Vehicle Routing and Adaptive Iterated Local Search within the *HyFlex* Hyper-heuristic Framework

James D. Walker[1], Gabriela Ochoa[1],
Michel Gendreau[2], and Edmund K. Burke[3]

[1] School of Computer Science, University of Nottingham, UK
[2] CIRRELT, University of Montreal, Canada
[3] Department of Computing Science and Mathematics, University of Stirling, UK

Abstract. HyFlex (Hyper-heuristic Flexible framework) [15] is a software framework enabling the development of domain independent search heuristics (hyper-heuristics), and testing across multiple problem domains. This framework was used as a base for the first *Cross-domain Heuristic Search Challenge*, a research competition that attracted significant international attention. In this paper, we present one of the problems that was used as a hidden domain in the competition, namely, the capacitated vehicle routing problem with time windows. The domain implements a data structure and objective function for the vehicle routing problem, as well as many state-of- the-art low-level heuristics (search operators) of several types. The domain is tested using two adaptive variants of a multiple-neighborhood iterated local search algorithm that operate in a domain independent fashion, and therefore can be considered as hyper-heuristics. Our results confirm that adding adaptation mechanisms improve the performance of hyper-heuristics. It is our hope that this new and challenging problem domain can be used to promote research within hyper-heuristics, adaptive operator selection, adaptive multi-meme algorithms and autonomous control for search algorithms.

1 Introduction

There is an increasing and renewed research interest in developing heuristic search methods that are more generally applicable [7,8]. The goal is to reduce the role of the human expert in the design of effective heuristic methods to solve hard computational search problems. Researchers in this field, however, are often constrained on the number of problem domains in which to test their adaptive, self-configuring algorithms. This can be explained by the inherent time and effort required to implement a new problem domain, including efficient data structures and search operators; initialisation routines; the objective function; and an varied a set of benchmark instances.

The HyFlex framework has been recently proposed to assist researchers in hyper-heuristics and autonomous search control. HyFlex features a common

Y. Hamadi and M. Schoenauer (Eds.): LION 6, LNCS 7219, pp. 265–276, 2012.

software interface for dealing with different combinatorial optimisation problems, and provides the algorithm components that are problem specific.In this way, it simultaneously liberates algorithm designers from needing to know the details of the problem domains; and it prevents them from incorporating additional problem specific information in their algorithms. Efforts can instead be focused on designing high-level strategies to intelligently combine the provided problem-specific algorithmic components. The framework also served as the basis of an an international research competition: the first *Cross-domain Heuristic Search Challenge (CHeSC 2011)*[1] [5], which successfully attracted the interest and participation of over 40 researchers at universities and academic institutions across six continents. This competition differed from other challenges in heuristic search and optimisation in that the goal was to design a search algorithm that works well, not only across different instances of the same problem, but also across different problem domains. It can be considered as the first *Decathlon* challenge of search heuristics. For testing purposes, four domain modules were provided to the participants, each containing around 10 low-level heuristics of the types discussed below, and 10 instances of medium to hard difficulty. The domains provided were: permutation flowshop, one dimensional bin packing, Boolean satisfiability (MAX-SAT) and personnel scheduling.

For calculating the final competition scores, two additional (hidden) domains were implemented and used: the traveling salesman problem, and the capacitated vehicle routing problem with time windows. This paper describes the design of the vehicle routing domain. It also tests this domain using two adaptive variants of a multiple neighborhood iterated local search algorithm. The first variant was the best performing algorithm in [6], while the second is a modification that improves the local search stage of the algorithm by incorporating an adaptive mechanism.

The next section briefly overviews the HyFlex framework, whilst section 3 describes the design of the vehicle routing domain. Section 4 describes the adaptive iterated local search hyper-heuristics that have been developed. Section 5 summarises the experiments and results and section 6 concludes and discusses our contributions.

