Compendium: “Machine Learning Constructives and Local Searches for the Travelling Salesman Problem”

Tommaso Vitali[1], Umberto Junior Mele[1,2], Luca Maria Gambardella[1,2], and Roberto Montemanni[3]

[1] Università della Svizzera Italiana, Lugano 6900, Switzerland
[2] Dalle Molle Institute for Artificial Intelligence, IDSIA-SUPSI, Lugano 6900, Switzerland
[3] University of Modena and Reggio Emilia, Department of Sciences and Methods for Engineering, Reggio Emilia 42122, Italy

1 Introduction

The *ML-Constructive* heuristic is a recently presented method and the first hybrid method capable of scaling up to real scale traveling salesman problems. It combines machine learning techniques and classic optimization techniques. In the paper we presented improvements to the computational weight of the original deep learning model. In addition, as simpler models reduce the execution time, the possibility of adding a third local-search phase is explored to further improve performance. Experimental results corroborate the quality of the proposed improvements.

In this Compendium we show all the details required to replicate our experiments, and the complete results we obtained. All the code for the experiments, along with the trained *ML models*, can be found in the GitHub repository at https://github.com/tommivitali/ML-Constructive_LS.

2 Machine Learning setup

To find a ML model that works accurately and in a short time, several ML models were tried out and tested. In this compendium, the aim is to go into detail about the ML models used, specifying their hyper-parameters and the training decisions made. The models trained for the task are: a logistic regression (**Linear**), a linear Support Vector Machine (**SVM**), and an Ensemble of different classifiers (**Ensemble**). The scikit-learn software available for Python was used to edit these models.

2.1 Linear

For the Linear model, few changes have been made in the search for the best hyper-parameters, since it is used more as a test and benchmark model. The only change made is that of increasing the number of iterations available to the solver to converge to 1000.

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(max_iter=1000)
```

Listing 1.1. Linear ML model training

2.2 Support Vector Machine

The final hyper-parameters for the linear SVM were chosen after testing several variants. Based on the performance on the evaluation set, the best results were achieved with the hyper-parameters shown in Listing 1.2.

```
from sklearn.svm import LinearSVC

clf = svm.LinearSVC(penalty='l2',
                    loss='squared_hinge',
                    dual=False,
                    max_iter=100000)
```

Listing 1.2. Linear SVM ML model training

2.3 Ensemble

The final version of the Ensemble model was also identified after testing various conformation possibilities. The best results were obtained using three basic classifiers (Logisti Regression, Gradient Boosting and Linear SVM) and Random Forest as final estimator.

```
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier, StackingClassifier,
    RandomForestClassifier

estimators = [
    ('lin', LogisticRegression(max_iter=1000)),
    ('xgb', GradientBoostingClassifier(loss='deviance',
                                       learning_rate=0.5,
                                       n_estimators=200,
                                       min_samples_split=4,
                                       max_depth=4,
                                       n_iter_no_change=100)),
    ('svm', LinearSVC(penalty='l2',
                      loss='squared_hinge',
                      dual=False,
                      max_iter=100000))
]

