Co-evolving add and delete heuristics

Jerry Swan · Ender Özcan · Graham Kendall

Received: date / Accepted: date

Abstract Hyper-heuristics are (meta-)heuristics that operate at a high level to choose or generate a set of low-level (meta-)heuristics to solve difficult search and optimisation problems. Evolutionary algorithms are well-known nature-inspired meta-heuristics that simulate Darwinian evolution. In this article, we introduce an evolutionary-based hyper-heuristic in which a set of low-level heuristics compete to solve timetabling problems.

Keywords Hyper-heuristics · Coevolution · Ruin-and-recreate

1 Introduction

Hyper-heuristics are (meta-)heuristics that operate at a high level to choose or generate a set of low level (meta-)heuristics to solve difficult search and optimisation problems [1],[6]. Heuristics can be used to search the solution space directly or construct a solution based on a sequence of moves. In most of the previous studies, the type of the low-level heuristics used is uniform, i.e. they are either constructive or perturbative (improvement) heuristics. Hyper-heuristics aim to replace bespoke approaches by general methodologies for solving different problems. They provide a *"good enough - soon enough - cheap enough"* framework for problem solving.

Timetabling problems are NP hard [2], real-world constraint optimisation problems. A timetabling problem requires scheduling of given events using limited resources, subject to a set of constraints. Evolutionary algorithms are well-known nature-inspired meta-heuristics that simulate Darwinian evolution. In this paper, we introduce an evolutionary based hyper-heuristic in which a set of low level heuristics compete to solve timetabling problems.

Automated Scheduling, Optimisation and Planning (ASAP) Research Group,
School of Computer Science, University of Nottingham,
Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, UK.
{jps,exo,gxk}@cs.nott.ac.uk

```
(* Generate feasible binary strings. '+' denotes concatenation.
   Initialize with: generateAddDeleteLists( n, "01" ); *)
function generateAddDeleteLists( var n : Int, var bits :
    BinaryString ) : ListOfBinaryString
begin
  var result : ListOfBinaryString;
  if( n = 1 )
  begin
    result := result + bits;
  end;
  else
  begin
    result := result + generateAddDeleteLists(n-1,bits+ "01");
    result := result + generateAddDeleteLists(n-1,"0"+bits+"1");
    if( bits doesn't have prefix "01" )
    begin
      result := result + generateAddDeleteLists( n-1,"01"+bits);
    end;
  end;
  return result;
end.
```

Listing 1 Generate feasible binary strings

1.1 Low-level heuristics

A solution to a timetabling problem can be reconstructed from a previous solution by successively deleting and adding (re-scheduling) events. In our hyper-heuristic framework, we propose a sequence of delete-add (0-1) operations with a fixed length. Add and delete operations can be handled in many different ways. The simplest approach for the delete operation is choosing an event randomly and putting it into an unscheduled events list. On the other hand, the add operation requires two consecutive actions to be taken. Firstly, an event should be selected from the list of unscheduled events and then a suitable period should be selected for scheduling.

A fixed-length binary string can be used to represent a series of add (1) and delete (0) operations for reconstructing a new solution from an existing one. A feasible string of length $2n$ can be formed by respecting the following rules:

1. The number of ones must be equal to the number of zeros, that is n. Hence, the string length is an even number.
2. The first entry in the string is always the delete operation, since there is no unscheduled event at the start.
3. For any prefix of the string, the number of ones must be less than or equal to the number of zeroes.

