Learning a Stopping Criterion for Local Search

Alejandro Arbelaez and Barry O'Sullivan

Insight Centre for Data Analytics
Department of Computer Science, University College Cork, Ireland
{alejandro.arbelaez|barry.osullivan}@insight-centre.org

Abstract. Local search is a very effective technique to tackle combinatorial problems in multiple areas ranging from telecommunications to transportations, and VLSI circuit design. A local search algorithm typically explores the space of solutions until a given stopping criterion is met. Ideally, the algorithm is executed until a target solution is reached (e.g., optimal or near-optimal). However, in many real-world problems such a target is unknown. In this work, our objective is to study the application of machine learning techniques to carefully craft a stopping criterion. More precisely, we exploit instance features to predict the expected quality of the solution for a given algorithm to solve a given problem instance, we then run the local search algorithm until the expected quality is reached. Our experiments indicate that the suggested method is able to reduce the average runtime up to 80% for real-world instances and up to 97% for randomly generated instances with a minor impact in the quality of the solutions.

1 Introduction

Local search is a popular technique to solve many combinatorial problems. These problems can be classified as either decision or optimisation problems. A combinatorial decision problem consists in finding a solution that satisfies the constraints of the problem. The satisfiability (SAT) problem is perhaps one of the most important decision problems and consists in determining whether a given Boolean formula in conjunctive normal form (i.e., conjunction of clauses) is satisfiable or not. In combinatorial optimisation the goal is to find a solution that satisfies the constraints and proving that the solution is optimal. The Travelling Salesman Problem (TSP) is a well-known combinatorial optimisation problem that involves finding the shortest Hamiltonian circle in a complete graph.

A local search algorithm starts with a given initial solution and iteratively improves the solution, little by little, by performing small changes while a given stopping criterion is not met. In the context of SAT the algorithm typically starts with a random assignment for the variables and the algorithm flips one variable at a time until either a solution is obtained or a time limit is reached. Alternatively, to solve a TSP the algorithm heuristically builds a starting solution and then applies the k-opt move, i.e., k-links of the current solution are replaced with k new links by improving the current solution until: the optimal solution is reached;

or no improvement is observed after applying the *k*-opt move; or a given time limit is reached.

In this paper, we focus our attention in local search to tackle combinatorial optimisation problems. Ideally the algorithm is executed until a given target solution is reached, (i.e., optimal or near-optimal). However, in many real-world problems such solution is unknown. Therefore, this paper exploits the use of instance features to predict the expected quality of the solution for a given problem instance. We then execute the local search algorithm until the desired quality is reached. A proper stopping criterion might considerably reduce the runtime of the algorithm, and therefore accurate predictions are very important in particular with the increasing computational power available in cloud systems, e.g., Amazon Cloud EC2, Google Cloud, and Microsoft Azure. Typically, these systems can be rented by processor-hour therefore reducing the computational time and still providing high quality solutions is expected to be vary valuable in the near future.

This paper is structured as follows. Sections 2 and 3 provide background material of the two target problems, i.e., CRP and TSP. Section 4 details the machine learning methodology to design the stopping criterion. Section 6 reports on the experimental validation. Related work is discussed in Section 6 before final conclusions are presented in Section 7.

2 The Cable Routing Problem

The *Cable Routing Problem* (CRP) that we are tackling in this paper relates to the bounded spanning tree problem with side constraints. In particular we focus on a network design problem arising in optical networks where a bound is given on the number of carries and each selected carrier is connected to a set of clients using a tree topology that respects distance, degree, and capacity constraints.

The long-reach passive optical network (LR-PON) architecture consists of three subnetworks: (1) an Optical Distribution Network (ODN) for connecting customers to facilities; (2) a backhaul network for connecting facilities to metro-nodes; and (3) a core network connecting pairs of metro-nodes. In this paper we focus our attention in the ODN part, where the fibre cable is rounded from a facility to a set of customer forming a tree distribution network. A PON is a tree network associated with each cable and the signal attenuation in a PON is due the number of customers in the PON and the maximum length of the fibre between the facility and the customer. Additionally, non-root nodes are limited to maximum branching of p nodes. In [1] the authors show the relationship between the size and the maximum length of the PON.