clf = StackingClassifier(estimators=estimators,
                         final_estimator=RandomForestClassifier())
```

Listing 1.3. Ensemble ML model training

Algorithm 1 ML-Constructive

Require: TSP graph $G(V, E)$
Ensure: a feasible tour X

1: **procedure** ML-CONSTRUCTIVE($G(V, E)$)
2: create CL for each vertex
3: insert the shortest two vertices for each CL into L_P
4: sort L_P according to the position in the CL and the ascending costs $c_{i,j}$
5: $X = \bar{\mathbf{0}}$
6: **for** l in L_P **do**
7: select the extreme vertices i, j of l
8: **if** vertex i **and** vertex j have exactly one connection each in X **then**:
9: **if** l do not creates a inner-loop **then**:
10: **if** the ML agrees the addition of l **then**: $x_{i,j} = 1$
11: **else**
12: **if** vertex i **and** vertex j have less than two connections each in X **then**:
13: **if** vertex i **or** vertex j has zero connections in X **then**:
14: **if** the ML agrees the addition of l **then**: $x_{i,j} = 1$
15: find the hub vertex h
16: select all the edges that connects free vertices and insert them into L_D
17: compute the saving values with respect to h for each edge in L_D
18: sort L_D according to the descending savings $s_{i,j}$
19: $t = 0$
20: **while** the solution X is not complete **do**
21: $l = L_D[t], \quad t = t + 1$
22: select the extreme vertices i, j of l
23: **if** vertex i **and** vertex j have exactly one connection each in X **then**:
24: **if** l do not creates a inner-loop **then**: $x_{i,j} = 1$
25: **else**
26: **if** vertex i **and** vertex j have less than two connections each in X **then**:
27: **if** vertex i **or** vertex j has zero connections in X **then**: $x_{i,j} = 1$

3 The Original *ML-Constructive* Algorithm

The *ML-Constructive* is a constructive hybrid algorithm composed by two phases. The first phase exploits Machine Learning's ability in detecting specific patterns to create an initial partial solution (lines 1-14). The edges chosen in this are locked and will not be modified later. During the second phase the well-known Clarke-Wright heuristic is executed to complete the solution (lines 15-27).

The input of the algorithm is given by the network $G(V, E)$ whose TSP tour we want to find (V is the set of vertices and E the set of edges). The output is a feasible tour described by an X matrix with entries x_{ij}. The variable x_{ij} defines if the found route picks the edge that goes from vertex i to vertex j with $x_{ij} = 1$, if the route does not pick such edge then $x_{ij} = 0$.

Initially, the candidate list (CL) for each vertex in the instance is found. Promising edges (shortes two edge per CL) are inserted in the list L_P. The list is sorted according to the position in the candidate list and the cost values. All the edges that are the first nearest will be found first and sorted according to c_{ij}, then the second nearest and so on.

The Machine Learning insertion phase starts, after checking that the inserting edge complies with the TSP constraints (line $8, 9, 12$ and 13), the ML model is asked to confirm the insertion (line 10 and 14). Once the L_P list has been searched whole, the first phase ends and the algorithm concludes the tour by using the Clarke-Wright heuristics.

