A computational evaluation of online ATSP algorithms

Michele Barbato, Alberto Ceselli, and Filippo Mosconi

Abstract We prove that state-of-the-art online asymmetric traveling salesman problem algorithms can successfully be used in real time practical systems, in terms of both solutions quality and computational efficiency. At the same time, we show that such a good behaviour can only be obtained by a careful fine tuning of the algorithms, often clashing with their theoretical analysis.
Keywords: Asymmetric Traveling Salesman; online algorithms; experimental analysis

1 Introduction

Server routing problems involve a set of requests to be served by a server in the minimum amount of time. Usually, every request is identified with a point in a space. A request is then served if the server visits the corresponding point. In the *online asymmetric traveling salesman problem* (OL-ATSP), requests are generated in sequence along time and each request must be served after it has been generated. In general, the generation time of the next request is unknown to the server. The goal in the OL-ATSP is to minimize the time at which all requests have been served.

The OL-ATSP and its variations arise in a number of real-world applications. Our interest in the OL-ATSP actually stems from its application to the management of automated warehouses [3], which is critical in highly custom production contexts like cosmetics manufacturing [1].

A warehouse consists of racks. They are typically identical, growing in vertical shelves of the same height, with an aisle between them which is traversed both horizontally and vertically by a stacker crane. The stacker crane moves containers to and from the racks: we refer to any such a movement as a *task*. One may assume

Michele Barbato, Alberto Ceselli, Filippo Mosconi
Università Degli Studi di Milano, Dipartimento di Informatica, via Celoria 18, 20133 Milano (Italy)
e-mail: michele.barbato@unimi.it, alberto.ceselli@unimi.it, filippo.mosconi@studenti.unimi.it

that the complete set of possible tasks is known in advance, and so are both the cost to perform each task and the time needed to start a task immediately after another one has been executed. Given a reasonable time horizon, only a small subset of all possible tasks appear: the goal is to find a sequence of them that can be performed by the stacker crane minimizing the total completion time.

Often in practical circumstances there is no knowledge on the temporal distribution of the tasks. That is, the order in which tasks will be revealed, as the time at which this will happen, is unknown. Algorithms operating the stacker crane in the above situation must therefore act in an online fashion: decisions at a given instant are taken by looking only at the sequence of requests generated until that instant. Under this assumption, the problem of optimally sequencing the completion of the tasks is exactly an OL-ATSP where the requests are the tasks to be performed and the server is the stacker crane.

The quality of OL-ATSP solutions, as in general with online algorithm, is assessed by comparing its cost to that potentially achievable by an exact offline algorithm, that is one where decisions are taken with complete knowledge on the sequence of requests. The worst-case ratio between the former and the latter is called competitive ratio. The lower the competitive ratio, the better the corresponding online algorithm. In this regard, the OL-ATSP is well understood, since algorithms providing optimal competitive ratio are known for several variants [5].

However these proven competitive algorithms do not necessarily exhibit good performance in practice, and a corresponding suitable experimental validation is often missing. The OL-ATSP is a relevant case: despite its high potential in applications, no experimental evaluation is carried out in the literature. Additionally, the complexity of the sub-problems that need to be solved undermines their practical applicability, finally imposing to resort to heuristics, thereby losing quality guarantees.

In this paper we fill in such a gap by performing an extensive experimental evaluation of OL-ATSP algorithms. We prove that state-of-the-art OL-ATSP algorithms can successfully be used in real time practical systems, in terms of both solutions quality and computational efficiency. At the same time, we show that such a good behaviour can only be obtained by a careful fine tuning of the algorithms, often clashing with the theoretical analysis.

More precisely, we experimentally study SmartStart, a family of algorithms for the OL-ATSP presented in [5] (Section 2). The theoretical analysis performed in [5] shows that, for several variations of the OL-ATSP, SmartStart yields algorithms with the best possible competitive ratio. We extensively test SmartStart algorithms, designing instances with specific spatial and temporal distribution of the requests (Section 3). We analyze the discrepancy between an experimental fine tuning of SmartStart and the theoretically predicted guarantees as the choice of these distributions changes (Section 3.1). We also consider lower bounding procedures, and use them to evaluate the actual gap obtained by SmartStart under various settings, thereby checking the tightness of the theoretical analysis in practical instances (Section 3.2). Finally, we evaluate the potential performance of SmartStart algorithms when employed in real-time applications (Section 3.3).

