Netnomics manuscript No.
(will be inserted by the editor)

A Learning-based Variables Assignment Weighting Scheme for Heuristic and Exact Searching

Fan Xue · C.Y. Chan · W.H. Ip · C.F. Cheung

Received: date / Accepted: date

Abstract Many search algorithms have been successfully employed in combinatorial optimization in logistics practice. This paper presents an attempt to weight the variables assignments through supervised learning in subproblems. Heuristic and exact searching methods can therefore test promising solutions first. The Euclidean Traveling Salesman Problem (ETSP) is employed as an example to demonstrate the presented method. Analysis shows that the rules can be approximately learned from the training samples from the subproblems and the near optimal tours. Experimental results on large-scale local search tests and small-scale branch-and-bound tests validate the efficiency and effectiveness of the approach, especially when it is applied to industrial problems.

Keywords Supervised learning · Metaheuristics · Euclidean traveling salesman problem · Class association rules · Large-scale optimization

1 Introduction

The search for a "best" configuration of a set of variables to achieve certain goals plays a pivotal role in many important combinatorial optimization problems in logistics practice, such as vehicle routing, warehouse location decision, production planning and scheduling.

Many heuristic (including metaheuristic) and exact search algorithms have been developed, for example Laporte surveyed classes of algorithms for the vehicle routing problem [1]. Each search algorithm has its own strategy of exploiting the search space. The strategies can be categorized into different classes, such as random sampling, greedy (e.g., first-come-first-served), metaheuristic update (e.g., neighborhood

F. Xue · C.Y. Chan(†) · W.H. Ip · C.F. Cheung
Department of Industrial and Systems Engineering,
The Hong Kong Polytechnic University,
Hunghom, Kowloon, Hong Kong
Tel: +852 2766 4980
E-mail: mfcychan@inet.polyu.edu.hk

of local search, and mutation and crossover of genetic algorithm), systematic enumeration (e.g., branch-and-bound), exhaustive enumeration.

Apart from the algorithms aforementioned, machine learning techniques have received increasing attention. The supervised learning, a class of machine learning, aims at building a concise model of the distribution of class labels in terms of predictor features [2]. Some supervised learning techniques have been successfully adopted in logistics optimization applications, e.g. in scheduling [3,4,5,6], in production control [7] and in binpacking [8].

This paper presents a fast supervised-learning-based approach to determine *a priori* weights for the assignments of each variable. Based on the weights derived, many search algorithms can test promising solutions first.

The remainder of this paper is organized as follows. In Section 2, the variables assignment weighting approach is presented. In Section 3, the *Euclidean traveling salesman problem* (ETSP) is introduced as an example. The parameters of learning, the sizes of subproblems, and the complexity of the approach are discussed in Section 4. Section 5 presents numerical results and analysis of experiments of local search metaheuristics on large-scale ETSPs and those of branch-and-bound on small-scale ETSPs. Conclusions are finally given in Section 6.

2 The variables assignment weighting scheme

In the proposed weighting approach, the objective is to advise assignments for all variables, while the learning is only carried out on a subset of variables in a subproblem. It is assumed that every variable is "homogeneous," or obeying the same patterns of decision making. The assumption is naturally held in many optimization problems, such as vehicle routing and location decision. In fact the assumption may also enable learning from near optimal solutions, especially if most variable assignments in the optimal solutions can be found in the near optimal solutions.

The proposed method consists of four phases, as shown in Fig. 1:

1. Start with a problem P,
2. Learn pattern of decision making from a small *representative* subproblem P^* and its solution S^*,
3. Interpret the patterns to weights (or weight-based sorting or interchanges) of possible assignments of the variables,
4. Reform the assignment process of a heuristic or exact search algorithm h in the problem P.

The pattern of decision making, the key of this method, can usually be represented in the forms of rules or statistical values. In this paper, the *class association rules* (CARs) [9,10] are employed to represent the pattern. The learning results are therefore a set of CARs. Each CAR is in the form of "$A \Rightarrow c$," where c denotes a class attribute related to the variables assignments, $A \subset \mathscr{A}$ stands for some decision attributes, and $\mathscr{A}$ denotes all decision attributes. There are two typical parameters for the CAR learning: the minimum confidence and the minimum support. The two parameters can be easily determined in this paper: they are set to very low, e.g., minimum confidence $= 0.01$, minimum support $= 0.001$ for a set of ten thousand training

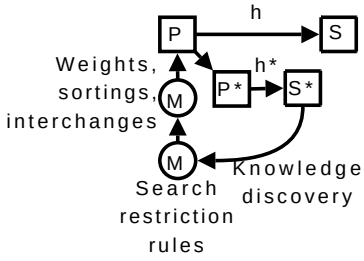


Fig. 1 The variables assignment weighting scheme

samples. It is because the rules with different confidences and different supports can be potentially useful to weight and order the assignments.

The learning phase is carried out in a *representative* subproblem P^* and its solution S^*. The representativeness denotes the similarity between the CARs learned from P^* and S^* and those learned from the given problem P. The algorithm h^* that solves P^* can be certain effective exact or heuristic algorithm, such as *linear programming*. If the size of P^* is much smaller than the size of original problem, the time cost of solving P^* and learning can usually be very low.

The learned CARs can be interpreted to different forms of weights compatible with different search algorithm h. Examples are:

- Numeric weights for greedy assignments (see dispatching rules),
- Sorting for tests of a local search heuristic, and
- Grouping for a rank-based constructive heuristic.

In the first example of interpretations, the top weighted possible assignment of each variable can be chosen greedily. It is very close to learning dispatching rules, see [3,4,5,6]. Weights in some other cases, e.g. the the second example, can be involved in more sophisticated search algorithms.

Besides the idea of weighting assignments, other researchers have proposed different cooperation plans between learning and search algorithms, e.g. a general framework of guiding constructive search with learning [11], employing genetic procedures to control neighborhood functions[1], and the adaptive combinatorial search for complex problems[2].

3 The Euclidean traveling salesman problem

The *traveling salesman problem* (TSP) aims at finding the shortest possible tour in which each given city is visited exactly once. This problem is one of the most comprehensively studied problems in operational research, theoretical computer science, and industrial engineering. The *Euclidean* TSP (ETSP), in which the distance between two cities is the Euclidean distance, is the most common form of the TSP.

[1] See `http://www.info.univangers.fr/pub/saubion/PHD/PHddetails.htm`.

[2] See `http://www.msr-inria.inria.fr/Projects/adaptive-comb-search`.