Informally speaking, in the CRP we want to determine a set of spanning trees with side constraints, i.e., distance, degree, and capacity, such that each tree is rooted at the facility, each customer is present in exactly one tree and the total cost or quality of the solution is minimised. The number of trees denotes the number of optical fibres that run from the exchange-site to the customers. Typically, the distance is limited to up 10 km, 20 km, and 30 km for respectively at most 512, 256, and 128 customers, and due to hardware constraints the branching factor is restricted to 32.

Local Search for CRP

In [2] the authors propose a general constraint-based local search to tackle bounded minimum spanning tree with side constraints. Broadly speaking, the local search algorithm comprises two phases. First, in an intensification phase, the algorithm improves the current solution, little by little, by performing small changes. Generally speaking, it employs a move-operator in order to move from one solution to another in the hope of improving the value of the objective function. Second, in the diversification phase, the algorithm perturbs the incumbent solution in order to escape from difficult regions of the search. The algorithm switches from intensification to diversification when a local minimum is reached.

The subtree operator (Figure 1) moves a given node e_i and the subtree emanating from e_i from the current location to another in the tree. As a result of this, the predecessor of e_i is not connected to e_i, and all successors of e_i are still directly connected to the node. e_i can be placed as a successor for another node or in the middle of an existing edge. In order to complete the intensification and diversification phases, the subtree operator requires four main functions: removing a subtree; checking whether a given solution is feasible or not; inserting a subtree; finding the best location of a subtree in the current subtree; finding the best location of a subtree in the current network. The general idea of the LS algorithm in the intensification phase is described as follows:

1. Randomly select a node (e_i);
2. Delete the emanating subtree of e_i in the current solution;
3. Identify the best location, i.e., a new predecessor e_p and a potential successor e_s for e_i satisfying all constraints;
4. Insert e_i as a new successor of e_p, and if needed, add es as a new predecessor of e_p.

During the diversification phase the third step performs a random selection of the new candidate location of e_i in the tree. It is worth noticing that the local search operator has been used in several network design applications such as [3] and [4].

A solution is represented by a tree whose root-node is the facility and the number of immediate successors of the root-node is the number of cables starting at the facility. Notice that the facility acts as the root-node of each cable tree network. Without loss of generality we add a set of dummy clients (or copies of the facility) to the original set of clients for the purpose of ease of representation to distinguish the cable tree-networks. More precisely the set of clients, $\{v_0, \ldots, v_n\}$, is modified to $\{v_0, v_1, \ldots, v_m, v_{1+m} \ldots, v_{n+m}\}$. Recall that n is the number of clients and m is the upper bound on the number of cables that can start at the facility. In the latter set v_0 is the original facility, each $v_i \in \{v_1 \ldots v_m\}$ denote the starting point of a cable tree network, and $\{v_{1+m}, \ldots, v_{n+m}\}$ denote the original set of clients. We further enforce that each dummy client is connected to v_0 and the distance between the dummy client and the facility is zero, i.e., $\forall 1 \leq i \leq m, d_{0i} = 0$.

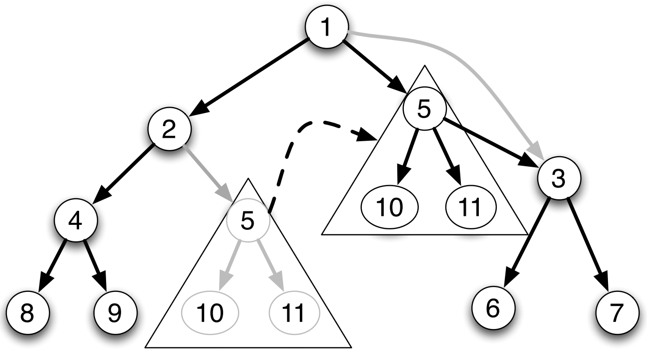


Fig. 1. Example of the subtree move-operator

Default stopping criterion. In this paper we use search stagnation as the default stopping criterion and compute a bound in the solution. In particular, we stop the algorithm whenever no improvement greater than 0.01% in the incumbent solution has been observed for 30 consecutive seconds.

3 Traveling Salesman Problem

The *Traveling Salesman Problem* (TSP) is a well-known combinatorial optimisation problem with applications in multiple areas ranging from transportation to VLSI circuit design, and bioinformatics. The TSP consists in finding the shortest tour among a predefined list of n cities and all cities are visited only once. In this paper we focus our attention in the 2D Euclidean TSP in which we are given a set of points in a plane, and the distance between two points is the Euclidean distance between their corresponding coordinates.