2 Definitions and SmartStart Algorithms

Throughout let V be a set of points in a space with a *distance* $d\colon V \times V \longrightarrow \mathbb{R}_{>0}$ satisfying the *triangular inequality* $d(u,v) + d(v,w) \geq d(u,w)$ for every $u, v, w \in V$. In general, $d(u,v) \neq d(v,u)$, hence we model such a space as a complete digraph $D = (V, A)$, having weight $d(u,v)$ on arc $(u,v) \in A$. According to [5], asymmetry makes the problem harder. We select an *origin* vertex $O \in V$, where a server is located at time 0. To reach $v \in V$ from $u \in V$ the server employs a time $t_{uv} = d(u,v)$. A *request* is a pair $r = (v,t)$ with $v \in V \setminus \{O\}$ and $t \in \mathbb{R}_+$. Given request $r = (v,t)$, we define $v(r) = v$ and $\tau(r) = t$. We say that request r is *served* if the server visits $v(r)$ at any time $t \geq \tau(r)$, *unserved* otherwise. Let $\mathcal{R} = \{r_1, r_2, \ldots, r_k\}$ be a sequence of requests such that $k \leq |V|$ and $\tau(r_i) \leq \tau(r_j)$ whenever $1 \leq i < j \leq k$. In the *homing* OL-ATSP, the server must minimize the total *completion time*, that is, the time required to serve all requests in $\mathcal{R}$ and subsequently return at O. Note that the homing OL-ATSP where $k = |V|$ and $\tau(r) = 0$ for every $r \in \mathcal{R}$ is the classical offline asymmetric traveling salesman problem (ATSP), thoroughly described *e.g.,* in [7]. Online algorithms operate the server by taking decisions on the fly, considering at time $t \geq 0$ only the requests r such that $\tau(r) \leq t$. Let $c_{\mathcal{A}}(\mathcal{R})$ be the completion time of a server operated by algorithm $\mathcal{A}$, defined as the time to serve all requests in $\mathcal{R}$ and return to O afterwards. Let $c_{\text{BEST}}(\mathcal{R})$ be the best possible completion time for the request sequence $\mathcal{R}$. Given $K \geq 0$, algorithm $\mathcal{A}$ is *K-competitive* if $c_{\mathcal{A}}(\mathcal{R})/K \leq c_{\text{BEST}}(\mathcal{R})$ for every request sequence $\mathcal{R}$. The real value K is the *competitive ratio* of $\mathcal{A}$.

SmartStart Algorithms. SmartStart is a family of algorithms first introduced in [4] in the context of online dial-a-ride problems. It has been used for the OL-ATSP in [5]. Let us assume that requests are generated over time at some vertices of the digraph D described above. The behavior of SmartStart is parametric on a real value: for a fixed $\alpha > 0$ the corresponding algorithm $\mathcal{A}(\alpha)$ works as follows. At every $t \geq 0$ let $\mathcal{R}_t \subseteq \mathcal{R}$ be the subset of unserved requests r with $\tau(r) \leq t$. The server computes $d(\mathcal{R}_t)$, the value of the tour S needed to visit all requests of $\mathcal{R}_t$ starting from and returning to O. At the minimum t' such that $t' \geq \alpha d(\mathcal{R}_t)$ the server executes S serving all requests in $\mathcal{R}_t$ and ignoring all requests generated meanwhile. When the server is back at O the above process is repeated.

If the weights on the arcs of D obey the triangular inequality, the competitive ratio of $\mathcal{A}(\alpha)$ only depends on whether the last request arrives while the server is idle at O. Exploiting this fact the authors of [5] prove the following:

Theorem 1 ([5])

For every $\alpha > 0$, $\mathcal{A}(\alpha)$ is $\max\{1+\alpha, 2+1/\alpha\}$-competitive for the homing OL-ATSP. The best SmartStart algorithm is $\mathcal{A}(\phi)$ where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.

