Sharing Rides with Friends: a Coalition Formation Algorithm for Ridesharing

Filippo Bistaffa and **Alessandro Farinelli**
Department of Computer Science
University of Verona
Verona, Italy
{filippo.bistaffa,alessandro.farinelli}@univr.it

Sarvapali D. Ramchurn
Electronics and Computer Science
University of Southampton
Southampton, UK
sdr@ecs.soton.ac.uk

Abstract

We consider the Social Ridesharing (SR) problem, where a set of commuters, connected through a social network, arrange one-time rides at short notice. In particular, we focus on the associated optimisation problem of forming cars to minimise the travel cost of the overall system modelling such problem as a graph constrained coalition formation (GCCF) problem, where the set of feasible coalitions is restricted by a graph (i.e., the social network). Moreover, we significantly extend the state of the art algorithm for GCCF, i.e., the CFSS algorithm, to solve our GCCF model of the SR problem. Our empirical evaluation uses a real dataset for both spatial (GeoLife) and social data (Twitter), to validate the applicability of our approach in a realistic application scenario. Empirical results show that our approach computes optimal solutions for systems of medium scale (up to 100 agents) providing significant cost reductions (up to -36.22%). Moreover, we can provide approximate solutions for very large systems (i.e., up to 2000 agents) and good quality guarantees (i.e., with an approximation ratio of 1.41 in the worst case) within minutes (i.e., 100 seconds).

1 Introduction

The concept of real-time ridesharing, where people arrange one-time rides at short notice with their private cars, is rapidly shifting the way people commute for their daily activities. Companies such as *Uber* or *Lyft* allow users to quickly share their positions and arrange rides with other people they know/trust within minutes, hence providing a credible alternative to standard transportation systems (such as taxis or public transport). A clear trend for such companies is to build a community of users, where commuters can rate drivers/passengers, and then use such information to automatically form groups of commuters that know/trust each other.

Following this trend, here we focus on providing an approach that, given the desired starting points and destinations of a community of commuters, can share cars to lower associated transportation costs (i.e., travel time and fuel), while considering the constraints imposed by the social network that connects such commuters. We call this problem the Social Ridesharing (SR) problem.

In particular, we provide a model for the SR problem casting this as a Graph Constraint Coalition Formation (GCCF) problem, where commuters (i.e., riders and drivers) form coalitions (i.e., join in a car) meeting the constraints imposed by the social network (i.e., users prefer to join a car with their friends). Specifically, following relevant literature on GCCF (Myerson 1977; Voice, Ramchurn, and Jennings 2012), we consider a coalition to be *feasible*, only if the commuters involved in such coalition form a connected subgraph of the social network.

Recently, (Kamar and Horvitz 2009) addressed the computational aspects related to ridesharing, proposing an interesting model to evaluate ridesharing plans, on which we base our model for SR. However, such work is mostly focused on incentive design aspects for ridesharing while here we focus on the optimisation problem posed by SR. Moreover, they do not consider the role of the social network in their work. Now, the optimisation problem related to coalition formation (i.e., Coalition Structure Generation) has been addressed by several researchers throughout the years (Sandholm et al. 1999; Service and Adams 2011; Rahwan, Michalak, and Jennings 2012; Shehory and Kraus 1998). Consequently the relevant literature offers a wealth of optimal solution techniques as well as approximate algorithms that can provide quality guarantees. However, as pointed out in (Voice, Ramchurn, and Jennings 2012), all such algorithms do not consider graph constraints that limit feasible coalitions, hence they can not be applied to the GCCF problem as their behaviour for such specific problem is not well defined.

Recently, several approaches have been explicitly designed to deal with the GCCF problem (Voice, Ramchurn, and Jennings 2012; Voice, Polukarov, and Jennings 2012; Bistaffa et al. 2014). Among these approaches, here we consider as a solution technique the CFSS algorithm proposed in (Bistaffa et al. 2014) which, unlike previous approaches, can provide approximate solutions with quality guarantees for large-scale systems (including thousands of agents). Hence, this approach is particularly suitable for SR, in which good solutions for a large number of agents are needed in near real-time. On the other hand, one key element for the efficiency of such approach is the possibility of finding an effective and efficient bounding function to prune significant part of the search space. Now, such a bounding function is easy to

provide only for specific characteristic functions which can be decomposed into a sum of monotonic and anti-monotonic functions ($m + a$). Unfortunately, the SR problem does not exhibit such a property, hence we significantly extend the CFSS algorithm to provide solutions for large-scale systems (e.g., up to 2000 agents) in this scenario.

In more detail, this paper advances the state of the art in the following ways:

- We provide a formalisation of the ridesharing optimisation problem as GCCF.
- We significantly extend the state-of-the-art approach for solving GCCF to provide optimal solutions, as well as approximate solutions with quality guarantees for large-scale systems.
- We evaluate our approach with realistic datasets, i.e., GeoLife from Microsoft Research for the geospatial data and Twitter for social networks. Results show that our approach computes optimal solutions in minutes for systems including up to 100 agents, and provides approximate solutions for systems including up to 2000 agents, with good quality guarantees (i.e., with an approximation ratio of 1.41 in the worst case).