Local Search for TSP

The Lin-Kernighan heuristic (LKH) [5] is an efficient local search implementation to tackle TSPs capable of solving instances with tens of thousands of cities. Although this is an incomplete algorithm it is known to provide near-optimal solutions within very short computational time.

The 2-opt operator [6] is one of the first local search move operators to solve TSP instances. Figure 2 shows an example of the operator by removing (black dotted lines) edges (a, b) and (c, d) from the current tour and reconstructing the tour, in the new solution, by connecting a with d and b with c (grey lines). Notice that after removing the edges in the 2-opt there is only one feasible way of reconnecting the tour. For instance, in our example adding edges (a, c) and (b, d) will not result in a valid solution.

In general, the k-opt move removes k edges from the current tour, k ranges from two to the number of cities. For $k > 2$ there are multiple ways of reconstructing the solution, in this case the algorithm typically uses the one with the shortest tour.

Currently, there is a large variety of heuristics to create the initial tour or solution such as Nearest Neighbour and Quick-Boruvka (see [7] for a complete description of the algorithms). However, LKH suggests the use of randomly generated tours as an initial solution. Other features of the LKH algorithm include the ability to restrict the use in the tour of certain edges within a given distance of a given city, and efficient data structures to check whether a solutions is a valid tour or not.

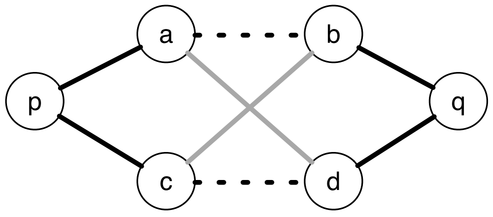


Fig. 2. Example of the 2-opt

Default stopping criterion. The algorithm stops when no improvement can be observed for a given solution, that is, when no neighbour solution can improve the current tour. Additionally, we also use a certain time limit to stop the execution of the algorithm.

4 Machine Learning Methodology

Supervised Machine Learning exploits data labelled by the expert to automatically build hypotheses emulating the expert's decisions. Formally, a learning algorithm processes a training set $\mathcal{E} = \{(x_i, y_i), x_i \in \Omega, y_i \in \{1, -1\}, i = 1 \ldots n\}$ made of n examples (x_i, y_i), where x_i is the example description (e.g. a vector of values, $\Omega = R^d$) and y_i is the associated output. The output can be numerical (i.e., regression) or a class label (i.e., classification). The learning algorithm outputs a hypothesis $f : \Omega \mapsto Y$ associating to each example description x

the output $y = f(x)$. Among ML applications are pattern recognition, ranging from computer vision to fraud detection [8], predicting protein function [9], game playing [10] or autonomic computing [11].

In Figure 3 we depict the general methodology to learn a stopping criterion for a given problem instance. Similar to other machine learning applications, the learning process takes place offline and involves computing features and the quality of the solution for each instance in the training set. Later on in the online testing phase, for each unseen instance, we compute the feature set and use the ML model to compute the expected outcome (i.e., solution quality) of the algorithm for the instance.

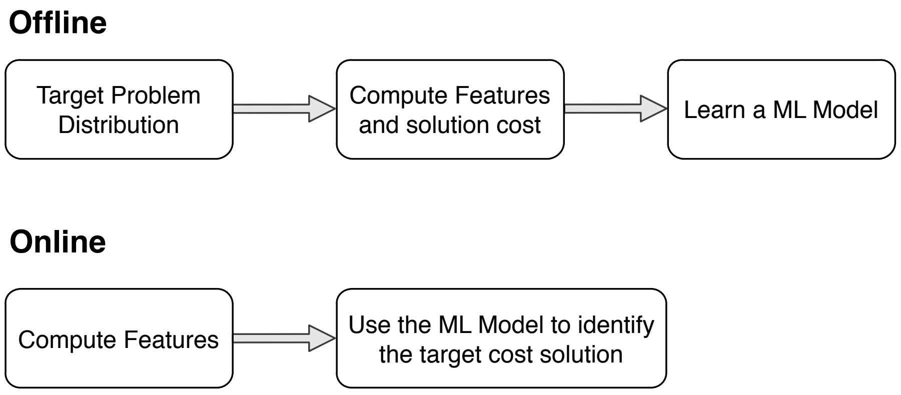

Offline
Target Problem Distribution
Compute Features and solution cost
Learn a ML Model
Online
Compute Features
Use the ML Model to identify the target cost solution