The value ϕ minimizes $\max\{1+\alpha, 2+1/\alpha\}$. Since $1+\phi = 2+1/\phi$, algorithm $\mathcal{A}(\phi)$ is $(1+\phi)$-competitive. The study in [5] also shows that $(1+\phi)$ is a lower bound on the competitive ratio of any possible OL-ATSP algorithm. Thus, $\mathcal{A}(\phi)$ yields the best competitive ratio among online algorithms for the homing OL-ATSP.

3 Experimental analysis

In this section we address two experimental questions. The first concerns solutions quality, that is low completion time objective values, as defined in Section 2. In particular we investigate whether the theoretical best setting of α is effective also in experiments (subsections 3.1 and 3.2). We stress that, by definition, SmartStart algorithms need an exact solver for the ATSP as a subroutine to serve partial sets of requests every time the server leaves the origin. Therefore, the second question concerns computing times, that is if the need of an exact ATSP solver turns out to be a bottleneck for the applicability of SmartStart or not (subsection 3.3).

Dataset Design. Our investigation started from preliminary experiments on TSPLib instances [9]. We found that the behavior of SmartStart on these instances was affected by requests appearing in clusters. In fact, assuming constant inter-arrival times, if requests are generated spatially very close to each other and far from the origin O, a good heuristic for the OL-ATSP would be to never return to O before having served all of them, as this would imply unnecessary round-trips between the origin and their locations. The same applies to the temporal distribution of the requests: if all data was known in advance a good heuristic would merge requests generated in short time intervals, exploiting large time gaps for returning back to O.

Therefore, we decided to build a more reliable experimental setting, by controlling both spatial and temporal distributions of the requests. Indeed, spatial and temporal distributions are intertwined in determining the complexity of an instance.

Concerning the spatial distribution, SmartStart algorithms are designed to start a tour serving subsets of requests based on the trade-off between distance to be traveled and elapsed time. As sketched above, we argue space cluster tendency to be predictive of easy instances. Oppositely, when requests are uniformly scattered, no particular rationale is obviously yielding good policies; in particular, SmartStart algorithms has no design elements to perform a wise choice of the subsets of requests to serve.

Motivated by the above arguments, digraphs $D = (V, A)$ in our dataset are constructed as follows. In the bi-dimensional Euclidean space, we initially consider a point O and six points $m_1, \ldots, m_6$ uniformly disposed on a large circumference with centre in O and radius 500. Next, we generate clusters $C_1, \ldots, C_6$ each including 20 points drawn at random from a bi-variate Gaussian distribution centred at m_i, with covariance matrix $[[50^2, 0], [0, 50^2]]$, for every $i = 1, \ldots, 6$. The origin of D is located at O, and all other vertices are located at the points of the clusters (note that the m_i's may not be vertices of D).

In order to have asymmetric distances between vertices, we define $d \colon A \longrightarrow \mathbb{R}_{\geq 0}$ as:

$$d(v, w) = \left\lceil \Big(||p(v) - p(w)||^2 + G \cdot \max\{0, y(w) - y(v)\} \Big)^{1/2} \right\rceil \quad \forall v, w \in V \qquad (1)$$

where for every $v \in V$, $p(v)$ (resp. $y(v)$) is the Euclidean point (resp. its y-coordinate) where vertex v is located, $||p - q||$ is the Euclidean distance between points p and q and $G > 0$ is an *asymmetry parameter*. It is not difficult to show that the arc weight function d defined in (1) satisfies the triangular inequality. Moreover, by varying the asymmetry parameter, we can easily control the total amount of *asymmetry degree* of D defined in [5] as $\sup_{v,w \in V} \frac{d(x,y)}{d(y,x)}$, which is relevant in the competitive analysis of online algorithms for the OL-ATSP. We remark that, given this setting, O is not guaranteed to be the center of the smallest circle containing all the points. On the whole, we created 50 weighted digraphs (D, d) with d having asymmetry parameter $G = 1$ and 50 additional digraphs with $G = 10$.