In what follows, we first provide necessary background on the GCCF problem and on the CFSS algorithm, then we describe our model of the SR and our solution technique. Finally we present the results of our empirical evaluation.

2 Background

The purpose of this section is twofold. First, in Section 2.1 we define the GCCF problem. Second, in Section 2.2 we provide some background on the state of the art algorithm for solving GCCF, namely the CFSS algorithm.

2.1 GCCF Problem Definition

A *coalitional game* or *characteristic function game*[1] consists of a finite set of players $\mathcal{A}$ and a characteristic function $v : 2^{\mathcal{A}} \to \mathbb{R}$, that maps each coalition $C \in 2^{\mathcal{A}}$ to its value, describing how much collective payoff a set of players can gain by forming a coalition. A coalition structure CS is a partition of the set of agents into disjoint coalitions. The set of all coalition structures is $\Pi(\mathcal{A})$. The value of a coalition structure CS is assessed as the sum of the values of its composing coalitions:

$$V(CS) = \sum_{C \in CS} v(C) \tag{1}$$

The *coalition formation problem* (Shehory and Kraus 1998; Sandholm et al. 1999) (or *coalition structure generation problem*) takes as input a coalitional game and aims at identifying CS^*, the most valuable coalition structure, i.e., $CS^* = \arg\max_{CS \in \Pi(\mathcal{A})} V(CS)$.

Now, given a graph $G = (\mathcal{A}, E)$, where $E \subseteq \mathcal{A} \times \mathcal{A}$ is a set of edges between agents, representing their relationships (i.e., friendship), (Myerson 1977) considers a coalition C to be feasible if all of their members are connected in the subgraph of G induced by C. That is, if for each pair of players from $a, b \in C$ there is a path in G that connects them without going out of C. Given a graph G the set of feasible coalitions is

$$\mathcal{FC}(G) = \left\{ C \subseteq \mathcal{A} \;\middle|\; \begin{array}{l}\text{The subgraph induced by } C \text{ on } G \\ \text{is connected}\end{array} \right\}.$$

Consequently, a graph constrained coalition formation (GCCF) game is a coalitional game together with a graph G, where a coalition C is considered feasible if $C \in \mathcal{FC}(G)$. In GCCF games a coalition structure CS is considered feasible if each of its coalitions is feasible, i.e.,

$$\mathcal{CS}(G) = \{CS \in \Pi(\mathcal{A}) \mid CS \subseteq \mathcal{FC}(G)\}.$$

Hence, the goal for a GCCF problem is to identify CS^*, which is the most valuable coalition structure, i.e., $CS^* = \arg\max_{CS \in \mathcal{CS}(G)} V(CS)$.

After the definition of the GCCF problem, we now present the state of the art algorithm to solve it, namely the CFSS algorithm.

2.2 CFSS

CFSS (Bistaffa et al. 2014) is based on the concept of *edge contraction* on the graph G, which represents the merging of the coalitions associated to its incident vertices. Such an operation can be used to generate the entire search space $\mathcal{CS}(G)$ and organise it as a rooted tree $\mathcal{T}_G$, which can be traversed with polynomial memory requirements in order to find the optimal solution. Each feasible coalition structure $CS \in \mathcal{CS}(G)$ is represented only once, one per each node of $\mathcal{T}_G$, by means of a *2-coloured graph* G^c, in which the nodes of G^c represent the coalitions of CS and the edges are marked either in green (i.e., such edge can still be contracted) or in red, meaning that a previous contraction of that edge has already been done, and its endpoints must not be in the same coalition in the following phases of the algorithm. Figure 1(a) shows an example of a 2-colour graph in which edge $(\{A\}, \{D\})$ is red: hence, in any subsequent step of the algorithm it is impossible to contract it. This marking ensures that each feasible coalition structure is represented in $\mathcal{T}_G$ without any redundancy. In particular, given a node $\mathcal{T}_G$ representing a feasible coalition structure CS, its children are assessed contracting each green node in the corresponding 2-colour graph G^c. We refer to the subtree of $\mathcal{T}_G$ rooted at CS as $\mathcal{ST}(CS)$.

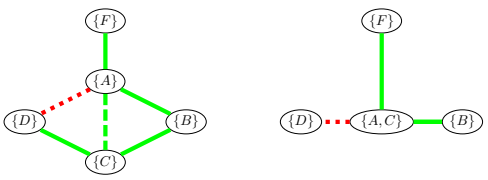

Figure 1: Example of a green edge contraction.

In the next section, we show how we represent a Social Ridesharing (SR) problem as a GCCF problem.

