A Cooperative Game-Theoretic Approach to the Social Ridesharing Problem

Filippo Bistaffa[a,*], Alessandro Farinelli[a], Georgios Chalkiadakis[b], Sarvapali D. Ramchurn[c]

[a] *Department of Computer Science, University of Verona, Italy*
[b] *School of Electrical and Computer Engineering, Technical University of Crete, Greece*
[c] *Electronics and Computer Science, University of Southampton, United Kingdom*

Abstract

In this work, we adopt a cooperative game theoretic approach in order to tackle the *social ridesharing* (SR) problem, where a set of commuters, connected through a social network, form coalitions and arrange one-time rides at short notice. In particular, we address two fundamental aspects of this problem. First, we focus on the optimisation problem of forming the travellers coalitions that minimise the travel cost of the overall system. To this end, we model the formation problem as a Graph-Constrained Coalition Formation (GCCF) one, where the set of feasible coalitions is restricted by a graph (i.e., the social network). Our approach allows users to specify both *spatial* and *temporal* preferences for the trips. Second, we tackle the *payment allocation* aspect of SR, by proposing the first approach that computes *kernel-stable payments* for systems with thousands of agents. We conduct a systematic empirical evaluation that uses real-world datasets (i.e., GeoLife and Twitter). We are able to compute optimal solutions for medium-sized systems (i.e., with 100 agents), and high quality solutions for very large systems (i.e., up to 2000 agents). Our results show that our approach improves the social welfare (i.e., reduces travel costs) by up to 36.22% with respect to the scenario with no ridesharing. Finally, our payment

*Corresponding author
Email addresses: filippo.bistaffa@univr.it (Filippo Bistaffa), alessandro.farinelli@univr.it (Alessandro Farinelli), gehalk@intelligence.tuc.gr (Georgios Chalkiadakis), sdr@ecs.soton.ac.uk (Sarvapali D. Ramchurn)

allocation method computes kernel-stable payments for 2000 agents in less than an hour—while the state of the art is able to compute payments only for up to 100 agents, and does so 84 times slower than our approach.
Keywords: Coalition formation, ridesharing, social networks, graphs

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 *Maramoja*[1] or *Lyft*[2] 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 provide an approach that, given the desired starting point and destination, and the time constraints on the pick-up and the arrival of the commuters, can form groups that share cars to lower associated transportation costs (i.e., travel time and fuel), while considering the constraints imposed by the social network. We call this problem the *social ridesharing* (SR) problem. In particular, we provide a model for the SR problem casting this as a Graph-Constrained Coalition Formation (GCCF) problem. Specifically, following relevant literature on GCCF [36, 45], we consider a coalition to be *feasible*, only if the commuters involved in such coalition form a connected subgraph of the social network.

Within such SR scenario, we first address the optimisation problem of minimising the total cost of all the coalitions formed by the system. Given this, we define the value of each coalition as the travel cost of the associated car. Specifically, we present the first model that encodes the above discussed scenario as a

[1] `https://maramoja.co.ke`.
[2] `https://www.lyft.com`.

GCCF problem, and we provide a novel formulation of coalitional values on the basis of the spatial preferences of the agents. Subsequently, we generalise our model incorporating the temporal preferences of the agents, so to allow them to express constraints on the departure and the arriving time. Our approach allows us to derive efficient methods for the computation of the path and the departure time of the driver, which provide the optimal solution the considered model.[3] Finally, we show how to solve the GCCF problem associated to the SR scenario by means of a modified version of CFSS (i.e., the state of the art approach for solving GCCF), a branch and bound algorithm proposed in previous work [7] that is based on the graph-theoretic concept of *edge contraction*. Specifically, in this paper we propose SR-CFSS, which significantly extends CFSS by implementing a novel bounding technique devised for the SR scenario, and by ensuring the validity of the constraints imposed by our SR model.

We empirically evaluate our approach on real-world datasets for both spatial (i.e., GeoLife by Microsoft Research [50]) and social data (i.e., Twitter [30]). Results show that our approach can produce significant cost reductions (up to -36.22% with respect to no ridesharing) and it scales to large numbers of agents, computing approximate solutions for very large systems (i.e., up to 2000 agents) and providing good quality guarantees (i.e., 71% of the optimal in the worst case) within minutes.

Having solved the optimisation problem posed by GCCF, we then turn to the problem of splitting the travel costs (corresponding to each car) among its passengers, i.e., we solve the *payment allocation* problem. Payments to the commuters need to be computed given the passengers' distinct needs (e.g., shorter/longer trips), roles (e.g., drivers/riders, less/more socially connected) and opportunity costs (e.g., taking a bus, their car, or a taxi).

One key aspect of payment allocation in Coalition Formation (CF) is the game-theoretic concept of stability, which measures how agents are keen to maintain the provided payments instead of deviating to a configuration deemed

[3] In general, both these problems are not tractable [27, 31].

to be more rewarding from their individual point of view. Here, we induce stable payments in the context of the SR problem, employing *the kernel* [14] stability concept. Kernel-stable payoffs are perceived as fair, since they ensure that agents do not feel compelled to claim part of their partners' payoff. Kernel stability has been widely studied in cooperative game theory, and various approaches have been proposed to compute kernel-stable payments [29, 39]. However, as we discuss in Section 5, state of the art approaches exhibit inefficiency (i.e., they do not avoid considering infeasible solutions) and redundancy (i.e., they consider coalitions more than once). These drawbacks severely limit their scalability. In contrast, a better way to tackle this problem is to exploit the structure of the graph in order to consider *only* the coalitions that are indeed feasible, so to avoid any unnecessary computation.

We achieve this by means of the PK (Payments in the Kernel) algorithm, our method to compute a kernel-stable allocation given a coalition structure, and we apply it to the SR scenario. In particular, we address the shortcomings of the state of the art algorithm [39] in real-world scenarios, by designing an efficient parallel approach that scales up to thousands of agents. Specifically, we benchmark PK adopting the same realistic environment used for testing SR-CFSS, showing that our method computes payments for 2000 agents in less than an hour and it is 84 times faster than the state of the art in the best case. Moreover, our method can be efficiently parallelised, i.e., it achieves a speed-up of 10.6× on a 12-core CPU with respect to the serial approach.

Finally, we develop new insights into the relationship between payments incurred by a user by virtue of its position in its social network and its role (rider or driver). In general, our experimental results suggest that the kernel can be a valuable stability concept in the context of SR, as it results in a reasonable payment allocation that can be directly correlated with some simple properties of the agents in the system (i.e., network centrality and being a driver/rider).

In more detail, this paper advances[4] the state of the art as follows:

[4]Aspects of this work had already been presented in [8, 9]. This paper presents a signif-

- We model SR as GCCF that considers the desired starting point and destination, and the time constraints on the pick-up and the arrival.
- We propose SR-CFSS, a significantly extended version of CFSS (i.e., the state of the art approach for solving GCCF), to provide optimal solutions, and approximate solutions with quality guarantees for large-scale systems.
- We propose PK, the first approach able to compute kernel-stable payments for systems with thousands of agents.
- We evaluate our algorithms with realistic datasets, i.e., GeoLife from Microsoft Research for the geospatial data and Twitter for social networks. Results show that SR-CFSS 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 a maximum performance ratio of 1.41 in the worst case). PK is able to compute payments for 2000 agents in less than an hour and it is 84 times faster than the state of the art in the best case.
- We analyse the relationship between payments incurred by a user by virtue of its position in the social network and its role (rider/driver).