2 The HyFlex Framework

HyFlex (Hyper-heuristics Flexible framework) [15] is a Java object oriented framework for the implementation and comparison of different iterative general-purpose (domain independent) heuristic search algorithms (also called hyper-heuristics). The framework appeals to modularity and is inspired by the notion of a domain barrier between the low-level heuristics and the hyper-heuristic [9,7]. HyFlex provides a software interface between the hyper-heuristic and the problem domain layers, thus enabling a clearly defined separation and communication protocol between the domain specific and the domain independent algorithm components.

[1] `http://www.asap.cs.nott.ac.uk/external/chesc2011/`

HyFlex extends the conceptual framework discussed in [9,7] in that a population of solutions (instead of a single incumbent solution) is maintained in the problem layer. Also, a richer variety of low-level heuristics is provided. Another relevant antecedent to HyFlex is PISA [2], a text-based software interface for multi-objective evolutionary algorithms, which divides the implementation of an evolutionary algorithm into an application-specific part and an algorithm-specific part. HyFlex differs from PISA in that its interface is not text-based but is instead given by an abstract Java class. Moreover, HyFlex provides a rich variety of combinatorial optimisation problems including real-world instance data. Each HyFlex problem domain module consists of:

1. A user-configurable memory (a population) of solutions, which can be managed by the hyper-heuristic.
2. A routine to initialise randomised solutions in the population.
3. A set of heuristics to modify solutions classified into four groups:
   mutational : makes a (randomised) modification to the current solution.
   ruin-recreate : destroys part of the solution and rebuilds it using a constructive procedure.
   local search : searches in the *neighbourhood* of the current solution for an improved solution.
   crossover : takes two solutions, combines them and returns a new solution.
4. A varied set of instances that can be easily loaded.
5. A fitness function, which can be called to obtain the objective value of any member of the population. HyFlex problem domains are always implemented as minimisation problems, so a lower fitness is always superior. The fitness of the best solution found so far in the run is stored and can be easily obtained.
6. Two parameters: α and $\beta, (0 <= [\alpha, \beta] <= 1)$, which are the 'intensity' of mutation and 'depth of search', respectively, that control the behaviour of some search operators.

Currently, six problem domain modules are implemented (which can be downloaded from the CHeSC 2011 website [14]). These are the original four test domains: permutation flow shop, one-dimensional bin packing, maximum satisfiability (MAX-SAT) and personnel scheduling; and the two additional (hidden) domains used for the competition: the traveling salesman problem and the vehicle routing problem with time windows.

3 The Vehicle Routing HyFlex Domain

The vehicle routing problem was implemented using the HyFlex software framework interface[2]. Specifically, a java class derived from the HyFLex *ProblemDomain* class was implemented, following the descriptions below.

[2] The API documentation can be found at:
`http://www.asap.cs.nott.ac.uk/external/chesc2011/javadoc/help-doc.html`

3.1 Problem Formulation

The vehicle routing problem can be described as the task of meeting the demand of all customers, using as few vehicles as possible, and satisfying all constraints, such as vehicle capacity. Furthermore, the variant of the vehicle routing problem which is modelled here is the vehicle routing problem with time windows. This variant includes extra time window constraints, whereby a customer must be served between two time points for a solution to be valid.

There is a base location, or depot, from where each vehicle must start and end its route. A route is a series of location visits for a single vehicle. The objective function for this domain balances the dual objectives of minimising the number of vehicles needed and minimising the total distance travelled. It was defined as follows:

$$objectivefunction = c \times numVehicles + distance,$$

where c is a constant that we empirically set to 1000 to give higher importance to the number of vehicles in a solution.

The problem domain offers a set of operators to initialise and modify solutions which are commonly found in effective meta-heuristics and a set of benchmarks instances (due to [21]) that are readily available.

3.2 Solution Initialisation

The initialisation method is stochastic, generating solutions based upon the given seed. Customers are inserted into the solution one at a time, with the customer to be inserted being chosen by a metric measuring the proximity of a customer in terms of distance and time to the most recently inserted customer. The metric also includes a stochastic element to ensure different solutions are generated. If it is not possible to insert any customer into the current route, a new route is generated. This process is repeated until all customers have been scheduled.

