A Hyperheuristic Based on Dynamic Iterated Local Search

Mark Johnston[1], Thomas Liddle[1], Joel Miller[1], and Mengjie Zhang[2]

[1] School of Mathematics, Statistics and Operations Research
[2] School of Engineering and Computer Science
Victoria University of Wellington, PO Box 600, Wellington, New Zealand
mark.johnston@vuw.ac.nz

Abstract. Two hyperheuristics are proposed as entries for the 2011 Cross-domain Heuristic Search Challenge (CHeSC), namely DynamicILS and BiasILS; both are simple adaptations of Iterated Local Search (ILS).

1 Iterated Local Search

Iterated Local Search (ILS) [3] is a metaheuristic search technique which aims to find good solutions to difficult combinatorial optimisation problems in a reasonable amount of computational time. It is an iterative procedure which performs a biased sampling of local optima (hilltops) in the search space by randomly changing the current solution at each iteration (performing a "kick") and then using a local search heuristic to find a new local optima. Thus ILS aims to avoid getting stuck in local optima (the main disadvantage of heuristics). ILS can operate with "black-box" components; it does not need to know the local search heuristic or how the kick is performed.

Algorithm 1 Iterated Local Search [3]

s_0 = GenerateInitialSolution
s^* = LocalSearch(s_0)
repeat
  s' = PerformKick($s^*, history$)
  s'^* = LocalSearch(s')
  s^* = AcceptanceCriterion($s^*, s'^*, history$)
until TerminationCriterion

ILS begins (see Algorithm 1) with a local optima from using a local search heuristic on an initial solution (generated by a construction heuristic). It then iteratively performs a kick on the current solution (by randomly changing some elements of the solution), obtains a new local optima (using local search) and then determines if this new solution should be used as the current solution for the next iteration by testing it against some *acceptance criterion*, e.g., only accepting

a better solution or probabilistically accepting a worse solution. The search is terminated when a *termination criterion* is reached, e.g., the computation time limit expires. Within the HyFlex package, a kick (PerformKick) corresponds to a predefined mutation or a ruin-and-recreate low-level heuristic.

The current solution at each iteration in ILS has a *neighbourhood*, which is the set of solutions that all possible locally optimised kicks can reach (the set of all neighbouring hilltops). Standard implementations of ILS perform a single kick blindly without searching for the "best" kick to perform (to reach the best neighbouring hilltop). In addition, the *strength* of the kick (how much change is made to the current solution) determines the solutions that are contained within the neighbourhood (hilltops closer or further away).

2 Dynamic Iterated Local Search

A key decision when using ILS is the determination of the kick's nature and strength. A new method of dynamically adjusting the kick strength *during* the search was proposed by Liddle [2] and applied to the Travelling Salesman Problem (TSP). Here we extend this idea within the HyFlex package for the CHeSC competition.

We propose a variation of ILS, called Dynamic Iterated Local Search (DynamicILS, see Algorithm 2). DynamicILS dynamically samples from a set of available kick strengths K, choosing more often those kick strengths in K that seem to be performing well so far. For ILS, $|K|$ experiments (one for each kick strength) would be needed to establish the most effective $k \in K$, but DynamicILS has the potential to achieve the same or similar results using only a single experimental run.

Algorithm 2 Dynamic Iterated Local Search

s_0 = GenerateInitialSolution
s^* = LocalSearch(s_0)
$\boldsymbol{w}$ = InitialiseWeights(s^*)
repeat
  s' = PerformDynamicKick($s^*, \boldsymbol{w}, history$)
  s'^* = LocalSearch(s')
  $\boldsymbol{w}$ = UpdateWeights($s^*, s', \boldsymbol{w}$)
  s^* = AcceptanceCriterion($s^*, s'^*, history$)
until TerminationCriterion

Weights for Kick Strengths: We use *weights* w_k for each $k \in K$ in a probabilistic selection method. These w_k are updated during the search. This concept is similar to the use of *pheromones* in Ant Colony Optimization [1]. K is specified by lower and upper kick strength limits k_l and k_u, thus $K = \{k_l, k_l + 1, \ldots, k_u - 1, k_u\}$, with $m = k_u - k_l + 1$ possible kick strengths to sample from. Within the HyFlex package, these are subsequently scaled to $(0, 1)$.

Memory: A simple implementation of dynamic sampling is to use a single weights vector $\boldsymbol{w}$, where the next kick strength is chosen according to $\boldsymbol{w}$ only, that is, no memory is retained. Alternatively, one level of memory could be incorporated into the kick strength choice, in order to evolve a balance between intensification and diversification. In this case a weights *matrix* W is kept, where each row j of W ($j = 1, \ldots, m$) corresponds to the weights vector $\boldsymbol{w}_j$: the weights for choosing the next kick strength given that the previous kick strength was j. This is not included in the HyFlex implementation as it was found not to be as effective as the simpler version over the given test domains and the TSP.

