Supporting Information for

Quantifying the benefits of vehicle pooling with shareability networks

Paolo Santi, Giovanni Resta, Michael Szell, Stanislav Sobolevsky, Steven Strogatz & Carlo Ratti

In this supporting information we present the detailed methods, including the handling of the data set, the formal derivation of the network-based approach used to quantify the benefits of a shared taxi system, and essential extended details of the analysis.

Data set and pre-processing

The data set contains origin-destination data of all 172 million trips with passengers of all 13,586 taxicabs in New York during the calendar year of 2011. Each vehicle is associated with a license, a so-called *medallion*, which is synonymously used as a name for the vehicles. These medallion taxis are the only vehicles in the city permitted to pick up passengers in response to a street hail. A medallion may be purchased from the City at infrequent auctions, or from another medallion owner. Because of their high prices medallions and most cabs are owned by investment companies and are leased to drivers. There are 39,437 unique driver IDs in the data set, which corresponds to 2.9 drivers per medallion on average. Note that we unfortunately do not have explicit information on the number of passengers per vehicle, however, following the data reported by [1], that the average daily number of passengers served by NY taxis is 600,000, with 450,000 trips on average, the average number of passengers per trip is around 1.3. The data set contains a number of fields from which we use the following: medallion ID, origin time, destination time, origin longitude, origin latitude, destination longitude, destination latitude. Times are accurate to the second, positional information has been collected via Global Positioning System (GPS) technology by the data provider. Out of our control are possible biases due to urban canyons which might have slightly distorted the GPS locations during the collection process [2]. All IDs are given in anonymized form, origin and destination values refer to the origins and destinations of trips, respectively.

For creating the street network of Manhattan we used data from openstreetmap.org. We filtered the streets of Manhattan, selecting only the following road classes: primary, secondary, tertiary, residential, unclassified, road, living street. Several other classes were deliberately left

out, such as footpaths, trunks, links or service roads, as they are unlikely to contain delivery or pickup locations. Next we extracted the street intersections to build a network in which nodes are intersections and directed links are roads connecting those intersections (we use directed links because a non-negligible fraction of streets in Manhattan are one-way). The extracted network of street intersections was then manually cleaned for obvious inconsistencies or redundancies (such as duplicate intersection points at the same geographic positions), in the end containing 4091 nodes and 9452 directed links. This network was used to map-match the GPS locations from the trip data set. We only matched locations for which a closest node in the street intersection network exists with a distance less than than $100\,\mathrm{m}$. We matched GPS points to street intersections rather than to points on the closest street segments as a reasonable compromise between high accuracy (the average length of street segments in Manhattan is $126\,\mathrm{m}$) and granularity of discretization that is mainly relevant in the estimation of travel times, see SI Section *Computing travel times*. Finally, from the remaining 150 million trips we discarded about 2 million trips that had identical starting and end points, and trips that lasted less than one minute.

Static and dynamic implementations, and the relevance of empty trips

Before addressing the aim of this work of developing a theoretical framework for the rigorous quantification and optimization of general spatio-temporal sharing problems, in particular of trip sharing in taxi systems, we draw in this section the connection to the corresponding practical issues that come along with concrete implementations of such systems. A street hailing based, conventional taxi system, ideally features taxis that i) when empty, try to find a passenger as fast as possible by choosing an optimal passenger-finding strategy [3, 4], ii) when occupied, deliver passengers as fast as possible to their destination. For taxi operations that are not based on street hailing but on trip queries, as facilitated by modern mobile phone apps and services, the sub-problem i) becomes an issue of efficiently locating, scheduling and dispatching the closest empty taxis.

In conventional taxi systems, the problem of dispatching taxis to a trip request is conceptually straightforward: Find the closest empty taxi – where the metric for closeness can involve vehicle velocities, traffic conditions [5], etc. In dynamic approaches to taxi sharing however, in which taxis are allowed to re-route and to pick up new passengers on the fly, the concrete operational issues of the detailed spatial query setup and the communication protocol design between taxis and dispatch system becomes potentially intricate. This set of problems has been satisfactorily solved by a recently published service model called T-share [6]. Using empirical data and simulations, the study has pointed out the impressive potential of a heuristic, dynamic approach, and has provided efficient and scalable algorithms to solve the taxi searching and scheduling sub-problems which occur when taxis are allowed to change their routes on the fly. It found that a dynamically scheduled taxi system is able to handle ridesharing, allowing in the city of Beijing to service 25% additional taxi users while saving 13% travel distance compared to services without ridesharing.

As we derive formally in the following sections, it is already possible to achieve high levels of trip sharing and extraordinary benefits with simple static implementations rather than with dynamic ones. A static perspective additionally allows rigorous quantification of the benefits using provably optimal algorithms from graph theory. Such a static implementation of taxi sharing can be imagined as follows, from the perspective of a passenger: 1) Submit trip request source and destination to the central system, for example via mobile phone app, 2) wait up to $\delta = 1\,\mathrm{min}$ for the system to respond with a sharing option presenting the information of estimated arrival time without sharing and how, due to sharing, the trip might be prolonged by a time up to Δ (or equivalently, the passenger is presented a time window of possible arrival times), 3) either confirm or deny the presented sharing option. After the k passengers have confirmed their participation, the shared trip becomes one assignment of bundled, unchangeable requests with a well-defined route having a starting point, intermediate points, and an end point, which can be handled by any conventional taxi dispatch system. Only at this step do the locations of the surrounding taxis become relevant. Therefore, a static implementation builds directly on existing taxi systems without the need to reconsider the dispatch process.

Since such a static implementation of a shared taxi system can be regarded as independent of the problem of dispatching and can build directly on top of existing dispatching solutions, it is not in the scope of our work to revisit implementation issues concerning search and scheduling of empty taxis. Still, it is helpful to understand quantitatively the relevance of empty trips in the general picture of taxi systems since one could suspect that the cruising of empty taxis might be the main source of wasting energy in terms of the effective mile per gallon. For this reason, we measured the distribution of durations of occupied trips versus the durations in-between occupied taxi trips, Fig. S1. The figure shows the empirical probability distribution of durations of occupied trips, and of the time spans in-between the occupied trips which comprise both empty trips and all activities where taxis are not being used to transport passengers such as shift changes, lunch breaks, vehicle maintenance. Specific information for distinguishing between the empty trips and the rest of these downtime activities is unfortunately not available, as the data points only include the spatial and temporal information of the pickups and dropoffs of occupied trips. In any case, the in-between durations peak below two minutes, substantially lower than the durations of occupied trips which peak at around six minutes, showing that taxis tend to find new passengers relatively quickly and that taxis spend about 75% of their on-service time performing occupied trips. This is at least the case for Manhattan; it is an open question whether the dispatching sub-problem becomes more relevant in cities with lower taxi demand. Further, the distribution of in-between durations is long-tailed due to the various types and occurences of taxi downtimes, see Fig. S1 inset. For example, bumps in the distribution at 6 and 14 hours possibly indicate shift-related durations. However, the vast majority of empty trips are covered by the durations that are shorter than half an hour, which includes over 96% of all downtimes, shown in the main panel of Fig. S1.

For the specific problem of implementing a dynamic taxi dispatching and trip sharing system these observations suggest that while efficient dispatching and routing of empty taxis is doubtlessly a non-negligible issue, especially for the possible small fraction of cases where

taxis do not find passengers quickly, it might be of higher importance to solve the main problem related to the occupied trips, namely of matching trips efficiently. In any case, to understand thoroughly a taxi system's improvability of the handling of empty trips, explicit data of empty trips must be available, preferably containing frequently sampled points per trip, which is not the case in the available data set.

A network-based approach for sharing taxi rides

In contrast to typical approaches based on linear programs [7, 8] and references therein, we show in this supplementary information in detail how our new approach allows polynomial-time, i.e., feasible, computation of the optimal ride sharing strategy when at most two trips can be combined, and polynomial-time computation of a constant-factor approximation of the optimal solution when $k > 2$ trips can be shared. Notice, though, that the degree of the involved polynomials increases with k. In practice, the approach turns out to be computationally feasible for $k = 3$, while it becomes impractical for larger values of k.

The goal of the trip sharing strategy can be either minimizing the number of trips performed or the total travel cost for a given set of trips, subject to a quality of service constraint (maximum allowed delay at delivery/destination). The former goal allows quantifying the actual number of taxis needed to satisfy the current taxi demand with a shared taxi service. By assuming that cost of a trip is proportional to the travel time, the latter goal becomes a proxy of the carbon emissions generated by the shared taxi fleet to accommodate the total traffic demand. By comparing the total travel time of the shared taxi service with that of the traditional, non-shared taxi service, we can thus quantify the expected reduction in pollution achieved by a shared versus a traditional taxi service. Of course the factors which determine vehicle emissions can be complicated and highly non-linear, such as the most important factor of speed and engine load, which are themselves affected by traffic congestion, driver mentality, traffic signals, posted speed limits, etc. [9, 10]. However, *all things being equal*, as general traffic conditions presumably remain largely unaltered by the sharing service, the cumulative emissions can be treated as proportional to the travel time.

