Pearl Hunter: A Hyper-heuristic that Compiles Iterated Local Search Algorithms

Fan Xue, C.Y. Chan, W.H. Ip, and C.F. Cheung

Department of Industrial and Systems Engineering, The Hong Kong Polytechnic University, Hunghom, Kowloon, Hong Kong
{mffxue,mfcychan,mfwhip,mfbenny}@inet.polyu.edu.hk

Extended Abstract

1 Introduction

Searching for best configurations to achieve some goal is a fundamental trait of intelligence and can be found in many ancient stories, such as the Carthaginian bullhide rope in 800s B.C. and *Sun*'s strategy of rearranging of best, average and inferior horses to race against king's in 340s B.C. in ancient China[1]. In the contemporary world, almost all branches of engineering have been highly dependent on optimization techniques. Over the history of optimization, enormous of optimization algorithms have been proposed to address problems on four key issues:

Effectiveness able to find highly satisfactory solutions,
Efficiency with quick running,
Ease easy to understand and deploy, and
Portability scalable to different problem domains,

where the first three are more fundamental and more straightforward in practice than the last one. However, cross-domain algorithms and frameworks are receiving increasing attention in recent decades for a great number of new problems are emerging from modern engineering domains.

Many well-known portable optimization algorithms, such as Genetic Algorithm, Ant Colony, and Iterated Local Search (ILS), are usually classified as metaheuristics. The suffix "meta" means "beyond, in an upper level." Another class of portable optimization algorithms are hyper-heuristics. A hyper-heuristic is a search method or learning mechanism for selecting or generating heuristics to solve computational search problems[2]. Hyper-heuristics can therefore be distinguished from metaheuristics by search space — a hyper-heuristic searches in a space of "low level" heuristics. However there is still no widely accepted definition on metaheuristics. So hyper-heuristics sometimes are treated as a subclass of metaheuristics, such as the definition in [3].

One trend in developing heuristic algorithms is to employ machine learning to obtain feedback and control solution-oriented and domain-dependent heuristics. Such approaches can be found even before proposals of the concepts of metaheuristics and hyper-heuristics[2]. This trend is closely related to the emerging and development of hyper-heuristics. In practice, algorithms enhanced with learning are generally more competitive or at least comparable to their predecessors, e.g., SATzilla[1] with off-line learning on selection of algorithms in SAT (Boolean satisfiability) competitions[4] and Lin-Kernighan-Helsgaun[2] with online learning on initial tour construction and local search in open Traveling Salesman Problem solving[5].

2 Pearl Hunter

Pearl hunting is a traditional way of diving to retrieve pearl from pearl oysters or to hunt some other sea creatures. In some areas, hunters need to repeatedly dive and search sea floor at several meters depth for pearl oysters.

In a search perspective, pearl hunting which consists of repeated diversification (surface and change target area) and intensification (dive and find pearl oysters) is in the paradigm of ILS[6]. A Pearl Hunter (PHunter) hyper-heuristic is inspired by the pearl hunting, as shown in Fig. 1. PHunter groups, tests, sorts, selects and combines low level heuristics to imitate a rational diver. In PHunter, low level algorithms which are classified as "local search" are considered as actions of "dives" (intensification) due to the explicitly monotonic objective value. Non-local search methods are treated as "surface moves" ("moves," diversification), though some moves, such as mutation and certain ruin-recreate heuristic consisting of an inner local search, have a limited capability of intensification.

There are two forms of dives in PHunter: "snorkeling" (Line 17 in Fig. 1) and "deep dive" (scuba, Line 20 in Fig. 1). Given a set of initial positions (solutions), snorkeling involves a number of given local search algorithms with a very low "depth of search" and stops if an improvement is found. Deep dive stops if no improvements can be further found, where the local search algorithms are given a high depth of search. A hashed cache is employed to record courses of deep dives and also used as an unwanted (inferior to tabu) list for surface moves at the same time. A typical "move-dive" iteration in PHunter consists of a large set of initial positions generated by some moves, a number of snorkeling dives on selected positions and a few deep dives. Deep dives only handles promising positions which are usually the best (occasionally the worst) ranked results of snorkeling.