The rest of the paper is organised as follows. Section 2 illustrates the background on the GCCF problem, the kernel, and discusses the relationship between our work and the existing literature. Section 3 details our GCCF model for SR and Section 4 extends such model to include time constraints. In Section 5 we discuss PK, our approach to compute kernel-stable payments. Sections 6 and 7 present our experimental evaluation, where we benchmark SR-CFSS and PK, respectively. Section 8 concludes the paper and outlines future work.

icantly extended model, which allows each commuter to express temporal preferences, i.e., agents can specify an ideal pick-up and arriving time. This extension is of utmost importance in order to provide a SR model that can be applied in realistic scenarios, in which time significantly influences the travel needs of the commuters and, hence, plays a fundamental role for SR. Furthermore, we conduct an additional experimental evaluation on real-world datasets, in order to investigate the influence of the introduction of time constraints on our approach.

2. Background & related work

The purpose of this section is threefold. In Section 2.1 we define the GCCF problem, and we provide some background on the state of the art algorithm to solve it, i.e., CFSS. In Section 2.2 we discuss payment computation and the stability concept adopted by our approach, i.e., the kernel. Finally, in Section 2.3 we discuss the relationship between our work and the literature on ridesharing.

2.1. GCCF problem definition

The *Coalition Structure Generation* (CSG) problem [38] takes as input a finite set of n agents $A = \{a_1, \ldots, a_n\}$ and a characteristic function $v : 2^A \rightarrow \mathbb{R}$, that maps each coalition $S \in 2^A$ 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 $\Pi(A)$. The value of a coalition structure CS is assessed as the sum of the values of its composing coalitions, i.e., $V(CS) = \sum_{S \in CS} v(S)$.

CSG aims at identifying CS^*, the most valuable coalition structure, i.e., $CS^* = \arg\max_{CS \in \Pi(A)} V(CS)$. The computational complexity of the CSG problem is due to the size of its search space. In fact, a set of n agents can be partitioned in $\Omega((\frac{n}{\ln(n)})^n)$ ways, i.e., the n^{th} Bell number [6], since, in standard CSG, every possible subset of agents is potentially a valid coalition.

In many realistic scenarios, constraints influence the process of coalition formation. Following the work of Myerson [36] and Demange [16], and more recent work by Voice et al. [44, 45], in this paper we focus on a specific type of constraints that encodes synergies or relationships among the agents and that can be expressed by a graph, where nodes represent agents and edges encode the relationships between the agents. In this setting, edges enable connected agents to form a coalition and a coalition is considered feasible only if its members represent the vertices of a connected subgraph. In order to model these settings, Myerson [36] first proposed a definition of *feasible* coalition by considering an undirected graph $G = (A, E)$, where $E \subseteq A \times A$ is a set of edges between agents, representing the relationships between them:

Definition 1 (feasible coalition)**.** *A coalition S is* feasible *if all of its members are connected in the subgraph of G induced by S, i.e., for each pair of players $a_i, a_j \in S$ there is a path in G that connects a_i and a_j without going out of S.*

Thus, given a graph G the set of feasible coalitions is $\mathcal{FC}(G) = \{S \subseteq A \mid$ The subgraph induced by S on G is connected$\}$. A Graph-Constrained Coalition Formation (GCCF) problem is a CSG problem together with a graph G, where a coalition S is considered feasible if $S \in \mathcal{FC}(G)$. In GCCF problems a coalition structure CS is considered feasible if each of its coalitions is feasible, i.e., $\mathcal{CS}(G) = \{CS \in \Pi(A) \mid CS \subseteq \mathcal{FC}(G)\}$. The goal for a GCCF problem is to identify CS^*, which is the most valuable feasible coalition structure, i.e., $CS^* = \arg\max_{CS \in \mathcal{CS}(G)} V(CS)$. After the definition of the GCCF problem, we now present CFSS, the state of the art algorithm to solve it.

2.1.1. The CFSS algorithm

CFSS [7] is a search-based algorithm that solves the GCCF problem. CFSS works by representing the solution space $\mathcal{CS}(G)$ of the GCCF problem as a rooted search tree, which is guaranteed to contain each $CS \in \mathcal{CS}(G)$ only once without any redundancy. Each solution in such search tree is constructed by a unique sequence of *edge contraction* operations starting from the initial graph G. Intuitively, the contraction of an edge represents the merging of the coalitions associated to its incident vertices, as shown in Figure 1. Such an operation can be used to generate the entire search space $\mathcal{CS}(G)$, which can be traversed with polynomial memory requirements in order to find the optimal solution. This is possible because CFSS is based on a depth-first traversal (see Algorithm 9 in Appendix C) of the search tree, and, hence, at each point of the search, only the ancestors of the current node need to be maintained in memory. For this reason, CFSS can solve large-scale GCCF instances with more than 2000 agents [7]. The superior scalability of CFSS with respect to classic CSG solution algorithms is also possible thanks to the reduced search space that must be explored, due to the presence of the graph that constrains the number of feasible coalitions. An in-depth discussion about the CFSS algorithm is provided by Bistaffa et al. [7].

(a) Before contraction (b) After contraction

Figure 1: Example of an edge contraction.

Having discussed the optimisation part of GCCF, we now discuss the second fundamental aspect of CF, i.e., payment computation, which, as mentioned in the introduction, is crucial in the context of SR.

2.2. Payment computation

The *payment computation* problem involves the computation of a *payoff vector* x, which specifies a payoff $x[i]$ for each agent $a_i \in A$ as a compensation of their contributions. This problem has been thoroughly studied in the cooperative-game theory literature, thus we suggest the reader to refer to the book by Chalkiadakis et al. [10] for a more extensive discussion of all the technical aspect on this subject. In the context of this discussion, we are particularly interested in computing payoff vectors that are *efficient* (i.e., the entire value of S is split among the members of S) and *individually rational* (i.e., each agent a_i receives a payoff $x[i]$ that is *at least* the value of its singleton). Efficiency and individual rationality are fundamental in real-world applications such as SR, as they formalise natural properties that are often assumed in practice. Efficiency expresses the principle that the entire travel cost of each car should be divided among its passengers, while individual rationality states that a rational agent does not join a car if such an action results in a cost higher than going alone.

Furthermore, computing payments that are *stable* is of utmost importance in systems with selfish rational agents, i.e., agents who are only interested in the maximisation of their payoffs [10]. As such, payoffs have to be distributed among agents to ensure that members are rewarded according to their bargaining power [10]. Stability ensures that agents will not deviate from the provided solution to a different one that is better from their individual point of view. In

cooperative game-theory, stability has been defined with several concepts, e.g., the stable set, the nucleous, the kernel, and the core [10]. The *core* is one of the most widely studied stability concepts, but its computation has an exponential complexity with respect to the number of agents [11]. As such, it is not suitable for large-scale systems, as confirmed by our past research [43]. Furthermore, it is not guaranteed that core-stable solutions always exist [10], as evidenced by our experiments in Appendix B. Thus, in this paper we focus on the kernel.