Probabilistic Selection Method: The proposed method for probabilistically selecting a kick strength j from the set of possible kick strengths K, when performing a dynamically sampled kick in D-ILS is as follows. First determine the row vector of weights $\boldsymbol{w}$. If not using memory, this is just $\boldsymbol{w}$, the vector of weights. If using memory assign $\boldsymbol{w} = \boldsymbol{w}_i$, row i of the weights matrix W, which corresponds to the previous kick strength $i \in \{1, \ldots, m\}$. Then calculate the probability vector $\boldsymbol{p}$ from $\boldsymbol{w}$ as follows. Normalise the weights to be non-negative:

$$v_j = \begin{cases} w_j - w_{min} & \text{if } \; w_{min} < 0 \\ w_j & \text{otherwise} \end{cases}, \quad \text{for } j = 1, \ldots, m \tag{1}$$

where $w_{min} = \min_j w_j$. Calculate a normalisation increment a as:

$$a = \frac{w_{max} - w_{min}}{k_u - k_l} \tag{2}$$

where $w_{max} = \max_j w_j$. Add the normalisation increment to $\boldsymbol{v}$:

$$v_j \leftarrow v_j + a, \quad \text{for } j = 1, \ldots, m. \tag{3}$$

Finally, calculate p_j for each $j \in K$ by normalising the v_j between 0 and 1:

$$p_j = \frac{v_j}{\sum_{i=1}^{m} v_i}, \quad \text{for } j = 1, \ldots, m \tag{4}$$

The probability vector $\boldsymbol{p}$ can then be used to sample a kick strength to use from K. After a kick strength j is selected, the kick is performed, a local optima is found using local search, and the weight corresponding to the chosen kick strength is then updated via

$$w_j \leftarrow w_j + \text{fitness}_{\text{last}} - \text{fitness}_{\text{new}} \tag{5}$$

where $\text{fitness}_{\text{last}}$ and $\text{fitness}_{\text{new}}$ are the objective function values of the current (before the kick) and new solutions' local optima respectively. This update formula increases w_j (and the chance of j being selected next time) if the new solution is better than the current one and conversely reduces w_j if it is worse. If selection with memory is used, then replace w_j with w_{ij} in equation (5) where i is the kick strength chosen the previous iteration.

Extension to Multiple Low-Level Heuristics: In HyFlex, several mutation and ruin-and-recreate low-level heuristics are available to apply to a given (current) solution. These can easily be incorporated into DynamicILS by considering the set K to include both the kick strength and the kick type (particular low-level heuristic). Thus the idea (and hope) is that DynamicILS learns (or evolves) which low-level heuristics to apply and which strengths to apply them at throughout a run.

3 Non-Improvement Bias

Using a small kick strength has the potential for getting stuck in a local optima [3]. We introduce a *non-improvement bias* (NIB), which biases the strength of the kick increasingly towards larger values as the number of non-improving iterations increases. The NIB (set before the search) is used as a power to exponentially increase the bias for higher values of NIB.

Calculate the non-improvement biases b_j:

$$b_j = \begin{cases} 0 & \text{if NIB} = 0 \\ (\beta j)^{\text{NIB}} & \text{otherwise} \end{cases}, \quad \text{for } j = 1, \ldots, m \tag{6}$$

where β is the number of non-improving ILS iterations that have elapsed. Then equation (5) is modified to add the non-improvement biases to $\boldsymbol{v}$:

$$v_j = w_j + a + b_j, \quad \text{for } j = 1, \ldots, m. \tag{7}$$

The result is the hyperheuristic called BiasILS.

4 Summary

DynamicILS (without the non-improvement bias) and BiasILS (with the non-improvement bias) have been implemented within HyFlex for the CHeSC competition. In summary, the idea behind DynamicILS is to have weight vector which determines the probability that each particular mutation and ruin-recreate heuristic will be used and the associated strength at which it will be used. This weight vector is updated based on the historical performance of each mutation and ruin-recreate low-level heuristic. BiasILS includes a non-improvement bias so that if an improvement hasn't been made in a long time, the weights will favour using a greater kick strength for mutations and ruin-recreates.

References

1. M. Dorigo and T. Stutzle. *Ant Colony Optimization*. MIT Press, 2004.
2. T. Liddle. Kick strength and online sampling for iterated local search. In *45th Annual Conference of the Operations Research Society of New Zealand (ORSNZ)*, pages 130–139, 2010.
3. H. R. Lourenco, O. Martin, and T. Stutzle. Iterated local search. In F. Glover and G. Kichenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer, 2003.