The industrial applications of the ETSP include vehicle routing, electric power cable networks, and VLSI (Very-Large-Scale Integration) chip fabrication. [12, 13].

The ETSP is known in $\mathscr{NP}$-complete (*Non-deterministic Polynomial time* complete) class [14]. Nevertheless, as a result of decades of research, the ETSP, can be viewed as well-handled in practice. On one hand, codes that find provably optimal tours are now capable of handling relatively large instances, e.g., the `Concorde` code solved the 85,900 cities in 136 CPU-years using 2.4 Ghz processors [13]. On the other hand, fast heuristics typically can quickly find tours within a few percent of optimal on real-world instances [15], and more sophisticated heuristics can get within 1% of optima in reasonable amounts of time [16, 17].

It is believed that one of the major reasons for the efficiencies of the heuristics, especially local search, is the high effectiveness of the edge candidate set structures. The main concern of the candidate set is that it is not possible for most of very long edges to appear in optimal tours. So a candidate set usually suggests a limited number of *promising* edges for each city according to the geometry. Popular candidate sets includes the *Nearest-neighbor-first* [18], the *Delaunay* [19], the *k-quadrant* [20], and the α-nearness [16].

The following show how the presented variables assignment weighting scheme can be employed in the ETSP.

3.1 Learning from subproblems

According to experiences and tests, the representativeness of a subproblem P^* is highly associated with its density of cities. The density of the subproblem should be close to that of the given ETSP in order to find the closet rules. It is not difficult to find such a subproblem with a known density from a given ETSP. Linear time algorithms for finding such subproblems can be easily developed.

The size (number of cities) of the subproblem is another parameter to determine. In this paper the size is set to 3,000 (cities) according tests, see Subsection 4.2 for details.

When the subproblem is determined, it can be solved by effective algorithms. In this paper the algorithm is the Bentley's Greedy heuristic tour construction [21] followed by a 5-Opt local search by the LKH[3] [16] over an α-nearness candidate set. The parameters of the local search are set to the suggested values in literature [16, 22]. The trials of the local search is set to 300 and the number of runs is set to 1. The solution S^* can be typically found in tens of seconds for a 3,000 ETSP and the tour quality is typically within 1% over the optimum.

Decisions of assigning variables in the ETSP can be viewed as: "what kind of (short) edges can outperform other competitors departing from one city." Fourteen *prima-facie* geometric decision attributes and one label are selected for the CAR learning, as listed in Table 1. Without loss of generality, all the edge candidates departing from a city c_i are put in an ordered set:

$$\mathscr{N}_{c_i} = \left\{ \langle c_i, n_1 \rangle, \langle c_i, n_2 \rangle, \ldots, \langle c_i, n_{k_i} \rangle \right\}, \tag{1}$$

[3] Available at:`http://www.akira.ruc.dk/~keld/research/LKH/`, Version 2.0.3.

where the distances obey $d(c_i,n_1) \le d(c_i,n_2) \le \cdots \le d(c_i,n_{k_i})$. The asymmetry should be noted: an edge $\langle c_i,c_j \rangle$ may be contained in $\mathscr{N}_{c_i}$ and out of $\mathscr{N}_{c_j}$.

Table 1 Definitions of attributes and label for the edge-selection function for edge $\langle c_i,n_j \rangle$

Id	Type	Definition	Remarks
$G1$	Boolean	$G1=1$, iff. $\underset{c_x \in \mathscr{C}}{\forall} \underset{c_y \in \mathscr{C}-\{c_x\}}{\forall} d(c_i,n_j) \le d(c_x,c_y)$	Globally nearest
		$G1=0$, otherwise	
$R1$	Integer	$R1 = LenIdx(c_i,n_j,n_1)$ †	Length index against n_1 for c_i
$R2$	Integer	$R2 = LenIdx(c_i,n_j,n_2)$	Length index against n_2 for c_i
$R3$	Integer	$R3 = LenIdx(c_i,n_j,n_3)$	Length index against n_3 for c_i
$S1$	Boolean	$S1=1$, iff. $d(c_i,n_1) \le d(c_i,n_2)/2$	n_1 is significant closer than n_2 for c_i
		$S1=0$, otherwise	
$S2$	Boolean	$S2=1$, iff. $d(c_i,n_2) \le d(c_i,n_3)/2$	n_2 is significant closer than n_3 for c_i
		$S2=0$, otherwise	
$P1$	Integer	$P1 = LenIdx(n_j,c_i,m_1)$, iff. $c_i \in \mathscr{N}_{n_j}$	Length index against m_1 for n_j
		$P1=-1$, otherwise	
$P2$	Integer	$P2 = LenIdx(n_j,c_i,m_2)$, iff. $c_i \in \mathscr{N}_{n_j}$	Length index against m_2 for n_j
		$P1=-1$, otherwise	
$P3$	Integer	$P3 = LenIdx(n_j,c_i,m_3)$, iff. $c_i \in \mathscr{N}_{n_j}$	Length index against m_3 for n_j
		$P1=-1$, otherwise	
$Q1$	Boolean	$Q1=1$, iff. $d(n_j,m_1) \le d(n_j,m_2)/2$	m_1 is significant closer than m_2 for n_j
		$Q1=0$, otherwise	
$Q2$	Boolean	$Q2=1$, iff. $d(n_j,m_2) \le d(n_j,m_3)/2$	m_2 is significant closer than m_3 for n_j
		$Q2=0$, otherwise	
Ag	Integer	$Ag = round(MinDirGap(c_i,\mathscr{N}_{c_i}) \times 15/\pi)$ ‡	Minimal angular gap around c_i
Ah	Integer	$Ah = round(MaxDirGap(c_i,\mathscr{N}_{c_i}) \times 15/\pi)$	Maximal angular gap around c_i
An	Integer	$An = NumOfDir(c_i,\mathscr{N}_{c_i})$ *	Number of directions around c_i
Opt	Boolean	$Opt=1$, iff. $\langle c_i,n_j \rangle$ is in the given tour	Appearing in the training sample
		$Opt=0$, otherwise	

† : $LenIdx(c_x,c_y,c_z) = \min(\lfloor d(c_x,c_y)/d(c_x,c_z) \times 3 \rfloor, 10)$, returns an index of relative length,
‡ : *MinDirGap* (*MaxDirGap*) returns the minimal (maximal) gap of included angles among the edges candidates, in a resolution of $\pi/15$;
* : *NumOfDir* returns how many directions (in a resolution of $\pi/15$) the edges distribute, if they are considered as vectors from c_i.