2.2.1. The kernel

The *kernel* is a stability concept introduced by Davis and Maschler [14]. A key feature of the kernel is that it is always possible to compute a kernel-stable payoff allocation. Moreover, a number of approaches [8, 29] can compute an approximation of the kernel when the size of coalitions is limited in a polynomial time with respect to the number of agents. The kernel provides stability within a given coalition structure, and under a given payoff allocation, by defining how payoffs should be distributed so that agents cannot *outweigh* (cf. below) their current partners, i.e., the other members of their coalition. Kernel-stable payoffs are perceived as fair,[5] since they ensure that agents do not feel compelled to claim part of their partners payoff. We define the *excess* of a coalition S with respect to a given payoff vector x as $e(S,x)=v(S)-x(S)$, where $x(S) = \sum_{a_i \in S} x[i]$. A positive excess is interpreted as a measure of threat: in the current payoff distribution, if some agents deviate by forming coalition with positive excess, they are able to increase their payoff by redistributing the coalitional excess among themselves. On this basis, we define the notion of *surplus*.

Definition 2 (surplus)**.** *Given a coalition structure CS and a coalition $S \in CS$,*

[5] Fairness can also be achieved by considering the *Shapley value* [10]. Nonetheless, computing the Shapley value is computationally intractable in general and next to impossible in large settings (see, e.g., [17, 33]). Of course, approximation approaches exist for specific classes of games [4] and in fact a fully polynomial-time Shapley-value approximation scheme does exist for super-modular environments [32]; however, super-modularity cannot be readily assumed in our domain. At the same time, practical algorithms for approximating the Shapley value in graph-restricted games have recently appeared [40]. Testing these approaches in our domain is future work.

we consider $a_i, a_j \in S$*. Then, the* surplus s_{ij} *of* a_i *over* a_j *with respect to a given payoff configuration* x*, is defined by*

$$s_{ij} = \max_{\substack{S' \in 2^A \\ a_i \in S', a_j \notin S'}} e\left(S', x\right), \tag{1}$$

In other words, s_{ij} is the maximum of the excesses of all coalitions S' that include a_i and exclude a_j, with S' not in the given coalition structure CS (since under CS agents a_i and a_j belong to the same coalition S). We say that agent a_i outweighs agent a_j if $s_{ij} > s_{ji}$. When this is the case, a_i can claim part of a_j's payoff by threatening to walk away (or to expel a_j) from their coalition. When any two agents in a coalition cannot outweigh one another, the payoff vector lies *in the kernel* – i.e., it is stable. In addition, the set of kernel-stable payoff vectors is always non-empty [10].

Stearns [41] provides a *payoff transfer scheme* which converges to a vector in the kernel by means of payoff transfers from agents with less bargaining power to their more powerful partners, until the latter cannot claim more payoff from the former. Unfortunately, this may require an infinite number of steps to terminate. To alleviate this issue, Klusch and Shehory [29] introduced the ϵ-kernel in order to represent an allocation whose payoffs do not differ from an element in the kernel by more than ϵ. The current state of the art approach to compute an ϵ-kernel payoff allocation for classic CF has been proposed by Shehory and Kraus [39] (see Algorithm 4 in Section 5). Such an algorithm does not specify how x should be initialised, and assumes that a payoff vector is provided as an input. The first (and most expensive) phase is the computation of the *surplus matrix* s, which iterates over the entire set of coalitions to assess the maximum excess (Equation 1) for each pair of agents in each coalition. Once the surplus matrix has been computed, a transfer between the pair of agents with the highest surplus difference (i.e., $s_{ij} - s_{ji}$) is set up, while ensuring that each payment is individually rational. This scheme is iteratively executed until the ratio between the maximum surplus difference δ and the value of the considered coalition structure is within a predefined parameter ϵ. This ensures

that the computed payoff allocation is ϵ-kernel stable. On the one hand, the computation of Equation 1 is a key bottleneck for classic CF, since it involves enumerating an exponential number of coalitions, i.e., $\Theta(2^n)$. On the other hand, when the size of the coalitions is limited to k members, such an algorithm has polynomial time complexity [39], since the coalitions are $O(n^k)$ [29].

Despite having polynomial time complexity under certain assumptions, such an approach has some drawbacks that hinder its applicability in real-world scenarios, and especially in the SR scenario we consider. First, it is designed for classic CF, failing to exploit the graph-constrained nature of this problem. Second, this algorithm assumes that computation of the characteristic function has a $O(1)$ time complexity (e.g., coalitional values are stored in memory or provided by an oracle). This hypothesis, although appropriate in several settings, does not apply to SR, in which the value of a coalition is the solution of a routing problem, which cannot be assessed with a $O(1)$ time complexity (see Sections 3.1 and 3.2). Furthermore, coalitional values cannot be stored in memory, as it would require tens of GB even for medium-sized instances (e.g., 100 agents). These shortcomings lead to inefficiencies that prevent the application of the method proposed by Shehory and Kraus in our case (see Section 5).

Having discussed the approaches to address the two main aspects of CF, in the next section we elaborate on the existing literature on ridesharing.

2.3. Related work

We now elaborate on related work in the areas of ridesharing and temporal constraint optimisation.

2.3.1. Ridesharing

Ridesharing poses several challenges that have been addressed by a number of works in the Artificial Intelligence literature. In the context of optimisation, most studies [5, 22, 46, 47] tackle only one or two particular objectives [1] among the followings: minimise the overall distance travelled by the cars in the system, minimise the overall travel time, or maximise the number of participants. This

allows to achieve solutions of tractable computational complexity, but, on the other hand, these approaches do not generalise to scenarios such as SR, in which a more complex cost model is considered (see Section 3). Specifically, Baldacci et al. [5] adopt a Lagrangian column generation method in order to compute the optimal way of assigning commuters to cars while minimising unmatched participants. A similar objective is pursued by Ghoseiri et al. [22], who also try to fulfil individual preferences such as age, gender, smoke, and pet restrictions. Several papers [46, 47] consider an agent-based system where autonomous rider and driver agents locally establish ride-shares with the objective of maximising the number of served riders. Winter and Nittel [46] consider a setting where short-range wireless communication devices (e.g., Bluetooth or WiFi) are used, showing that limiting the information dissemination between agents provides a benefit in terms of computation requirements, while it does not significantly impact the solution quality. Xing et al. [47] consider a highly dynamic ride-share system where drivers and riders are matched *en-route*.

Recently, Kamar and Horvitz [26] addressed the computational aspects related to ridesharing, proposing an interesting model to evaluate ridesharing plans, on which we base our model for SR. In particular, such work is mostly focused on incentive design aspects for ridesharing. Similarly, Kleiner et al. [28] and Zhao et al. [49] tackle the same challenge adopting a mechanism design perspective. Now, we also focus on the computation of incentives (in the form of payments), but, in contrast with the above works, we focus on the computation of payoffs that are stable in a game-theoretic sense, which is fundamental in contexts with selfish rational agents.