The high-level idea, elaborated rigorously in the following sections, is to cast the problem of identifying the best trip sharing strategy as a network problem, where nodes of the network represent taxi trips, and links connect trips that can be combined. The resulting network is called the *shareability network*. A maximum tolerated time delay Δ at both pickup and delivery location regulates the density of the shareability network – the higher Δ the more sharing opportunities arise but the lower the quality of service becomes due to the increased delays. We show that the problem of finding the optimal trip sharing strategy when at most two trips can be combined is equivalent to the problem of finding the maximum matching in the shareability network, which can be solved in time $O(m\sqrt{n})$, where n is the number of nodes and m the number of links in the network. Notice that the shareability network is likely sparse, i.e. the average node degree is a constant which does not depend on n, hence the above time complexity reduces to $O(n\sqrt{n})$. More specifically, the maximum matching in the shareability network corresponds

to the trip combination strategy that minimizes the number of performed trips. If links in the trip graph are weighted with the travel cost reduction, i.e. the difference between the duration of combined ride and the two single rides, then the problem of finding the trip combination that minimizes the total travel cost is equivalent to the problem of finding the maximum weighted matching in the shareability network, which is also solvable in polynomial time.

If we relax the assumption that at most two trips can be combined, the complexity of the problem increases. In fact, the problem(s) at hand becomes equivalent to the weighted matching problem on k-bounded hyper-networks, which is NP-complete when $k > 2$ in general hyper-networks. However, polynomial-time algorithms are known that compute a solution which is within a constant factor from optimal. In particular, when the number of combined trips is at most k, for any constant $k > 2$, simple greedy algorithms can be used to produce a solution within a factor k from optimal.

We first present the case in which at most two trips can be combined, and then proceed to present the more general (and complex) case of an arbitrary number of combined trips. To ease presentation, we assume single passenger trips.

The two-trips sharing case

In this section, we assume that at most two trips can be combined. Notice that this is a stricter condition than assuming that the maximum taxi capacity is two. The difference between the two assumptions is exemplified in Fig. 1G of the main text. We have three trips T_1, T_2, and T_3. Assuming that delay constraints on passenger delivery are satisfied, the three trips can be combined in a single trip even using a taxi with capacity two if the passenger of T_2 is loaded onboard taxi performing T_1 at time t_2, unloaded at time $t_2^{'} > t_2$, and the passenger of trip T_3 is loaded at time $t_3 > t_2^{'}$ – cfr. the middle case in Fig. 1G of the main text. This combination of trips is not allowed in our model, since only two trips at most can be combined. Notice, on the other hand, that any shared trip obtained by combining at most two single trips can be realized using a taxi with capacity two. More generally, any k combination of trips can be performed using a taxi with capacity k. Hence, the trip combination solutions presented in the following can be accomplished using a taxi fleet where each taxi has capacity k, where k is the upper bound on the number of trips combined in a single trip.

Let $S = (T, L)$ be the (undirected) *shareability network* defined as follows. The node set $T = \{T_1, \ldots, T_n\}$ corresponds to the set of all possible n trips. The link set $L = \{L_1, \ldots, L_m\}$ is built as follows: link $(T_i, T_j) \in L$ connects nodes T_i and T_j if and only if trips T_i and T_j can be combined. The trip obtained combining trips T_i and T_j is denoted $T_{i,j}$ in the following. Whether trips T_i, T_j can be combined depends on spatial/temporal properties of the two trips, and on an upper bound Δ on the maximum delivery delay customers can tolerate. How to derive the set of combinable trips (link set L) given a travel data set such as the one at hand is a problem we defer to the next section. In the remainder of this section, we assume that L is known and given as input to the problem.

Definition 1. *A set $\mathcal{T}$ of* possibly combined trips*, where combined trips are composed of two*

single trips, is defined as $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$, where $\mathcal{T}_1 \subseteq T$ is a set of single trips, and $\mathcal{T}_2$ is a set of combined trips $T_{i,j}$, for some $i, j \in \{1, \ldots, n\}$.

Definition 2 (Feasible trip set)**.** *A set $\mathcal{T}$ of possibly combined trips is* feasible *if and only if all trips in T appear once in $\mathcal{T}$, formally:*

$$\forall T_i \in T, \quad (T_i \in \mathcal{T}_1) \vee (\exists_1 j \text{ s.t. } T_{i,j} \in \mathcal{T}_2) \; .$$

Notice that any trip $T_i \in T$ can appear only once in a feasible trip set, either as a single trip (if $T_i \in \mathcal{T}_1$), or combined with another trip (if $T_i \in \mathcal{T}_2$). The two travel optimization problems we solve in the following are formally defined as follows:

Definition 3 (MINIMUMNUMBERTRIP – MNT)**.** *Given the shareability network $S = (T, L)$, determine a feasible trip set of minimum cardinality.*

Definition 4 (MINIMIZETOTALTRAVELCOST – MTTC)**.** *Given the weighted shareability network $S = (T, L)$ where each link $(T_i, T_j) \in L$ is weighted with $w_{ij} = c(T_i) + c(T_j) - c(T_{i,j})$, where $c(T_x)$ denotes the cost of trip T_x, and $c(T_{i,j})$ the cost of the combined trip $T_{i,j}$; determine a feasible trip set such that the total travel cost is minimized.*

Regarding problem MTTC, we observe that we can assume w.l.o.g. that $c(T_{i,j}) < c(T_i) + c(T_j)$, since otherwise we can remove link (T_i, T_j) from L without impacting the optimal solution.

Definition 5. (MATCHINGS) *Let $S = (T, L)$ be a shareability network. A matching on S is a set of links $L' \subseteq L$ such that no two links in L' share an endpoint. A maximum matching on S is a matching on S of maximum cardinality. If links in S are weighted, a maximum weighted matching on S is a matching L' on S such that the sum of weights of links in L' is maximum.*

Lemma 1. *A set $\mathcal{T}$ of possibly combined trips is feasible if and only if its subset $\mathcal{T}_2$ of combined trips is a matching of S.*

Proof. Let $\mathcal{T}$ be any feasible set of combined trips. Since $\mathcal{T}$ is feasible, it contains either single trips, or trips obtained by combining two trips. Let $\mathcal{T}_2 \subseteq \mathcal{T}$ be the set of combined trips in $\mathcal{T}$. For any combined trip $T_{i,j} \in \mathcal{T}_2$, we consider the corresponding link (T_i, T_j) in the shareability network. Notice that, for any other link of the form (T_i, T_h) (or (T_h, T_j)) in the shareability network, the corresponding combined trip $T_{i,h}$ (or $T_{h,j}$) is not in $\mathcal{T}_2$, since otherwise conditions on maximal trip combination would be violated. It follows that no two links corresponding to the combined trips in $\mathcal{T}_2$ share a node, i.e., the links corresponding to trips in $\mathcal{T}_2$ are a matching on S.
The proof of the reverse implication is similar. Any matching M of S uniquely determines a set of combinable trips $\mathcal{T}_2$. A feasible set $\mathcal{T}$ of possibly combined trips is then obtained from $\mathcal{T}_2$ by adding as single trips all trips whose corresponding nodes in S are not part of the matching M. □

Theorem 1. *Let $M_{\max} \subseteq L$ be a maximum matching on S. Then, the minimum cardinality of a feasible set of possibly combined trips is $n - |M_{\max}|$.*

Proof. By Lemma 1, the cardinality of any feasible set $\mathcal{T}$ of possibly combined trips is $n-|M|$, where $|M|$ is the cardinality of the matching M corresponding to the subset $\mathcal{T}_2$ of combined trips in $\mathcal{T}$. The proof then follows by observing that the minimum cardinality of a feasible set is obtained when $|M|$ is maximum, i.e., when M is a maximum matching for S. □

The proof of Theorem 1 suggests a polynomial time algorithm for solving MNT, which is reported below. The feasible set $\mathcal{T}$ of trips to be performed is initialized to the entire set of single trips T. Given the shareability network S, a maximum matching $M_{\max}$ on S is computed using, e.g., Edmond's algorithm [11]. For any edge $(T_i, T_j) \in M_{\max}$, the combined trip $T_{i,j}$ is included in $\mathcal{T}$, while the individual trips T_i and T_j are removed from $\mathcal{T}$. After all edges in $M_{\max}$ have been considered and processed as above, $\mathcal{T}$ contains a set of (possibly combined) trips of minimum size that satisfies all customers, i.e., it is a feasible trip set of minimum size. The time complexity of MAXMATCH is determined by the complexity of the matching algorithm. Considering that the shareability network is likely to be very sparse in practice, the Edmond's matching algorithm yields $O(n\sqrt{n})$ time complexity.