Seven attributes ($G1$, $R1$, $R2$, $R3$, $P1$, $P2$, $P3$) are length indices which denote the competitiveness on distance. The rest decision attributes stand for some geometric features of the departure city and destination city, for example $Ah = \pi$ means the departure city is on some side of the map. The label Opt stands for whether the edge is a part of the tour S^* or not. The attributes are applicable for all the ETSPs, and some of them are compatible to general TSPs. However it should be noted that they may not always maintain the best distinguishability in every problem. For example in printed circuit problems, attributes such as the "same y coordinate" might be an optional attribute.

Supervised training examples can be populated according to the attributes and the subproblem. *Apriori* [9,23], a well known CAR learning algorithm, can obtain rules from the training examples. The Apriori in this paper is partially based on implementation by Borgelt[4] [24]. The CARs in the form of "$A \Rightarrow Opt$=1" are employed to represent the pattern. The CARs implying "Opt=0" are omitted, because they are complementary. A few types of redundancy reductions were implemented, for example, if a rule is "$A,B \Rightarrow Opt$=1" and another rule is "$A \Rightarrow Opt$=1," and the confidence

[4] Available at `http://www.borgelt.net/apriori.html`, version 5.8.

of the first rule is equal or less than that of the latter; the first rule will be omitted as a useless (dominated) rule in this paper. The size of the body (left part of a rule) is also limited to a constant $|A|_{\max}$, so a CAR "$A \Rightarrow c$" will be omitted if the cardinality $|A| > |A|_{\max}$. More tests and analysis of $|A|_{\max}$ can be found in Subsection 4.1.

Table 2 shows a set of examples of the discovered CARs. For instance, the No. 30 rule in Table 2 stands for: "globally shortest edges should be in the optimal tour(s)" and it has a confidence of 0.913. This is what the *Greedy* heuristic follows first. In fact more rules relevant to conventional heuristics can be found in the full version of such a table.

Table 2 Samples of discovered CARs

Id	Rule	Support	Confidence
1	$R1{=}3, S1{=}1, Q1{=}1 \Rightarrow Opt{=}1$	0.013	1.000
2	$P1{=}3, S1{=}1, Q1{=}1 \Rightarrow Opt{=}1$	0.013	1.000
3	$R1{=}3, S1{=}1, Q2{=}0 \Rightarrow Opt{=}1$	0.012	1.000
⋮	⋮	⋮	⋮
30	$G1{=}1 \Rightarrow Opt{=}1$	0.022	0.913
⋮	⋮	⋮	⋮
983	$R3{=}8 \Rightarrow Opt{=}1$	0.048	0.010

3.2 Interpreting the learning results

Each learned CAR carries a confidence and covers certain edge candidates. For each edge candidate $\langle c_i, c_j \rangle$, a promising possibility $\tilde{p}_{ij}$ can be set to the highest confidence from the rules covering the edge. The probabilities of the excluded edges are set to 0. It should be noted that the possibilities are asymmetric, e.g., a promising possibility judged on behalf of $\langle c_i, c_j \rangle$ does not necessarily the same as that of $\langle c_j, c_i \rangle$. In practice, the promising possibility can be set to the greater one between the two for convenience.

For each city, only two edges can be chosen out of a set of edge candidates. However there usually are a lot of non-zero possibilities. It means every edge with a non-zero probability has a chance to be a part of the optimal tour(s), while every edge with a less-than-one probability has a chance to fail.

The conventional tour construction methods for the ETSP hardly make use of probability. So one way of taking advantage of the probabilities is to transform them into different weights, such as:

- Setting the weight to the promising possibility,
- *Weighted distance* (WD) = Euclidean distance $\times$ (1$-$ promising possibility), and
- Ranking candidates to Level A, B, and C.

In this paper, the WD is studied. The WD can be a substitution of the Euclidean distance in many heuristic construction and test procedures. Some search algorithms, such as the *nearest-neighbor insertion* and the *shortest fragment merge*, could directly

work on the WD; while others such as the local search could indirectly be benefited, e.g., from a reformed testing neighborhood by the WD. There are some search techniques that are not directly based on the Euclidean distance, such as the α-nearness candidate set which is based on the α-nearness values. Pseudo-distances can be introduced in such cases to facilitate the presented method, for example a transform for the non-negative α-values:

$$pseudo\_distance = \ln(\alpha\text{-}value + 1). \tag{2}$$

4 Parameters and time cost analysis

The parameters of the presented method were tested and discussed in this section. Two indicators are used to measure the capability of the ordered edge candidates in including the optimal tour's edges (OEs). The *minimal population of candidates* ($Cand_{\min}$) stands for the least number of edge candidates to fully cover the $2N$ OEs. The *average index of OEs* (IOE_{avg}) denotes the average position of the OEs. The two indicators are not independent, but the first one emphasizes the range of coverage, and the second stresses average depth. Smaller values stand for better capability for both indicators. Their lower bounds can easily be obtained as $2N$ and 1.5 respectively.

4.1 Parameters of CAR learning

The CAR learning in this paper involves three parameters: the maximal cardinality of condition attributes ($|A|_{\max}$ for CARs in form of $A \Rightarrow c$), the minimum support (σ), and the minimum confidence (γ). The $|A|_{\max}$ is relatively independent, whereas σ and γ are correlated.

The maximal cardinality of condition attributes $|A|_{\max}$ was tested on two benchmark problems `E10k.0` and `xmc10150`. The first problem, from the DIMACS TSP Challenge benchmark data sets[5], consists of 10,000 random cities; the second one comes from the VLSI data set[6] and contains 10,150 cities. In the tests, σ was temporarily set to 0.001 and γ was temporarily set to 0.01. The sizes of the subproblems were set to the same sizes as the given problems. The input tours were two near optimal tours with lengths of 71,867,264 (0.002% excess best known) and 28,387 (best known) respectively. The learning were based on 50 quadrant edge candidates (50-Quad). Four indicators, including the number of discovered CARs, learning time, IOE_{avg} and $Cand_{\min}$, were considered to determine an appropriate cardinality for large-scale ETSPs, as shown in Figure 2.

Figure 2 seems to suggest $|A|_{\max}$=5 as an appropriate value. When $|A|_{\max} > 5$, despite the increasing amount of the learning time, the rest indicators seem not to be significantly changed. It could be hoped that if the learning time is sufficient, $|A|_{\max}$=6 can also be a possible alternative.