As a final remark, notice that none of the works in the literature consider the role of the social network as we do, which, as mentioned in the introduction, is crucial for real-world ridesharing services.

2.3.2. Temporal constraint optimisation

Problems involving time constraints arise in various areas of computer science, especially in the context of scheduling [15] and vehicle routing [13, 42]. In par-

ticular, Dechter [15] define the so-called *Simple Temporal Problems* (STP), a particular type of Constraint Satisfaction Problem (CSP) in which a variable τ_i corresponds to a continuous time point and a binary constraint (τ_i, τ_j) is associated to one time range that contains the valid values for $\tau_j - \tau_i$. In the context of SR, if τ_1 and τ_2 are respectively the departure and the arrival time for a particular agent, the constraint (τ_1, τ_2) associated to the range $[0', 60']$ means that its arrival cannot happen more than 60 minutes after its departure. Khatib et al. [27] later extended the concept of STP associating a function (i.e., a *preference*) to each constraint, in order to differentiate among valid solutions and select the one that best meets such preferences. The authors also characterise the complexity of solving such problem (denoted as STPP) as NP-Complete in the general case, while it is tractable if preferences are expressed by linear functions. Such complexity results by virtue of the fact that STPPs with linear preferences can be expressed as Linear Programming (LP) problems, which can be solved in polynomial time [12]. However, even if our preferences are linear (Equations 12 and 13), our time domain is discrete, resulting in a problem of Integer LP, which is NP-Hard in the general case [12]. Nonetheless, our formalisation allows us to restrict such problem to a particular, tractable case. Specifically, our scenario requires to compute only the optimal departure time for the first point in the path, i.e., τ_S^*, since we assume no delay between the arrival to a point and the departure for the next point in the path (Equation 11).[6]

These challenges have also been studied in the Vehicle Routing Problem (VRP) literature [13, 42], and, specifically, in the context of the Vehicle Routing Problem with Time Windows (VRPTW) [25]. To the best of our knowledge, these works adopt a different perspective with respect to our approach, as their main focus is on logistics challenges (e.g., routing, scheduling). While these

[6] If we drop such an assumption (i.e., we allow a delay between the arrival to each point and the departure for the next one), our model can be easily formalised as an STPP. Even if such an STPP is still untractable in the general case due to the discretisation of the time domain, it can be transformed to a tractable LP problem by means of LP relaxation techniques [12]. Further investigation is required to determine the effectiveness of such an approach, which will be considered as future work.

aspects also appear in our paper, here we are mainly interested in studying and solving the SR problem from a CF perspective, i.e., solving the CSG and the payment computation problems.

3. A GCCF approach for SR

The *social ridesharing* (SR) problem considers a set of riders $A = \{a_1, \ldots, a_n\}$, where $n > 0$ is the total number of riders, and a non-empty[7] set of drivers $D \subseteq A$, containing the riders owning a car. Notice that our SR approach cannot increase the number of cars in the system, i.e., SR can only improve the traffic or leave it unaffected. Every driver $a_i \in D$ can host up to $seats\,(a_i)$ riders in his car, including himself, where the function $seats : A \to \mathbb{N}^0$ provides the number of seats of each car. If $a_i \notin D$, then $seats\,(a_i) = 0$.

The *map* of the geographic environment in which the SR problem takes place is represented by a connected graph $M = (P, Q)$, where P is the set of geographic points of the map and $Q \subseteq P \times P$ is the set of edges among these points. Each edge is associated to a *length* by means of the function $\lambda : Q \to \mathbb{R}^+$, where $\mathbb{R}^+ = \{i \in \mathbb{R} \mid i \geq 0\}$. Similarly, we define $\mathbb{R}^- = \{i \in \mathbb{R} \mid i \leq 0\}$. $\mathbb{N}^+$ and $\mathbb{N}^-$ are defined in the corresponding ways.

Definition 3 (path)**.** *A path composed by m points, each belonging to P, is represented as an m-tuple, denoting as $L[k]$ the k^{th} point of L. A path is allowed to cross the same point multiple times.*

Definition 4 (set of paths $\mathcal{L}$)**.** *$\mathcal{L}$ is the set of all paths among the points in P.*

Each rider $a_i \in A$ has to commute from a starting point $p_i^\sigma \in P$, i.e., its pick-up point, to a destination $p_i^\omega \in P$.

A key aspect of the SR scenario is the presence of a social network, modelled as a graph $G = (A, E)$ with $E \subseteq A \times A$, which restricts the formation of groups. To this end, we define a *feasible* coalition as follows.

[7] If $D = \emptyset$ the problem is trivial, as the only solution is represented by the singletons.

Definition 5 (feasible coalition for SR)**.** *Given a graph G and a set of riders $S \subseteq A$, S is a* feasible *coalition if it induces a connected subgraph on G, and if it contains at least one rider whose car has enough seats for all the members. Formally, we state such a requirement as follows.*

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

Notice that such a constraint allows a rider $a_i \notin D$ 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 $k\,(\{a_i\})$ for the ticket. Formally, the function $k : A_{single} - D_{single} \rightarrow \mathbb{R}^-$ provides such a cost, where $A_{single} = \{\{a_i\} \mid a_i \in A\}$, and $D_{single} = \{\{a_i\} \mid a_i \in D\}$.[8] If $a_i \in D$, then $\{a_i\}$ is not associated with any value by $k\,(\cdot)$, as we assume that such riders always prefer to use their car with respect to public transport.[9]

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 feasible set of riders S contains at most one driver. Formally, the following additional constraint must hold:

Constraint 2. $|S \cap D| \leq 1$*, i.e., the number of drivers per coalition can be 0 (i.e., S contains a single rider without a car) or 1.*

Notice that Constraint 2 is optional, but it holds in several established real-world services, arising from aspects of practical nature.[10] Nonetheless, since our approach supports a more general model, it can also be applied to scenarios where such a constraint does not hold. Having defined our notion of a feasible

[8] Notice that the function $k\,(\cdot)$ receives a *singleton* formed by a rider as an argument.

[9] This assumption does not impact on the generality of our model, as we assume that each commuter first evaluates its preferences and opportunity costs (i.e., whether it is more convenient to take the car or the public transport) and then, based on this, declares its status of driver or rider, before the execution of the algorithm. Given the short-lived nature of each run, we assume that each agent does not change its status during the execution of the algorithm. Notice that, before the next potential run, each agent is allowed to revise its decision.

[10] For instance, in the *BlaBlaCar* service each driver rides its own car.

coalition, in the following section we detail how we associate a value to each feasible coalition, i.e., we define the *characteristic function* properly.

3.1. Coalitional value definition

When a shared car is arranged, it drives through a path that contains all the starting points and destinations of its passengers. Notice that not all the permutation of these points are valid (e.g., it is not reasonable to go to a rider's destination and then to its starting point). More formally, a *valid* path must fulfil two constraints to correctly accommodate the needs of all the passengers.

