action is $P_{reinit} = 1/N_{Texp}$. In order to eliminate rapid re-initialization of the evaluation solution, a minimal time-interval between two consecutive re-initializations, T_{min}, is set to $T_{exp}/2$, see Fig. 2.

Note, the adaptation procedure is hardware-oriented as it relies on absolute time constants, T_{min} and T_{exp}, in seconds. These can easily be replaced with hardware-independent time values given as fractions of the total running time.

4 Experiments

This section presents experiments carried out with the two versions of the hyper-heuristic approach ISEA

- ISEA-chesc – this version, described in Section 2, took part in the final competition CHeSC 2011.
- ISEA-adaptive – this is the version with adapted re-initialization rate.

The two versions of the ISEA hyper-heuristic were implemented and tested with the use of the HyFlex framework [4]. The framework provides experimental environment with the following six hard optimization problems implemented:

– Max-SAT problem (MSAT),
– Bin packing problem (BP),
– Personnel scheduling problem (PS),
– Flow shop problem (FS),
– Traveling salesman problem (TSP),
– Vehicle routing problem (VRP).

The two ISEA algorithms were compared with the algorithms that took place in the competition of the Cross-domain Heuristic Search Challenge CHeSC 2011[3]. Likewise in the CHeSC 2011 competition, 31 runs per instance were conducted with each of the two ISEA algorithm variants. The two ISEA algorithms were compared in an indirect way so that each of them was added separately to the set of CHeSC 2011 competition algorithms and the following performance indicators were used for their mutual comparison:

– The final rank of each algorithm and total number of points received in the competition.
– Individual scores of each algorithm per domain.
– Median of the best objective values calculated for each instance from the set of 31 runs.

The competition results and the program for calculating the hyper-heuristics' score being kindly provided by the organizers of the CHeSC 2011 competition.

4.1 Experimental Setup

ISEA-chesc configuration for easy and hard instances, see Section 2.2:

– common parameters:
  • total running time: 600 [s]
  • probability of crossover: 75%
  • probability of mutation: 25%
  • tournament size: 2
– parameters for easy instances:
  • initial prototype length (i.e. a length of the LLHs sequence): 5
  • population size:50
  • action sequence length: 5
  • max. number of evaluations in one EA: 200
  • $T_1 = 45$ [s]; $T_2 = 10$ [s]
– parameters for hard instances:
  • initial prototype length: 3
  • population size: 30
  • action sequence length: 3
  • max. number of evaluations in one EA: 50
  • $T_1 = 60$ [s]; $T_2 = 10$ [s]

The same configuration was used for the ISEA-adaptive with one difference – parameters T_1 and T_2 were replaced with the parameters used for adapting the probability P_{reinit}, see Fig. 2 and accompanying text.

[3] `http://www.asap.cs.nott.ac.uk/chesc2011/index.html`

Table 1. Comparisons of ISEA-chesc and ISEA-adaptive on CHeSC 2011 test suite.

algorithm		Total	SAT	BP	PS	FS	TSP	VRP
ISEA-chesc	rank	8	9	2	5	11	7	13
	points	71.0	6.0	30.0	14.5	3.5	12.0	5.0
ISEA-adaptive	rank	**2**	14	**1**	**4**	**2-3**	**3**	**3**
	points	**145.75**	0.25	**42.0**	**22.5**	**34.0**	**24.0**	**23.0**

4.2 Results

Results are summarized in Table 1 and Table 2. The main observation is that ISEA-adaptive significantly improved the final score over ISEA-chesc. ISEA-adaptive was ranked as the second best hyper-heuristic with the total score of 145.75 points among all competition hyper-heuristics while ISEA-chesc placed at eighth position with the total number of 71 points, see Table 1. Recall, that the results presented in Table 1 come from two separate competitions. We can see that ISEA-adaptive outperformed ISEA-chesc on all problems but the Max-SAT one.

Table 2. Results obtained by ISEA-chesc and ISEA-adaptive on CHeSC 2011 test suite. Median values from 31 final objective values per instance are presented. Statistically significant improvements of the ISEA-adaptive over the ISEA-chesc are emphasized in bold as confirmed by the Wilcoxon rank sum test at the 1% level.

		instance				
problem	algorithm	1	2	3	4	5
SAT	ISEA-chesc	5	11	4	9	11
	ISEA-adaptive	7	12	5	11	10
BP	ISEA-chesc	0.034223	0.003284	0.003655	0.108625	0.006400
	ISEA-adaptive	0.034025	**0.002936**	**0.001669**	**0.108386**	**0.002487**
PS	ISEA-chesc	20	9966	3308	1660	315
	ISEA-adaptive	20	9841	3274	**1587**	315
FS	ISEA-chesc	6262	26844	6366	11419	26663
	ISEA-adaptive	**6242**	**26811**	**6325**	**11364**	**26636**
TSP	ISEA-chesc	48194.9	20868203.1	6832.6	67282.1	54129.2
	ISEA-adaptive	**48194.9**	**20827231.2**	6832.8	**66983.3**	55248.6
VRP	ISEA-chesc	70471.7	13339.8	149149.6	20657.2	150474.0
	ISEA-adaptive	67582.3	13338.1	**145280.0**	**20653.8**	**148476.1**

The same trend can be seen in Table 2, where ISEA-adaptive outperformed ISEA-chesc on most of the competition instances with exception of the Max-SAT ones. On BP, PS, FS, TSP and VRP problems we can observe significant improvement of the median best value and corresponding score gain.