[5] Available at: `http://www.research.att.com/~dsj/chtsp/`.

[6] Available at: `http://www.tsp.gatech.edu/vlsi/`.

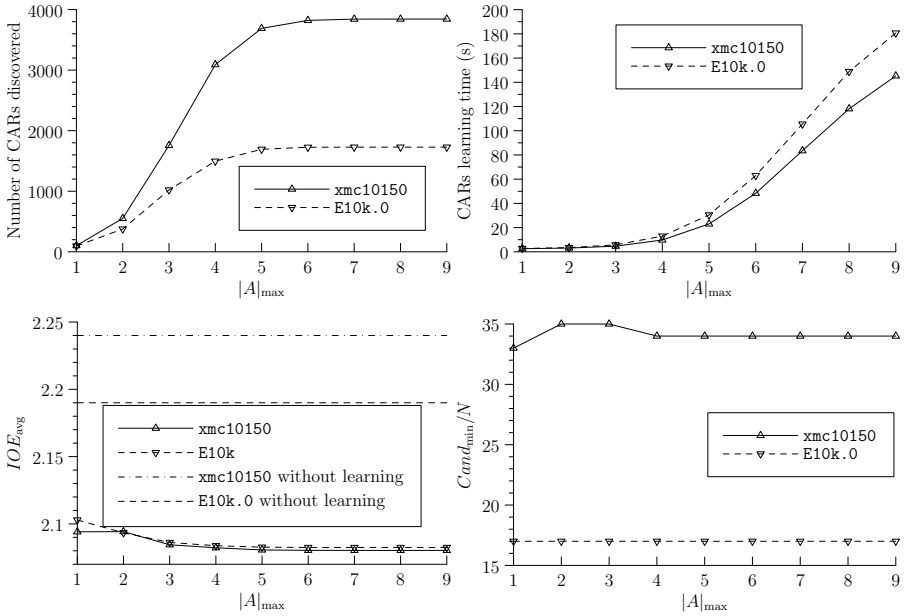


Fig. 2 Trends of the indicators against the maximal cardinality of condition attributes ($|A|_{\max}$)

Thresholds σ and γ were also tested on the same base candidate sets on the two problems, where $|A|_{\max}$=5. The main objective of the CAR learning is to harbor more CARs and cover more OEs. So the main performance indicator of the CARs was the coverage of OEs. The percentage trends of the covered OEs can be seen in Figure 3 against the combination of the values of σ and γ.

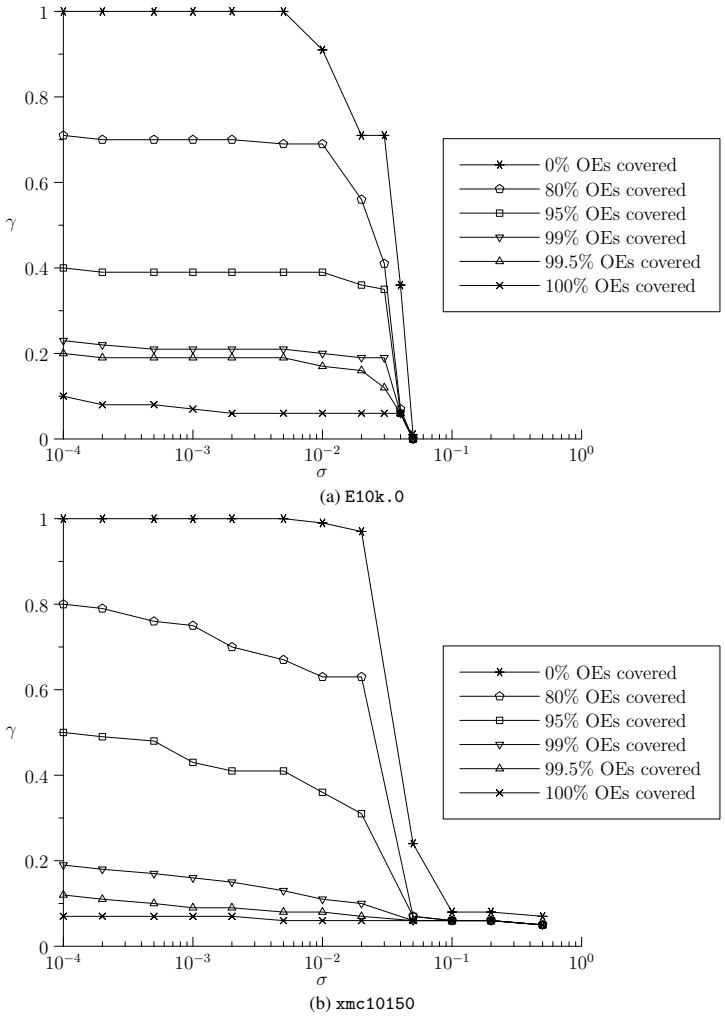
Figure 3 shows that σ should be less than 0.05 to approximately fully cover the OEs in the two problems. Further when $\sigma \leq 0.01$, the majority (about 80%) of OEs can be covered by the rules with high confidence (approximately 0.7), as shown as the area between the 0% coverage curve and the 80% coverage curve in Figure 3. So 0.01 seemed an upper limit for σ in practice. However, a too small value of σ may bring overfitting of the CARs. So another lower bound N^{-1} is suggested for σ, where N is the number of the cities. In practice, σ can be set to 0.001 if $N \geq 1,000$, or to N^{-1} if $N < 1,000$.

The minimum confidence γ is easier to determine. According to Figure 3, γ should be no more than about 0.07 to cover all the OEs. In practice, it is suggested to be 0.01.

4.2 Sizes of subproblems

The sizes of the subproblems were tested on the `E10k.0` and the `xmc10150`. The possible edge candidates are set to the 50 quadrant candidates. Figure 4 shows the changes of the indicators when different sizes (N′) of subproblems were selected.

It can be determined that the CARs were not abundantly discovered when $N' < 1,000$, where N' was the number of cities in a subproblem. It is mainly because of



(a) `E10k.0`

(b) `xmc10150`

Fig. 3 OEs covered in combinations of different σ and γ

the increasing threshold N'^{-1} when N' decreases from 1,000. In general, the overall time cost increases steadily when N' increases. According to experience, the time for finding a near optimal tour usually took an increasingly large share of the overall time cost.