Fig. 3. ML methodology to learn a stopping criterion

In recent years, machine learning has been used extensively to design portfolios of algorithms. Informally speaking, a portfolio of algorithms involves using machine learning techniques to identify the most suitable algorithm to solve a given problem instance (see [12] for a recent survey). SATzilla [13] is probably the most popular portfolio of algorithms for SAT solving. Informally speaking, SATzilla uses a set of features to describe a given problem instance, the feature set encodes general information about the target instance, e.g., number of variables, clauses, fraction of Horn clauses, number of unit propagations, etc. Similar to our approach, SATzilla employs a training and testing phases. During the training phase the portfolio requires a set of target instances and a set of SAT solvers. During the testing phase, a set of pre-solvers are executed in a pre-defined manner for a short period and if no solution is obtained during the pre-solving time, the algorithm with minimal expected runtime is executed. SATzilla has shown impressive results during the past SAT competition winning several tracks in the annual SAT competitions.

Reinforcement learning, another branch of machine learning, has also been an effective alternative in the development of self-tuning algorithms. In contrast with the previous portfolio of algorithms approach, here the algorithms have the ability to adjust the parameters and reach to fast chaining conditions while solving a problem instance. An interesting application of these algorithm's is the reactive search framework described and analysed in [14] where the authors proposed a mechanism for adjusting the size of the tabu list by increasing (resp. decreasing) its size according to the progress of the search

CRP Instance Features

For each instance we compute two main type of features: basic and initial solution features. While a few of these features had already been used in the context of portfolio of algorithms for the TSP problem, many descriptors had to be added to consider properties of the CRP with distance, degree, and capacity constraints.

Basic features. These encode general information of a given problem instance:

1. Number of nodes (1 feature);
2. Distance matrix (3 features): mean, median, std;
3. Distance to root-node (3 features): mean, median, std;
4. Pairwise distance (3 features): mean, median, std.

Initial solution features. For each instance we compute multiple initial solutions to extract the following features, for each of the following five categories we compute the mean of 5 independent initial solutions:

1. Node out-degree (3 features): mean, median, std;
2. Root-node out-degree (3 features): mean, median, std;
3. Max. distance to root (3 features): mean, median, std;
4. Size of the cable tree networks (3 features): mean, median, std;
5. Number. of leaf nodes (3 features): mean, median, std.

TSP Instance Features

In [15] the authors describe 50 descriptors to characterise TSP instances. Computing the full set of features is computationally very expensive. Therefore we limit our attention to the following twelve features divided in three categories:

1. Problem size features (1 feature): number of cities;
2. Distance matrix features (3 features): mean, variation coefficient, skew;
3. Minimum spanning tree (8 features): after constructing the minimum spanning tree there are four features describing the distance (mean, variation coefficient, skew) and four features describing node degree statistics: mean, variation coefficient, and skew).

Stopping Criterion

The pseudo-code for a generic local search algorithm with the proposed stopping criterion is depicted in Algorithm 1. The algorithm requires the target instance I, a regression model m, and a discrepancy $d \in (0, 1]$ indicating how far from the target solution should be from the expected solution. Certainly, the choice of a value close to zero for d leads to better solutions at a cost of using more time to reach the target solution.

The algorithm starts by computing the vector of features v for a given instance I (line 1), then we use the regression model m to compute the expected quality of the solution (line 2). Without loss of generality we assume a minimisation problem. Thus, we compute the target quality as the tolerance between the computed solution and the outcome of the regression model (line 3). Lines 4-7 sketch a general description of local search solvers, computing an initial solution (line 4), and iteratively moving from one solution to another using a given move operator, e.g., subtree or k-opt operators. We recall that we use the target solution in addition to the default stopping criterion of the algorithm. Thus, for instance, if the algorithm reaches a time limit we also stop the execution.

Algorithm 1 LocalSearchWithStoppingCriterion(*Problem-Instance I*, *ML-Model m*, *Discrepancy d*)

```
1: $v \leftarrow$ Compute-features(I)
2: $c \leftarrow$ Compute-expected-quality(m, v)
3: $target\_quality \leftarrow c + c \cdot d$
4: $s \leftarrow$ Initial-solution(I)
5: while default stopping criterion is not met and quality($s$) $>$ $target\_quality$ do
6:     $s \leftarrow step(I, s)$
7: end while
8: return $s$
```

5 Empirical Evaluation

In this experiments we consider a collection of 500 real-world CRP instances from a major broadband provider in Ireland. For TSP instances we randomly generated a set of 300 instances with the *portcgen* problem generator varying the number of cites from 500 to 5000. We ran each local search algorithm 20 times on each instance (each time with a different random seed) with a 5-minute time cutoff and report the average time and solution quality for the instances. All the experiments were performed on a 39-node cluster, each node features a Intel Xeon E5430 processors at 2.66 Ghz and 12 GB of RAM memory.