[1]Note that, to be consistent with relevant literature (Myerson 1977), here we use the name game. However, we do not consider stability of coalitions and we do not address issues such as payment schemes for the agents, which are typical of mechanism design.

3 Social Ridesharing Model

In this section we define our model for the SR problem. In particular, we consider a set of riders $\mathcal{R} = \{r_1, \ldots, r_R\}$, where $R > 0$ is the total number of riders, and a non-empty[2] set of drivers $\mathcal{D} \subseteq \mathcal{R}$, containing the riders owning a car. Every driver $r_i \in \mathcal{D}$ can host up to $s(r_i)$ riders in his car, including himself, where the function $s : \mathcal{R} \to \mathbb{N}^0$ provides the number of seats of each car. If $r_i \notin \mathcal{D}$, then $s(r_i) = 0$. Given a set of riders $C \subseteq \mathcal{R}$, C is said to be *valid* if the following constraint holds:

Constraint 1. $|C| > 1 \implies \exists r_i \in C : s(r_i) \geq |C|$*, i.e., at least one rider has a car with enough seats for all the riders.*

Notice that such a constraint allows a rider $r_i \notin \mathcal{D}$ to be in a singleton. In fact, if the total number of available seats is less than the total number of riders in the system, such a rider might need to resort to public transport paying a cost $k(\{r_i\})$ for the ticket. Formally, the function $k : \mathcal{R}^0 \to \mathbb{R}^-$ provides such a cost, where $\mathcal{R}^0 = \{\{r_i\} \mid r_i \in \mathcal{R} - \mathcal{D}\}$ is the set of all the singletons excluding the drivers, as we assume that such riders always prefer to use their car w.r.t. public transport.

Now, in several ridesharing online services (e.g., *Lyft* and *Uber*) a commuter declares whether he is available as a driver or as a rider, hence the two sets are disjoint and a valid set of riders C contains at most one driver. Formally, the following additional constraint must hold:

Constraint 2. $|C \cap \mathcal{D}| \leq 1$*, i.e., the number of drivers per car can be at most 1.*

Notice that Constraint 2 is optional, but it holds in several established real-world services, arising from aspects of practical nature. Nonetheless, since our approach supports a more general model, it can also be applied to scenarios where such a constraint does not hold.

We consider a *map* of the geographic environment in which the SR problem takes place, represented by a connected graph $\mathcal{M} = (\mathcal{P}, \mathcal{Q})$, where $\mathcal{P}$ is the set of geographic points of the map and $\mathcal{Q}$ is a set of edges (each associated to a positive weight) among these points. In what follows, we assume that a path going through n points is represented as a tuple $P \in \mathcal{P}^n$, denoting as P^i the i-th point in such a tuple.

Each rider $r_i \in \mathcal{R}$ has to move from a starting point p_i^a, i.e., its pick-up point, to a destination point p_i^b in the map $\mathcal{M}$ and each car drives through a path that contains all the starting and destination points of his passengers. Not all paths are valid, and a *valid* path must fulfil two constraints to correctly accommodate the needs of all the passengers. Formally, given a valid set of riders C and a path P of n points, P is said to be *valid* if the following constraints hold:

Constraint 3. $\exists r_i \in C : s(r_i) \geq |C| \wedge P^1 = p_i^a \wedge P^n = p_i^b$*, i.e., P goes from the driver's starting point to its destination.*

Constraint 4. $\forall r_i \in C\, \exists x, y : P^x = p_i^a \wedge P^y = p_i^b \wedge x < y$*, i.e., for each rider, its starting point precedes its destination.*

Henceforth, we refer to the set of all valid paths for a given set of riders C with $\mathcal{VP}(C)$.

[2] If $\mathcal{D} = \varnothing$ the problem is trivial and it is not taken into account.

Following (Kamar and Horvitz 2009), we define the total cost $v(C)$ of a valid set of riders C as:

$$v(C) = \begin{cases} t(P_C) + c(P_C) + f(P_C), & \text{if } C \cap \mathcal{D} \neq \varnothing \\ k(C), & \text{otherwise.} \end{cases} \quad (2)$$

where P_C is the optimal path for C, and $t : \mathcal{P}^n \to \mathbb{R}^-$, $c : \mathcal{P}^n \to \mathbb{R}^-$ and $f : \mathcal{P}^n \to \mathbb{R}^-$ are negative[3] cost functions respectively representing the time cost, the cognitive cost[4] and the fuel cost of driving through a given path. We denote the sum of these three cost functions as $cost(\cdot)$. Notice that if $C \cap \mathcal{D} = \varnothing$, Constraint 1 imposes that C is formed by a single rider without a car, hence its cost is provided by $k(\cdot)$. Furthermore, P_C is defined as follows:

$$P_C = \underset{P_i \in \mathcal{VP}(C)}{\arg\max}\, cost(P_i) \quad (3)$$

Finally, as mentioned before, a key aspect of ridesharing is the presence of a social network G that restrict the formation of groups. Hence, our SR model considers the previous definition of *feasible* coalition from Section 2.1 and we define a valid set of agent as a set that does not violate Constraint 1 and whose members induce a connected subgraph on G.