In the following we assume that the clusters $C_1, \ldots, C_6$ are disposed in clockwise order around O with C_i being the predecessor of C_{i+1}, for every $i = 1, \ldots, 5$. We consider three types of order in which the requests are generated: *(i)* CLUSTER order: 20 requests are generated consecutively in the same cluster, one at each vertex of that cluster. Then the operation repeats on successive clusters by following the cluster order $C_1, C_2, C_3, C_4, C_5, C_6$; *(ii)* JUMPING order: 20 requests are generated *consecutively in the same cluster*, one at each vertex of that cluster. The operation is iterated following the cluster order $C_1, C_4, C_6, C_2, C_5, C_3$; *(iii)* RANDOM order: *one request is generated in one vertex of a cluster*, among the vertices where no request has been generated yet. The operation is iterated cyclically following the cluster order $C_1, C_4, C_6, C_2, C_5, C_3$.

Concerning the temporal distribution we designed three scenarios: (a) Uniform: requests appear at constant time intervals; (b) Dense-first: let T be the time at which the last request appears. The first 75% requests are generated uniformly between 0 and time $T/2$; the remaining 25% requests are generated uniformly between time $T/2$ and T. (c) Sparse-first: as in Dense-first case but generating the first 25% requests before time $T/2$ and the last 75% between $T/2$ and T. In all three temporal distributions above, we let the time of the last generated request be equal to the length of the optimal ATSP solution on the corresponding instance, computed in preprocessing, in such a way that the inter-arrival times are of the same order of magnitude of the traveling time between requests.

For each of the 100 digraphs created as above we combine each spatial order from types *(i)*–*(iii)* with each temporal distributions from types (a)–(c). Hence, overall, our dataset consists of 900 instances.

3.1 Effect of α

We first analyze how the behavior of SMARTSTART algorithm $\mathcal{A}(\alpha)$ varies according to the choice of the parameter α. According to Theorem 1, the value of α yielding the best competitive ratio on generic instances is $\alpha = \phi \simeq 1.6$. We performed tests with $\alpha \in \{0.2, 0.4, 0.8, 1.0, 1.4, 1.6\}$ on our dataset.

In Table 1 we provide the average completion times corresponding to several values of α in our set. In this table the first column corresponds to the three orders

Table 1 Average completion times for varying values of α.

Spatial Distribution	Temporal Distribution	α					
		0.2	0.4	0.8	1	1.4	1.6
C	Uniform	11063.22	**11027.58**	11154.02	13883.85	16699.46	18091.09
	Dense-first	**10476.31**	10895.53	12760.20	14903.36	16699.46	18091.09
	Sparse-first	**11783.40**	11923.03	11854.01	11831.93	12696.33	17664.01
J	Uniform	**12143.51**	12303.67	12186.78	13999.04	16699.46	18091.09
	Dense-first	**11507.02**	11654.39	12862.41	14877.36	16699.46	18091.09
	Sparse-first	**13102.64**	13214.90	13096.30	13176.66	14018.01	17782.99
R	Uniform	15175.50	15503.38	15728.75	**14215.77**	16699.46	18091.09
	Dense-first	15122.27	**13193.61**	15847.49	14990.32	16699.46	18091.09
	Sparse-first	14680.25	15928.97	14809.05	**14336.07**	16699.46	18091.09

of request generation (C is for the CLUSTER order, J for JUMPING, R for RANDOM); similarly, the second column of Table 1 specifies the temporal distribution followed to generate the instances. The best average completion times of Table 1 are reported in boldface for every combination of spatial and temporal distributions. Table 1 highlights that on seven out of the nine types of instances the best average completion times correspond to $\alpha \in \{0.2, 0.4\}$. The other two types of instances are instead optimized in average by the value $\alpha = 1$ and belong to the random spatial distribution. We report that the theoretically optimal algorithm $\mathcal{A}(\phi)$ yielded the best completion time only on 1 instance out of the 900 tested instances.[1] Oppositely, for about 77% of instances the best α belongs to $\{0.2, 0.4, 0.8\}$. This suggests that our instances are better solved by SMARTSTART algorithms serving small sets of unserved requests, almost immediately after their generation. From these results we conclude that when considering structured OL-ATSP instances, the theoretical analysis of [5] can be imprecise in predicting the value of α giving the best SMARTSTART algorithm in practice. We mention that a similar result was reported in [2] on the online *symmetric* traveling salesman problem. On the other hand, for less structured instances, our experiments seem to indicate that the theoretically best α is closer to the best experimental α.