Given a set of local search algorithms $\{A, B, C, D\}$ and an initial solution, the result of applying in order "$ABCD$" is usually different from that in "$DCBA$" in practice. Possible reasons include complex shape of solution space and occasionally inconsistency of local search algorithms. In PHunter, such a set is tested and ordered before the main search phase. Assume the ordered set is $\{A, B, C, D\}$. Deep dives exploit parallel sequences (such as "$ABACBADCBA$" and "$DCBACBABAA$"), choose the best, and repeat until no improvements can be further found. The complex sequences introduce potential redundancy but

[1] See `http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/`

[2] See `http://www.akira.ruc.dk/~keld/research/LKH/`

```
procedure PHunter()
  random_initialization ();
  test_and_order_dives ();
  test_and_order_moves();
  test_rehearsal_runs ();
  $mo$ ←select_move_mode_according_to_rehearsal_runs();
  $env$ ←discover_environment_of_diving ();
  loop while (global_terminate_condition_not_met())
    $M$ ←select_moves_by_mode($mo$);
    for each move $m$ in $M$
      $P$ ←Ø;
      loop for $Num\_of\_Snorkeling$ times
        $p$ ←apply_move_to_pool($m$);
        loop while (trapped_around_buoy($p$))
          $p$ ←apply_more_moves($p$);
        end loop
        $p$ ←snorkeling($env$, $p$);
        $P \leftarrow P \cap p$;
      end loop
      $p^*$ ← select_promising_positions ($P$);
      deep_dive($env$, $p^*$);
    end for
    if (mission_terminate_condition_is_met())
      clear_pool ();
      random_initialization ();
    end if
  end loop;
  return $BestEverFound$;
end procedure
```

Fig. 1. Pseudo code of Pearl Hunter

return better results generally. Some less complex sequences are also introduced to snorkeling dives.

PHunter determines a portfolio ("mode") of categorized moves and a way of diving for a given problem. Some trial search procedures are tested for information gathering and the environment of diving is discovered before a final mode is determined. Therefore PHunter can generates different ILSs for different problems. In rehearsal test runs (Line 5 in Fig. 1), PHunter employs a number of counters to record how many sub-optimal solutions are found by different groups of moves and different actions of dives. The final mode is determined according to the objective values of obtained solutions and the counters. The environment of diving is discovered by the reaction of dives (Line 7 in Fig. 1). For example, if every dive can be finished by at most one local search heuristic, the environment is flagged as "Shallow Water." In Shallow Water, PHunter simplifies the test sequences in snorkeling and deep dives, such as parallel "A," "B," "C," and "D." Another case is "Sea Trench," where at least one local search consumes too much time (e.g., 3% of overall time) in a single execution. In this case, the depth of search is tuned to a relatively low value and the test sequences in snorkeling and deep dives are also simplified.

There can be a number of consecutive moves before a dive (Line 14-16 in Fig. 1). PHunter tries more moves if the newly moved position is still trapped around a "buoy in water," which is an objective value obtained in advance. A tabu list is designed to prevent recent moves around the buoy. In other words, more diversifications will be applied if the solution is trapped in a local optimum.

A restart mechanism called "mission restart" (Line 20-23 in Fig. 1) can reset the search procedure if some condition is met. The condition can be an over-converged pool or no better solutions for a certain amount of time.

3 Implementation and Experiments

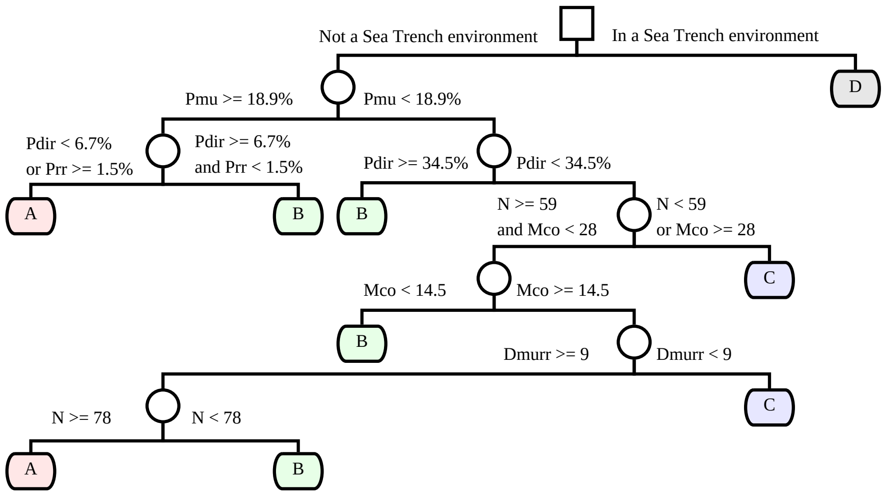
PHunter was implemented on a java cross-domain platform named HyFlex[3] (Hyper-heuristics Flexible framework). For a given problem in a given domain, HyFlex provides a random initialization, a set of low level heuristics in 4 groups (Crossover, Mutation, Ruin-recreate and Local search), two parameters of the "intensity" of mutation and "depth of local search," and a population of easily accessible (but functionally limited) solutions.