The charts also show that IOE_{avg} and $Cand_{\text{min}}$ curves gradually become flat when N' increases. It means the WD obtained from the middle sizes of subproblems are not much different from those from the complete problems. Therefore it is feasible to find discover the patterns of large-scale ETSPs from their subproblems, without significant losses of effectiveness. The two indicators IOE_{avg} and $Cand_{\text{min}}$ seems not much affected by the size of the subproblem when $N' > 100$.

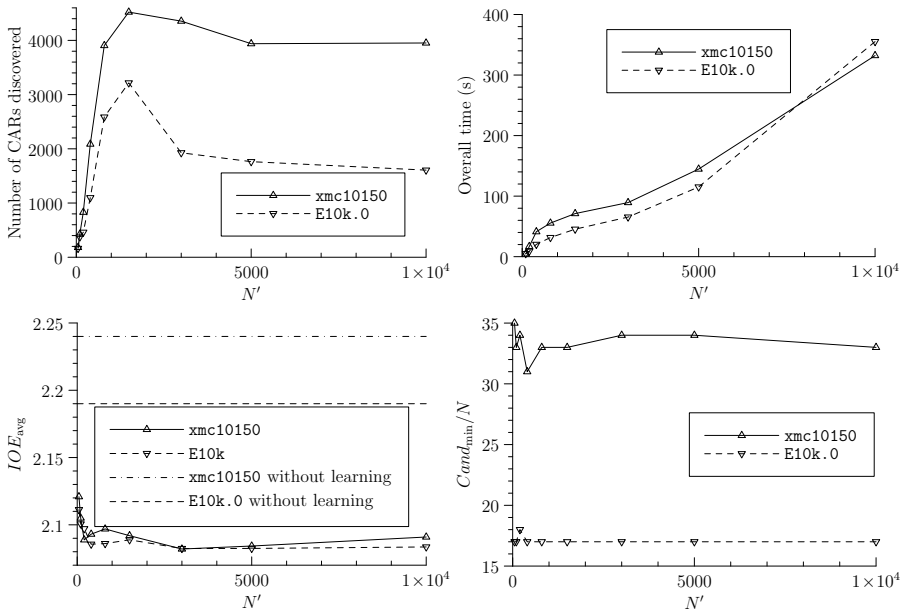


Fig. 4 Trends of indicators against the size of the subproblem (N')

Experience shows the trends are general in large-scale ETSPs and almost independent from the size N of the original problems. So an appropriate constant, e.g. $N' = 3,000$, can be determined as a general size for subproblems when solving large-scale ETSPs. Therefore one can solve a constant size of a subproblem to provide metric weights for a given large-scale ETSP.

4.3 Expected time cost

If the size of the subproblem is a constant, the time cost of the presented method can be bounded in time $O(Nk_{\max} \log k_{\max})$, where N is the size of the given ETSP, and $k_{\max}$ is the max number of edge candidates for one city. The upper bounds of the running time of the sub procedures are listed in Table 3 in detail. The most expensive sub procedure in this approach is weighting and reordering the edge candidate sets.

5 Experimental results

Several popular Euclidean data sets were selected as the sources of the test problems, including the VLSI industrial data sets, the Euclidean random (E) data sets from the DIMACS TSP Challenge, and the TSPLIB data sets[7] from various industries and geographic maps. A number of ETSPs from the data sets were selected and clustered

[7] Available at: `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`.

Table 3 Upper bounds of the running time of the sub procedures if the size of the subproblem is a constant

No Procedure	Upper bound of time
1 Finding a subproblem	$O(N)$
2 Solving the subproblem	$O(1)$
3 Generating training examples	$O(1)$
4 Learning CARs	$O(1)$
5 Examining promising probabilities	$O(Nk_{\max})$
6 Weighting and reordering edge candidates	$O(Nk_{\max}\log k_{\max})$
7 Reducing the sizes of the edge candidates (optional)	$O(N)$
Overall	$O(Nk_{\max}\log k_{\max})$

by their size into different categories: 3k, 10k, …, 1M, as shown in Table 4. The "BKS" stands for the best known solution. Some BKSs of the E problems came from Helsgaun's results[8]. Using BKSs instead of Held-Karp lower bounds in this paper mainly aims at a comparable metric of heuristics improvement over different data sets.

Table 4 The selected large-scale ETSPs for experiments

Category	VLSI (BKS)	E (BKS)		TSPLIB (Optimum)
3k	lsn3119 (9114*)	E3k.0 (40634081*)	E3k.1 (40315287*)	pr2392 (378032)
	lta3140 (9517*)	E3k.2 (40303394*)	E3k.3 (40589659*)	pcb3038 (137694)
	fdp3256 (10008*)	E3k.4 (40757209)		fnl4461 (182566)
10k	dga9698 (27724)	E10k.0 (71865826)	E10k.1 (72031630)	pla7397 (23260728)
	xmc10150 (28387)	E10k.2 (71822483)		brd14051 (469385)
31k	pbh30440 (88328)	E31k.0 (71865826)		pla33810 (66048945)
	xib32892 (96757)	E31k.1 (72031630)		
100k	sra104815 (251433)	E100k.0 (225787421)	E100k.1 (225659006)	pla85900 (142382641)
316k	ara238025 (578775)	E316k.0 (401307462)		—
	lra498378 (2168067)			
1M	lrb744710 (1612132)	E1M.0 (713189834)		—

*: Also proved optimal.

The tests were conducted on an Intel Core2 Duo E6750 (2.66GHz) CPU on a Ubuntu Linux 8.04.1 (Kernel 2.6.24-19, 64-bit) with 4 Gbytes of memory. A method of normalization of running time was described by Johnson, et al.[17], so that results on any machine could be compared with other published results. A set of tests of the CPU speed were carried out, in comparison to a 500 MHz EV6 Compaq Alpha processor, as shown in Table 5. In the following, all the running time are normalized to a 500 MHz Alpha.

Table 5 Results of running time (in seconds) of the Greedy [17] heuristic on different CPUs

CPU	3k×316	10k×100	31k×32	100k×10	316k×3	1M×1
Intel Core2 Duo E6750 (2.66GHz)	2.735	2.809	3.330	5.466	8.187	11.247
Compaq ES40 Alpha (500 MHz)	14	16	23	36	55	88

[8] Available at: http://www.akira.ruc.dk/~keld/research/LKH/DIMACS_results.html.