Algorithm MAXMATCH
Input: the shareability network $S = (T, L)$
Output: the set $\mathcal{T}$ of (possibly combined) trips to be performed
1. $\mathcal{T} = T$
2. Build a maximum matching $M_{\max}$ on S
3. **for each** $(T_i, T_j) \in M_{\max}$ **do**
4. $\quad \mathcal{T} = \mathcal{T} \cup \{T_{i,j}\}; \mathcal{T} = \mathcal{T} - \{T_i, T_j\}$
5 **return** $\mathcal{T}$

Algorithm MAXMATCH for optimally solving MNT.

Theorem 2. *Let $M_{\max_w} \subseteq L$ be a maximum weighted matching on S, where S is link weighted as described in Definition 4. Then, the feasible set of possibly combined trips of minimum total travel cost has cost*

$$c_{\min} = \sum_{i=1,\dots,n} c(T_i) - \sum_{(T_i,T_j)\in M_{\max_w}} c(T_i) + c(T_j) - c(T_{i,j}) \; .$$

Proof. By Lemma 1, the subset $\mathcal{T}_2$ of combined trips of any feasible trip set $\mathcal{T}$ corresponds to a matching M on S. For any edge $(T_i, T_j) \in M$, the travel cost reduction due to the combined trip $T_{i,j}$ with respect to the cost of the two single trips T_i, T_j is $c(T_i) + c(T_j) - c(T_{i,j})$. Thus, the total travel cost for any feasible trip set $\mathcal{T}$ is given by the total travel cost of the single trips ($\sum_{i=1,\dots,n} c(T_i)$), minus the sum of the cost savings achieved by the combined trips in M, i.e.,

$\sum_{(T_i,T_j)\in M} c(T_i) + c(T_j) - c(T_{i,j}) = \sum_{(T_i,T_j)\in M} w_{ij}$. The proof then follows by observing that, if $M_{\max_w}$ is a maximum weighted matching on S, then the sum of the cost savings is maximized, and the total travel cost of $\mathcal{T}$ is minimized. □

The algorithm WEIGHTEDMAXMATCH to find the feasible trip set of minimum travel cost is similar to MAXMATCH, the only difference being that the maximum matching algorithm in step 2 is replaced with a maximum weighted matching algorithm. For instance, we can use Edmond's algorithm for weighted matching [11] , which on a sparse graph yields time complexity $O(n^2 \log n)$.

The k-trips sharing case

In this section, we generalize the results presented in the previous section to the more challenging scenario in which an arbitrary number $k > 2$ of trips can be combined. The only assumption we make about the value of k in this section is that k does not depend on the total number of trips n, i.e., $k = O(1)$ in asymptotic notation. Considering that k is also an upper bound on taxi capacity needed to accommodate the k combined trips, assuming k a small constant is reasonable in any practical case.

We first present how a combination of up to k trips can be represented in form of a k-bounded shareability hyper-network. Some definitions are in order before proceeding further.

Definition 6. *A set $\mathcal{T}$ of* possibly combined trips*, where combined trips are composed of at most $k \geq 2$ single trips, is defined as $\mathcal{T} = \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_k$, where $\mathcal{T}_1 \subseteq T$ is a set of single trips, and $\mathcal{T}_h$, with $2 \leq h \leq k$, is a set of combined trips $T_{i_1,\ldots,i_h}$, for some $i_1, \ldots, i_h \in \{1, \ldots, n\}$.*

Definition 7 (Feasible trip set)**.** *A set $\mathcal{T}$ of possibly combined trips is* feasible *if and only if all trips in T appear once in $\mathcal{T}$, formally:*

$$\forall T_j \in T, \quad (T_j \in \mathcal{T}_1) \vee (\exists_1 h, \ell \text{ s.t. } (T_{i_1,\ldots,i_h} \in \mathcal{T}_h) \wedge (i_\ell = j)) \ .$$

Notice that also in this case, any trip $T_i \in T$ can appear only once in a feasible trip set, either as a single trip, or in a combined trip.

Definition 8. *A* hyper-network *H is a pair $H = (T, \mathcal{L})$ where $T = (T_1, \ldots, T_n)$ is a set of nodes (representing single trips in the context at hand), and $\mathcal{L}$ is a set of non-empty subsets of T called hyper-links. The* size *of a hyper-link is the number of nodes it connects. A hyper-network whose hyper-links have size $\leq k$, for some integer $k \geq 2$, is called a k-*bounded hyper-network*.*

Definition 9. *A (hyper-)matching M on the hyper-network $H = (T, \mathcal{L})$ is a subset of the hyper-links in $\mathcal{L}$ such that each node in T appears in at most one hyper-link.*

Similarly to the previous section, given a set of trips T, we can represent all possible combinations of up to k trips – defined according to some quality of service criterion – with a k-bounded hyper-network, which we call the *shareability hyper-network*. Formally, the trip hyper-newtork is defined as the k-bounded hyper-network $H = (T, \mathcal{L})$, where $L_i = (T_{i_1}, T_{i_2}, \ldots) \in \mathcal{L}$

if and only if trips $T_{i_1}, T_{i_2}, \dots$ can be combined. Notice that, if hyper-link $L_i = (T_{i_1}, T_{i_2}, \dots)$ belongs to the shareability hyper-network, so do all hyper-links formed of any subset of the nodes connected by L_i. This is due to the fact that, if, say, trip $T_{1,2,3,4}$ is feasible, so do trips $T_{1,2}$, $T_{1,2,3}$, etc. We call a hyper-link in H *maximal* if its incident nodes are not a subset of any other hyper-link in H.

We are now ready to formally define the two considered optimization problems.

Definition 10 (k-MINIMUMNUMBERTRIP – kMNT)**.** *Given the shareability hyper-network $H = (T, \mathcal{L})$, determine a feasible trip set $\mathcal{T}$ of minimum cardinality.*

Definition 11 (k-MINIMIZETOTALTRAVELCOST – kMTTC)**.** *Given the weighted shareability hyper-network $H = (T, \mathcal{L})$ where each link $L_i = (T_{i_1}, T_{i_2}, \dots) \in \mathcal{L}$ is weighted with $w_i^c = \sum_{i_j \in L_i} c(T_{i_j}) - c(T_{i_1,i_2,\dots})$, where $c(T_{i_j})$ denotes the cost of trip T_{i_j}, and $c(T_{i_1,i_2,\dots})$ the cost of the combined trip $T_{i_1,i_2,\dots}$; determine a feasible trip set $\mathcal{T}$ such that the total travel cost is minimized.*

Lemma 2. *A set $\mathcal{T}$ of possibly combined trips is feasible if and only if its subset $\mathcal{T}_c = \mathcal{T} - \mathcal{T}_1$ of combined trips is a (hyper-)matching of H.*

Proof. The proof is along the same lines of the proof of Lemma 1. □

Theorem 3. *Let $H = (T, \mathcal{L})$ be a shareability hyper-network, and assign weight $w_i = |L_i| - 1$ to each hyper-link $L_i \in \mathcal{L}$. Then, the minimum cardinality of a feasible set of possibly combined trips is $n - \sum_{L_i \in M_{\max_w}} w_i$, where $M_{\max_w}$ is a maximum weighted matching of H.*

Proof. By Lemma 2, any feasible trip set $\mathcal{T}$ uniquely defines a matching M in the shareability hyper-network H. Consider any hyper-link L_i in the matching M. By definition, w_i represents the number of trips that are saved by performing the combined trip corresponding to hyper-link L_i instead of performing all single trips in L_i. For instance, if $L_i = \{T_{i_1}, \dots, T_{i_k}\}$, the combination of k trips allows reducing the number of performed trips from k to 1; i.e., the total number of trips is reduced of $w_i = k - 1$. Based on this observation, the total number of trips performed for feasible trip set $\mathcal{T}$ equals $n - \sum_{L_i \in M} w_i$. The proof then follows by observing that the total number of trips is minimized when $\sum_{L_i \in M} w_i$ is maximized, i.e., when M is a maximum weighted matching for H. □

Unfortunately, the maximum (weighted) matching problem on k-bounded hyper-networks is NP-complete for $k > 2$ on general hyper-networks, hence finding the optimal solution to kMNT is likely computationally hard. However, a simple greedy heuristic can be used to find a k-approximation of the optimal solution in time $O(m \log m)$, where m is the number of hyper-links in the hyper-network, which yields $O(n \log n)$ complexity under our working assumption of sparse shareability hyper-network. In the greedy heuristic, a hyper-link L_i of maximum weight is added to the current matching at each iteration, and hyper-links sharing at least one node with L_i are removed from the set of candidate hyper-links for matching before proceeding to the next iteration. Observe that better approximation ratios can be obtained at the price

of increased (but still polynomial) time complexity using, for instance, the algorithm of [12] which finds a $2(k+1)/3$ approximation of the optimal solution. Note that the weighted maximum matching problem on hyper-networks is equivalent to the weighted set packing problem. The greedy algorithm for finding a k-approximation to the optimal kMNT solution is reported below.