Four move modes were defined: average calls (A), Crossover emphasized (B), Crossover only (C), and average calls with an online pruning (D). The first three modes sorts selected low-level heuristics by groups, and calls them by their order in their group in the first half of running time then by their merits (counters of sub-optimal solutions) in the later half. The last mode replaces the ordering procedure in the first mode with an online pruning.

An off-line classification procedure was carried out to identify the best mode. Decision attributes were gathered from a 1-minute test on iterated Crossover

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

moves followed by dives and a 1-minute test on iterated Mutation and Ruin-recreate moves followed by dives. In the Crossover test, missions were restarted if no improvement was found in the iterations of a whole group of moves. The test on Mutation and Ruin-recreate moves started with the pool found by the Crossover test and had no mission restart. A decision tree was discovered by the Best-first tree classifier provided by WEKA[4] with default parameters, as shown in Fig. 2.



D_{murr}: Depth of the mission in the Mutation and Ruin-recreate test,
M_{co}: Number of missions completed in the Crossover test,
N: Number of sub-optimal solutions found in total,
P_{dir}: Percent of sub-optimal solutions found right after some moves (before any dive),
P_{mu}: Percent of sub-optimal solutions found in iterations started with Mutation moves,
P_{rr}: Percent of sub-optimal solutions found in iterations started with Ruin-recreate moves,

Fig. 2. Decision tree on modes of moves obtained by off-line learning

Tests have been conducted on 4 problem domains: MAX-SAT, 1D Bin-packing, Personnel Scheduling and Flow Shop, each with 10 difficult instances. The results are shown in Table 1. The scoring system was the Formula 1 point system provided by HyFlex, greater number meant better. Each solution of PHunter was an average of 3 independent trials. HH1 to HH8 were 8 default hyper-heuristics and their results were provided in HyFlex. The computational time was benchmarked to be equal to 10 CPU minutes on a standard machine.

[4] Version 3.5.6, see `http://www.cs.waikato.ac.nz/ml/weka/`

Table 1. Scores of hyper-heuristics on the HyFlex framework

Domain	HH1	HH2	HH3	HH4	HH5	HH6	HH7	HH8	PHunter
MAX-SAT	53.75	70.75	36.5	24.5	1.0	46.0	51.75	14.5	**91.25**
1D Bin-packing	48.0	51.0	67.0	53.0	10.0	41.0	30.0	1.0	**89.0**
Personnel Scheduling	**67.0**	61.5	23.0	48.5	53.0	0.0	53.0	29.0	55.0
Flow Shop	28.0	19.5	22.0	69.0	16.5	63.5	18.5	56.0	**97.0**
Overall	196.75	202.75	148.5	195.0	80.5	150.5	153.25	100.5	**332.25**

4 Discussion

As shown in Table 1 scores of PHunter were very competitive. In fact some results of the tests approximated or had broken the best-known solutions. One exception is the Personnel Scheduling domain. A possible reason was the standard deviation of objective values were significantly greater than those in other domains, e.g., 343.8 ± 35.7 (about 10%) in a Personnel Scheduling problem versus 213 ± 1.2 ($< 1\%$) in a MAX-SAT problem.

Acknowledgements

The work described in this paper was partly supported by a grant from the Department of Industrial and Systems Engineering of The Hong Kong Polytechnic University (No. RP1Z) and an internal grant from the Hong Kong Polytechnic University (POLYU5110/10E).

References

1. Sima, Q.: Biographies of Sun Tzu and Wu Qi, in Records of the Grand Historian, Vol. 65. (91B.C.) (In Chinese)
2. Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., and Qu, R. Hyper-heuristics: A Survey of the State of the Art, School of Computer Science and Information Technology, University of Nottingham, Computer Science. Technical Report No. NOTTCS-TR-SUB-0906241418-2747. (2010)
3. Stützle, T. Local Search Algorithms for Combinatorial Problems—Analysis, Improvements, and New Applications. PhD Thesis. Department of Computer Science, University of Essex, Colchester, UK. (1998)
4. Xu, L., Hutter, F., Hoos, H., and Leyton-Brown, K. SATzilla: Portfolio-based Algorithm Selection for SAT. Journal of Artificial Intelligence Research. (2008) Vol. 32 565–606
5. Helsgaun, K. An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. European Journal of Operational Research. (2000) 126:1, 106–130
6. Lourenço, H., Martin, O., Stützle, T. Iterated Local Search. In Glover, F., Kochenberger, G. (eds.) Handbook of Metaheuristics. Springer New York. (2003) 320–353.