The tests were conducted on different candidate sets: 5 nearest neighbors (5-NN), 5 quadrant nearest candidates (5-Quad), 5 α-nearest candidates (5-α), and edge candidates by weighted distances 5-WD_{NN}, 5-WD_{Q}, and 5-WD_{α}. The later three candidate sets were weighted, reordered and trimmed by the presented approach on some larger candidate sets in the subproblems, i.e. the 50-NN, the 50-Quad, and the 50-α candidate sets, respectively. In the learning and reordering phases, the parameters were: $N' = 3,000$, $|A|_{\max}$=5, σ=0.001, γ=0.01.

The normalized running time of the preparations of edge candidates on different categories of E problems are shown in Figure 5. because the curves were very close to other data sets. It can be observed that the time cost increased as expected. The faction of the presented approach kept decreasing when the size of the problem increased, especially for the α-nearness candidate set.

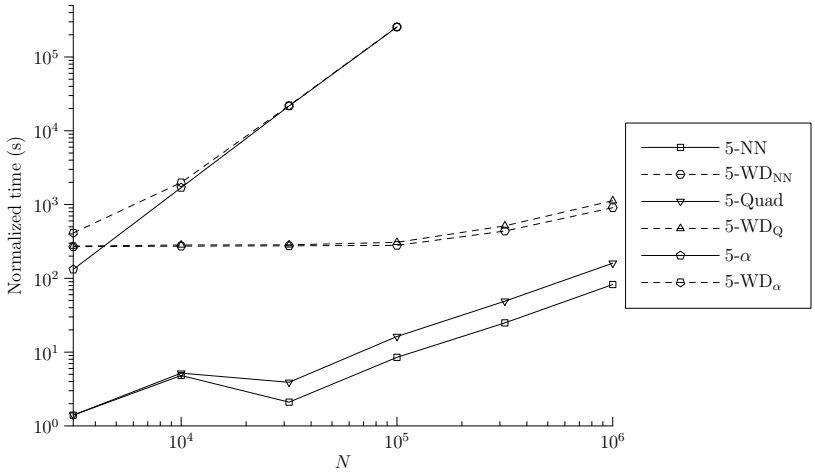


Fig. 5 Typical normalized running time of the presented approach and well known candidate sets

Two local search algorithms were employed to examine the presented method. One algorithm was the Bentley's Greedy heuristic tour construction followed by a 5-Opt local search of LKH, Greedy+5-Opt in short. The other was the Greedy followed by a 2-Opt (Greedy+2-Opt), a simplified version. Each algorithm ran 100 times independently for each problem. The average tour qualities and the running time of the algorithms are grouped by the categories, as shown in Tables 6, 7, 8, and 9 respectively. The "Imp" columns in the tables stand for the significance (in percentage) of the tour length (or time) improvements.

It can be seen from Table 6 and Table 7 that the industrial ETSPs benefit more from the metric weights, and the Greedy+5-Opt benefit more that Greedy+2-Opt from the metric weights, except for those on the nearest neighbor candidate sets. A possible reason could be that more nontrivial information can be found in the industrial problems than in random problems. Another possible reason can be that the redressed and reordered edge candidates are more compatible with stronger search algorithms.

Table 6 Results of average tour qualities of the Greedy+5-Opt algorithm on different candidate sets (100 runs)

Data set	Category	Average percentage excess over BKS or optimum								
		5-NN	5-WD_{NN}	Imp (%)	5-Quad	5-WD_Q	Imp (%)	5-α	5-WD_α	Imp (%)
VLSI	3k	3.889	2.663	31.5	0.695	0.649	6.7	0.361	0.327	9.3
	10k	4.236	3.300	22.1	0.863	0.693	19.7	0.526	0.503	4.5
	31k	4.169	2.913	30.1	0.814	0.642	21.2	0.454	0.437	3.7
	100k	6.657	6.467	2.9	0.842	0.752	10.7	0.339	0.328	3.2
	316k	9.959	7.950	20.2	1.183	0.917	22.5	-	-	-
	1M	4.682	4.385	6.3	0.857	0.762	11.1	-	-	-
E	3k	0.703	0.487	30.7	0.346	0.338	2.3	0.156	0.156	0.3
	10k	0.862	0.490	43.1	0.375	0.370	1.4	0.179	0.178	0.2
	31k	1.262	0.659	47.8	0.527	0.526	0.2	0.343	0.341	0.6
	100k	1.851	0.646	65.1	0.438	0.434	0.9	0.252	0.250	0.8
	316k	1.660	0.679	59.1	0.43	0.422	1.9	-	-	-
	1M	1.176	0.911	22.5	0.381	0.379	0.5	-	-	-
TSPLIB	3k	0.456	0.358	21.4	0.34	0.321	5.4	0.143	0.134	6.5
	10k	2.878	2.234	22.4	0.427	0.395	7.6	0.253	0.278	-10.1
	31k	2.297	1.677	27.0	0.913	0.517	43.4	0.560	0.617	-10.2
	100k	2.065	1.476	28.5	0.761	0.445	41.5	0.932	0.978	-4.9

Table 7 Results of average tour qualities of the Greedy+2-Opt algorithm on different candidate sets (100 runs)

Data set	Category	Average percentage excess over BKS or optimum								
		5-NN	5-WD_{NN}	Imp (%)	5-Quad	5-WD_Q	Imp (%)	5-α	5-WD_α	Imp (%)
VLSI	3k	5.196	4.234	18.5	2.177	2.019	7.3	1.376	1.625	-18.1
	10k	5.949	4.943	16.9	2.716	2.144	21.1	2.101	1.956	6.9
	31k	5.660	4.221	25.4	2.421	2.162	10.7	1.675	2.052	-22.5
	100k	8.101	7.945	1.9	2.472	2.344	5.2	1.244	2.006	-61.3
	316k	11.503	4.942	57.0	3.004	2.746	8.6	-	-	-
	1M	6.125	5.710	6.8	2.505	2.380	5.0	-	-	-
E	3k	2.250	1.616	28.2	1.412	1.645	-16.5	0.791	0.996	-25.8
	10k	1.849	1.575	14.8	1.439	1.495	-3.8	0.756	1.113	-47.3
	31k	2.007	1.648	17.9	1.604	1.678	-4.6	0.881	1.314	-49.1
	100k	2.320	1.611	30.6	1.553	1.550	0.2	0.791	1.237	-56.5
	316k	2.764	2.370	14.3	2.154	2.179	-1.2	-	-	-
	1M	2.235	1.884	15.7	1.452	1.460	-0.6	-	-	-
TSPLIB	3k	1.735	1.470	15.3	1.554	1.433	7.8	0.793	0.751	5.3
	10k	3.832	3.371	12.0	1.817	2.040	-12.3	1.177	1.485	-26.1
	31k	3.176	2.610	17.8	2.469	2.232	9.6	1.694	1.914	-13.0
	100k	3.017	2.591	14.1	2.211	2.022	8.5	1.589	1.978	-24.5

It can be observed that the presented approach was very competitive in the 316k and 1M groups of problems, where the α-nearness candidate set cost too much time.