In order to validate our algorithms we used the traditional 10-fold cross-validation technique [16], that is, the entire dataset D is divided into 10 disjoint sets $\{D_1, D_2, \ldots, D_{10}\}$. For each dataset $D_{i \in 10}$, the regression model is learned with $D \setminus D_i$ and tested with D_i. In the following experiments we use the linear regression implementation available in the Weka toolbox (version 3.6.2) [17]. We use to state-of-the-art local search solvers [2, 5] to tackle CRPs and TSPs, respectively.

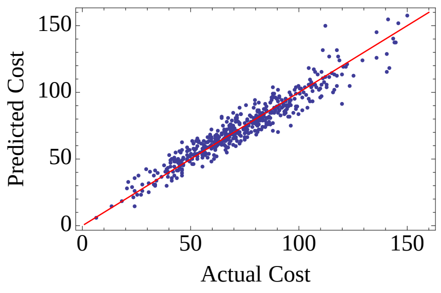
We computed the baseline solutions using the default stopping criterion of each respective local search solver with a time limit of 300 seconds. We use this criterion as reference to compute the gap of the solutions as follows:

$$GAP(I) = \frac{\textit{ls-quality}(I) - \textit{baseline-quality}(I)}{\textit{baseline-quality}(I)}$$

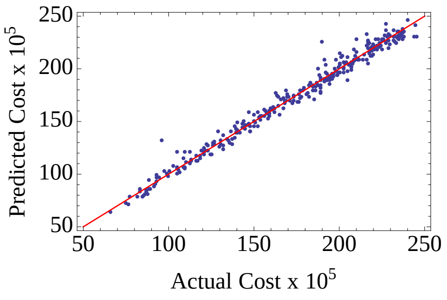
where *baseline-quality* and *ls-quality* indicate respectively the quality of the baseline solutions and the quality of the local search with the suggested stopping criterion for a given instance I.

Table 1. Correlation Coefficient and Root mean square error of estimations and runtime for the baseline computation

Problem	CC	RMSE	Runtime (s)	Feature time (s)
CRP	0.95	7.2	89.0	2.05
TSP	0.99	6.2	258.2	0.40



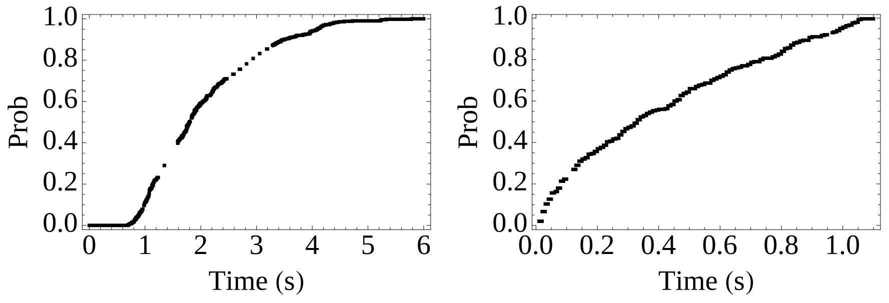
(a) 500 CRP instances



(b) 300 TSP instances

Fig. 4. Actual vs. Estimated quality for each CRP and TSP instance

We start our analysis with Table 1 in which we analyse the quality of the regression model to predict the quality of the baseline solutions. We report the Correlation Coefficient (CC) and the Root Mean Square Error (RMSE) between the baseline solutions and our predictions, the average runtime and the average time for feature computation. As it can be observed the regression model greatly predicts the quality of the actual baseline solutions with a CC of 0.95 (CRP) and 0.95 (TSP), and RMSE of 7.2 (CRP) and 6.2 (TSP). We also remark that the average feature computation time is considerably lower than the actual average runtime of the algorithms. In Figure 4 we provide a visual comparison of the actual baseline solution quality and the outcome of the regression model for the entire dataset. The red diagonal line represents a perfect prediction, as it can be observed the predictions are highly correlated with the baseline results.