3.3 Low Level Heuristics

The module includes 12 low level heuristics $h_1, \ldots, h_{12}$ across the four categories of heuristics, as specified within HyFlex. They are described below, sorted by category.

Mutational Heuristics

h_1: **Two-opt**[3]. Swaps two adjacent customers within a single route.
h_2: **Or-opt**[16]. Moves two adjacent customers to a different place, within a single route.
h_3: **Shift**[18]. Moves a single customer from one route to another.
h_4: **Interchange**[18]. Swaps two customers from different routes.

Ruin and Recreate Heuristics

h_5: **Time-based radial ruin**[19]. Chooses a number of customers to be removed from the solution, based upon the proximity of their time window to a given time. Each remaining customer is inserted into the best route possible, based on a metric of distance and time proximity. If it is not feasible to insert into any route, a new route is created.

h_6: **Location-based radial ruin**[19]. Chooses a number of customers to be removed from the solution, based upon the proximity of their location to a given location. Each remaining customer is inserted into the best route possible, based on a metric of distance and time proximity. If it is not feasible to insert into any route, a new route is created.

Local Search Heuristics. These heuristics implement 'first-improvement' local search operators. In each iteration, a neighbour is generated, and it is accepted immediately if it has superior or equal fitness. If the neighbor is worse, then the change is not accepted. The *depth-of-search parameter* (see section 2) controls the number of iterations to attempt to obtain an improved solution.

h_7: **Shift**[18]. Moves a customer from one Route, to another providing that the new position yields an improvement in objective function score.

h_8: **Interchange**[18]. Swaps two customers from different routes, providing that the new routes yield an improvement in objective function score.

h_9: **Two-opt***[17]. Takes the end sections of two routes, and swaps them to create two new routes.

h_{10}: **GENI**[11]. A customer is taken from one route, and placed into another route, between the two customers of that route which are closest to it. Re-optimisation is then performed on the route.

Crossover Heuristics

h_{11}: **Combine**. A random percentage of routes (between 25% and 75%) are kept from one of the solutions (chosen randomly.) Then all routes which don't contain any conflicts with the routes already chosen are taken from the other solution. Finally, all unrouted customers are inserted into the solution.

h_{12}: **Longest Combine**. All routes from both solutions are taken and ordered by length (here length is defined as the number of customers served in a route.) The routes are taken from longest to shortest, providing there are no customer conflicts. Then, all unrouted customers are inserted into the solution.

3.4 Problem Instances

The problem instances provided in this module are taken from two sources. The first is the Solomon data set of 100 customer problems. The second is the Gehring and Homberger data set of 1000 customer problems. For both data sets, there are three types of instances. These are:

R: **Random**. The customers' locations are determined in a uniformly random way.
C: **Clustered**. The customers' locations are grouped in a number of clusters.
CR: **Clustered Random**. The customers' locations are in a mix of random and clustered locations.

4 Adaptive Iterated Local Search Hyper-heuristics

Iterated local search is a relatively simple but successful algorithm. It operates by iteratively alternating between applying a move operator to the incumbent solution and restarting local search from the perturbed solution. This search principle has been rediscovered multiple times, within different research communities and with different names [1,13]. The term *iterated local search* (ILS) was proposed in [12]. The algorithms compared in this article can be considered as ILS with multiple perturbation heuristics and multiple local search heuristics. They can be considered to be hyper-heuristics as they both coordinate several low-level heuristics and operate in a domain-independent fashion. Three variants were considered as described below.

4.1 The Baseline ILS Hyper-heuristic

The ILS implementation proposed in [4] contains a perturbation stage during which a neighborhood move is selected uniformly at random (from the available pool of mutation and ruin-recreate heuristics) and applied to the incumbent solution. This perturbation phase is then followed by an improvement phase, which works as follows. Each of the local search heuristics is independently applied to the incumbent solution. Providing at least one of the applications has yielded an improvement, then the application resulting in the greatest improvement in objective function is kept. The process is then repeated until no improvement in objective function value is found. If the resulting new solution is better than the original solution then it replaces the original solution, otherwise the new solution is simply discarded. This last stage corresponds to a greedy (only improvements) acceptance criterion. The pseudo-code of this iterated local search algorithm is shown below (Algorithm 1), notice that this differs from traditional implementations of ILS in that multiple heuristics are used in both the improvement and perturbation stages. We refer to this algorithm as *Rnd-ILS*.