Algorithm GREEDYKMATCHING
Input: the shareability hyper-network $H = (T, \mathcal{L})$ with weights w_i on hyper-links
Output: the set $\mathcal{T}$ of (possibly combined) trips to be performed
1. $\mathcal{T} = T$
2. Build a weighted matching M_w of H using the greedy heuristic
3. **for each** $L_i = (T_{i_1}, \ldots, T_{i_j}) \in M_w$ **do**
4. $\quad \mathcal{T} = \mathcal{T} \cup \{T_{i_1,\ldots,i_j}\}; \mathcal{T} = \mathcal{T} - \{T_{i_1}\} - \cdots - \{T_{i_j}\}$
5 **return** $\mathcal{T}$

Algorithm GREEDYKMATCHING for finding a k-approximation to kMNT.

Theorem 4. *Let $H = (T, \mathcal{L})$ be the shareability hyper-network, where each $L_i = (T_{i_1}, T_{i_2}, \ldots) \in \mathcal{L}$ is weighted with the weight $w_i^c = \sum_{i_j \in L_i} c(T_{i_j}) - c(T_{i_1,i_2,\ldots})$ representing the cost saving in performing the combined trip versus the collection of single trips. Then, the feasible set of possibly combined trips of minimum total travel cost has cost*

$$c_{\min} = \sum_{i=1,\ldots,n} c(T_i) - \sum_{L_i \in M_{\max_w}} w_i^c \,,$$

where $M_{\max_w}$ is a maximum weighted matching of H.

Proof. The proof is along the same lines of the proof of Theorem 3. □

The greedy heuristic for computing a k approximation of the optimal solution to kMTTC can be straightforwardly obtained from Algorithm GREEDYKMATCHING by using weights w_i^c instead of w_i to label hyper-links in the shareability hyper-network.

Building the shareability network

In this section, we describe a method for producing the shareability (hyper-)network, given a set of single trips $T = \{T_1, \ldots, T_n\}$ and a quality of service criterion Δ. We present in detail the method for $k = 2$, and shortly describe how the technique can be generalized to arbitrary values of k.

Each trip $T_i \in T$ is characterized by the following quantities: the trip origin o_i and destination d_i, that we can think of as pairs of (lat, lon) coordinates; the start time st_i; and the arrival time at_i. We start defining a notion of feasible trip combination based on a quality of service criterion Δ.

Definition 12. *The combined trip $T_{i,j}$ is* feasible *if and only if a trip route can be found such that the following conditions are satisfied:*

a) $st_i \leq pt_i \leq st_i + \Delta$*;*

b) $st_j \leq pt_j \leq st_j + \Delta$*;*

c) $dt_i \leq at_i + \Delta$*;*

d) $dt_j \leq at_j + \Delta$*;*

where pt_x is the pickup time at o_x in the combined trip, and dt_x is the delivery time at d_x in the combined trip.

The above definition is motivated by the fact that a customer might be willing to wait at most some extra time Δ at her pickup location (and in general she might not be able to show up at o_i before time st_i), as well as to arrive at destination with delay at most Δ (early arrivals are likely not to be a problem for customers).

Theorem 5. *Building the shareability network $S = (T, L)$ starting from the trip set $T = \{T_1, \ldots, T_n\}$ requires $O(n^2)$ time.*

Proof. In the worst-case, we have to consider all $O(n^2)$ possible pairs of trips T_i, T_j. For each pair, the feasibility condition for the combined trip $T_{i,j}$ can be verified in $O(1)$ as follows. Observe that only four routes are possible for trip $T_{i,j}$: $o_i \rightarrow o_j \rightarrow d_i \rightarrow d_j$, $o_i \rightarrow o_j \rightarrow d_j \rightarrow d_i$, $o_j \rightarrow o_i \rightarrow d_i \rightarrow d_j$, and $o_j \rightarrow o_i \rightarrow d_j \rightarrow d_i$. Let us consider a specific route, e.g., $o_i \rightarrow o_j \rightarrow d_i \rightarrow d_j$. Condition $a)$ for feasibility is always satisfied by setting a pickup time at o_i in the desired time window. The pickup time at o_j can then be computed as follows: $pt_j = pt_i + tt(o_i, o_j)$, where $tt(x, y)$ denotes the travel time between x and y. The delivery time at d_i is defined as follows: $dt_i = pt_j + tt(o_j, d_i)$. Finally, the delivery time at d_j is defined as $dt_j = dt_i + tt(d_i, d_j)$. Thus, the feasibility condition for route $o_i \rightarrow o_j \rightarrow d_i \rightarrow d_j$ can be verified by checking whether a value of pt_i that simultaneously satisfies the four conditions below exists, which requires $O(1)$ time:

$$st_i \leq pt_i \leq st_i + \Delta \quad \text{(S1)}$$

$$st_j \leq pt_i + tt(o_i, o_j) \leq st_j + \Delta \quad \text{(S2)}$$

$$pt_i + tt(o_i, o_j) + tt(o_j, d_i) \leq at_i + \Delta \quad \text{(S3)}$$

$$pt_i + tt(o_i, o_j) + tt(o_j, d_i) + tt(d_i, d_j) \leq at_j + \Delta \quad \text{(S4)}$$

The feasibility conditions for the other routes can be verified similarly. If there exists at least one route which satisfies the feasibility condition, then trip $T_{i,j}$ is feasible, and link (T_i, T_j) is included in the trip graph. Otherwise, trips T_i and T_j cannot be combined.
Observe that the number of trip pairs to consider for combination can be reduced by considering only trip pairs T_i, T_j such that: $i)$ $st_j \leq at_i + \Delta$; and $ii)$ $st_i \leq at_j + \Delta$. Simultaneously

satisfying conditions $i)$ and $ii)$ is a necessary (but not sufficient) condition for feasibility of trip $T_{i,j}$. In practice, this heuristic considerably reduces the running time of the shareability network construction algorithm, although the worst-case time complexity remains $O(n^2)$. □

The algorithm for building the trip graph reported in the proof of Theorem 5 can be extended in a straightforward way to the case of combinations of up to k trips, yielding a time complexity of $O(n^k)$; in particular, all possible routes connecting k origins with k destinations, subject to the condition that each origin must precede the respective destination in the route, must be considered. The number of possible such routes grows exponentially with k, which is however assumed to be a small constant in our model. For instance, 60 possible routes connecting origins with destinations must be considered when $k = 3$.

Runtime and feasibility considerations

Although we have proven above theoretically that the trip sharing problem can be solved by maximum matching algorithms with polynomial time complexity, it is important to demonstrate that the computational implementation indeed runs fast enough on practical problem sizes and that the solution is feasible to operate in an actual real-time taxi sharing system. On average, in every two minutes there are about 600 trips requested in NYC, which constitutes the typical number of nodes in the shareability network that have to be matched optimally. To account for possible extraordinary spikes in the demand, we measured the runtime for sharebility networks of size 10,000. Here, the runtime of the software implemented in the programming language C on a Linux-based workstation equipped with i7-3930K CPU and 32GB of RAM is less than 0.1 seconds. For the more realistic size of 1,000 nodes the runtime is even less than 0.01 seconds. These measurements show that the proposed graph matching-based solution is highly efficient and feasible for an online taxi sharing service. Further, the procedures for inserting/deleting nodes and corresponding links in the shareability network are also very efficient, with a running time that increases only linearly with the number of nodes in the network.

Oracle and Online model

Trip sharing opportunities are investigated according to two different models, called the *Oracle* and the *Online* model. The difference between the two models lies in the set of links in the shareability network that are considered for computing the maximum matching. In the Oracle model, all possible links between shareable trips as determined by spatial and temporal conditions are retained. Thus, two trips T_1, T_2 are considered to be shareable also if their starting times are separated by a long time interval (say, 30 min), as long as the two trips can be combined without imposing delays exceeding Δ on both trips. Note that this is in principle possible if, say, T_1 is a very long trip, and the pickup location of T_2 is close to the trajectory of T_1. Since in this paper we consider only static trip sharing, combining trips in a situation like the one described above would require the presence a reservation system, in which requests for taxi trips

are issued well ahead of time (e.g., at least $30\ \mathrm{min}$ in the example above). Since in many cases a reservation system is not present or not allowed by regulations (as it is the case in the city of NY), we have introduced also the Online model, which can be easily turned into a practical real time, on-demand taxi system. The idea is to reduce the number of links in the shareability network, by filtering out trip sharing opportunities for trips T_1, T_2 whose starting times are more than δ apart. In other words, we retain in the shareability network only trip sharing opportunities that can be exploited with a real-time, on-demand taxi system.

Computing travel times

Knowledge of estimated travel times between arbitrary origin/destination in the road map is a pre-requisite for checking the trip sharing conditions, and, hence, to build the shareability network. Since we cannot use directly GPS taxi traces for this purpose due to lack of trajectory and speed information in the data set, we designed a travel time estimation heuristic starting from the pickup/drop off times recorded in the data set.