(a) CRP (b) TSP

Fig. 5. Feature computation time

Figure 5 shows the cumulative distribution of the feature computation time for all instances for both solvers. The feature computation time ranges from 0.6 to 5.7 seconds for CRP instances and from 0.01 to 1.1 seconds for TSP instances. We would like to recall that we use a subset of the complete feature set described in [15] for the TSP problem, the complete feature set required up to several minutes to compute the descriptors.

We now focus our attention on the evaluation of the local search algorithms with the suggested stopping criterion. To this end, we evaluate eight values for the minimum desired discrepancy of the target solution (i.e., 0%-5%, 10% and 15%). Tables 2 and 3 depict the results showing: the average runtime, the gap with respect to the baseline solution, and the output of the Kolmogorov-Smirnov (KS) and Wilcoxon Signed-Rank (WSR) tests to check whether the solution with the new stopping criterion and the baseline solutions are statistically different or not. Bold entries indicate that the solution with the new stopping criterion is not statistically different than the baseline solution. Interestingly, in nearly all scenarios we observed that the results with the suggested stopping criterion are statistically the same as the baseline results, only 2 scenarios with KS and 4 scenarios with WSR failed the test with a 5% confidence level.

As expected, there is a trade-off between quality and speed. In our experiments we observed for CRP instances a reduction of up to 80% in the runtime to reach a solution 5.6% far from the baseline solution. Certainly, the runtime increases when enforcing better quality solutions, in particular we observed that for a discrepancy of 0% (i.e., setting as a target solution the predicted quality) the gap with respect to baseline solution is 3.5% and the runtime reduction is about 39%. A similar scenario can be observed for TSP instances where we

Table 2. CRP: statistics varying the discrepancy target to the expected solution

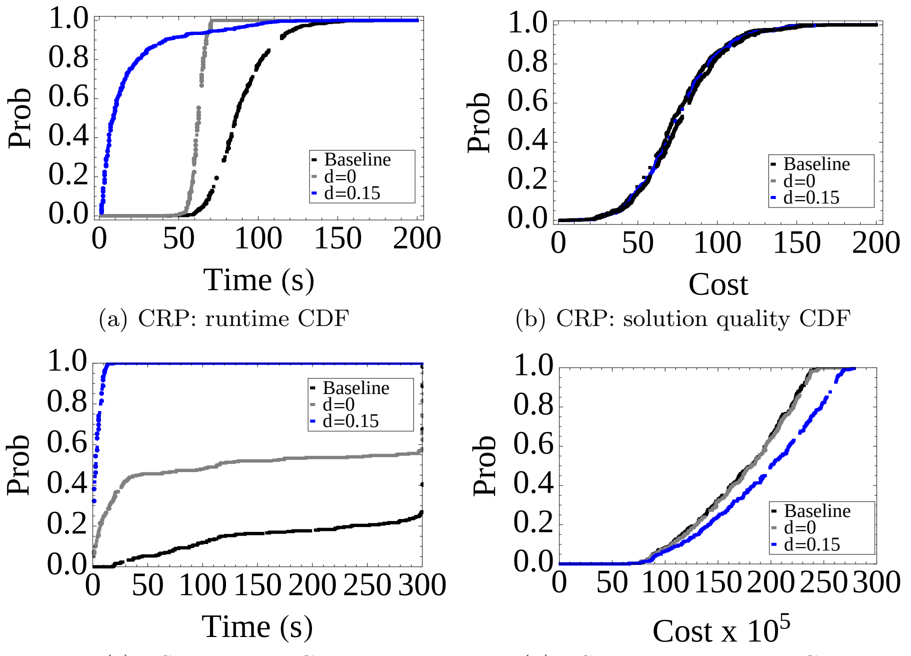
Discrepancy (%)	Time (s)	GAP(%)		KS(%)	WRS(%)
	avg	avg	std		
0	53.9	3.55	5.79	**55.9**	**20.4**
1	49.3	4.30	6.44	**36.9**	**9.63**
2	45.3	4.39	6.40	**32.9**	**8.46**
3	41.1	4.51	6.36	**25.7**	**7.45**
4	37.9	4.63	6.30	**22.6**	**6.44**
5	35.1	4.76	6.25	**22.6**	**5.56**
10	24.3	5.27	6.01	**9.5**	2.95
15	17.4	5.62	5.84	**6.9**	1.91

observe a reduction of 97% (with discrepancy=5%) in the runtime to reach a solution with a gap of 3.4%, and enforcing the best possible quality we observe a solution with a gap of 0.7% with a reduction of 40% in the runtime with respect to the baseline solution.