Definition 6 (valid path)**.** *Given a feasible set of riders S and a path $L \in \mathcal{L}$ of m points, L is said to be* valid *if the following constraints hold:*

Constraint 3. $\exists a_i \in S : seats\,(a_i) \geq |S| \wedge L[1] = p_i^{\sigma} \wedge L[m] = p_i^{\omega}$*, i.e., L goes from the driver's starting point to its destination.*

Constraint 4. $\forall a_i \in S\, \exists x, y : L[x] = p_i^{\sigma} \wedge L[y] = p_i^{\omega} \wedge x < y$*, i.e., for each rider, its starting point precedes its destination.*

Notice that a valid path can cross the same point multiple times.[11] We refer to the set of all valid paths for a given feasible set of riders S with $\mathcal{VL}\,(S)$. $\mathcal{VL}\,(S)$ is always non-empty, since we assume that M is a connected graph.[12]

Following Kamar and Horvitz [26], we define $v\,(S)$ as

$$v\,(S) = \begin{cases} k\,(S)\,, & \text{if } S \cap D = \emptyset, \\ t\,(L_S^*) + c\,(L_S^*) + f\,(L_S^*)\,, & \text{otherwise,} \end{cases} \tag{2}$$

where L_S^* represents the optimal path for S (Equation 3). On the one hand, if $S \cap D = \emptyset$, Constraint 1 imposes that S is formed by a single rider without a

[11] If a path goes through the starting point/destination of an agent more than once, we assume that the car stops only the first time.

[12] A valid path that always exists starts from the driver's starting point, then, for each passenger, goes to its starting point and to its destination, and ends at the driver's destination.

car, hence its cost is provided by $k(S)$. On the other hand, if S contains at least one driver, its value is the sum of the following negative[13] cost functions:

- $t : \mathcal{L} \to \mathbb{R}^-$, i.e., the time cost of driving through a given path,
- $c : \mathcal{L} \to \mathbb{R}^-$, i.e., the cognitive cost[14] of driving through a given path,
- $f : \mathcal{L} \to \mathbb{R}^-$, i.e., the fuel cost of driving through a given path,

Finally, we define the function $value : \mathcal{L} \to \mathbb{R}^-$ as the sum of the above three functions, i.e., $value(L) = t(L) + c(L) + f(L)$. We assume that such functions are *additive*, as defined in what follows.

Definition 7 (additivity)**.** *A function $z : \mathcal{L} \to \mathbb{R}^-$ is said to be* additive *if, given two paths $L_1, L_2 \in \mathcal{L}$ such that the last point of L_1 is the first of L_2, then $z(L_1 \oplus L_2) = z(L_1) + z(L_1)$, where $\oplus$ represents the concatenation of paths.*

Additivity trivially applies to any cost function in real-world ridesharing scenario. Notice that we do *not* assume that the above cost functions are *monotonic* with respect to the length of the path, i.e., longer paths can results in lower costs. Finally, L_S^* represents the optimal path for S, defined as

$$L_S^* = \arg\max_{L \in \mathcal{VL}(S)} value(L). \tag{3}$$

Considering this, a SR problem can be easily translated into a GCCF problem, as each feasible set of riders is indeed a feasible 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 is NP-hard [31], which would not be solvable in realistic scenarios. Hence, in the next section we show how a reasonable assumption allows us make such computation tractable, by means of well-known optimisation techniques [20].

[13] Since we consider a maximisation problem, we represent costs as negative values.
[14] The fatigue incurred by the driver during the trip [26].

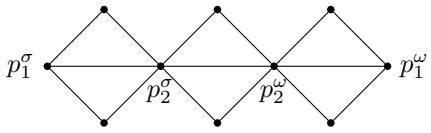

Figure 2: Example starting points and destinations for 2 riders.

3.2. Optimal path computation

The computational complexity of Equation 3 is due to the size of its search space, formed by all the *valid* paths for S, i.e., all the paths in the graph M that contain the starting points and destinations of the members of S in an order that satisfies Constraints 3 and 4. Notice that, given a particular sequence of starting points and destinations that satisfies such constraints, the solution space of Equation 3 contains multiple valid paths, as the following example shows.

Figure 2 shows an example map containing the starting points and destinations of 2 agents, in which only one sequence of points is valid, i.e., $L = \langle p_1^\sigma, p_2^\sigma, p_2^\omega, p_1^\omega \rangle$. Nonetheless, the set of valid paths is much larger (i.e., $3^3 = 27$ valid paths), since there exist 3 possible paths for each of the 3 pairs of consecutive points in L. However, it is reasonable to assume that the driver will go through the shortest path for each of these 3 pairs of points.[15]

Assumption 1. *When the driver has to go from one point in L to the next one, it will choose the* shortest path *(considering $\lambda(\cdot)$) connecting such points.*

Assumption 1 allows us to collapse the search space of Equation 3 to $\mathcal{VT}(S)$.

Definition 8 ($\mathcal{VT}(S)$)**.** *Given a feasible coalition S, $\mathcal{VT}(S)$ is the set of tuples that contain all and only the starting points and destinations of the members of S (without repetitions) and that satisfy Constraints 3 and 4.*

In order to explain how to simplify the solution of Equation 3 given the above assumption, we define the function $concat(\cdot)$.

[15] In the case of a coalition formed by a single driver, the path does not depend on Equation 2, since the driver will always go through the shortest path from its starting point to its destination. In the case of multiple agents per car, the driver can choose among all the valid sequences of points (see Equation 4), and the optimal total path may not be the shortest one.

Definition 9 ($concat\,(\cdot)$)**.** *Given $L \in \mathcal{VT}\,(S)$, the function $concat : \mathcal{VT}\,(S) \to \mathcal{L}$ provides the path obtained as the concatenation of all the shortest paths between one point in L and the following one. Formally, $concat\,(L)$ is a tuple defined as $concat\,(L) = \bigoplus_{k=1}^{|L|-1} sp\,(L[k], L[k+1])$, where $sp : P \times P \to \mathbb{R}^+$ provides the shortest path between two points, considering the length provided by $\lambda\,(\cdot)$.*

The function $concat\,(\cdot)$ can be computed in $O\,((|L|-1) \cdot (|Q| + |P| \cdot \log |P|))$, assuming that $sp\,(\cdot)$ is implemented using Dijkstra's algorithm [18]. Moreover, if M is a euclidean graph, $concat\,(\cdot)$ can be computed in $O\,((n-1) \cdot |Q|)$ with the A* algorithm [24]. Against this background, Equation 3 can be rewritten as

$$L_S^* = \underset{L \in \mathcal{VT}(S)}{\arg\max}\, value\,(concat\,(L))\,, \tag{4}$$

by exploiting the additivity property (Definition 7) of the $value\,(\cdot)$ function. Notice that the search space of Equation 4 is $\mathcal{VT}\,(S)$, which is significantly smaller than $\mathcal{VL}\,(S)$ in Equation 3, and although being still exponential with respect to $|S|$, such computational complexity is manageable for reasonably sized groups of riders. In fact, $\mathcal{VT}\,(S)$ contains only 2520 valid tuples for $|S| = 5$ (i.e., the number of seats of an average car). Such a result allows us to evaluate each coalition S by means of Equation 2, and hence we can address SR as GCCF.