Given is a set of actually performed trips $\mathcal{T} = \{T_1, \ldots, T_k\}$, where each trip $T_i = (o_i, d_i, tt_i)$ is defined by an origin location o_i, a destination location d_i, and a travel time tt_i. While in the original data set origin and destination of a trip are defined as raw GPS (lat, lon) coordinates, in the following we assume origin and destination of a trip are taken from the set $\mathcal{I}$ of street intersections in the road map. To convert raw GPS coordinates into an intersection in $\mathcal{I}$, we associate o_i (or d_i) to the closest intersection based on geodesic distance, subject to the condition that the distance to the closest intersection is below a threshold such as twice the average GPS accuracy, set to $100\,\mathrm{m}$. Thus, in the following we assume o_i and d_i are indeed distinct elements of the set $\mathcal{I}$ of possible intersections in the road map, i.e., $\forall T_i \in \mathcal{T}, o_i, d_i \in \mathcal{I}$. We also define the set $\mathcal{S} = \{S_1, \ldots, S_h\}$ of *streets* as the set of all road segments connecting two adjacent intersections in the road map.

Given the trip set $\mathcal{T}$ as defined above, the problem to solve is estimating the travel time x_i for each street $S_i \in \mathcal{S}$, in such a way that the average relative error (computed across all trips) between the actual travel time tt_i and the estimated travel time et_i for trip T_i computed starting from the x_is (compound with a routing algorithm) is minimized[1]. Once error minimizing travel times for each street in $\mathcal{S}$ are determined, the travel time between any two intersections $I_i, I_j \in \mathcal{I}$ can be computed starting from the x_is, using a routing algorithm that minimizes the travel time between any two intersections. Besides the trip set $\mathcal{T}$, we are also given the array $Le = (l_i)$ of the lengths of the streets in $\mathcal{S}$.

In the following, we define the problem at hand more formally. First, we partition the trip set in time sliced subsets $\mathcal{T}_1, \ldots, \mathcal{T}_{24}$, where subset $\mathcal{T}_i$ contains all trips whose starting time is in hour i of the day. Finer partitioning (e.g., per hour and weekday, per hour and weekday and month, etc.) is possible, if needed. In the following, to simplify notation, we re-define $\mathcal{T}$ as any of the time-sliced subsets $\mathcal{T}_i$. In fact, the travel time estimation process

[1]Formally, the average relative error is defined as $\epsilon_i = |tt_i - te_i|/tt_i$.

can be performed independently on each of the time-sliced trip subsets. When a time-sliced trip set $\mathcal{T}$ is considered, classes $\mathcal{T}^1, \ldots, \mathcal{T}^h$ of *equivalent trips* are formed, where two trips T_u, T_v are said to be equivalent if and only if $(o_u = o_v) \wedge (d_u = d_v)$. Notice that, under the assumption that the routing algorithm is deterministic (i.e., it always computes the same route given the same starting and ending intersections I_i and I_j), the set of streets in the optimal route from origin to destination is the same for any two trips $T_u, T_v \in \mathcal{T}^{i,j}$, where $\mathcal{T}^{i,j}$ is the class of trips with origin I_i and destination I_j. Thus, all the trips in equivalence class $\mathcal{T}^{i,j}$ can be considered as multiple samples of the travel time on the same set of streets. All trips in $\mathcal{T}^{i,j}$ are then replaced by a single trip $T_{i,j}$ with corresponding origin and destination, and travel time $\bar{tt}_{i,j}$ equal to the average of the travel times of all trips in $\mathcal{T}^{i,j}$. After this step is performed for all equivalence classes, we are left with an aggregate set $\mathcal{T}_{\text{agg}}$ of singleton, non-equivalent trips $T_{i,j}$, and corresponding travel times $\bar{tt}_{i,j}$.

The travel time estimation heuristic is reported below. Initially, trips are filtered to remove "loop" trips (i.e., trips with the same origin and destination), as well as excessively "short" or "long" trips. After a step in which initial routes are computed using a pre-selected initial speed v_{init} (the same for all streets), a second trip filtering step is performed, in which excessively "fast" and "slow" trips are removed from the travel time estimation process. The rationale for this filtering is removing "noisy" data which could have been resulted from very specific conditions (say, a snowstorm could have caused many slow trips). Including "noisy" data in the travel time estimation process would bias the estimation process to partially compensate for "noisy" trips, increasing the error experienced in the remaining portion of trips.

An iterative process is started after the second trip filtering step. The iterative process is composed of two nested iterations. In the outer iteration, new routes for the trips are computed based on the updated travel time estimation of street segments. After routes are computed, new trip travel time estimations are determined, and the average relative error across all trips is computed. Furthermore, an offset value is computed for each street segment, indicating whether travel times of all trips in which a street segment is included are under- or over-estimated. Then, an inner loop is started, with the purpose of refining street travel time estimations based on the computed offset values: an increase/decrease step $\mathtt{k}$ is initialized, and used to tentatively change street travel time estimates based on the offset value (tentative updated trip travel times are accepted only if the resulting average speed v on the trip is such that $0.5\,\mathrm{m/sec} < v < 30\,\mathrm{m/sec}$). The tentative estimations are accepted if the newly computed average relative error is decreased with respect to the current value. Otherwise, another iteration of the inner loop is started with a smaller value of $\mathtt{k}$. This process is repeated until either the street travel time estimations are updated, or the value of $\mathtt{k}$ has reached a specified minimum value. The outer iterative process terminates when there is no updated street travel time estimation after the execution of the inner loop.

After the iterative process, the algorithm has produced a travel time estimation for each street included in at least one optimal route for at least one trip in the data set (set $\mathcal{S}_{\text{trip}}$). The travel time for the remaining streets is then computed according to a simple heuristic: the travel time for each street s having an intersection in common with at least one street in $\mathcal{S}_{\text{trip}}$ is estimated

based on the average speed estimated in adjacent streets, i.e., streets s' such that s and s' share an intersection. This process is repeated until the travel time on all streets can be estimated. Finally, at step 7 the travel time between any two possible intersections I_i, I_j in the street map is computed by first computing the optimal route between I_i and I_j using Dijkstra algorithm with the estimated trip travel times, and then computing the travel time by summing up the travel time of the streets in the optimal route. Notice that we use the Dijkstra algorithm [13] (repeated $|\mathcal{I}|^2$ times) to compute all-to-all shortest paths instead of the classical Floyd-Warshall algorithm since the graph corresponding to the street network is very sparse. Thus, repeating $|\mathcal{I}|^2$ times the Dijkstra algorithm yields $O(|\mathcal{I}|^2 \log |\mathcal{I}|)$, which is lower than the $O(|\mathcal{I}|^3)$ complexity of Floyd-Warshall.

The travel time estimation algorithm has been executed on the set of about 150 millions trips performed in New York City during weekdays, in year 2011. The performance of the travel time estimation algorithms for the 24 trip classes (corresponding to time of day) is summarized in Table S1. The table reports the average relative error computed on all trips retained after the filtering steps, the percentage of trips retained in the data set after filtering, and the number of streets included in at least one optimal route. As seen from the table, the algorithm provides travel time estimations incurring an average relative error of 15%. The vast majority of trips is retained in the data set after filtering (more than 97% on the average). Furthermore, the vast majority of street segments are included in at least one optimal route: considering that the total number of (directed) street segments in Manhattan is 9452, on average 91.7% of the streets are included in at least one optimal route. For the remaining streets, step 6 of the algorithm is used to estimate street travel time.

To study the travel speeds estimated by our algorithm we calculated travel speeds across different times of the day. The travel time estimations are reasonable, with a relatively lower average speed of around $5.5\,\mathrm{m/sec}$ estimated during rush hours (between 8am and 3pm), and peaks around $8.5\,\mathrm{m/sec}$ at midnight. Further evidence for a reasonable estimation is highlighted also by Fig. S2, reporting the estimated travel speed on each street segment at four different times of day: 0am, 8am, 4pm, and 22pm. As expected, travel speeds tend to reduce during daytime. Also, the algorithm is able to faithfully model the higher speed on highways (on the left-hand side of Manhattan).