4 Additional Results on TSPLIB instances

As described in the paper, one of our contributions was to introduce a third local-search phase. With reference to Figure 1, the right image shows the final tour obtained after a 2-opt local search on the solution shown on the left. The red edges are confirmed by the ML decision-taker model, and locked in such a way that they cannot be modified later. Black edges are obtained through the Clarke-Wright heuristic, and the corrected by the 2-opt local search. On this instance this third phase has improved the tour length over 3%. Over the 54 TSPLIB instances, we had an average improvement of about 2.6 %, with a maximum of 11.15 %.

Overall, results in terms of gap from the optimal tour length are shown in Table 1. Corresponding times in seconds to execute the heuristic on such instances are presented in Table 2.

As it can be seen, the changes we made led to better performance with respect to the original ML-Constructive, apart from a few particular instances. The best ML model applied to the ML-Constructive heuristic, according to our experiments, is the SVM: on average it can do even better with respect to the original ResNet. Of course, times get a lot better since we do not need to create images. On average, we use about 1/4 of the time.

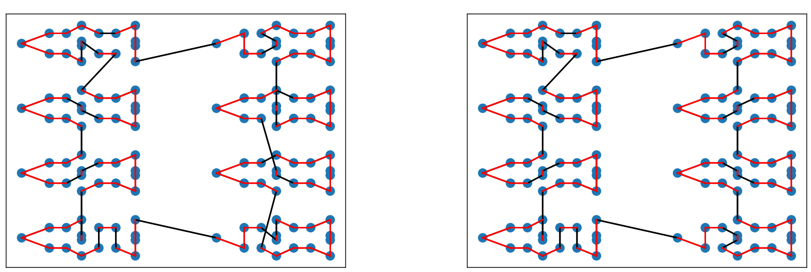

Fig. 1. TSPLIB pr136 instance solved by our ML-Constructive modified heuristic. On the left the instance solved by the first and the second phases, 4.435% above the optimal length. Red edges are locked by the first ML step. On the right side the previous solution enhanced by the 2-opt local search, 1.463% above the optimum.

Table 1. Gap from the optimal tour length computed on the results of the modified ML-Constructive heuristic executed on 54 TSPLIB instances. Each column refers to a different ML model decision-taker.

	B	**NN**	**Lin**	**SVM**	**ENS**	**ML-G**	**SVM + LS**	**OPT**	**OPT + LS**
kroA100	17,36	9,62	13,46	6,71	7,08	**6,48**	**3,41**	**3,79**	**0,47**
kroC100	9,56	8,36	5,24	7,25	**5,19**	10,34	6,34	**4,78**	**1,09**
rd100	15,66	11,58	**8,41**	9,61	9,61	8,56	**7,97**	**6,74**	**6,50**
eil101	12,72	8,90	4,45	4,77	6,20	**4,29**	**3,66**	**0,64**	**0,00**
lin105	14,27	**7,18**	8,18	8,46	14,01	8,49	**3,90**	8,78	8,78
pr107	9,02	**8,98**	10,62	9,79	10,96	11,15	**7,93**	**0,00**	**0,00**
pr124	16,70	**4,91**	13,02	5,69	13,56	6,99	**4,89**	**3,00**	**2,78**
bier127	9,55	4,50	4,28	**3,93**	6,33	6,60	**3,32**	**0,00**	**0,00**
ch130	10,62	7,41	7,01	5,34	5,58	**4,98**	**3,00**	7,77	**0,00**
pr136	13,23	13,28	13,81	**7,80**	10,90	11,15	**5,14**	**3,71**	**1,42**
gr137	11,52	9,74	8,71	**5,16**	5,56	7,33	**4,45**	**3,19**	**2,81**
pr144	5,85	6,63	5,48	5,79	7,26	**4,47**	**1,58**	**3,96**	2,05
kroA150	12,39	10,51	12,52	7,10	**6,73**	6,88	**6,50**	**1,14**	**0,58**
pr152	12,36	7,20	7,20	8,29	7,43	**6,92**	**2,76**	**3,79**	**2,28**
u159	10,12	8,87	12,48	9,26	17,25	**7,95**	**7,82**	8,14	**7,43**
rat195	12,53	7,58	**7,06**	9,00	7,49	7,53	**7,06**	**0,00**	**0,00**
d198	18,28	7,51	**5,09**	6,17	5,52	6,26	**4,92**	**4,01**	**2,64**
kroA200	10,73	11,10	10,86	9,24	7,29	**6,68**	**6,15**	**2,09**	**1,26**
