Iterated Search Driven by Evolutionary Algorithm Hyper-Heuristic

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

1 Introduction

Iterated Search Driven by Evolutionary Algorithm Hyper-Heuristic (ISEA) is based on an evolutionary-based iterative local search algorithm called POEMS [1]. 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.

Effectiveness of POEMS has been demonstrated on several hard discrete optimization problems [1], [2], [3]. It was also successfully applied to multiobjective optimization problems [5].

2 ISEA

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. ISEA maintains a set of four solutions solutionMemory[] to the given problem via methods provided by the AbstractClasses.ProblemDomain. The meaning of the solutions is as follows:
solutionMemory[0] – current working solution, it is used for evaluating candidate sequences of low level heuristics (LLHs) defined for the problem at hand.

solutionMemory[1] – a solution obtained as a result of an application of some candidate sequence of LLHs to the solution stored in solutionMemory[0].


Prototype. Unlike other traditional applications of POEMS, the ISEA operates on a prototype that is a sequence of LLHs defined for the problem at hand. So, the prototype is not a solution to the given problem. Instead, the prototype is a sequence of LLHs that are to be modified by evolved action sequences and resulting sequence of LLHs is applied to the working solution (solutionMemory[0]) yielding a new candidate solution to the problem.

Only local search, mutation and ruin-recreate heuristics are considered for the generated sequences of LLHs. The length of the prototype can vary in certain range. When initialized, the prototype’s length is set to $p$ (it is a parameter of the algorithm) and the particular LLHs are assigned to conform with the following schema:

- At the first position of the prototype, either local search, mutation or ruin-recreate heuristic can be generated.
- Either local search or mutation heuristic can be assigned to any remaining position but the last one.
- Only local search heuristic can be assigned to the last position of the prototype.

The idea behind this scheme is that at when looking for better solution in the neighborhood of the current working solution, one might try any type of LLHs first (including the ruin-recreate heuristic), then make some mutations and/or local search and finalize the new solution by applying some of the local search heuristics only.

Actions. Note, that the prototype is permanently modified by evolved action sequences. So, the particular sequence of LLHs can vary along the run. In other words, it is the task for the EA to evolve such a modification of the current prototype that yields the best suitable sequence of LLHs at each moment of the run. Consequently, at some stages sequences that realize rather disturbing moves can be generated while at other stages sequences realizing fine-tuning moves are prevailing.

Following three simple actions are proposed for the ISEA:

- addLLH(pos, LLHType, params) – adds a new LLH of type LLHType with parameters set to params to the prototype at position pos.
- removeLLH(pos) – removes LLH that is at position pos in the prototype.
ISEA Hyper-Heuristic

input: set local search, mutation and ruin-recreate heuristics
output: best solution found

1. \( \text{lastRestart} \leftarrow \text{currentTime} \)
2. \( \text{lastImprovement} \leftarrow \text{currentTime} \)
3. \( \text{evaluations} \leftarrow 0 \)
4. \( \text{initialize} (\text{solutionMemory}[\text{]}) \)
5. \( \text{initialize} (\text{prototype}) \)
6. \( \text{initialize} (\text{populationAS}) \)  // initialize population of action sequences
7. \( \text{while} (\text{hasTimeExpired}()) \)
8. \( \text{calculateGenerationEA()} \)  // calculates one generation of the EA
9. \( \text{evaluations} += 2 \)  // increment number of evaluations calculated in the EA
10. \( \text{if} ((\text{currentTime} – \text{lastRestart}) > T_1) \) and \((\text{currentTime} – \text{lastImprovement}) > T_2)\)
11. \( \text{reinitializeWorkingSolution()} \)  // initializes solutionMemory[0]
12. \( \text{solutionMemory}[2] \leftarrow \text{solutionMemory}[0] \)  // update working solution
13. \( \text{lastRestart} \leftarrow \text{currentTime} \)
14. \( \text{if} ((\text{evaluations}) > N) \)  // number of evaluations calculated in one EA
15. \( \text{evaluations} \leftarrow 0 \)  // start new EA
16. \( \text{solutionMemory}[0] \leftarrow \text{solutionMemory}[2] \)  // update working solution
17. \( \text{initialize} (\text{prototype}) \)
18. \( \text{initialize} (\text{populationAS}) \)  // initialize population of action sequences
19. \( \text{end while} \)
20. \( \text{return} \text{solutionMemory}[3] \)  // return best-so-far solution

---

- \( \text{changeLLH(pos, params)} \) – modifies LLH at position \( \text{pos} \) in the prototype according to the new parameters. It can change the type of the LLH (mutation / local search / ruin-recreate) and/or its parameters.

2.1 ISEA Pseudo code

At the beginning of the run the following variables are initialized

- \( \text{prototype, populationAS, solutionMemory}[] \) – prototype, population of action sequences and the solution memory,
- \( \text{lastRestart, lastImprovement} \) – time stamps of the last restart of the working solution \( \text{solutionMemory}[0] \) and the last improvement of the \( \text{solutionMemory}[2] \),
- \( \text{evaluations} \) – number of fitness evaluations calculated in one EA.

Then the algorithm iterates steps until the time expires (steps 7-19 in Fig. 1). First, one generation of the EA is calculated. This means that a pair of new
action sequences is generated either by means of crossover and mutation. The newly generated action sequences are evaluated on the current working solution \( \text{solutionMemory}[0] \) and the result is stored in \( \text{solutionMemory}[1] \). Whenever the new solution \( \text{solutionMemory}[1] \) is at least as good as the solution \( \text{solutionMemory}[0] \) the solution \( \text{solutionMemory}[0] \) is replaced with the solution \( \text{solutionMemory}[1] \). Moreover the solution \( \text{solutionMemory}[1] \) replaces \( \text{solutionMemory}[2] \) and solution \( \text{solutionMemory}[3] \), respectively, if it is better than or equal to the solutions.

If it is already longer than \( T_1 \) seconds since the last restart of the working solution and it is already longer than \( T_2 \) seconds since the last improvement of the working solution then the working solution gets reinitialized (step 11 in Fig. 1) – the current working solution is replaced with its perturbed version.

If the number of fitness evaluations exceeds the specified maximum number of fitness evaluations then the prototype and the population of action sequences are re-initialized (steps 15-18 in Fig. 1).

### 2.2 ISEA Control Parameters’ Setting

Before the main procedure of the ISEA starts, it is checked whether a hard or simple problem instance is being solved. The time complexity of the low level heuristics for the particular problem is roughly estimated. Based on the estimate one of two possible configurations is chosen. The configuration for simple problems uses longer prototypes, longer action sequences, bigger population of action sequences and larger number of fitness evaluations calculated in one EA. The configuration for hard problems uses shorter prototype, shorter action sequences, smaller population of action sequences and smaller number of fitness evaluations calculated in one EA. Note, that just the time complexity of the problem are examined. There is no attempt to disclose the type of the problem.

### References