Algorithm for travel time estimation

Input: the (sub)set $\mathcal{T}$ of performed trips;the set $\mathcal{I}$ of intersections;
the set $\mathcal{S}$ of streets; the vector Le of lengths for streets in $\mathcal{S}$

Output: a travel time estimation matrix $ET(i,j)$, where $et_{i,j}$ is the estimated time for traveling from intersection I_i to intersection I_j

1. **Equivalent trip reduction**
   - group in class $\mathcal{T}^{i,j}$ all trips T_u such that $o_u = I_i$ and $d_u = I_j$
   - for each class $\mathcal{T}^{i,j}$, replace all trips in $\mathcal{T}^{i,j}$ with a single trip $T_{i,j}$ with $o_{i,j} = I_i, d_{i,j} = I_j$, and $tt_{i,j} = \bar{tt}_{i,j} = \frac{\sum_{T_u \in \mathcal{T}^{i,j}} tt_u}{|\mathcal{T}^{i,j}|}$
   - let $\mathcal{T}_{\mathrm{agg}}$ be the collection of trips $T_{i,j}$
2. **First trip filtering**
   - for each $T_{i,j} \in \mathcal{T}_{\mathrm{agg}}$, remove $T_{i,j}$ from $\mathcal{T}_{\mathrm{agg}}$ if $i = j$ *//remove "loop" trips*
   - for each $T_{i,j} \in \mathcal{T}_{\mathrm{agg}}$, remove $T_{i,j}$ from $\mathcal{T}_{\mathrm{agg}}$ if $(\bar{tt}_{i,j} < 2min)$ or $(\bar{tt}_{i,j} > 1h)$ *//remove "short" and "long" trips*
3. **Initial route computation**
   - for each $S \in \mathcal{S}$, set same initial speed $v_S = v_{\mathrm{init}}$; set travel time to $t_S = \frac{L(S)}{v_S}$
   - for each $T_{i,j} \in \mathcal{T}_{\mathrm{agg}}$, compute optimal route $I_i \rightarrow I_j$ using Dijkstra algorithm
   - for each $T_{i,j} \in \mathcal{T}_{\mathrm{agg}}$, let $\mathcal{S}^{i,j} = \{S_1^{i,j}, \ldots, S_k^{i,j}\}$ be the set of streets in the optimal route for $T_{i,j}$
4. **Second trip filtering**
   - for each $T_{i,j} \in \mathcal{T}_{\mathrm{agg}}$, compute the average speed $as_{i,j} = \frac{\sum_h L(S_h^{i,j})}{tt_{i,j}}$
   - for each $T_{i,j} \in \mathcal{T}_{\mathrm{agg}}$, remove $T_{i,j}$ from $\mathcal{T}_{\mathrm{agg}}$ if $(as_{i,j} < 0.5\,\mathrm{m/sec})$ or $(as_{i,j} > 30\,\mathrm{m/sec})$ *//remove "slow" and "fast" trips*
5. **Iterative steps**
   5.1 set `again`=**true**
   5.2 **while** `again` **do**
   - `again`=**false**
   - for each $T_{i,j} \in \mathcal{T}_{\mathrm{agg}}$, compute optimal route $I_i \rightarrow I_j$ using Dijkstra algorithm
   - for each $T_{i,j} \in \mathcal{T}_{\mathrm{agg}}$, let $\mathcal{S}^{i,j} = \{S_1^{i,j}, \ldots, S_k^{i,j}\}$ be the set of streets in the optimal route for $T_{i,j}$
   - for each $T_{i,j} \in \mathcal{T}_{\mathrm{agg}}$, compute $et_{i,j} = \sum_{S \in \mathcal{S}^{i,j}} t_S$ *//trip travel time estimation*
   - let $\mathcal{S}_{\mathrm{trip}} = \bigcup_{\mathcal{T}^{i,j} \in \mathcal{T}_{\mathrm{agg}}} \mathcal{S}^{i,j}$
   - `RelErr`=$\sum_{T_{i,j} \in \mathcal{T}_{\mathrm{agg}}} \frac{|et_{i,j} - \bar{tt}_{i,j}|}{\bar{tt}_{i,j}}$
   - for each $S \in \mathcal{S}_{\mathrm{trip}}$, let $T_S = \{T_{i,j} \in \mathcal{T}_{\mathrm{agg}} | S \in \mathcal{S}^{i,j}\}$ *//set of trips including S in the current route*
   - for each $S \in \mathcal{S}_{\mathrm{trip}}$, compute $O_S = \sum_{T_{i,j} \in T_S} (et_{i,j} - \bar{tt}_{i,j}) \cdot |\mathcal{T}_{i,j}|$ *//offset computation*
   - `k`=1.2
   5.3 **while true do**
   - for each $S \in \mathcal{S}_{\mathrm{trip}}$, do the following
   - if $O_S < 0$, then $t_S = k \cdot t_S$; else $t_S = \frac{t_S}{k}$ *//street travel time estimate is increased/reduced based on offset*
   - for each $T_{i,j} \in \mathcal{T}_{\mathrm{agg}}$, compute $et'_{i,j} = \sum_{S \in \mathcal{S}^{i,j}} t_S$ *//tentative updated trip travel time estimation*
   - `NewRelErr`=$\sum_{T_{i,j} \in \mathcal{T}_{\mathrm{agg}}} \frac{|et'_{i,j} - \bar{tt}_{i,j}|}{\bar{tt}_{i,j}}$ *//compute new relative error*
   - if `NewRelErr`<`RelErr` then do the following *//new estimates better than previous ones*
   - for each $T_{i,j} \in \mathcal{T}_{\mathrm{agg}}$, $et_{i,j} = et'_{i,j}$ *//update travel time estimates*
   - `RelErr`=`NewRelErr`
   - `again`=**true**; **goto** step 5.2 *//perform another iteration*
   - else *// new estimates worse than previous ones*
   `k`=1 + (`k`−1) · 0.75 *//reduce the street travel time increase/decrease step*
   if `k`< 1.0001 then exit from loop at step 5.3 *//if `k` is too small, exit from inner loop*
   else **goto** step 5.3 *//otherwise, perform another iteration with smaller `k`*
6. **Computation of estimated travel time for remaining streets**
   - $\mathcal{ES} = \mathcal{S}_{\mathrm{trip}}$; $\mathcal{NS} = \mathcal{S} - \mathcal{S}_{\mathrm{trip}}$
   - let $N(S)$ be the set of streets sharing an intersection with street S
   - for each $S_i \in \mathcal{NS}$ compute $n_{S_i} = |N(S_i) \cap \mathcal{ES}|$
   - order the streets in $\mathcal{NS}$ in decreasing order of n_{S_i}
   - for each $S_i \in \mathcal{NS}$ in the ordered sequence
   $v_{S_i} = \frac{\sum_{S_j \in N(S_i) \cap \mathcal{ES}} v_{S_j}}{|N(S_i) \cap \mathcal{ES}|}$; $t_{S_i} = \frac{L(S_i)}{v_{S_i}}$; $\mathcal{ES} = \mathcal{ES} \cup \{S_i\}$;$\mathcal{NS} = \mathcal{NS} - \{S_i\}$
   - repeat above step until $\mathcal{NS} = \emptyset$
6. **Travel time estimation**
   -for each possible pair of intersections (I_i, I_j), compute optimal route $I_i \rightarrow I_j$ using Dijkstra algorithm with estimated travel time t_S for each street S
   - let $\mathcal{S}^{i,j}$ be the set of streets in the optimal route for (I_i, I_j)
   - $ET(i,j) = \sum_{S_h \in \mathcal{S}^{i,j}} \frac{L(S)}{v_S}$
   - **return** ET

Travel time estimation algorithm.

Robustness of day of week (Oracle model)

To assess whether there is any noticeable difference in terms of trip sharing opportunities between weekend and week days, we have repeated the analysis above for the 104 weekend days. There is no major difference in terms of trip sharing opportunities in weekend versus weekdays. However, the average number of trips per day during weekend days is about 17% lower than that during week days ($\approx 350K$ versus $\approx 418K$ trips per day). Only minimal differences in total trip travel time savings between week and weekend days are observed, with slightly better savings achieved during weekdays. As shown next, this is due to the strong relation between trip sharing opportunities and the number of performed trips.

Shareable trips versus trips per day (Oracle model)

To better understand the relationship between trip sharing opportunities and number of trips performed in a day, we have ordered the days for increasing number of performed trips, and plotted the corresponding percentage of shared trips in the day. The resulting plot is reported in Fig. 3C in the main text. Typical days in New York City feature around 400,000 trips with almost near maximum shareability. Days with a small number of trips are rare and happen mostly during special events. For example, the most noticeable drop in trip sharing opportunities occurs on day 240, August 28th 2011, during which hurricane Irene hit New York City. On this (weekend) day, only about $26,500$ trips were performed, and trip sharing opportunities dropped to about 87% when $\Delta = 2\,\mathrm{min}$. As such, data points below 300,000 trips per day are too sparse to make statistically reasonable assessments. Hence we have generated additional low density situations by subsampling our dataset, randomly removing various fractions of vehicles from the system in the following way: For each day in the data set, we randomly selected a percentage c of the taxis in the trace, and deleted the corresponding trips from the data set. We varied c from 95% down to 1%, generating a number of trips per day as low as 1,962.