Such SR problem can be easily translated into a GCCF problem, as each valid set of riders is indeed a coalition and $v(\cdot)$ provides its coalitional value. Hence, CS^* represents the optimal coalition structure which maximises the social welfare (i.e., minimises the total cost) for the system. However, the computation of the optimal path in Equation 3 represents a hard problem, which could be not solvable in realistic scenarios. Hence, in the next section we show how a reasonable assumption on the cost functions allows to reduce this complexity making such computation tractable.

4 Route computation

The computational complexity of Equation 3 derives from the lack of assumptions on the cost functions $t(\cdot)$, $c(\cdot)$ and $f(\cdot)$. However, in many urban scenarios the cost of driving through a path is determined by the length of the path itself, and longer paths usually result in higher costs. Hence, we assume that $t(\cdot)$, $c(\cdot)$ and $f(\cdot)$ are *antimonotonic* functions, i.e., given two paths P_i and P_j, if the length of P_i is greater than the length of P_j, then $t(P_i) < t(P_j)$, $c(P_i) < c(P_j)$ and $f(P_i) < f(P_j)$. Against this background, the following proposition holds:

Proposition 1. *If $t(\cdot)$, $c(\cdot)$ and $f(\cdot)$ are antimonotonic functions, the optimal path P_C for a set of riders C is the shortest path $P_i \in \mathcal{VP}(C)$.*

Proof. See Appendix. □

Given a path $P \in \mathcal{P}^n$, the function $best : \mathcal{P}^n \to \mathcal{P}^m$ is defined as $\bigoplus_{k=1}^{n-1} sp\left(P^k, P^{k+1}\right)$, where the function $sp(\cdot)$ provides the shortest path between two points, $\oplus$ represents the

[3] Since we consider a maximisation problem, we represent costs as negative values.

[4] Following (Kamar and Horvitz 2009), this cost represents the fatigue incurred by the driver during the trip.

concatenation of tuples and m is the number of points resulting from all such concatenations. The function $best(\cdot)$ can be computed efficiently in $O((n-1)\cdot(|\mathcal{Q}|+|\mathcal{P}|\log|\mathcal{P}|))$, assuming that $sp(\cdot)$ is implemented using Dijkstra's algorithm. Moreover, if $\mathcal{M}$ is an euclidean graph $sp(\cdot)$ can be computed in $O((n-1)\cdot|\mathcal{Q}|)$ with an A* algorithm.

Proposition 2. *Given a tuple of points T, $best(T)$ is the shortest path going through such points in order.*

Proof. See Appendix. □

Finally, given a set of riders C, we define $\mathcal{VT}(C)$ as the set of tuples which contains only and all the start and destination points of its riders (without repetitions) and which satisfy Constraints 3 and 4. Against this background, we prove the following theorem:

Theorem 1. *If $t(\cdot)$, $c(\cdot)$ and $f(\cdot)$ are antimonotonic functions, the optimal path P_C for a car C is P_C^*, the shortest among all paths in $\{best(P_i) \mid P_i \in \mathcal{VT}(C)\}$.*

Proof. See Appendix. □

Theorem 1 provides an affordable algorithm to compute the optimal path for a set of riders assuming that the cost functions are antimonotonic. Notice that its search space is $\mathcal{VT}(C)$, whose size is significantly smaller than $\mathcal{VP}(C)$ of Equation 3, and although being still exponential w.r.t. $|C|$, $|\mathcal{VT}(C)|$ is manageable for reasonably sized groups of riders. In fact, it is only 2520 for $|C| = 5$ (i.e., the number of seats of an average car).

In the following section, we detail how the CFSS algorithm, can be adapted to tackle the above defined problem.

5 CFSS for Ridesharing

In order to solve the SR problem, the original version of CFSS must be modified to correctly assess the additional constraints introduced in Section 3. In particular, to ensure that Constraints 1 and 2 hold, we must avoid the formation of coalitions which are not valid sets of riders. This can be achieved by avoiding the contractions of the green edges which would result in the violation of such constraints. Notice that such edges must be marked in red, even if we are not visiting the corresponding subtrees: in fact, this is equivalent to traversing such search spaces and discarding any possible solution they may contain, because such solutions would violate one of the above mentioned constraints.

A key enhancement for the efficiency of CFSS is the use of a branch and bound search strategy to prune significant parts of the search space. In (Bistaffa et al. 2014), authors provide a general bounding technique for a particular class of characteristic functions, namely $m+a$ functions.