For example, given a solution S and a feasible string "0011", two randomly selected events are deleted from S, and S' is formed by rescheduling them, successively. On the other had, "0110" is not a feasible string, since after

```
(*  Co-evolve binary strings to be used for reconstructing
  a new solution to a timetabling problem from an old one. *)
procedure coevolve( var pop : Population )
begin
  (* let individuals compete for generating a better individual *)
  for( each individual in pop )
  begin
    var previousTimetable = getTimetable( individual );
    var newTimetable := applyADL( individual.getTimetable(),
        individual.getADL() );
    if( newTimeTable is better than previousTimetable )
    begin
        individual.setTimetable( newTimetable );
    end;
  end;
  sortByFitness( pop );
  bestIndividual = pop(0); (* Keep the best individual *)

  (* divide the population into 4 parts *)
  (* copy individuals in 1st quarter into 2nd quarter *)
  copy(pop, 0, popSize/4, popSize/4, 2*popSize/4);
  (* copy individuals in 1st quarter into 3rd quarter*)
  copy(pop, 0, popSize/4, 2*popSize/4, 3*popSize/4);

  (* mutate timetable state and randomize associated add-delete
      list in 1st quarter *)
  mutateTimetableAndRandomizeADL(pop, 2, popSize/4);
  (* mutate timetable state in 2nd quarter *)
  mutateTimetable(popSize/4, 2*popSize/4);
  (* randomize add-delete-list in the 3rd quarter *)
  mutateADL(2*popSize/4, 3*popSize/4, addDeleteLists);
  (* randomize state and add-delete list in 4th quarter *)
  randomizeTimetableAndADL( pop, 3*popSize/4, popSize );
end
```

Listing 2 Co-evolve timetables and add-delete lists

deleting and rescheduling an event, the unscheduled event list becomes empty and the add operation at the third location is not possible. Listing 1 shows a divide-and-conquer approach that generates such strings for a given n. For $n = 1$, the only such string is "01", for $n = 2$, we have "0101", "0011" and for $n = 3$, "010101", "001011", "001101", "000111", "010011". The running time of the algorithm is $\mathcal{O}(3^n)$. Hence, we will use practical values for n and after generating the feasible strings, they compete for survival while constructing timetables within an evolutionary framework, as described in listings 2 and 3.

2 Results

The results presented here are for the 8 publicly-available datasets of the exam-timetabling track from the ITC2007 competition [4], with Müller's celebrated

```
(*  Co-evolution framework. *)
procedure framework( var posSize : Int )
begin
  var allAddDeleteLists, coevolvedAddDeleteLists :
      ListOfBinaryString;
  var population : ListOfTimetable;

  (* Generate feasible add-delete strings *)
  allAddDeleteLists := generateString(n, "01");

  population := generateRandomPopulation( popSize,
      allAddDeleteLists );

  while( termination condition (e.g. num-iterations ) not met )
  begin
    (* co-evolve a population of timetables and add-delete
        operators *)
    coevolve( pop );
  end;
end.
```

Listing 3 Top-level co-evolutionary framework

hybrid solver [5] ranking first on each dataset[1]. Gogos et al. [3] are second and have rankings [3, 4, 3, 2, 3, 3, 2, 3] over these 8 datasets[2] with average rank of 2.85. Table 1 gives the scores obtained by our program on a Pentium 4 dual-core 3GHz PC with 2GB of RAM. The maximum runtime of our program was set to 377 seconds (as determined by the competition benchmark program[3]. The respective rankings for our program were [2, 3, 2, 4, 5, 2, 3, 2], yielding an average rank of 2.875, which would place us in joint second with Gogos over these 8 datasets. Table 2 shows $(\bar{x}, \sigma)$ of the results of each dataset for M̈ller, Gogos and our program for the cases where all 10 sample runs are feasible (i.e. are of the form $0, x$ for some x).

3 Conclusion

We have presented a co-evolutionary variant on the 'ruin-and-repair' strategy that ranks joint second with the pre-existing finalists on the publicly-available datasets of the ITC2007 competition exam-timetabling track.

Future work involves investigating alternative entity relationships between solution and add-delete string (e.g. N:1 and 1:N as opposed to the 1:1 approach adopted here) and associating some probability (possibly adaptively determined) with this relationship.