4.2 The Adaptive ILS Hyper-heuristics

The adaptive versions of the base-line ILS hyper-heuristic described above, incorporate adaptive mechanisms in the perturbation and/or the improvement stages. The most successful adaptive ILS hyper-heuristic suggested in [6] implements an online learning mechanisms for selecting the move operators in the perturbation stage, instead of selecting then uniformly at random at each iteration. Specifically, it implements an adaptive operator selection mechanisms. As discussed in

Algorithm 1. *Iterated Local Search Hyper-heuristic.*

s_0 = GenerateInitialSolution
s^* = ImprovementStage(s_0)
repeat
  s' = PerturbationStage (s^*)
  $s^{*'}$ = ImprovementStage(s')
  if $f(s^{*'}) < f(s*)$ **then**
    $s^* = s^{*'}$
  end if
until time limit is reached

[10], an adaptive operator selection scheme consists of two components: a *credit assignment*mechanisms and a *selection* mechanism. The algorithm proposed in [6]used *extreme value* credit assignment, which is based on the principle that infrequent, yet large, improvements in the objective score are likely to be more effective than frequent, small improvements [10]. It rewards operators which have had a recent large positive impact on the objective score, while consistent operators that only yield small improvements receive less credit, and ultimately have less chance of being chosen. Following the application of an operator to the problem, the change in objective score is added to a window of size W, which works on a FIFO mechanism. The credit for any operator is the maximum score within the window. Window size plays an important part in the mechanism. If it is too small then the range of information on offer is narrowed, meaning that useful operators are missed. If it is too large then information is considered from many iterations ago, when the position in the search space might have meant that the operator performed differently to how it would at the latest iteration. However, the window size is the only parameter that needs to be tuned, which is a desirable property when the goal is to achieve robust and general algorithms. After testing several values of (W), we decided upon a value of 25. The credit assignment mechanism is combined with a selection strategy that uses the accumulated credits to select the operator to apply in the current iteration. Operator selection strategies in the literature, generally assign a probability to each operator and use a roulette wheel-like process to select the operator according to them. We use here one of these rules, namely, *adaptive pursuit*, originally proposed for learning automata and adapted to the context of operator selection in [22]. With this method, at each time step, the operator with maximal reward is selected and its selection probability is increased (follows a winner-take-all strategy.), while the other operators have their selection probability decreased. We refer to this adaptive ILS hyper-heuristic as *Ad-ILS*.

The variant proposed in this article keeps the adaptive selection of operators in the perturbation stage described above, but modifies the improvement stage by incorporating a simple adaptive mechanisms that considers the past performance of the local search heuristics. The mechanism works as follows: each heuristic has a score attributed to it, which is updated after each application of that heuristic. The score corresponds to the mean improvement in objective function

obtained from that heuristic's applications (from all applications across whole search). These scores are then used to order the local search heuristics, with the best performing heuristics being placed to the front of the list. The local search heuristics are then applied in sequence following this order. This new improvement stage is illustrated below (see Algorithm 2). We refer to the ILS hyper-heuristic with this modified component as *AdOr-ILS*.

Algorithm 2. *Ordered ImprovementStage.*

```
repeat
  ls ← OrderLocalSearches(scores)
  for i = 0 → numLocalSearchers, in the order ls do
    s' = LocalSearch(s',i)
    scores ← UpdateScores
  end for
until no improvement found
```

5 Experiments and Results

For testing the three algorithm variants described above, *Rnd-ILS*, *Ad-ILS* and *AdOr-ILS*, 10 instances were chosen, representing a range of instance types from both the Solomon and Gehring-Homberger data sets (see Table 1). These 10 instances are those currently available in the version of the HyFlex software used for the competition (which can be downloaded from the CHeSC 2011 website [14]).