Unfortunately, the characteristic function defined in Equation 2 is not an $m+a$ function, since it depends on P_C, and in particular on the actual position of the start and destination points of the riders. As an example, consider Figure 2, which shows the start and destination points for 3 riders, i.e., $\mathcal{R} = \{r_1, r_2, r_3\}$, in which only r_1 owns a car, i.e., $\mathcal{D} = \{r_1\}$. For simplicity, we assume that $v(C)$ is equal to the length of P_C, and $k(\{r_2\}) = k(\{r_3\}) = -1$.

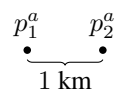

Figure 2: Example start and destination points for 3 riders.

In this example, $v(\{r_1\}) = -3$, $v(\{r_2\}) = -1$, $v(\{r_3\}) = -1$. However, we notice that p_2^a and p_2^b are actually part of the path travelled by r_1, hence it is reasonable for r_2 to join r_1 in the coalition $\{r_1, r_2\}$. In fact, $v(\{r_1, r_2\}) = v(\{r_1\}) = -3 > v(\{r_1\}) + v(\{r_2\}) = -3 - 1 = -4$. On the other hand, r_3 start and destination points are outside r_1's path,[5] hence ridesharing is not effective in this case: $v(\{r_1, r_3\}) = -7 < v(\{r_1\}) + v(\{r_3\}) = -3 - 1 = -4$. Notice that this particular characteristic function cannot be seen as the sum of a monotonic and an antimonotonic part, since it exhibits a monotonic behaviour for some coalition structures, i.e., $v(\{r_1, r_2\}) > v(\{r_1\}) + v(\{r_2\})$, while it is antimonotonic for some others, i.e., $v(\{r_1, r_3\}) < v(\{r_1\}) + v(\{r_3\})$.[6]

Hence, in the next section we provide alternative bounding techniques that can be used in our ridesharing scenario.

5.1 Bound Computation

Given a feasible coalition structure CS in our search tree, we now show how to compute an upperbound $M(CS)$ for the values assumed by the characteristic function in $\mathcal{ST}(CS)$, i.e., $M(CS) \geq V(CS_i)\ \forall CS_i \in \mathcal{ST}(CS)$. This value can be used to avoid visiting $\mathcal{ST}(CS)$ if $M(CS)$ is not greater than the current best solution.

First, we provide a method to compute $M(CS)$ in scenarios where Constraint 2 holds. In these environments it is not possible to merge coalitions both containing a driver, since only single riders not owning a car are allowed to join existing groups. Notice that the addition of a rider to car can only result in a greater cost, if we consider antimonotonic cost functions. Therefore, the sum of the costs of all the cars (i.e., all the coalitions containing a driver) can only increase after such an addition. More formally:

Proposition 3. *If $cost(\cdot)$ is an antimonotonic function and Constraint 2 holds, then $M(CS) = \sum_{C \in \mathcal{R}^d(CS)} v(C)$, where $\mathcal{R}^d(CS) = \{C \in CS \mid C \cap \mathcal{D} \neq \varnothing\}$.*

We now detail how to compute an upperbound without assuming Constraint 2. We define $\mathcal{S}_1$ as the set of the start and destination points of all the riders, i.e., $\{p_i \in \mathcal{P} \mid \exists r_j \in \mathcal{R} : p_i = p_j^a \vee p_i = p_j^b\}$, and $\mathcal{S}_2$ as the set of all the couples of different points in $\mathcal{S}_1$, i.e., $\{(p_i, p_j) \in \mathcal{S}_1 \times \mathcal{S}_1 \mid p_i \neq p_j\}$. Then, we define $\mathcal{S}_{1,a}(r_i)$ as the set of all the shortest paths from r_1's starting point to the start and destination points of any other rider, i.e., $\{P_i \in \mathcal{P}^n \mid P_i = sp(p_i^a, p_j)\ \forall p_j \in S_1 : p_j \neq p_i^a\}$. Similarly, we define $\mathcal{S}_{1,b}$ considering r_1's destination point.

[5] The optimal path for $C = \{r_1, r_3\}$ is $P_C = (p_1^a, p_3^a, p_3^b, p_1^b)$.

[6] This notion of antimonotonic function should not be confused with the one previously defined for $t(\cdot)$, $c(\cdot)$ and $f(\cdot)$, which take paths as arguments, while characteristic functions are defined on coalition structures.

Also, we define $\mathcal{S}_{2,a}(r_i)$ as the concatenation of all the couples of shortest paths from r_1's starting point to the start and destination points of any other rider, i.e., $\{P_i \in \mathcal{P}^n \mid P_i = sp(p_i^a, p_j) \oplus sp(p_i^a, p_k) \; \forall (p_j, p_k) \in S_2 : p_j \neq p_i^a \wedge p_k \neq p_i^a\}$. Again, we define $\mathcal{S}_{2,b}$ considering r_1's destination point. Finally, we define $m : \mathcal{R} \rightarrow \mathbb{R}^-$ as:

$$m(r_i) = \begin{cases} \max\limits_{P_i \in \mathcal{S}_{1,a}} cost(P_i) + \max\limits_{P_i \in \mathcal{S}_{1,b}} cost(P_i), & \text{if } r_i \in \mathcal{D} \\ \max\limits_{P_i \in \mathcal{S}_{2,a}} cost(P_i) + \max\limits_{P_i \in \mathcal{S}_{2,b}} cost(P_i), & \text{otherwise.} \end{cases}$$