The results are very promising and indicate that a proper re-initialization scheme is crucial for optimal performance of the algorithm. It shows that the scheme with adapted re-initialization probability is better suited for this kind of iterated local search algorithm than the rather hard-wired re-initialization strategy that is dependent on properly set user-defined parameters T_1 and T_2.

The question is why ISEA-adaptive performed almost consistently worse than ISEA-chesc on SAT instances. One reason for this observation could be that the time-intervals T_{exp} and T_{min} (and subsequently the P_{accept}), derived based on the expected number of non-improving LLH sequences, were too short so that there was not enough time to converge to a local optimum before new re-initialization was carried out.

5 Conclusions

This paper presented two versions of an evolutionary-based iterative local search hyper-heuristic called Iterated Search Driven by Evolutionary Algorithm Hyper-Heuristic (ISEA). The two versions differ in the re-initialization scheme. The first one, ISEA-chesc, relies on strictly defined time-based conditions that if fulfilled than the re-initialization of the current evaluation solution takes place. The second one, ISEA-adaptive, uses a probability P_{reinit} to determine whether the evaluation solution will be re-initialized or not.

The performance of the two algorithms was experimentally evaluated on six hard optimization problems using the HyFlex experimental framework [4] and the two algorithms were compared using the results of the algorithms that took part in the CHeSC 2011 challenge [10].

The achieved results are very promising, the ISEA-adaptive variant would place second in the CHeSC 2011 final competition. It also shows that the re-initialization scheme used in ISEA-adaptive is better suited for this optimization algorithm as the ISEA-chesc placed on eighth place and attained almost consistently worse results than ISEA-adaptive.

There are several directions for future research:

- In order to avoid cycles in the search process we plan to incorporate a Tabu list containing several recent *evaluation solutions*. Any newly generated solution that appears in the Tabu list must not be accepted as the new *evaluation solution*.
- Another subject of research is how to implement the perturb operator so that it enables sampling regions of attraction of local optima in the vicinity of the current local optimum. We would like to investigate a potential of adaptive perturb operators.
- Third main direction of our research is towards utilization of information gathered during the optimization process such as which low level heuristics

or their combinations appeared frequently in improving LLH sequences and in which situations. Such information can be used for example in biased sampling of low level heuristics to the LLH sequences.

Acknowledgement

The work presented in this paper has been supported by the research program No. MSM 6840770038 "Decision Making and Control for Manufacturing III" of the CTU in Prague. I would like to thank Dr. Gabriela Ochoa for kindly providing me with complete outcome data of the ISEA hyper-heuristic calculated for the CHeSC 2011 competition.

References

1. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., zcan, E., Woodward, J.R.: Exploring hyper-heuristic methodologies with GP. *Artificial Evolution*, 1:177-201, 2009.
2. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., zcan, E., and Woodward, J.R.: A Classification of Hyper-heuristic Approaches. *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, Vol. 146, pp. 449-468, 2010.
3. Burke, E.K., Curtois, T., Hyde, M.R., Kendall, G., Ochoa, G., Petrovic, S., Vzquez Rodrguez, J.A., Gendreau, M.: Iterated Local Search vs. Hyper-heuristics: Towards General-Purpose Search Algorithms. IEEE Congress on Evolutionary Computation CEC'10, pp. 1-8, 2010.
4. Burke, E., Curtois, T., Hyde, M., Ochoa, G., Vazquez-Rodriguez, J. A.: HyFlex: A Benchmark Framework for Cross-domain Heuristic Search, ArXiv e-prints, arXiv:1107.5462v1, July 2011.
5. Garrido, P. and Riff, M.C.: DVRP: A Hard Dynamic Combinatorial Optimisation Problem Tackled by an Evolutionary Hyper-Heuristic. *Journal of Heuristics*, Volume 16, Issue 6, pp. 795-834, 2010.
6. Kubalik J. and Faigl J.: Iterative Prototype Optimisation with Evolved Improvement Steps. In: P. Collet, M. Tomassini, M. Ebner, A. Ekart and S. Gustafson (Eds.): Genetic Programming. Proceedings of the 9th European Conference, EuroGP 2006. Heidelberg: Springer, 2006, pp. 154–165
7. Kubalik, J.: Solving the Sorting Network Problem Using Iterative Optimization with Evolved Hypermutations. In Genetic and Evolutionary Computation Conference 2009 [CD-ROM]. New York: ACM, pp. 301–308, 2009.
8. Kubalik, J.: Efficient stochastic local search algorithm for solving the shortest common supersequence problem. In Proceedings of the 12th Genetic and Evolutionary Computation Conference, New York: ACM, 2010, ISBN 978-1-4503-0073-5, pp. 249–256, 2010.
9. Luke, S.: *Essentials of Metaheuristics*. Lulu, 2009, available at http://cs.gmu.edu/ sean/book/metaheuristics/
10. The results of the first Cross-domain Heuristic Search Challenge, CHeSC 2011, http://www.asap.cs.nott.ac.uk/chesc2011/index.html