Twenty runs were performed for each instance and algorithm. Following the experimental set up used in the CHeSC competition, the running time was set to 10 CPU minutes. The machine running the tests has a 2.27GHz Intel(R) Core(TM) i3 CPU and 4GB RAM.

Table 1. Capacitated vehicle routing problem instances, taken from [20]

Instance	name	no. vehicles	vehicle capacity
0	Solomon/RC/RC207	25	1000
1	Solomon/R/R101	25	200
2	Solomon/RC/RC103	25	200
3	Solomon/R/R201	25	1000
4	Solomon/R/R106	25	200
5	Homberger/C/C1-10-1	250	200
6	Homberger/RC/RC2-10-1	250	1000
7	Homberger/R/R1-10-1	250	200
8	Homberger/C/C1-10-8	250	200
9	Homberger/RC/RC1-10-5	250	200

Table 2 shows the average and standard deviation of the best objective function value at the end of the run, from the ten runs per instance. The adaptive ILS

hyper-heuristic that incorporates both adaptive operator selection and adaptive ordering of the local searchers (*AdOr-ILS*)outperforms the other two variants in 9 out otf the 10 instances. Only for one of of the smallest and less constrained instances (instance 1), it is the base-line ILS hyper-heuristic the one producing the best performance. It seems that the added complexity of the adaptive mechanisms does not help in this case. The experiments also suggest that for the smaller Solomon instances (instances 0 to 4), the difference in performance among the competing algorithms is less noticeable.

Table 2. Vehicle routing results for the 10 instances in Table 1. The entries account for the average and standard deviation of objective function values (out of 20 runs).

instance	*AdOr-ILS*	*Ad-ILS*	*Rnd-ILS*
0	$\mathbf{5281.71}_{334.614}$	$5406.48_{404.159}$	$5292.43_{337.186}$
1	$21291.89_{482.56}$	$21212.60_{509.28}$	$\mathbf{21054.87}_{500.73}$
2	$\mathbf{13605.03}_{451.64}$	$13932.67_{616.29}$	$13827.54_{516.39}$
3	$\mathbf{6564.42}_{554.77}$	$7055.26_{748.15}$	$6760.62_{597.41}$
4	$\mathbf{14280.79}_{319.54}$	$14549.22_{449.1}$	$14600.09_{471.7}$
5	$\mathbf{155305.46}_{6154.24}$	$163041.76_{11226.39}$	$180301.07_{2921.14}$
6	$\mathbf{77302.72}_{3384.83}$	$79175.63_{3431.57}$	$82316.66_{2326.49}$
7	$\mathbf{163177.74}_{2100.09}$	$164341.16_{1550.06}$	$169729.31_{1721.3}$
8	$\mathbf{158941.93}_{2460.71}$	$163332.72_{4314.93}$	$172007.42_{2055.46}$
9	$\mathbf{149447.68}_{1500.9}$	$150276.89_{1644.28}$	$153648.66_{1079.4}$

The boxplots shown in Figure 1 illustrate the magnitude and distribution of the best objective values for 4 of the harder Homberger instances (instances 5, 7, 8 and 9). Each plot summarises the result of 20 runs from each algorithm. It can be clearly observed that the best performing hyper-heuristic is *AdOr-ILS*, followed by *Ad-ILS*. The base-line non-adaptive ILS hyper-heuristic is the less competitive in these challenging instances.

To test statistical significance between the performances of Ad-ILS and the new variant, AdOr-ILS, the two sided Wilcoxon Signed Rank test has been used. The test is performed at the 95% confidence level, where a p value of less than 0.05 indicates a rejection of the null hypothesis - this being that there is no difference between the results. The following table shows the p values for each instance. From the table, we can see that in seven out of the ten instances, there is a statistical difference between the results.

Table 3. p-values resulting from comparisons of Ad-ILS and AdOr-ILS. Values of less than 0.05 (shown in bold) indicate statistical significance.