gr202	13,30	7,37	6,01	4,75	5,50	**4,44**	**2,93**	**1,83**	**1,31**
ts225	10,51	8,92	**2,58**	6,52	6,52	6,52	6,40	11,33	11,33
tsp225	13,91	9,79	11,37	9,97	**9,45**	11,29	**5,21**	**4,40**	**0,00**
pr226	17,99	10,78	16,81	11,60	14,78	**8,60**	**8,15**	**5,37**	**3,94**
gr229	24,16	7,59	9,64	**6,00**	10,47	7,50	**4,81**	**2,81**	**1,45**
gil262	10,39	7,65	10,51	8,37	8,37	**6,22**	6,35	8,12	**5,64**
pr264	14,50	**4,92**	6,34	5,87	7,04	6,04	**3,25**	**3,74**	3,49
a280	17,10	13,61	13,61	12,68	12,68	**11,44**	**8,14**	**0,00**	**0,00**
pr299	15,80	9,17	8,31	**6,32**	11,67	8,64	**4,88**	**0,91**	**0,37**
lin318	9,80	**6,07**	7,81	8,10	8,26	6,68	**5,54**	**5,50**	**3,69**
rd400	10,33	8,19	9,98	**5,82**	7,37	8,11	**3,70**	**3,35**	**2,36**
fl417	11,75	11,04	10,51	10,26	11,19	**8,10**	**3,51**	**7,45**	3,52
gr431	13,55	**6,94**	11,28	8,06	9,72	11,55	**6,42**	**4,53**	**4,20**
pr439	13,86	9,02	8,86	7,99	10,09	**7,66**	**6,16**	**7,21**	**5,47**
pcb442	13,06	11,55	9,45	**7,31**	9,73	10,17	**4,90**	**2,79**	**1,78**
d493	9,07	8,54	9,45	7,10	7,24	**6,82**	**4,75**	**4,35**	**3,05**
att532	11,37	**7,60**	10,34	8,17	9,18	8,44	**5,81**	**3,83**	**2,00**
u574	11,04	8,47	9,36	**6,85**	7,02	9,79	**5,01**	**4,50**	**3,37**
rat575	11,31	10,32	8,28	8,34	8,46	**6,20**	6,38	**4,43**	**3,47**
d657	13,95	7,69	7,15	**6,94**	8,45	7,59	**5,05**	**3,83**	**2,31**
gr666	14,99	9,31	11,92	**8,46**	10,55	9,64	**6,74**	**6,79**	**5,86**
u724	10,61	7,38	10,06	7,79	8,25	**6,54**	**5,97**	**3,05**	**2,24**
rat783	9,92	7,33	9,28	8,35	7,06	**5,93**	6,41	**4,75**	**4,13**
pr1002	14,75	9,69	11,23	8,58	11,31	**8,53**	**5,44**	**5,36**	**3,39**
u1060	13,15	9,87	9,98	9,40	10,89	**8,95**	**7,28**	**6,70**	**5,36**
vm1084	12,19	9,62	10,96	9,80	11,04	**9,12**	**7,14**	**6,95**	**6,14**
pcb1173	11,20	10,02	11,34	10,16	**9,16**	9,99	**7,64**	**5,79**	**2,44**
d1291	8,81	**7,36**	9,40	7,38	7,66	8,54	**4,24**	**3,08**	**2,32**
rl1304	10,32	**7,49**	12,21	8,33	10,65	9,51	**6,52**	**5,23**	4,06
rl1323	11,32	7,11	7,75	8,12	9,95	**6,54**	6,56	**3,70**	**2,60**
nrw1379	11,35	8,13	9,78	8,47	8,19	**8,00**	**6,34**	**3,74**	**1,93**
fl1400	13,36	10,09	**8,84**	9,15	11,25	11,27	**3,70**	**8,47**	4,57
u1432	13,68	11,61	**8,29**	8,97	9,73	8,44	**6,60**	**5,73**	**4,10**
fl1577	14,59	11,83	13,02	11,23	11,53	**10,46**	**7,12**	**4,37**	**0,48**
d1655	11,92	10,13	11,10	10,91	10,21	**8,27**	**7,39**	**5,66**	**3,79**
vm1748	12,08	10,03	10,17	9,33	12,10	**9,30**	**7,07**	**6,40**	**3,78**
avg	12,66	8,82	9,46	7,98	9,20	8,03	5,56	4,47	2,96
std	2,99	1,95	2,75	1,84	2,57	1,87	1,61	2,45	2,35
best	0/54	9/54	6/54	10/54	4/54	25/54	48/54	49/54	47/54
time (sec)	0,932	1,909	3,286	2,605	5,559	9,822	631,665	0,483	36,632

Table 2. Time in seconds for the execution of the ML-Constructive heuristic on 54 TSPLIB instances. Each column refers to a different ML model decision-taker. Corresponding results are shown in Table 1.

	B	NN	Lin	SVM	ENS	ML-G	SVM + LS	OPT	OPT + LS
kroA100	0,116	0,060	0,108	0,140	0,353	1,423	0,331	0,030	0,042
kroC100	0,099	0,070	0,081	0,149	0,297	1,706	0,221	0,020	0,019
rd100	0,104	0,070	0,114	0,147	0,385	1,608	0,359	0,020	0,010
eil101	0,128	0,070	0,125	0,334	0,389	2,178	0,119	0,020	0,013
lin105	0,106	0,090	0,083	0,120	0,359	1,445	0,410	0,020	0,017