To the set of resulting shareability values we have fit a saturation curve of the form $f(x) = \frac{Kx^n}{1+Kx^n}$, where K and n are two (non-integer) parameters, Fig. 3C in the main text. Curves of this form appear in the well-known Hill equation in biochemistry, describing saturation effects in the binding of ligands to macromolecules and in similar processes [14]. Fits to both the shared trip maximization and time minimization conditions match very well (for both, $R^2 > 0.99$), we used a standard Levenberg-Marquardt algorithm for obtaining least squares estimates. The best fit parameters read $K = 1.1 \times 10^{-4}$, $n = 0.92$ for time minimization, and $K = 1.5 \times 10^{-6}$, $n = 1.39$ for shared trip maximization. Since for time minimization we have $n \approx 1$, the fit works here almost as well (again $R^2 > 0.99$) with the functional form $f(x) = \frac{Kx}{1+Kx}$ that has only the one parameter K, known as the Langmuir equation [15], with $K = 4.4 \times 10^{-5}$. The Langmuir equation describes the relationship between the concentration of a gas (or compound) adsorbing to a solid surface (or binding site) and the fractional occupancy of the surface. Since an increasing density of taxis – the "particles" – implies that more trip pairs – the "surface" – can be covered, the Langmuir equation can thus be seen as an analogy to the saturation effects in

shareability if a homogeneous distribution of trips and taxis is assumed. The second parameter n that appears in the Hill equation is used as a measure for cooperative binding in enzyme kinetics: If $n > 1$, an enzyme which has already a bound ligand increases its affinity for other ligand molecules. It is unclear if the analogy can be stretched to understand why $n = 1.39$ works best for shared trip maximization. In any way, the fast, hyperbolic saturation implies that taxi sharing could be effective even in cities with taxi vehicle densities much lower than New York, and in case of low market penetration of the sharing system a high return on investment.

Increasing the number of shared trips

We next investigate what happens when we increase the number k of sharable trips from 2 to 3. We remark that the computational complexity of the trip matching task with $k = 3$ is orders of magnitude higher than the same task with $k = 2$, for the following reasons:

- The computation of the shareability hyper-network is challenging. In fact, we now have to compare triplets, instead of pairs, of candidate trips. For each triplet, we have 60 possible valid routes connecting the three sources/destinations of the trips, instead of 4 possible routes with $k = 2$. For each valid route, we then have to check whether the trips can actually be shared, meaning that the computational time for calculating the shareability network with $k = 3$ is at least 15 times higher than that needed to compute the trip sharing graph with $k = 2$.

- We now have to solve a matching problem on hyper-networks, instead of on simple networks. While matching on graphs can be solved in polynomial time, matching on general hyper-networks belongs to the class of NP-hard problems, i.e. problems that are "difficult to solve". To get around this computational challenge, we use a greedy, polynomial-time heuristic that first builds the maximum matching considering only triplets of trips, then applies standard matching on the remaining trips. This heuristic is known to build a solution which is, in the worst-case, within a constant factor from the optimal solution.

To tackle these computational challenges, we computed the number of shared trips and the fraction of saved travel time only in the Online model, and for selected days of the year. Notice that in the Online model trips can be shared only when their starting times are within a temporal window of δ, thus significantly reducing the number of candidate trips for sharing (and, hence, computational time needed to compute the shareability hyper-network) with respect to the Oracle model in which also trips with starting times in time windows larger than δ can be shared.

We first present the results referring to a day (day 300) in which about $450,000$ trips were performed, which is about the average number of trips per day recorded in our data set. Figures 2D and E in the main text report the percentage of saved taxi trips and of saved travel time as a function of the quality of service parameter Δ, when the time window parameter δ is set to $1\,\mathrm{min}$. As seen from the figure, increasing the number of sharable trips provides some

benefit only when the quality of service parameter Δ is large enough for such trips taxi sharing opportunities to become available. This value of Δ is approximately equal to $150\,\mathrm{sec}$. For larger values of Δ, the advantage of triple trip sharing versus double trip sharing becomes perceivable. When $\Delta = 300\,\mathrm{sec}$, the number of saved taxi trips is increased from about 50% with $k = 2$ to about 60% with $k = 3$. While with $k = 2$ nearly all trips can be shared, resulting in about halving the number of performed trips, relatively less trips can be combined in a triple trips when $k = 3$. In fact, the achieved percentage of saved trips with $k = 3$ is 60%, which is below (but not too much) the percentage of 66.6% that would result if all trips would be shared in a triple trip.

Similarly to the percentage of saved trips, also when the percentage of total traveled time is considered the difference between double and triple trip sharing becomes perceivable only for values of $\Delta \geq 150\,\mathrm{sec}$. Increasing the number of shared trips from 2 to 3 allows a further saving of about 10% in terms of total traveled time, which is achieved when $\Delta = 300\,\mathrm{sec}$.

Figure S3 reports the percentage of saved taxi trips and of saved total travel time as a function of the time window parameter δ, for two specific values of Δ. While increasing δ beyond $120\,\mathrm{sec}$ provides little benefits in terms of saved taxi trips when $k = 2$, we still can observe some benefit for $\delta > 120\,\mathrm{sec}$ with $k = 3$. This is due to the fact that with $\delta = 120\,\mathrm{sec}$ we already obtain near-ideal performance when $k = 2$, corresponding to halving the number of trips. When $k = 3$, there is more room for improvement, and a near-ideal performance is approached only when $\delta = 180\,\mathrm{sec}$ and $\Delta = 5\,\mathrm{min}$. This means that all triple trip sharing opportunities can be exploited only with relatively "patient" taxi customers. Concerning percentage of saved travel time, we observe that triple trip sharing achieves as far as 45% saving, which is significantly higher than that achieved by double trip sharing. However, similar to the percentage of taxi trips, such high savings can be obtained only with relatively "patient" taxi customers.

Finally, we compare the potential of triple versus double taxi trip sharing in a relatively less crowded day (day 250), when only $\approx 250,000$ trips were performed. Figure S3 reports the achieved reduction in number of taxi trips and in total travel time as a function of δ, when $\Delta = 5\,\mathrm{min}$. While with $k = 2$ near-ideal taxi sharing can be achieved also with low taxi traffic, with $k = 3$ a higher number of taxi trip requests is needed to fully exploit the potential of triple trip sharing. The situation is different in terms of saved total travel time, which is consistently benefiting from a higher number of taxi requests for both double and triple trip sharing.

Summarizing, based on the analysis above we can state that triple trip sharing does provide substantial benefits versus double trip sharing, but for this to occur we need a reasonable number of taxi requests, and relatively "patient" taxi customers, for which waiting for a few minutes at taxi request time and upon arrival at destination is acceptable. With "impatient" customers, double trip sharing is much more effective than triple trip sharing: it is computationally efficient, and provides nearly the same performance as triple trip sharing. Since the benefits to the community in terms of reduced number of taxis and reduced pollution with triple trip sharing versus double trip sharing are considerable, an interesting question raised by our analysis is whether the New York City municipality can design a fare system that motivates customers to be "patient".

References

[1] Bloomberg, M. & Yassky, D. New york city 2014 taxicab factbook (2014).

[2] Grush, B. The case against map-matching. *Eur. J. Nav.* **6**, 2–5 (2008).

[3] Yamamoto, K., Uesugi, K. & Watanabe, T. Adaptive routing of cruising taxis by mutual exchange of pathways. In *Knowledge-Based Intelligent Information and Engineering Systems*, 559–566 (Springer, 2008).

[4] Li, B. *et al.* Hunting or Waiting? Discovering Passenger Finding Strategies from a Large-scale Real-world Taxi Dataset. In *Proc. IEEE* (2011).

[5] Yuan, J., Zheng, Y., Xie, X. & Sun, G. Driving with knowledge from the physical world. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 316–324 (ACM, 2011).

[6] Ma, S., Zheng, Y. & Wolfson, O. T-share: A large-scale dynamic taxi ridesharing service. In *Proc. of ICDE* (2013).

[7] Berbeglia, G., Cordeau, J. F. & Laporte, G. Dynamic Pickup and Delivery Problems. *Eur. J. Op. Res.* **202**, 8–15 (2010).

[8] Horn, M. *Transportation Research C* **10**, 35 (2002).

[9] Tong, H., Hung, W. & Cheung, C. On-road motor vehicle emissions and fuel consumption in urban driving conditions. *Journal of the Air & Waste Management Association* **50**, 543–554 (2000).

[10] Kean, A. J., Harley, R. A. & Kendall, G. R. Effects of vehicle speed and engine load on motor vehicle emissions. *Environmental Science & Technology* **37**, 3739–3746 (2003).

[11] Galil, Z. Efficient Algorithms for Finding Maximum Matching in Graphs. *ACM Comp. Surv.* **18**, 23–38 (1986).

[12] Chandra, B. & Halldorsson, M. Greedy Local Improvement and Weighted Set Packing Approximation. *J. Alg.* **39**, 223–240 (2001).

[13] Cormen, T. H., Leiserson, C. E. & Rivest, R. L. *Introduction to Algorithms* (MIT Press and McGraw-Hill, 1990).

[14] Hill, A. The possible effects of the aggregation of the molecules of haemoglobin on its dissociation curves. *J physiol* **40**, iv–vii (1910).

[15] Langmuir, I. The constitution and fundamental properties of solids and liquids. part i. solids. *Journal of the American Chemical Society* **38**, 2221–2295 (1916).