Instance	0	1	2	3	4	5	6	7	8	9
p-value	**0.017**	0.455	**0.023**	**0.021**	**0.005**	**0.04**	0.086	**0.048**	**0.005**	0.126

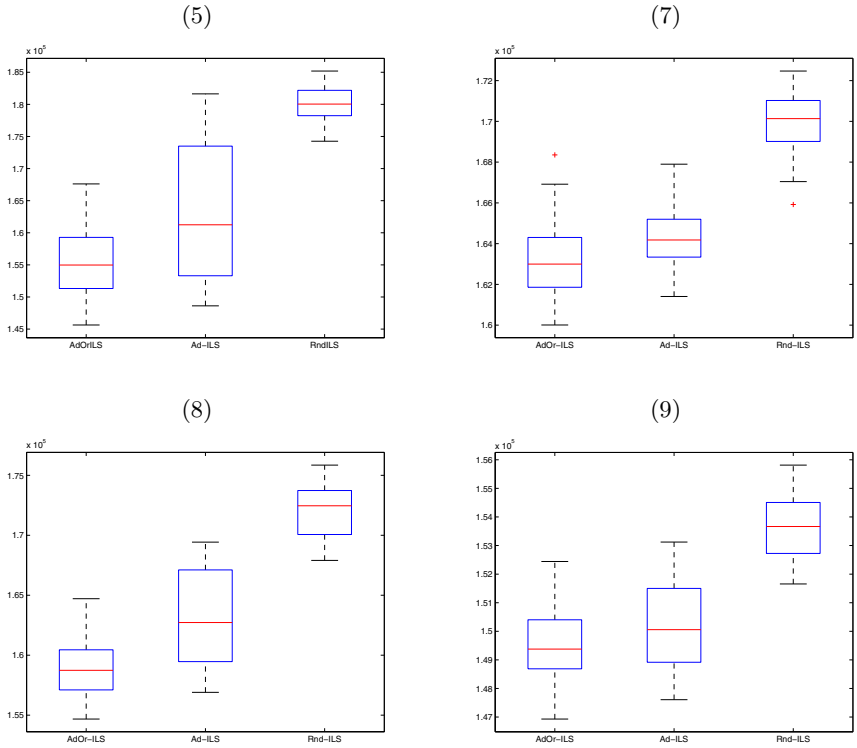

Fig. 1. Distribution of objective function values for the harder Homberger instances (instances 5, 7, 8 and 9 from Table 1)

6 Conclusions

This paper summarises the design of the capacitated vehicle routing domain for the HyFlex hyper-heuristic framework. The domain makes use of a large number of low-level heuristics, many of which are considered to be among the state-of the-art in current vehicle routing research. Considering this, the domain provides the opportunity to develop domain independent high-level algorithms or hyper-heuristics for the vehicle routing problem without the need to develop the low-level heuristics and underlying data structures. The domain was used as a hidden domain for the CHeSC competition and has been made freely available. It is our aim that it will be utilised within the hyper-heuristic, adaptive operator selection, adaptive multi-meme algorithms, and autonomous control for search algorithms research themes in intelligent optimisation.

The vehicle routing domain was used to test adaptive hyper-heuristics that can be considered as multiple-neighborhood iterated local search algorithms. This algorithmic scheme has proved to have good generalisation abilities. The inclusion of adaptive mechanisms both at the perturbation stage and at the

improvement stage of the ILS framework led to an improved performance on the challenging vehicle routing instances.

Future work will extend the the vehicle routing domain by including new low-level heuristics, and additional problem instances. New hyper-heuristics can be implemented using HyFlex. For example, we are currently exploring the use of a population and the crossover heuristics, which were not employed by our ILS hyper-heuristics. We are also exploring mechanisms for adapting the heuristic parameters provided in the framework, namely, *intensity-of-mutation* and *depth-of-search*, which have an important impact in the hyper-heuristic performance.

Finally, HyFlex can be extended to include new domains, additional instances and operators in existing domains; and multi-objective and dynamic problems. The current software interface can also be extended to incorporate additional feedback information from the domains to guide the adaptive search controllers. It is our vision that the HyFlex framework will continue to facilitate and increase international interest in developing domain independent and adaptive heuristic search methodologies, that can find wider application in practice.