pr107	0,107	0,070	0,082	0,254	0,578	1,298	0,318	0,020	0,011
pr124	0,160	0,060	0,081	0,326	0,614	1,565	0,262	0,020	0,026
bier127	0,154	0,090	0,094	0,180	0,416	2,554	0,482	0,020	0,010
ch130	0,157	0,110	0,176	0,233	0,554	2,507	0,714	0,040	0,074
pr136	0,173	0,100	0,078	0,250	0,64	2,308	0,433	0,040	0,104
gr137	0,166	0,100	0,091	0,196	0,439	2,557	0,351	0,060	0,033
pr144	0,174	0,110	0,121	0,307	0,676	0,640	1,345	0,100	0,103
kroA150	0,189	0,120	0,225	0,249	0,612	2,444	0,455	0,080	0,044
pr152	0,156	0,070	0,132	0,293	0,648	1,047	1,973	0,040	0,025
u159	0,186	0,080	0,126	0,247	0,716	2,926	0,544	0,040	0,038
rat195	0,247	0,170	0,347	0,257	0,722	3,553	1,295	0,030	0,020
d198	0,253	0,140	0,338	0,335	0,88	3,451	2,137	0,080	0,096
kroA200	0,272	0,180	0,253	0,328	0,849	3,975	4,241	0,060	0,155
gr202	0,252	0,150	0,267	0,332	0,982	4,018	3,267	0,060	0,139
ts225	0,285	0,140	0,144	0,457	0,878	3,955	0,797	0,050	0,026
tsp225	0,263	0,190	0,432	0,513	1,025	4,239	4,129	0,050	0,264
pr226	0,279	0,140	0,225	0,564	0,943	1,102	4,056	0,060	0,140
gr229	0,291	0,210	0,218	0,483	0,716	4,220	3,248	0,080	0,268
gil262	0,384	0,300	0,626	0,444	1,084	4,640	6,979	0,110	0,788
pr264	0,311	0,210	0,332	0,379	0,87	3,815	2,416	0,050	0,060
a280	0,351	0,370	0,740	0,800	1,748	5,368	8,835	0,100	0,057
pr299	0,446	0,340	0,483	0,760	1,707	5,429	8,472	0,110	0,336
lin318	0,427	0,400	0,577	0,514	1,337	4,871	12,913	0,200	1,373
rd400	0,592	0,590	1,373	0,960	2,397	7,594	32,630	0,270	1,344
fl417	0,558	0,560	1,452	1,595	2,612	7,020	72,036	0,210	5,376
gr431	0,683	0,890	0,825	1,324	2,244	8,016	34,707	0,210	1,859
pr439	0,619	0,690	0,600	1,467	1,925	7,093	28,204	0,270	3,261
pcb442	0,571	0,570	1,024	1,154	3,049	8,526	27,599	0,170	1,109
d493	0,702	0,670	1,787	1,721	3,165	10,235	58,458	0,220	3,873
att532	0,896	1,100	2,309	1,681	3,905	10,010	83,534	0,330	6,760
u574	1,113	1,200	2,731	1,889	4,125	10,652	94,139	0,400	6,686
rat575	0,895	1,290	2,757	1,668	5,084	11,079	90,455	0,350	2,488
d657	1,173	1,960	2,857	1,837	4,155	12,054	190,197	0,660	8,385
gr666	1,114	1,850	1,484	1,947	4,218	12,513	129,885	0,680	4,761
u724	1,190	2,210	4,046	2,197	6,024	12,734	243,995	0,400	3,992
rat783	1,542	2,660	6,141	2,629	7,981	14,551	249,006	0,720	7,815
pr1002	2,377	4,140	2,744	3,108	7,689	20,139	1073,307	0,990	37,998
u1060	2,956	4,520	4,639	5,312	11,29	20,074	1083,990	1,300	76,138
vm1084	2,834	4,470	3,583	5,210	10,945	17,057	1435,532	1,330	39,049
pcb1173	2,690	4,910	9,647	5,049	15,92	22,952	1664,702	0,810	49,543
d1291	2,859	4,620	12,269	4,545	19,56	20,517	2006,655	1,120	16,069
rl1304	2,933	5,600	3,226	6,995	12,626	19,792	1244,524	1,110	57,660
rl1323	3,197	6,120	3,020	6,068	12,894	20,215	1044,760	1,790	71,006
nrw1379	3,964	8,170	15,027	9,343	21,142	29,439	2782,291	1,430	96,179
fl1400	4,293	7,930	15,218	15,166	23,025	30,401	3065,200	3,670	722,733
u1432	3,520	6,450	12,562	14,598	24,715	32,086	1432,400	1,260	62,316
fl1577	3,375	6,400	20,865	11,423	24,665	25,673	3512,533	0,800	61,743
d1655	4,678	7,960	24,215	7,546	23,375	30,257	5736,207	1,370	86,879
vm1748	5,897	11,330	14,355	14,631	20,017	28,875	6621,856	2,610	538,814
avg	1,175	1,909	3,286	2,605	5,559	9,822	631,665	0,483	36,632
std	1,440	2,768	5,599	3,900	7,520	9,280	1378,889	0,714	121,692