Furthermore, on the basis of Assumption 1 we later formulate Proposition 2, the fundamental theoretical result that allows us to compute an upper bound for the SR characteristic function. This, in turn, allows us to use the CFSS algorithm [7] to solve the SR problem efficiently.

3.3. Solving the Social Ridesharing problem with CFSS

In order to solve the SR problem, the original version of CFSS [7] must be modified to assess the additional constraints introduced in Section 3. In particular, to ensure that Constraint 1 and Constraint 2 hold, we must avoid the formation of coalitions which are not feasible sets of riders. This is achieved by preventing the contractions of the green edges that would result in the violation of such constraints. Notice that such edges must be marked in red (see Section 2.1.1

above), even if we are not visiting the corresponding subtrees. In fact, this is equivalent to traversing such search spaces and discarding any solution they may contain, because such solutions would violate the above mentioned constraints.

A crucial feature of CFSS is the use of a branch and bound search strategy to prune significant parts of the search space, which can be used if the characteristic function is the sum of a superadditive and a subadditive part, i.e., an $m + a$ function [7]. However, Equation 2 is not $m + a$, since it depends on L^*_S, and in particular on the actual position of the starting points and destinations.

As an example, consider Figure 3, which shows the starting points and destinations for 3 riders, i.e., $A = \{a_1, a_2, a_3\}$, in which only a_1 owns a car, i.e., $D = \{a_1\}$. For simplicity, we assume that $v(S)$ is equal to the length of L^*_S, and $k(\{a_2\}) = k(\{a_3\}) = -1$. In this example, $v(\{a_1\}) = -3$, $v(\{a_2\}) = -1$, $v(\{a_3\}) = -1$. However, we notice that p^σ_2 and p^ω_2 are actually part of the path travelled by a_1, hence it is reasonable for a_2 to join a_1 in the coalition $\{a_1, a_2\}$. In fact, $v(\{a_1, a_2\}) = v(\{a_1\}) = -3 > v(\{a_1\}) + v(\{a_2\}) = -3 - 1 = -4$. The optimal path for $S = \{a_1, a_3\}$ is $L^*_S = \langle p^\sigma_1, p^\sigma_3, p^\omega_3, p^\omega_1 \rangle$. On the other hand, a_3's starting point and destination are outside a_1's path, hence ridesharing is not effective in this case: $v(\{a_1, a_3\}) = -7 < v(\{a_1\}) + v(\{a_3\}) = -3 - 1 = -4$. Notice that this particular characteristic function cannot be seen as the sum of a superadditive and a subadditive part, since it exhibits a superadditive behaviour for some coalition structures, i.e., $v(\{a_1, a_2\}) > v(\{a_1\}) + v(\{a_2\})$, while it is subadditive for some others, i.e., $v(\{a_1, a_3\}) < v(\{a_1\}) + v(\{a_3\})$.

Hence, in the next section we provide alternative bounding techniques that can be used in our ridesharing scenario to use branch and bound within CFSS.

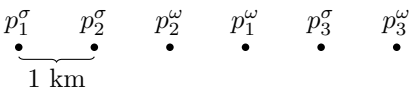

Figure 3: Example starting points and destinations for 3 riders.

3.3.1. Bound computation

Given a feasible coalition structure CS in our search tree, we now show how to compute an upper bound $M(CS)$ for the values assumed by the characteristic function in $ST(CS)$, i.e., $M(CS) \geq V(CS_i)\ \forall CS_i \in ST(CS)$, where $ST(CS)$ is the subtree (i.e., a portion of the entire search tree) rooted at the node corresponding to CS. $ST(CS)$ can be also seen as the set of all coalition structures that *cover* CS (i.e., $ST(CS) = \{CS' \mid \forall S \in CS\, \exists S' \in CS'$ such that $S \subseteq S'\}$). We use this value to avoid visiting $ST(CS)$ if $M(CS)$ is not greater than the current best solution. This allows us to realise the same pruning technique discussed in [7] in the context of $m + a$ functions.

In what follows, we provide a method to compute $M(CS)$ in scenarios where Constraint 2 holds. In these environments it is not possible to merge two coalitions that each contain one driver, since only single riders not owning a car are allowed to join existing groups. It is easy to see that the addition of a rider to a feasible coalition S can only result in a greater cost. This principle allows us to prove Proposition 1. First, we define $A_d(CS)$.

Definition 10 ($A_d(CS)$)**.** *Given a coalition structure CS, $A_d(CS)$ is the set of coalitions in CS that contain at least one driver, i.e., the set of cars. Formally, $A_d(CS) = \{S \in CS \mid S \cap D \neq \emptyset\}$.*

Proposition 1. *If Constraint 2 holds, for any feasible coalition structure CS*

$$M_1(CS) = \sum\nolimits_{S \in A_d(CS)} v(S) \tag{5}$$

is an upper bound for the value of any CS' in $ST(CS)$, i.e., the subtree rooted in CS. Formally, $M_1(CS) \geq V(CS')$ for all $CS' \in ST(CS)$.

Proof. See Appendix A. □

We now discuss how to compute an upper bound without assuming Constraint 2. We first make the following definitions.

Definition 11 (P_{ab})**.** *P_{ab} is the set of the starting points and destinations of all the riders, i.e., $P_{ab} = \{p \in P \mid \exists a_i \in A : p = p_i^{\sigma}$ or $p = p_i^{\omega}\}$.*

Definition 12 (P_{pairs})**.** *P_{pairs} is the set of all the pairs of different points in P_{ab}, i.e., $P_{pairs} = \{(p, q) \in P_{ab} \times P_{ab} \mid p \neq q\}$.*

Definition 13 ($P_{1,a}(a_i)$ and $P_{1,b}(a_i)$)**.** *Given a rider $a_i \in A$, $P_{1,a}(a_i)$ is the set of all the shortest paths from a_i's starting point to the starting points and destinations of any other rider, i.e., $P_{1,a} = \{L \mid L = sp(p_i^\sigma, p)\ \forall p \in P_{ab} : p \neq p_i^\sigma\}$. Similarly, we define $P_{1,b}(a_i)$ considering a_i's destination.*

Definition 14 ($P_{2,a}(a_i)$ and $P_{2,b}(a_i)$)**.** *Given a rider $a_i \in A$, $P_{2,a}(a_i)$ is the concatenation of all the pairs of shortest paths from a_i's starting point to the starting points and destinations of any other rider, i.e., $\{L \mid L = sp(p_i^\sigma, p) \oplus sp(p_i^\sigma, q)\ \forall (p, q) \in P_{pairs} : p \neq p_i^\sigma$ and $q \neq p_i^\sigma\}$. Similarly, we define $P_{2,b}(a_i)$ considering a_i's destination.*

Definition 15 ($m(\cdot)$)**.** *The function $m : A \to \mathbb{R}^-$ is defined as*

$$m(a_i) = \begin{cases} \max\limits_{L \in P_{1,a}(a_i)} value(L) + \max\limits_{L \in P_{1,b}(a_i)} value(L), & \text{if } a_i \in D \quad (6) \\ \max\limits_{L \in P_{2,a}(a_i)} value(L) + \max\limits_{L \in P_{2,b}(a_i)} value(L), & \text{otherwise.} \quad (7) \end{cases}$$

Intuitively, the purpose of $m(a_i)$ is to provide an upper bound on the $value(\cdot)$ function corresponding to the edges incident on p_i^σ and p_i^ω, when such edges are part of a path L driven by a car. If a_i is a driver, such an upper bound is calculated by considering the best edges (i.e., the ones that maximise $value(\cdot)$) incident on each point (Equation 6). In contrast, if a_i is not a driver, Equation 7 considers the best pairs of edges incident on each point. We now provide an example to better explain how the $m(\cdot)$ function is calculated.