References

1. Baxter, J.: Local optima avoidance in depot location. Journal of the Operational Research Society 32, 815–819 (1981)
2. Bleuler, S., Laumanns, M., Thiele, L., Zitzler, E.: PISA – A Platform and Programming Language Independent Interface for Search Algorithms. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 494–508. Springer, Heidelberg (2003)
3. Braysy, O., Gendreau, M.: Vehicle routing problem with time windows, part i: Route construction and local search algorithms. Transportation Science (2005)
4. Burke, E.K., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., Vazquez-Rodriguez, J.A., Gendreau, M.: Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms. In: IEEE Congress on Evolutionary Computation (CEC 2010), Barcelona, Spain, pp. 3073–3080 (July 2010)
5. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., McCollum, B., Ochoa, G., Parkes, A.J., Petrovic, S.: The Cross-Domain Heuristic Search Challenge – An International Research Competition. In: Coello, C.A.C. (ed.) LION 5. LNCS, vol. 6683, pp. 631–634. Springer, Heidelberg (2011)
6. Burke, E.K., Gendreau, M., Ochoa, G., Walker, J.D.: Adaptive iterated local search for cross-domain optimisation. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011, pp. 1987–1994. ACM, New York (2011)
7. Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology. In: Glover, F., Kochenberger, G. (eds.) Handbook of Metaheuristics, pp. 457–474. Kluwer (2003)
8. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.: A Classification of Hyper-heuristic Approaches. In: Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol. 146, ch. 15, pp. 449–468. Springer (2010)
9. Cowling, P.I., Kendall, G., Soubeiga, E.: A Hyperheuristic Approach to Scheduling a Sales Summit. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 176–190. Springer, Heidelberg (2001)

10. Fialho, Á., Da Costa, L., Schoenauer, M., Sebag, M.: Extreme Value Based Adaptive Operator Selection. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) PPSN X. LNCS, vol. 5199, pp. 175–184. Springer, Heidelberg (2008)
11. Gendreau, M., Hertz, A., Laporte, G.: A new insertion and postoptimization procedures for the traveling salesman problem. Operations Research (1992)
12. Lourenco, H.R., Martin, O., Stutzle, T.: Iterated Local Search, pp. 321–353. Kluwer Academic Publishers, Dordrecht (2002)
13. Martin, O., Otto, S.W., Felten, E.W.: Large-step Markov chains for the TSP incorporating local search heuristics. Operations Research Letters 11(4), 219–224 (1992)
14. Ochoa, G., Hyde, M.: The Cross-domain Heuristic Search Challenge, CHeSC 2011 (2011), http://www.asap.cs.nott.ac.uk/external/chesc2011/
15. Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J.A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A.J., Petrovic, S., Burke, E.K.: HyFlex: A Benchmark Framework for Cross-Domain Heuristic Search. In: Hao, J.-K., Middendorf, M. (eds.) EvoCOP 2012. LNCS, vol. 7245, pp. 136–147. Springer, Heidelberg (2012)
16. Or, I.: Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking. PhD thesis, Northwestern University, Evanston, IL (1976)
17. Potvin, J.-Y., Rousseau, J.-M.: An exchange heuristic for routeing problems with time windows. The Journal of the Operational Research Society (1995)
18. Savelsbergh, M.W.P.: The vehicle routing problem with time windows: Minimizing route duration. Informs Journal on Computing 4(2), 146–154 (1992)
19. Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G.: Record breaking optimization results using the ruin and recreate principle. Journal of Computational Physics (2000)
20. SINTEF. VRPTW benchmark problems, on the SINTEF transport optimisation portal (2011), http://www.sintef.no/Projectweb/TOP/Problems/VRPTW/
21. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research 64(2), 278–285 (1993)
22. Thierens, D.: An adaptive pursuit strategy for allocating operator probabilities. In: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, GECCO 2005, pp. 1539–1546. ACM, New York (2005)