In addition, it is interesting to note from Table that while our approach generally exhibits a higher standard deviation, it also yields a greater number

[1] http://www.cs.qub.ac.uk/itc2007/winner/tomasmuller.htm

[2] http://www.cs.qub.ac.uk/itc2007/winner/christosgogos.htm

[3] Available at http://www.cs.qub.ac.uk/itc2007/index_files/benchmarking.htm

dataset 1	dataset2	dataset3	dataset4	dataset5	dataset6	dataset7	dataset8
0,6133	0,942	0,13383	0,27333	0,5922	0,28800	0,5960	0,9590
0,6094	0,893	0,13364	0,30322	0,5333	0,30590	0,6179	0,9825
0,6027	0,933	0,13352	0,38651	0,5407	0,27830	0,6988	0,9738
0,6133	0,935	0,13367	0,29030	0,4734	0,31410	0,6914	0,9709
0,6487	0,927	0,14001	0,37850	0,5205	0,28055	0,6872	0,9772
0,6206	0,913	0,14298	0,29449	0,5831	0,27510	0,5950	0,9507
0,6131	0,972	0,13738	0,33802	0,4877	0,27890	0,5891	0,9814
0,5875	0,1009	0,14163	0,24174	0,5847	0,30560	0,5731	0,9587
0,6336	0,929	0,14334	0,27654	0,5023	0,29250	0,6842	0,9802
0,5992	0,948	0,14348	0,28195	0,5106	0,31600	0,7032	0,10164

Table 1 Results of co-evolutionary framework for public datasets of ITC2007 exam-timetabling track

Name	dataset 1	dataset2	dataset3	dataset4
Müller	(4574.9, 159.7731)	(414,11.49879)	-	-
Gogos	(6064,108.8087)	(1048.6,32.57879)	(14133.5,227.9553)	-
Swan	(6141.4, 173.3187)	(940.1,31.89375)	(13834.8,440.9013)	-
Name	dataset5	dataset6	dataset7	dataset8
Müller	(3320.7,209.9524)	(27808.5,1115.487)	(4399.143,123.4942)	(7922.429,126.9696)
Gogos	(4229.1,75.01178)	-	(6759.5,100.5134)	(10809,180.8265)
Swan	(5328.5,420.9968)	(29349.5,1567.905)	(6435.9,533.9795)	(9750.8,181.7635)

Table 2 $(\bar{x}, \sigma)$ of feasible solutions for public ITC2007 datasets

of feasible solutions, failing only on dataset 4 in this respect. This is perhaps a counter-intuitive result, since one might expect the 'repair' aspect of our 'ruin-and-repair' strategy to encounter many infeasible solutions. Hence, there is further work to be done in explaining the underlying reasons for this behaviour.

References

1. Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Handbook of Meta-Heuristics, chap. Hyper-Heuristics: An Emerging Direction in Modern Search Technology, pp. 457–474. Kluwer (2003). URL `http://www.asap.cs.nott.ac.uk/publications/pdf/hhchapv002.pdf`
2. Even, S., Itai, A., Shamir, A.: On the complexity of timetable and multicommodity flow problems. SIAM J. Comput. **5**(4), 691–703 (1976)
3. Gogos, C., Alefragis, P., Housos, E.: A multi-staged algorithmic process for the solution of the examination timetabling problem. In: The 7th International Conference on the Practice and Theory of Automated Timetabling (2008)
4. McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A.J., Gaspero, L.D., Qu, R., Burke, E.K.: Setting the research agenda in automated timetabling: The second international timetabling competition. INFORMS JOURNAL ON COMPUTING **22**(1), 120–130 (2010). DOI 10.1287/ijoc.1090.0320
5. Müller, T.: Itc2007 solver description: a hybrid approach. Annals OR **172**(1), 429–446 (2009)
6. Özcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive analysis of hyper-heuristics. Intell. Data Anal. **12**(1), 3–23 (2008)