Figure 4 shows the starting points and destinations of 3 riders, in which the edges represent the shortest paths between any pair of points (under Assumption 1). Furthermore, assume that a coalition is formed among such agents, and that a_1 and a_2 are drivers, while a_3 is not. Notice that we do not know in advance whether a_1 or a_2 will be the optimal driver of such a car. For the sake of brevity, in the following discussion we only refer to the agents' starting points, but the

same concepts apply symmetrically to the destinations. Notice that, since a_3 is not a driver, p_3^σ will necessarily be an inner point in L (Constraint 3). It follows that L contains exactly two undirected edges incident on p_3^σ. Since we are interested in computing an upper bound on $value(\cdot)$, we consider the pair of edges incident on p_3^σ that maximises such function (Equation 7).

On the other hand, since we do not know in advance if a_1 (resp. a_2) will be the optimal driver of the car, we cannot predict whether p_1^σ (resp. p_2^σ) will be the first point or an inner point in L. In other words, we do not know exactly whether one or two edges incident on p_1^σ (resp. p_2^σ) will be part of L. Therefore, in Equation 6 we assume that only one edge is present in L, as a conservative measure. This is guaranteed to provide an upper bound on $value(\cdot)$, as such a function is negative and, hence, the value of the best pair of edges is lower than the value of the best single edge. We now define $M_2(\cdot)$ on the basis of $m(\cdot)$.

Definition 16 ($M_2(\cdot)$)**.** *The function $M_2 : \mathcal{CS}(G) \to \mathbb{R}^-$ is defined as*

$$M_2(CS) = \frac{1}{2} \cdot \sum\nolimits_{a_i \in U_d(CS)} m(a_i), \quad \text{where } U_d(CS) = \bigcup\nolimits_{S \in A_d(CS)} S. \tag{8}$$

Intuitively, $U_d(CS)$ is the set of all agents (both riders and drivers) that are passengers of a car in CS. The $\frac{1}{2}$ term is necessary since, if we sum all the values of the couples of edges incident to the points that form a given path, we consider each edge twice.

We now prove the following lemma, that will support the proof of Proposition 2.

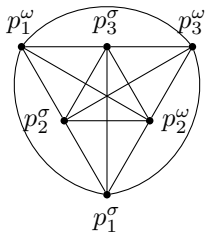

Figure 4: Example starting points and destinations for 3 riders.

Lemma 1. *Given a feasible coalition structure CS and a coalition structure $CS' \in ST(CS)$ such that $V(CS') > M_2(CS)$, then*

$$\exists S' \in A_d(CS') : v(S') > \frac{1}{2} \cdot \sum\nolimits_{a_i \in S'} m(a_i). \tag{9}$$

Proof. See Appendix A. □

Building upon this lemma, we now prove Proposition 2. Notice that, as previously mentioned, Proposition 2 is based on the validity of Assumption 1, which is also the key concept that allows us to compute the optimal path for a given coalition using Equation 4 in a feasible amount of time.

Proposition 2. *If Assumption 1 holds, for any feasible coalition structure CS $M_2(CS)$ is an upper bound for the value of any CS' in $ST(CS)$, i.e., the subtree rooted in CS. Formally, $M_2(CS) \geq V(CS')$ for all $CS' \in ST(CS)$.*

Proof. See Appendix A. □

Propositions 1 and 2 allows us to compute an upper bound on $V(\cdot)$ for all the coalition structures in $ST(CS)$. As a consequence, we can solve the SR problem by adopting a branch and bound approach based on CFSS [7], which we call SR-CFSS (Algorithm 1). In addition to the different bounding technique, SR-CFSS differs from CFSS as it enforces Constraint 1 and, optionally, Constraint 2, which ensure the computation of a correct solution for the SR problem. Specifically, this is achieved by including an additional check (i.e., line 4 in the SR-CHILDREN routine) that discards the solutions that violate such constraints. Notice that SR-CFSS derives all the anytime approximate properties of CFSS, since we can apply the technique discussed in [7] using $M_1(\cdot)$ or $M_2(\cdot)$ respectively defined in Equations 5 and 8. Intuitively, we can stop the execution of SR-CFSS after a given time budget and provide the best coalition structure found during the search. Furthermore, we employ our bounding techniques to compute an upper bound on the value of the characteristic function in the remaining part of the search space. In our experiments in Section 6.5, SR-CFSS provides solutions that are guaranteed to be at least the 71% of the optimal.

Algorithm 1 SR-CFSS(G)

$\mathcal{G}_c \leftarrow G$ with all green edges
$best \leftarrow \mathcal{G}_c$ {Initialise current best solution with singletons}
$Front \leftarrow \emptyset$ {Initialise search frontier $Front$ with an empty stack}
$Front$.PUSH $(\mathcal{G}_c)$ {Push $\mathcal{G}_c$ as the first node to visit}
while $Front \neq \emptyset$ **do** {Branch and bound loop}
  $node \leftarrow Front$.POP () {Get current node}
  if $M_2\,(node) > V\,(best)$ **then** {Can also use $M_1\,(node)$ with Constraint 2}
    if $V\,(node) > V\,(best)$ **then** {Check function value}
      $best \leftarrow node$ {Update current best solution}
    $Front$.PUSH (SR-CHILDREN $(node)$) {Update $Front$}
return $best$ {Return optimal solution}

The model defined in Section 3 takes into account only the spatial aspect of the SR problem. In what follows, we show how to incorporate the time preferences of the commuters in our model and algorithms.

Algorithm 2 SR-CHILDREN($\mathcal{G}_c$)

$\mathcal{G}'_c \leftarrow \mathcal{G}_c = (\mathcal{A}, \mathcal{E}, colour)$ {Initialise graph G' with $\mathcal{G}_c$}
$Ch \leftarrow \emptyset$ {Initialise the set of children}
for all $e \in \mathcal{E} : colour\,(e) = green$ **do** {For all green edges}
  if GREENEDGECONTR $(\mathcal{G}'_c, e)$ meets Constr. 1 (and Constr. 2) **then**
    $Ch \leftarrow Ch \cup \{$GREENEDGECONTR $(\mathcal{G}'_c, e)\}$
  Mark edge e with colour red in $\mathcal{G}'_c$
return Ch {Return the set of children}

4. Time constraints for Social Ridesharing

We now assume that each rider $a_i \in A$ specifies its desired departure time τ_i^σ within the *time window* $\theta_i^\sigma = [\tau_i^\sigma - \alpha_i^\sigma, \tau_i^\sigma + \beta_i^\sigma] \subseteq \mathbb{N}^+$.[16] Similarly, a_i expresses its preferences on the arriving time τ_i^ω by means of the time window $\theta_i^\omega = [\tau_i^\omega - \alpha_i^\omega, \tau_i^\omega + \beta_i^\omega] \subseteq \mathbb{N}^+$. Figure 5 shows an example of such time constraints. Without loss of generality, we assume that each agent a_i expresses *reasonable* time preferences on the arriving time (e.g., if a_i is a driver, the arriving time

[16] We consider a *discrete* time domain, e.g., seconds or minutes.