Hour	**Avg. Rel. Error**	**% trips after filtering**	$\lvert\mathcal{S}_{\text{trip}}\rvert$
0	0.1541	94.91	8582
1	0.1301	96.64	8664
2	0.1433	97.76	8781
3	0.1463	98.12	8673
4	0.1438	98.16	8707
5	0.1448	98.17	8443
6	0.1517	98.04	8485
7	0.1535	98.04	8554
8	0.1560	98.01	8600
9	0.1541	97.97	8596
10	0.1568	97.96	8629
11	0.1644	97.85	8650
12	0.1639	97.97	8833
13	0.1547	98.12	8820
14	0.1486	98.20	8622
15	0.1553	98.19	8866
16	0.1388	98.13	8687
17	0.1486	98.05	8853
18	0.1457	97.86	8845
19	0.1540	97.61	8718
20	0.1602	97.23	8698
21	0.1649	96.85	8600
22	0.1693	96.54	8641
23	0.1799	95.78	8578
avg	0.1534	97.59	8671.9

Table S1: Summary of travel time estimation performance.

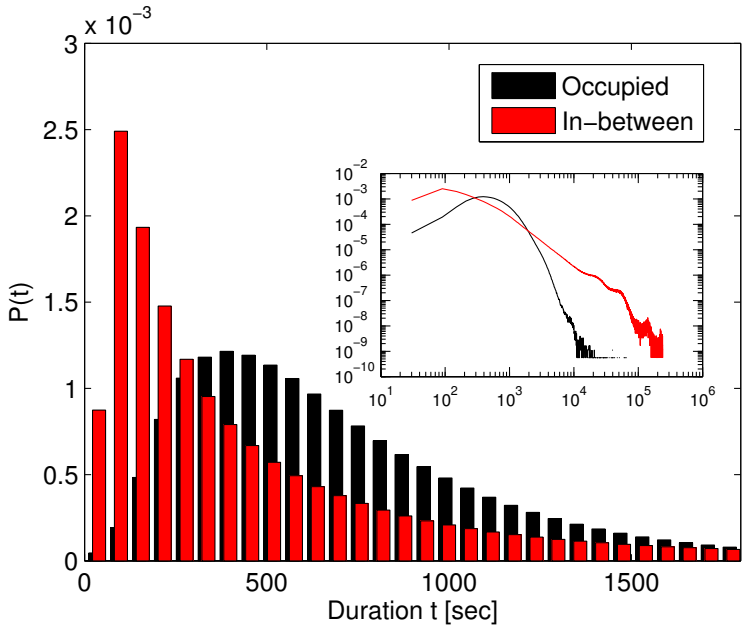

Figure S1: Empirical probability distribution of durations of occupied trips, and of the time spans in-between the occupied trips which comprise both empty trips and all activities where taxis are not being used for passenger transport such as shift changes, lunch breaks, vehicle maintenance. Specific data describing the empty trips alone is not available. The inset shows the same two histograms in log-log scale for all measured values. The in-between durations peak below two minutes, much lower than the durations of occupied trips which peak at around six minutes, showing that taxis tend to find new passengers relatively quickly.

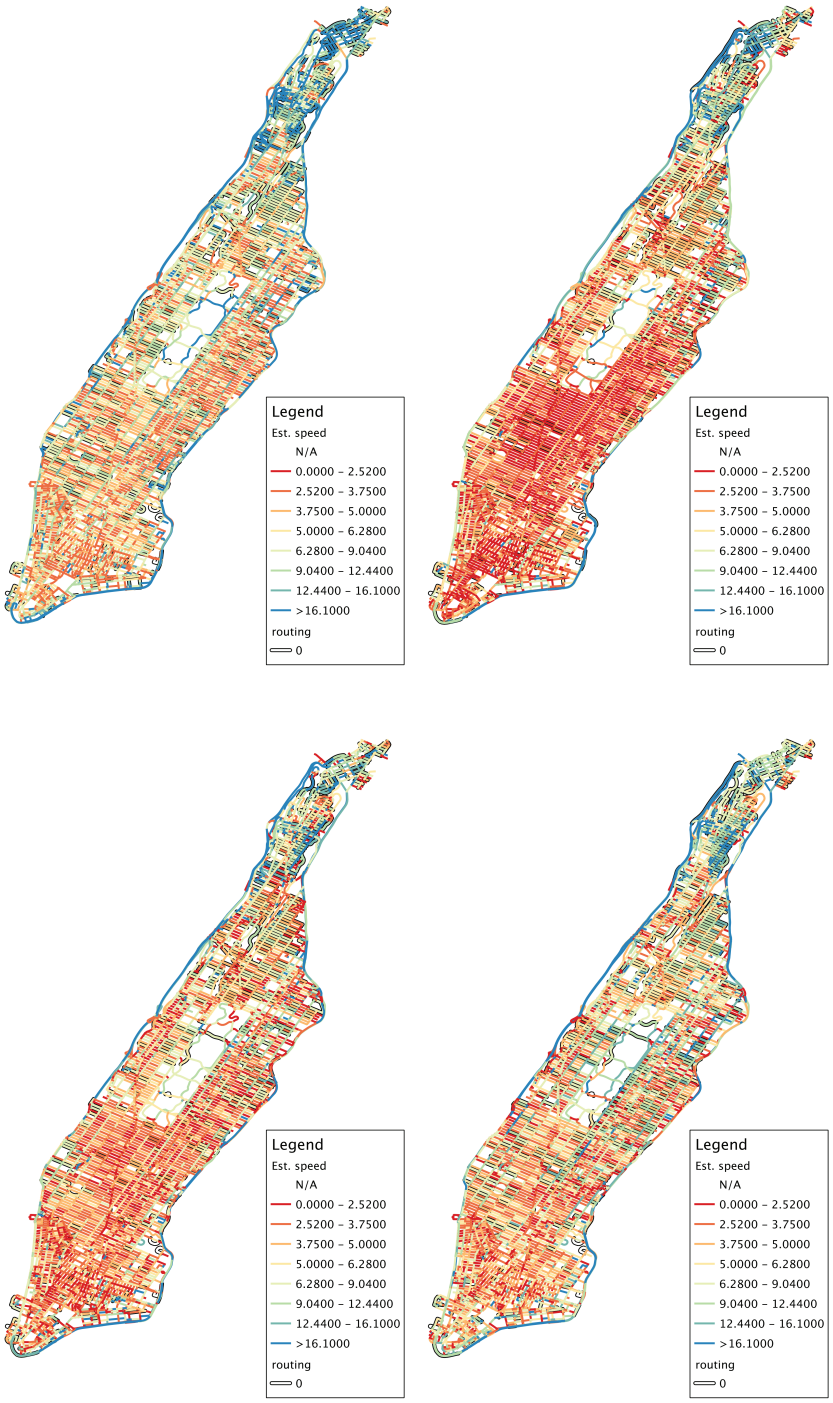

Figure S2: Estimated speed map at 0am (top left) and at 8am (top right). Estimated speed map at 4pm (bottom left) and 10pm (bottom right). Travel time for streets in bold (routing) is estimated at step 6 of the algorithm.

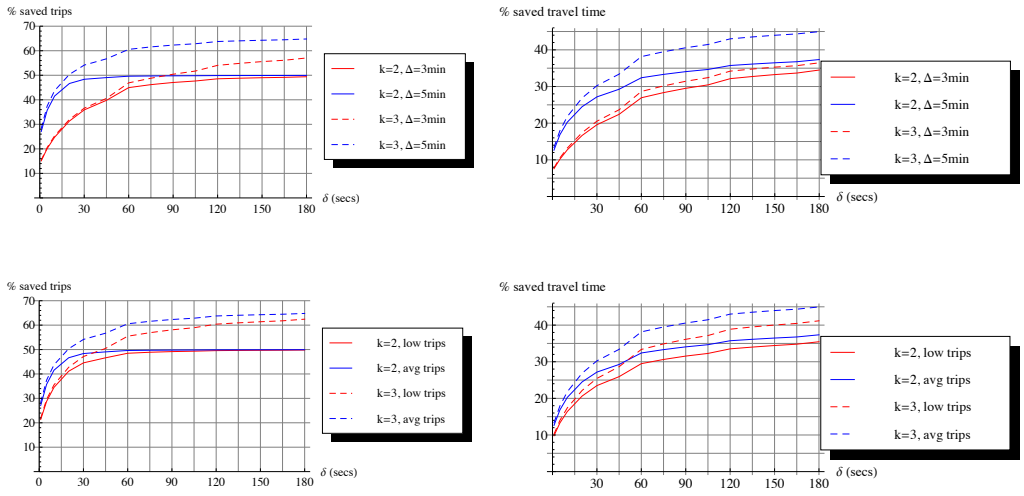

Figure S3: Percentage of saved taxi trips (top left) and percentage of saved travel time (top right) in New York City as a function of δ in the Online model. The quality of service parameter is set to $\Delta = 3\,\mathrm{min}$ and $\Delta = 5\,\mathrm{min}$. Percentage of saved taxi trips (bottom left) and percentage of saved travel time (bottom right) as a function of δ in the Online model, in a day with relatively low taxi traffic ("low"), and with average taxi traffic ("avg"). The quality of service parameter is set to $\Delta = 5\,\mathrm{min}$. Each plot reports two curves: one referring to the case where at most two trips can be shared ($k = 2$), and one referring to the case where at most three trips can be shared ($k = 3$).