3.2 Lower Bound Comparison

We now compare the results described in Section 3.1 with a theoretical lower bound for the OL-ATSP. Given an instance of the OL-ATSP, let $\bar{r}$ be its last generated request. Then a valid lower bound on the optimal completion time is $\tau(\bar{r}) + d(v(\bar{r}), O)$. Indeed, a server at least waits until time $\tau(\bar{r})$ for the last request to arrive and uses at least $d(v(\bar{r}), O)$ time units to go from $v(\bar{r})$ to O (visiting $v(\bar{r})$ is required to serve r).

[1] The instance has $G = 1$, following a CLUSTER ordering of the requests and a sparse-first distribution.

Table 2 Comparison of SmartStart with a theoretical lower bound.

Spatial Distribution	Temporal Distribution	$\mathcal{L}$	$\mathcal{A}$	Gap%
C	Uniform	7423.65	11027.58	48.55
	Dense-first	7500.01	10476.31	39.68
	Sparse-first	7422.70	11783.40	58.75
J	Uniform	7392.34	12143.51	64.27
	Dense-first	7468.70	11507.02	54.07
	Sparse-first	7391.39	13102.64	77.27
R	Uniform	7392.34	14215.77	92.30
	Dense-first	7468.70	13193.61	76.65
	Sparse-first	7391.39	14336.07	93.96

In Table 2, we compare SmartStart and the theoretical lower bound. Every row of the table represents a fixed instance type, obtained combining a spatial and a temporal distribution, as indicated by the first two columns while column $\mathcal{L}$ reports the average lower bounds on instances belonging to a given type. For each instance type, column $\mathcal{A}$ reports the average completion time of the SmartStart algorithms yielding the best average completion times on that instance type. Note that the values in column $\mathcal{A}$ correspond to the boldface values of Table 1. The last column of the table reports the relative gap between the values of column $\mathcal{A}$ and column $\mathcal{L}$, computed as $100 \cdot \frac{\mathcal{A}-\mathcal{L}}{\mathcal{L}}$.

On instances with sparse-first temporal distribution or random ordering generation of requests, the best SmartStart algorithm in average yields large relative increasing with respect to the average lower bound. However, on all other instances the best SmartStart algorithm in average yields completion times which are less than 60% greater than the corresponding average lower bound.

We recall that, by results in [5], the worst-case competitive ratio of $\mathcal{A}(\alpha)$ is $1 + \phi$ for every $\alpha > 0$. Since $\phi \simeq 1.6$ this results in a worst-case relative increasing with respect to an optimal offline algorithm of roughly 160%, for every SmartStart algorithm. Table 2 indicates that, in practice, SmartStart algorithms never attain this worst-case competitive ratio. In fact, this ratio is not reached even by comparing with the average lower bound in place of the optimum.

In Figure 1 we provide a more detailed view of the same experiment. There, the y-axis represents values of completion times; one boxplot summarizes the distribution of solution values for each choice of $\alpha \in \{0.2, 0.4, 0.8, 1.0, 1.4, 1.6\}$, as indicated on the x-axis. The leftmost (resp. rightmost) boxplot refers to LB values (resp. $(1 + \phi) \cdot LB$ values, that is, the *best* competitive ratio of SmartStart). Average values are marked with filled squares. First, we observe that the trend of each quartile reflects that of average values (see Table 1). Second, we notice that $\mathcal{A}(1.6)$ solution values are in general lower than $(1+\phi)\cdot LB$; in turn, we know the theoretical analysis to be tight, so we expect at least random instances to exist in our dataset on which $\mathcal{A}(1.6)$ approaches the theoretical worst case $(1+\phi)\cdot OPT$: in those instances