We can now state the following proposition:

Proposition 4. *If $cost(\cdot)$ is an antimonotonic function, then $M(CS) = \frac{1}{2} \cdot \sum_{r_i \in \mathcal{R}^d(CS)} m(r_i)$.*

Sketch of proof. See Appendix. □

These results allow us to bound $V(\cdot)$ in all the coalition structures in $\mathcal{ST}(CS)$. In the next section we detail how our bounding techniques, underlying our optimal branch and bound search strategy, can also be adapted to compute approximate solutions with quality guarantees.

5.2 Approximate Solutions

The bounding techniques described in previous section allow to prune significant parts of the search space, thus providing optimal solutions for system of significant size (i.e., up to 100 agents, see Section 6 for further details). However, a realistic application might involve community of thousands of agents, and, following the concept of real-time ridesharing, the system should be able to provide solutions in minutes. Consequently, for realistic applications approximate solutions must be considered. Now, following the approach proposed in (Bistaffa et al. 2014), the bounding techniques detailed above can be used to implement an approximate version the SR problem. In particular, we stop the search process after an amount of time t_s and compute the bound $M(CS)$ for all the leaf configurations of the search tree, the maximum among all these values is an admissible bound for the approximate solution found within t_s.

The quality of this approximate approach, together with a performance analysis of our optimal solution algorithm, will be object of our experimental evaluation, presented in the following section.

6 Experimental Evaluation

Having described and analysed our branch and bound approach for the SR problem, we now present the empirical evaluation. In what follows, we first present our evaluation methodology, then we discuss the achieved results.

6.1 Evaluation Methodology

The main goals of the empirical analysis are: i) to estimate the social welfare improvement when our ridesharing method is employed, ii) to evaluate the performance of our optimal algorithm in terms of runtime and scalability, iii) to evaluate the approximate performance and guarantees that our approach can provide when scaling to very large number of agents, i.e., up to 2000 agents.

Our experimental evaluation considers realistic datasets, both for spatial and social data. In particular, our map $\mathcal{M} = (\mathcal{P}, \mathcal{Q})$ is a realistic representation of the city of Beijing, with $|\mathcal{P}| = 8330$ points and $|\mathcal{Q}| = 13290$ edges, equivalent to an average resolution of a point every ~10 meters. This map has been derived from the GeoLife[7] dataset provided by Microsoft, which comprises 17621 trajectories with a total distance of about 1.2 million km, recorded by different GPS loggers and GPS-phones with a variety of sampling rates. This pool of trajectories is also adopted to sample random paths used to provide the start and destination points of the riders in our experiments. Moreover, in each experiment the graph G is a subgraph of a large crawl of the Twitter social graph completed in 2010. We adopted the Twitter dataset since it is freely available, free from privacy restrictions (unlike Facebook) and has been presented in a well established paper (Kwak et al. 2010). In particular, G is obtained by means of a standard algorithm (Russell 2013) to extract a subgraph from a larger graph, i.e., a breath-first traversal starting from a random node of the whole graph, adding each node and the corresponding arcs to G, until the desired number of nodes is reached. In our experimental evaluation there is no mapping between the trajectory data and the social graph, since they belong to independent projects.

In our experiments we adopt a cost model that only considers fuel expenses, i.e., $v(C) = K_{fuel} \cdot \overline{P_C}$, where $\overline{P_C}$ represents the length of P_C in km, $K_{fuel} = -0.0\overline{6}$ €/km (considering a fuel cost of −1 € per litre and an average consumption of 1 litre of fuel every 15 km) and $k(\{r_i\}) = -3$ € $\forall r_i \in \mathcal{R}$, which represents the average public transportation cost, i.e., a bus or a train ticket. Moreover, we assume that each car has a capacity of 5 seats, i.e., $s(r_i) = 5 \; \forall r_i \in \mathcal{D}$.

All our experiments are done considering Constraint 2 (drivers always drive their cars), as it models many real-world online services, e.g., *Lyft* and *Uber*. Hence, we employ both the bounding techniques detailed in Section 5.1 and we take the minimum one, since both are admissible. Each experiment is repeated on 20 random instances, and we report the average and the standard error of the mean of the results. Our approach is executed on a machine with a quad-core 3.40GHz processor and 16 GB of memory.