According to Table 8, the running time of the Greedy+5-Opt algorithm on the 5-WD_Q candidate sets was significantly shorter in industrial problems than those on the 5-Quad, as well as those on the 5-α, while the 5-WD_Q and the 5-Quad seemed not have significant differences in the random problems. However the same running time of the Greedy+2-Opt algorithm seemed not so competitive.

Another algorithm tested was a simple exact branch-and-bound search. In this algorithm, a string of cities were gradually tested and appended. Iterated minimum spanning 1 trees were examined at each branch to obtain a lower bound of the tours that contains the string. If the lower bound was no less than the best tour length which had been obtained, the branch would not be expanded for further search.

Table 8 Average normalized time cost of the Greedy+5-Opt on different candidate sets (100 runs)

Data set	Category	Average normalized time cost (s)								
		5-NN	5-WD_{NN}	Imp (%)	5-Quad	5-WD_Q	Imp (%)	5-α	5-WD_α	Imp (%)
VLSI	3k	2.20	2.27	-3.1	1.93	1.48	23.0	2.32	2.21	4.6
	10k	8.09	7.84	3.2	8.72	6.64	23.9	10.32	9.69	6.1
	31k	33.20	31.79	4.3	37.66	29.40	21.9	42.88	46.18	-7.7
	100k	88.57	86.00	2.9	147.62	133.05	9.9	158.95	169.7	-6.8
	316k	479.84	421.65	12.1	675.39	649.52	3.8	-	-	-
	1M	1123.66	949.97	15.5	1665.11	1500.81	9.9	-	-	-
E	3k	2.60	2.47	5.1	1.68	1.87	-11.6	1.98	2.08	-5.3
	10k	9.86	10.28	-4.2	7.94	7.56	4.8	8.91	8.56	3.9
	31k	41.81	45.40	-8.6	37.18	36.69	1.3	47.20	43.73	7.4
	100k	140.30	156.68	-11.7	141.62	139.84	1.3	161.85	167.15	-3.3
	316k	503.66	568.11	-12.8	601.57	596.53	0.8	-	-	-
	1M	2141.66	2432.96	-13.6	2986.15	3033.61	-1.6	-	-	-
TSPLIB	3k	2.71	2.68	1.3	2.18	1.84	15.6	1.88	4.07	-116.7
	10k	12.88	13.57	-5.3	12.60	11.17	11.3	13.40	13.57	-1.3
	31k	97.50	97.02	0.5	81.40	65.16	19.9	84.44	97.27	-15.2
	100k	149.99	159.61	-6.4	174.31	166.20	4.7	134.96	150.73	-11.7

Table 9 Average normalized time cost of the Greedy+2-Opt on different candidate sets (100 runs)

Data set	Category	Average normalized time cost (s)								
		5-NN	5-WD_{NN}	Imp (%)	5-Quad	5-WD_Q	Imp (%)	5-α	5-WD_α	Imp (%)
VLSI	3k	0.30	0.30	0	0.28	0.30	-7.1	0.24	0.30	-25.0
	10k	1.42	1.36	4.1	1.16	1.33	-15.0	1.10	1.27	-15.8
	31k	9.21	7.33	20.4	6.97	7.37	-5.7	6.21	4.27	31.3
	100k	32.23	31.02	3.8	26.58	26.58	0	22.74	28.06	-23.4
	316k	170.32	156.59	8.1	137.81	148.35	-7.6	-	-	-
	1M	460.39	341.91	25.7	275.1	297.95	-8.3	-	-	-
E	3k	0.48	0.57	-20.0	0.53	0.72	-36.4	0.31	0.35	-11.5
	10k	2.53	2.72	-7.6	2.70	2.91	-7.9	1.72	2.08	-21.3
	31k	12.20	12.38	-1.5	11.73	13.90	-18.5	8.34	10.25	-22.9
	100k	48.62	54.44	-12.0	50.53	51.11	-1.1	33.61	48.62	-44.6
	316k	205.92	212.52	-3.2	222.06	232.22	-4.6	-	-	-
	1M	914.88	932.81	-2.0	1175.30	1141.33	2.9	-	-	-
TSPLIB	3k	0.36	0.4	-11.1	0.36	0.40	-11.1	0.24	0.30	-25.0
	10k	1.56	1.76	-13.0	1.62	1.94	-19.6	1.22	1.59	-31.0
	31k	7.00	5.85	16.5	4.91	5.20	-5.9	4.41	5.34	-21.3
	100k	15.27	15.01	1.8	13.39	16.15	-20.6	13.79	16.55	-20.0

The branch-and-bound algorithm can not solve large-scale ETSPs as the previous local search algorithms. It can typically return the optimum of an ETSP with 30 cities around 10 normalized seconds, but unable to find the optimum of an ETSP with 40 cities in thousands of normalized seconds. Therefore some small-scale problems were generated, as listed in Table 10. The group of Euclidean random instances (E) were conducted by the DIMACS TSP Euclidean random generator. The group of VLSI problems were selected as subproblems from the VLSI industrial data sets.

The algorithm was tested with two orders of edge candidates. The first was the ascending order of Euclidean distance. The other was the ascending order of the weighted distance. The parameters of learning the weights were set as follows. The sizes of the subproblems were set to 15 (a half size of the given problems). $|A|_{\max}$ was set to 5. The minimum confidence and the minimum support were set to 0.01 and 0.05 (around 15^{-1}) respectively.

Table 10 The selected small-scale ETSPs for experiments

Category	VLSI (source)	E (source)
30	xqf30 (xqf131)	`E30.0` (generator)
	xqg30 (xqg237)	`E30.1` (generator)
	pma30 (pma343)	`E30.2` (generator)
	pka30 (pka379)	`E30.3` (generator)
	bcl30 (bcl380)	`E30.3` (generator)

The results of the tests can be shown in Table 11. The algorithm guarantees optima, so the number of expanded branches is the main indicator for comparison. The time cost however is a less accurate indicator than the number of branches.