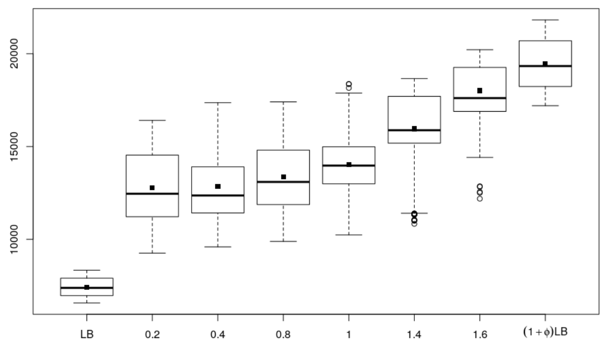

Fig. 1 Comparison of several SmartStart algorithms and the lower bounds computed on our dataset.

LB is approaching OPT; at the same time, the dispersion of LB values is low. This makes us conjecture that the LB is of good quality in our instances.

3.3 Realtime Applicability

As discussed, SmartStart involves the iterative resolution of ATSPs. While being an **NP**-hard problem, the size of ATSP instances manageable by current state-of-the-art solvers is matching the size of instances arising from real-world applications. Hence, it is of interest to evaluate the actual applicability of SmartStart in real-world OL-ATSPs.

We proceed as follows. For all $t \geq 0$, let $\mathcal{R}_t$ be the set of unserved requests, the last of which appearing at instant t and let $S(\mathcal{R}_t)$ be the time $\mathcal{A}(\alpha)$ employs to solve the ATSP on $\mathcal{R}_t \cup \{O\}$. It never pays off to postpone the resolution of such an ATSP. Then $\mathcal{A}(\alpha)$ has a *timeout* whenever there exists a new request r appearing between t and $t + S(\mathcal{R}_t)$. That is, in a real system, a new request appears while the algorithm is still evaluating the previous ones. We use the probability of occurrence of a timeout as a measure of applicability of SmartStart in a real-time context.

In fact, the occurrence of a timeout is a random variable $Y = 1$ if $T < S$, $Y = 0$ otherwise where in turn T is a random variable modeling the time interval between two consecutive requests, and S is a random variable modeling the ATSP resolution time using the ATSP solver. In our experiments we decided to map ATSP instances to instances of the symmetric traveling salesman problem by means of a standard Karp reduction [8], subsequently solved with Concorde [6], a state-of-the-art solver for the symmetric traveling salesman problem. We approximate the cumulative distribution function of S by its empirical counterpart through numerical simulation, considering all ATSP runs in our experiments (that is 401.925 heterogeneous Concorde single

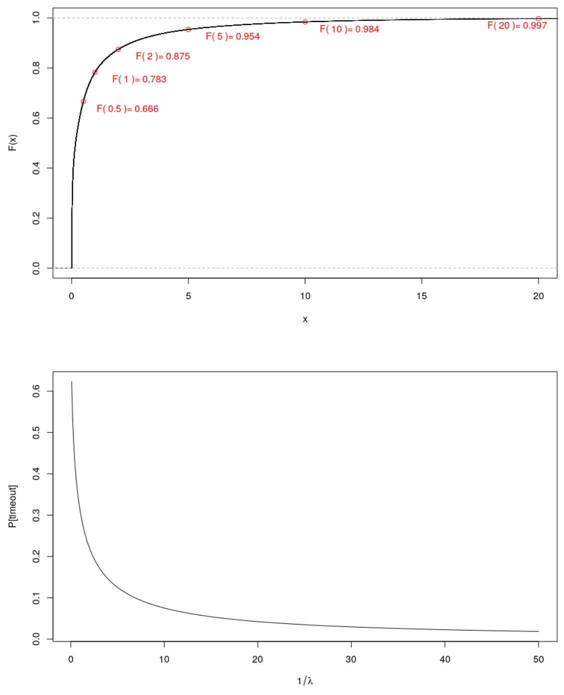

Fig. 2 Distribution of Concorde run times (top) and prob. of SmartStart timeouts (bottom).