6.2 Social Welfare Improvement

Here we consider the improvement of the social welfare (i.e., the cost reduction for the overall system) when using our automatic car formation approach as compared to the scenario in which every rider adopts its own conveyance (i.e., no ridesharing). This gives an indication of what gain can be achieved by the overall community when using our system for ridesharing. More formally, we define the social welfare improvement as $100 \cdot \left| \frac{V(CS^\star) - V(\mathcal{R}^0)}{V(\mathcal{R}^0)} \right|$. Such an improvement is influenced by the percentage of drivers in the system (Figure 4), which determines the number of available seats and the number of riders which can share a ride without having to resort to public transport. Moreover, with more drivers it

[7] http://research.microsoft.com/en-us/projects/geolife

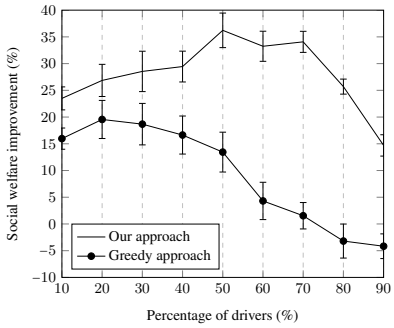

Figure 4: Social welfare improvement.

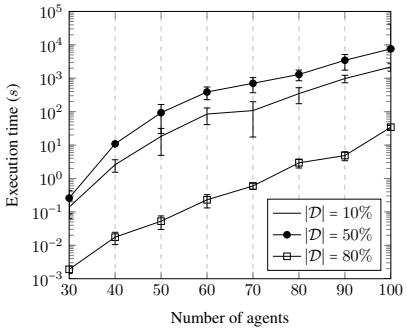

Figure 5: Runtime needed to compute the optimal solution.

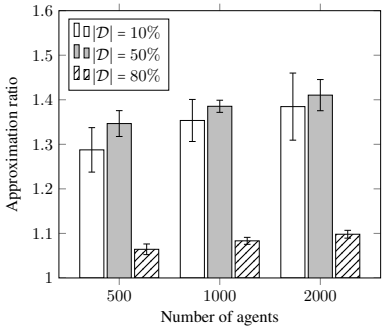

Figure 6: Approximation ratio of the solutions.

is more probable that a rider can join a car whose path is closer to him/her. On the other hand, if the majority of the riders own a car (i.e., > 80%), ridesharing is not very effective since too few riders without a car can benefit from sharing their commutes with a driver. In particular, when only the 10% of the total riders own a car, the average cost reduction (computed over 20 random instances with $|\mathcal{R}| = 50$ for each driver percentage) is −23.49%, reaching −36.22% when half of the riders owns a car.

To show the importance of an optimal approach, we benchmark our algorithm against a greedy one, in which every driver chooses its next stop as the closest among the destinations points of his current passengers and the starting points of the remaining riders. This choice is made considering the constraints imposed by the social network, avoiding the formation of unfeasible coalitions. As Figure 4 shows, our method allows superior cost reductions w.r.t. such a greedy approach, which can provide a maximum improvement of −19.55% for $|\mathcal{D}| = 20\%$. When the majority of the riders owns a car, such an algorithm provides solution whose costs are higher than the reference ones. In this scenario, where only few riders do not own a car, the greedy approach does not perform well since it might assign a rider to a suboptimal coalition before considering the optimal one.

6.3 Optimal Performance

Figure 5 shows the runtime needed to compute the optimal solution when increasing the number of agents. Our approach is tested in 3 scenarios, i.e., with low (10%), medium (50%) and high (80%) percentage of drivers, showing that this parameter has a significant influence on the performance of our algorithm. In fact, the size of the search space is determined by the the number of available seats (reduced when such a percentage is low) and the number of riders without a car who can benefit from sharing their commutes (reduced when the majority of the agents owns a car), consistently with the behaviour of the social welfare improvement detailed in the previous section. Notice that, in any case, our approach can solve systems with 100 agents in a reasonable amount of time, i.e., about 2 hours at most for $|\mathcal{D}| = 50\%$. This runtime is suitable for services with day-ahead or week-ahead requests (e.g., *Lyft*). Such a performance is possible thanks to our bounding techniques (see Section 5.1), which allow to prune a significant part of the search space. In more detail, such techniques allow an average pruning of the 97.5% of the search space (resulting in an average runtime improvement of about 4 hours) on 20 random instances with $|\mathcal{R}| = 60$ and $|\mathcal{D}| = 50\%$.

6.4 Approximate Performance

Figure 6 shows the value of the approximation ratio (i.e., the ratio between the bound and the approximate solution) provided by our approach when used to solve large systems with $|\mathcal{R}| \in \{500, 1000, 2000\}$. In particular, we adopted the approximate technique explained in Section 5.2, stopping the traversal of the search tree after $t_s = 100$ seconds. Our experiments show that, for $|\mathcal{R}| = 500$ and $|\mathcal{D}| = 80\%$, the provided bound is only 6.65% higher than the solution found within the time limit, reaching a maximum of +29.92% when $|\mathcal{R}| = 2000$ and $|\mathcal{D}| = 50\%$. In the worst case, CFSS provides an approximation ratio of 1.41 and thus solutions that are at least 71% of the optimal.