Table 3. TSP: statistics varying the discrepancy target to the expected solution, the last entry for the statistical tests are $5.4{\cdot}10^{-5}$ (KS) and $1.8{\cdot}10^{-5}$ (WRS)

Discrepancy (%)	Time (s)	GAP(%)		KS(%)	WRS(%)
	avg	avg	std		
0	153.4	0.76	1.72	**99.9**	**73.1**
1	121.9	1.09	1.91	**99.6**	**59.6**
2	84.61	1.54	2.11	**84.7**	**43.9**
3	58.7	2.10	2.32	**51.7**	**28.6**
4	39.0	2.73	2.48	**29.2**	**16.8**
5	22.8	3.43	2.62	**14.6**	**8.52**
10	5.5	7.50	3.11	0.01	0.02
15	3.7	11.51	3.62	0.00	0.00

We conclude our analysis with the cumulative distribution function (CDF) of the algorithms. Figure 6 describes the probability of a given algorithm to find a solution for a given instance with time (resp. quality) less than or equal to x. x-axis denotes the time (resp. quality) and y-axis denotes the probability to reach a certain time (resp. quality). Figures 6(b) and 6(d) visually confirm the output of the KS and WRS tests, that is, the cost of the solutions are very close to the baseline for two discrepancy values of the CRP. For the TSP we observe that the baseline is very close to the baseline solutions for d=0% and slightly different for d=15%. Figures 6(a) and 6(c) show that the baseline stopping criterion requires more time than the modified version of the algorithm with the suggested stopping criterion.



(a) CRP: runtime CDF

(b) CRP: solution quality CDF

(c) TSP: runtime CDF

(d) TSP: solution quality CDF

Fig. 6. CDF analysis for the baseline solutions vs. discrepancies 0% and 15%

6 Related Work

Random restarts are often used to avoid getting trapped in local minima and search stagnation. Typically, the algorithm is stopped after completing a certain number of restarts or when a target solution (optimal or near-optimal) is obtained. However, in many real-world problems such a solution is not known. In [18] the authors propose a methodology to on-the-fly construct a probabilistic rule to estimate the probability of finding a solution at least as good as the current best solution in future restarts. The methodology starts by approximating the quality of the solution of the first k restarts with a Normal distribution. This distribution is then used to estimate the number of high quality solutions that will be observed in future restarts. The authors present empirical results with reliable predictions when k is large enough.

Alternatively, [19] uses the empirical distribution to identify the optimal the number of diversification (or exploration) steps required to escape from a local minimum. The idea behind this approach is too avoid using too much computational time with unnecessary diversification steps, while still providing statistical evidence that future intensification steps will not end up in the same plateau area. In this paper we propose a stopping criterion to balance the trade-off between the quality of the solution and its runtime regardless the methodology to escape plateaus and local minima, and without particular assumptions on the objective function.

In the field of continuous optimization several authors have proposed several stopping rules for the multistart framework. Here we briefly describe the basic ideas of a few stopping rules, we refer the reader to [20] for a complete description.

In [21] stopping rules are proposed based on Bayesian methods to stop as soon as there is enough evidence that the best value will not change in future restarts.A Bayesian stopping rule for GRASP is proposed in [22]. The authors assume that the distribution function is known beforehand and derive explicit rules for two particular cases.

In [23] the authors estimate the probability p that the best value observed in last restart is within certain threshold ϵ of the global optimum. p is approximated with the best solutions observed in previous restarts that at within ϵ of the incumbent solution. This approach was extended in [24] by counting the number of solutions that are within ϵ since the last update of the incumbent solution.

We remark that the above mentioned stopping rules for continuous optimization assume that the objective function satisfy certain properties, e.g., continuity, differentiable, Lipshitz condition, or that one should be able to find explicit formulas for the distribution function.

7 Conclusions

In this work, our objective was to study the application of machine learning techniques to carefully craft a stopping criterion for local search algorithms where the optimal solution is typically unknown before hand. In particular, we use instance features to predict the cost of the solution for a given algorithm to solve a given problem instance. Interestingly, we have observed that machine learning can indeed provide very accurate estimations of the expected quality of two efficient local search solvers. We have observed that using the estimation of the regression model we can reduce the average runtime by up to 80% for real-world CRP instances and by up to 97% for randomly generated instances with a minor impact in the quality of the solutions.