thread calls on a 4.00GHz Intel(R) Core(TM) i7-6700K and 32GB RAM). In Figure 2 (top) we report such an empirical cumulative distribution function $F(x)$, that is estimating the probability that a Concorde run employs at most x seconds. For what concerns T, instead, we assumed an exponential distribution for the time intervals between two consecutive requests, whose density function is $q(x) = \lambda e^{-\lambda x}$, that is a standard modeling of independent inter-arrival times. We finally estimated

$$\mathbb{E}[Y] = P[Y = 1] = P[S > T] =$$
$$= \int_{x=0}^{+\infty} P[S > T|T = x]P[T = x]dx = \int_{x=0}^{+\infty} P[S > x|T = x]P[T = x]dx$$

by numerically computing $\int_{x=0}^{+\infty}(1 - F(x)) \cdot q(x) \cdot dx$.

That is, we statistically determine the probability of a SmartStart timeout in our setting, for varying values $1/\lambda$ of the average time intervals between consecutive requests. Results are plotted in Figure 2 (bottom). We deduce that the timeout probability approaches 10% already for average time intervals not smaller than 7 seconds. As a conclusion, SmartStart can be considered a viable solution to solve OL-ATSPs whose requests are generated at a high frequency (of the order of $\simeq$ 5 seconds) and hence successfully embeddable in modern systems handling real-time order satisfaction like automated warehouses.

4 Conclusions

In this paper we have performed an experimental analysis of SmartStart, a family of algorithms for the OL-ATSP. First, we have shown that the parameter α determining the SmartStart behavior requires careful fine-tuning to take advantage of a prior knowledge on instance structure. In particular, often on structured instances the best theoretical α resulted in SmartStart algorithms behaving poorly in practice. Even if more aggressive settings of α always pay off in our experiments, the theoretical analysis of [5] becomes closer to our results when requests are spatially more randomly distributed. We have additionally observed that well-tuned SmartStart algorithms may produce solutions whose value is nearly optimal. This result was shown by comparing the quality of SmartStart solutions with a theoretical lower bound. Finally, we have evaluated the usability of SmartStart in real-time applications by means of a statistical framework, concluding that, also thanks to the performance level reached by exact ATSP solvers and modern hardware, SmartStart can currently be embedded in realtime systems dealing with OL-ATSP applications.

Acknowledgement. This research was partially funded by Regione Lombardia, grant agreement n.E97F17000000009, project AD-COM.

References

1. AD-COM. Project website. https://ad-com.net/, last access April 2019.
2. M. Aprea, E. Feuerstein, G. Sadovoy, and A. S. de Loma. Discrete online TSP. In *International Conference on Algorithmic Applications in Management*, pages 29–42. Springer, 2009.
3. N. Ascheuer, M. Grötschel, and A. A.-A. Abdel-Hamid. Order picking in an automatic warehouse: Solving online asymmetric tsps. *Mathematical Methods of Operations Research*, 49(3):501–515, 1999.
4. N. Ascheuer, S. O. Krumke, and J. Rambau. Online dial-a-ride problems: Minimizing the completion time. In H. Reichel and S. Tison, editors, *STACS 2000*, pages 639–650, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.
5. G. Ausiello, V. Bonifaci, and L. Laura. The on-line asymmetric traveling salesman problem. *Journal of Discrete Algorithms*, 6(2):290–298, 2008.
6. Concorde. D. L. Applegate, R. E. Bixby, V. Chvatal and W. J. Cook. http://www.math.uwaterloo.ca/tsp/concorde.html, 2003.
7. G. Gutin and A. P. Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006.
8. R. Roberti and P. Toth. Models and algorithms for the asymmetric traveling salesman problem: an experimental comparison. *EURO Journal on Transportation and Logistics*, 1(1):113–133, 2012.
9. TSPLib. Maintained by G. Reinelt. https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/, 2013.