Table 11 Number of branches and time cost comparison of the branch-and-bound algorithm

Data set	Problem	Optimum	Expanded branches			Time
			BnB	BnB-WD	Imp (%)	Imp (%)
Random	E30.0	4620393	957	287	70.01	61.46
	E30.1	4539405	1370	1135	17.15	31.84
	E30.2	4778537	327	141	56.88	37.50
	E30.3	4779040	835	61201*	-7329.46	-80.92
	E30.4	4739803	610	976	-60.00	-84.87
VLSI	xqf30	128	258	214	17.05	5.19
	xqg30	158	379	143	62.27	36.90
	pma30	195	866	645	25.52	15.33
	pka30	184	563	547	2.84	8.35
	bcl30	149	46	45	2.17	4.23

*: Extraordinarily large.

It can be observed from Table 11 that the branch-and-bound can significantly benefit from the learned weights, but risks might be there especially when problem is small and the subproblem is even smaller. Similarly to the results in the large-scale tests, the average improvements in the industrial instances are more significant than those in the random instance.

In conclusion, the results show that the presented method is effective and efficient in learning and reforming the ETSP variables assignments. Both heuristic and exact search algorithms can benefit from the results. The industrial problems can typically benefit more than random problems. The presented method can also provide competitive results in large-scale problems where the α-nearness candidate set may take too much time to set up.

6 Conclusion

This paper presents an attempt to incorporate a supervised learning technique, CAR learning, into searching heuristics in the ETSP. The results obtained suggests that the presented approach can provide nontrivial information to determine some weights for variable assignments, if appropriate decision attributes can be defined. The determined weights can help heuristic and exact search algorithms in certain forms.

It is hoped that the presented research has made a preliminary contribution to take advantage of the power of machine learning in combinatorial optimization.

One of the significant results obtained is that rules can be approximately learned from subproblems and near optimal tours. This finding successfully restricts the expense of the auxiliary learning in large-scale problems.

It would be interesting to discover new decision attributes and learning procedures that can be applied to other combinatorial problems. Perhaps future studies can also be conducted for determining domain-specified (or even company-specific) decision attributes in practical industrial problems.

Acknowledgements The work described in this paper was fully supported by a grant from the Department of Industrial and Systems Engineering of The Hong Kong Polytechnic University (No. RP1Z).

References

1. Laporte, G.: Fifty years of vehicle routing. Transportation Science 43(4), 408–416 (2009)
2. Kotsiantis, S.B.: Supervised machine learning: A review of classification techniques. Informatica 31, 249–268 (2007)
3. Koonce, D.A., Tsai, S.C.: Using data mining to find patterns in genetic algorithm solutions to a job shop schedule. Comput. Ind. Eng. 38(3), 361–374 (2000)
4. Li, X., Olafsson, S.: Discovering dispatching rules using data mining. Journal of Scheduling 8(6), 515–527 (2005)
5. Sha, D.Y., Liu, C.H.: Using data mining for due date assignment in a dynamic job shop environment. The International Journal of Advanced Manufacturing Technology 25(11), 1164–1174 (2005)
6. Olafsson, S., Li, X.: Learning effective new single machine dispatching rules from optimal scheduling data. International Journal of Production Economics 128(1), 118–126 (2010)
7. Kwak, C., Yih, Y.: Data-mining approach to production control in the computer-integrated testing cell. IEEE Transactions on Robotics And Automation 20(1), 107–116 (2004)
8. Boyan, J., Moore, A.: STAGE learning for local search. Neural computing surveys 3, 35–38 (2000), available at: `http://ftp.icsi.berkeley.edu/ftp/pub/ai/jagota/vol3_1.pdf`
9. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data. pp. 207–216. ACM, New York, NY, USA (1993)
10. Liu, B., Hsu, W., Ma, Y.: Integrating classification and association rule mining. In: Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD). pp. 80–86 (1998)
11. Telelis, O., Stamatopoulos, P.: Guiding constructive search with statistical instance-based learning. International Journal on Artificial Intelligence Tools 11(2), 247–266 (2002)
12. Punnen, A.P.: The traveling salesman problem: Applications, formulations and variations. In: Gutin, G., Punnen, A.P. (eds.) The Traveling Salesman Problem and Its Variations, chap. 1, pp. 1–28. Kluwer Academic Publishers, Netherland (2002)
13. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The traveling salesman problem : a computational study. Princeton Series in Applied Mathematics, Princeton University Press, Princeton, N.J. (2006)
14. Papadimitriou, C.H.: The euclidean traveling salesman problem is NP-complete. Theor. Comput. Sci. 4(3), 237–244 (1977)
15. Johnson, D.S., McGeoch, L.A.: Local search in combinatorial optimization, chap. Traveling Salesman Problem: A Case Study in Local Optimization, pp. 215–310. John Wiley and Sons, Ltd. (1997)
16. Helsgaun, K.: An effective implementation of the lin-kernighan traveling salesman heuristic. European Journal of Operational Research 126(1), 106–130 (oct 2000)
17. Johnson, D.S., McGeoch, L.A.: The traveling salesman problem and Its Variations, Combinatorial Optimization, vol. 12, chap. Experimental analysis of heuristics for the STSP, pp. 369–444. Kluwer, Netherlands (2002)
18. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the travelling-salesman problem. Operations Research 21, 498–516 (1973)

19. Reinelt, G.: The Traveling Salesman: Computational Solutions for TSP Applications, Lecture Notes in Computer Science, vol. 840. Springer-Verlag, New York (1994)
20. Miller, D.L., Pekny, J.F.: A staged primal-dual algorithm for perfect *b*-matching with edge capacities. ORSA Journal on Computing 7, 298–320 (1995)
21. Bentley, J.L.: Fast algorithms for geometric traveling salesman problems. ORSA Journal on Computing 4, 387–411 (1992)
22. Helsgaun, K.: General k-opt submoves for the Lin-Kernighan TSP heuristic. Mathematical Programming Computation 1(2), 119–163 (2009)
23. Mannila, H., Toivonen, H., Verkamo, A.I.: Efficient algorithms for discovering association rules. In: Fayyad, U.M., Uthurusamy, R. (eds.) AAAI Workshop on Knowledge Discovery in Databases (KDD-94). pp. 181–192. AAAI Press, Seattle, Washington (1994), available at: `http://citeseer.ist.psu.edu/mannila94efficient.html`
24. Borgelt, C., Kruse, R.: Induction of association rules: Apriori implementation. In: Proceedings of the 15th Conference on Computational Statistics. pp. 395–400. Physica Verlag, Berlin, Germany (2002), available at: `http://www.borgelt.net/papers/cstat_02.pdf`