Further research involves the use of the estimated target solution in the context of tree-base algorithms to tackle optimisation problems in two main directions: first reducing the computation time to reach the target solution, and second, pruning the search space by eliminating candidate solutions as soon as we are able to detect that no improvement is expected in the target solution.

Acknowledgments

This work was supported by DISCUS (FP7 Grant Agreement 318137), and Science Foundation Ireland (SF) Grant No. 10/CE/I1853. The Insight Centre for Data Analytics is also supported by SFI under Grant Number SFI/12/RC/2289.

References

1. Davey, R., Grossman, D., Rasztovits-Wiech, M., Payne, D., Nesset, D., Kelly, A., Rafel, A., Appathurai, S., Yang, S.H.: Long-reach passive optical networks. Journal of Lightwave Technology **27**(3) (Feb 2009) 273–291
2. Arbelaez, A., Mehta, D., O'Sullivan, B., Quesada, L.: Constraint-based local search for the distance- and capacity-bounded network design problem. In: ICTAI'14, Limassol, Cyprus, November 10-12, 2014. (2014) 178–185
3. Arbelaez, A., Mehta, D., O'Sullivan, B., Quesada, L.: Constraint-based local search for edge disjoint rooted distance-constrainted minimum spanning tree problem. In: CPAIOR'15. (2015) 31–46
4. Arbelaez, A., Mehta, D., O'Sullivan, B.: Constraint-based local search for finding node-disjoint bounded-paths in optical access networks. In: CP'15. (2015) 499–507
5. Helsgaun, K.: An effective implementation of the lin-kernighan traveling salesman heuristic. European Journal of Operational Research **126**(1) (2000) 106–130
6. CROES, G.A.: A method for solving traveling salesman problems. Operations Res. **6** (1958) 791–812
7. Hoos, H., Stütze, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann (2005)
8. Larochelle, H., Bengio, Y.: Classification using Discriminative Restricted Boltzmann Machines. In: ICML'08, Helsinki, Finland, ACM (June 2008) 536–543
9. Al-Shahib, A., Breitling, R., Gilbert, D.R.: Predicting Protein Function by Machine Learning on Amino Acid Sequences – A Critical Evaluation. BMC Genomics **78**(2) (March 2007)
10. Gelly, S., Silver, D.: Combining Online and Offline Knowledge in UCT. In: ICML'07. Volume 227., Corvalis, Oregon, USA, ACM (June 2007) 273–280
11. Rish, I., Brodie, M., et al, S.M.: Adaptive Diagnosis in Distributed Dystems. IEEE Trans. on Neural Networks **16** (2005) 1088–1109
12. Kotthoff, L.: Algorithm selection for combinatorial search problems: A survey. AI Magazine **35**(3) (2014) 48–60
13. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: Portfolio-based algorithm selection for sat. Journal of Artificial Intelligence Research **32** (2008) 565–606
14. Battiti, R., Tecchiolli, G.: The reactive tabu search. INFORMS Journal on Computing **6**(2) (1994) 126–140
15. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: Methods & evaluation. Artif. Intell. **206** (2014) 79–111
16. Kahavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: IJCAI'95. (1995) 1137–1145
17. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explorations **11** (2009) 10–18
18. Ribeiro, C.C., Rosseti, I., Souza, R.C.: Effective probabilistic stopping rules for randomized metaheuristics: GRASP implementations. In: LION 5. (2011) 146–160
19. Bontempi, G.: An optimal stopping strategy for online calibration in local search. In: LION 5. (2011) 106–115
20. Pardalos, P.M., Romeijn, H.E.: Handbook of Global Optimization. Springer US (2002)
21. Boender, C.G.E., Kan, A.H.G.R.: Bayesian stopping rules for multistart global optimization methods. Math. Program. **37**(1) (1987) 59–80

22. Orsenigo, C., Vercellis, C.: Bayesian stopping rules for greedy randomized procedures. J. Global Optimization **36**(3) (2006) 365–377
23. Dorea, C.C.Y.: Stopping rules for a random optimization method. SIAM Journal on Control and Optimization (4) (1990) 841–850
24. Hart, W.E.: Sequential stopping rules for random optimization methods with applications to multistart local search. SIAM Journal on Optimization **9**(1) (1998) 270–290