7 Conclusions

We considered the Social Ridesharing problem, showing how it can be modelled as a GCCF problem and extending the state of the art algorithm for GCCF, i.e., CFSS, to solve it. Our empirical evaluation shows that our approach can lead to a cost reduction for the entire system that reaches the −36.22% and that our approximate technique can compute solutions for very large systems (i.e., up to 2000 agents) with good quality guarantees (i.e., with an approximation ratio of 1.41 in the worst case), hence being suitable for realistic applications.

Future work will look at extending our model to include more complex routing scenarios with traffic information, time constraints for riders as well as multi-hop ridesharing (Drews and Luxen 2013).

Acknowledgments.

This work was carried out as part of the ORCHID project funded by EPSRC (EP/I011587/1).

Appendix.

We report here the proofs of the propositions and the theorem stated in our paper.

Proposition 1. *If $t(\cdot)$, $c(\cdot)$ and $f(\cdot)$ are antimonotonic functions, the optimal path P_C for a set of riders C is the shortest path $P_i \in \mathcal{VP}(C)$.*

Proof. Since $t(\cdot)+c(\cdot)+f(\cdot)$ is an antimonotonic function, such a proposition follows as a direct consequence. □

Proposition 2. *Given a tuple of points T, $best(T)$ is the shortest path going through such points in order.*

Proof. By contradiction. Suppose there is a path P' going through all the points of T in order which is shorter than $best(T)$. Hence, there must exist two consecutive points in T, namely p_i and p_j, such that the subpath of P' going from p_i to p_j is shorter than the subpath of $best(T)$ going from p_i to p_j, which is a contradiction since it violates the definition of $best(T)$. □

Theorem 1. *If $t(\cdot)$, $c(\cdot)$ and $f(\cdot)$ are antimonotonic functions, the optimal path P_C for a car C is P_C^*, the shortest among all paths in $\{best(P_i) \mid P_i \in \mathcal{VT}(C)\}$.*

Proof. By contradiction. Suppose there is an optimal path $P' \neq P_C^*$. Then, P' must be shorter than P_C^* (Proposition 1). Since P_C^* is the shortest among all the paths going through all the tuples in $\mathcal{VT}(C)$ (Proposition 2), then it does not exists a tuple $T \in \mathcal{VT}(C)$ such that P' goes through all its points, which is a contradiction since T can be computed from P by removing any point which is neither a starting nor a destination point of a rider in C. □

Proposition 4. *If $cost(\cdot)$ is an antimonotonic function, then $M(CS) = \frac{1}{2} \cdot \sum_{r_i \in \mathcal{R}^d(CS)} m(r_i)$.*

Sketch of proof. This proof can be derived from the proof of the bound for the TSP problem with triangle inequality (Cormen et al. 2009; Brassard and Bratley 1996). □

References

Bistaffa, F.; Farinelli, A.; Cerquides, J.; Rodríguez-Aguilar, J.; and Ramchurn, S. D. 2014. Anytime coalition structure generation on synergy graphs. In *AAMAS*, pp. 13–20.

Brassard, G., and Bratley, P. 1996. *Fundamentals of Algorithmics*. Prentice-Hall, Inc.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2009. *Introduction to Algorithms*. The MIT Press, 3rd edition.

Drews, F., and Luxen, D. 2013. Multi-hop ride sharing. In *SOCS*. AAAI Press.

Kamar, E., and Horvitz, E. 2009. Collaboration and shared plans in the open world: Studies of ridesharing. In *IJCAI*, pp. 187–194.

Kwak, H.; Lee, C.; Park, H.; and Moon, S. 2010. What is twitter, a social network or a news media? In *WWW*, pp. 591–600.

Myerson, R. B. 1977. Graphs and cooperation in games. *MOR* 2(3):pp. 225–229.

Rahwan, T.; Michalak, T.; and Jennings, N. R. 2012. A hybrid algorithm for coalition structure generation. In *AAAI*, pp. 1443–1449.

Russell, M. A. 2013. *Mining the Social Web*. O'Reilly Media, 2nd edition.

Sandholm, T.; Larson, K.; Andersson, M.; Shehory, O.; and Tohmé, F. 1999. Coalition structure generation with worst case guarantees. *AIJ* 111(1):pp. 209–238.

Service, T. C., and Adams, J. A. 2011. Constant factor approximation algorithms for coalition structure generation. *JAAMAS* 23(1):pp. 1–17.

Shehory, O., and Kraus, S. 1998. Methods for task allocation via agent coalition formation. *AIJ* 101(1-2):pp. 165–200.

Voice, T.; Polukarov, M.; and Jennings, N. 2012. Coalition structure generation over graphs. *JAIR* 45:pp. 165–196.

Voice, T.; Ramchurn, S.; and Jennings, N. 2012. On coalition formation with sparse synergies. In *AAMAS*, pp. 223–230.