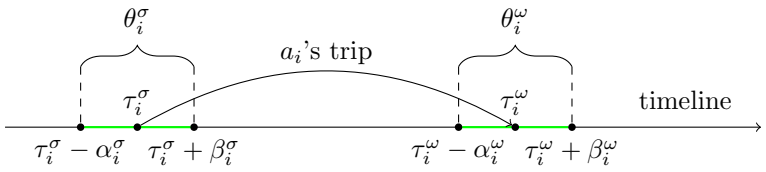

Figure 5: Example of departure and arriving time constraints.

under the hypothesis that she drives alone should not be outside θ_i^ω). Notice that, for the sake of generality, our model includes both the upper bound for start time and lower bound for arrival time, even if they may appear not very common. Such bounds can be easily "removed" by setting them to $+\infty$ and $-\infty$, respectively. We include these time preferences in Equation 2 as follows:

$$v(S) = \begin{cases} k(S), & \text{if } S \cap D = \emptyset, \\ t(L_S^*) + c(L_S^*) + f(L_S^*) + \theta_S(L_S^*, \tau_S^*), & \text{otherwise,} \end{cases} \tag{10}$$

where L_S^* and τ_S^* represent the optimal path and the optimal departure time for S (Equation 15), respectively. We define $value(L_S^*, \tau_S^*) = t(L_S^*) + c(L_S^*) + f(L_S^*) + \theta_S(L_S^*, \tau_S^*)$. We introduce the term $\theta_S : \mathcal{VL}(S) \times \mathbb{N}^+ \to \mathbb{R}^-$ as a measure of the extent to which the time preferences of the members of S have been fulfilled by a trip starting at a given time and going through a given valid path. We quantify such an extent with a cost for each starting point and destination that is proportional to the time difference between the actual pick-up/arriving time and the desired one. In order to formally define such a cost, denoted as $\theta_S(L, \tau)$ (where τ is the departure time), we first define the arrival time at each point p of L given τ as $time_L : L \times \mathbb{N}^+ \to \mathbb{N}^+$, i.e.,

$$time_L(p, \tau) = \tau + \sum_{k=1}^{j-1} \delta(L[k], L[k+1]), \quad \text{where } L[j] = p, \tag{11}$$

where $\delta : Q \to \mathbb{N}^+$ measures the travel time trough a given edge. We assume that $\delta(\cdot)$ does not change during the execution of the algorithm, given its short-lived nature. We define $\Delta_S^\sigma : S \times \mathcal{VL}(S) \times \mathbb{N}^+ \to \mathbb{R}^-$ and $\Delta_S^\omega : S \times \mathcal{VL}(S) \times \mathbb{N}^+ \to \mathbb{R}^-$

(for starting points and destinations respectively) for each $a_i \in S$, i.e.,

$$\Delta_S^{\sigma}(a_i, L, \tau) = \begin{cases} \gamma_1 \cdot |time_L(p_i^{\sigma}, \tau) - \tau_i^{\sigma}|, & \text{if } time_L(p_i^{\sigma}, \tau) \in \theta_i^{\sigma}, \\ -\infty, & \text{otherwise,} \end{cases} \tag{12}$$

$$\Delta_S^{\omega}(a_i, L, \tau) = \begin{cases} \gamma_2 \cdot |time_L(p_i^{\omega}, \tau) - \tau_i^{\omega}|, & \text{if } time_L(p_i^{\omega}, \tau) \in \theta_i^{\omega}, \\ -\infty, & \text{otherwise,} \end{cases} \tag{13}$$

Finally, we define $\theta_S(L, \tau)$ as

$$\theta_S(L, \tau) = \sum\nolimits_{a_i \in S} \Delta_S^{\sigma}(a_i, L, \tau) + \Delta_S^{\omega}(a_i, L, \tau), \tag{14}$$

where τ is the departure time and $\gamma_1, \gamma_2 \in \mathbb{R}^+$ are the costs associated to one time unit of delay/anticipation for starting points and destinations respectively. Notice that, even if other formulations for θ_S are possible, the crucial feature is the enforcement of the hard constraint (i.e., $\theta_S = -\infty$) outside θ_i.

Equations 12 and 13 induce hard constraints on the departure/arriving time for each $a_i \in S$, as each a_i is *not* willing to leave/arrive earlier that $\tau_i - \alpha_i$ nor later than $\tau_i + \beta_i$. Thus, we define $\theta_S = -\infty$ if any of the constraints is violated.

Definition 17 (temporally infeasible coalition)**.** *S is said to be* temporally infeasible *if $\theta_S(L, \tau) = -\infty$ for all $L \in \mathcal{VT}(S)$ and all $\tau \in \theta_j^{\sigma}$ $\forall a_j \in S \cap D$.*

We define the optimal path L_S^* and the optimal departure time τ_S^* as follows:

$$(L_S^*, \tau_S^*) = \underset{\substack{L \in \mathcal{VL}(S) \\ \tau \in \theta_j^{\sigma} \forall a_j \in S \cap D}}{\arg\max} \; value(L, \tau). \tag{15}$$

We reduce the search space for L_S^* in Equation 15 by applying the same techniques discussed in Section 3.2 (i.e., by considering Assumption 1) and obtaining

$$(L_S^*, \tau_S^*) = \underset{\substack{L \in \mathcal{VT}(S) \\ \tau \in \theta_j^{\sigma} \forall a_j \in S \cap D}}{\arg\max} \; value(concat(L), \tau). \tag{16}$$

In Equation 16, the computation of τ_S^* is still carried out in a naïve way, going through every possible timestep in the time windows specified by the drivers in S. In the following section, we explain a better approach to compute τ_S^*.

4.1. Optimal departure time computation

In this section we address the problem of computing the optimal departure time τ_S^* for a given coalition S. Specifically, we now propose an algorithm to compute the best departure time for a car S (given a tuple $L \in \mathcal{VT}(S)$ and a driver $a_j \in S \cap D$), so to avoid trying every possible departure time for the trip of S. Algorithm 3 achieves this by considering the ideal departure time of the driver, i.e., τ_j^σ, and by applying a sequence of shifts so to obtain the optimal τ.

First (lines 1–7), we initialise τ with the ideal departure time of the driver, and we initialise p, n and z, which will respectively count the number of points in which we register an arriving time that is late, early or on time, with respect to the desire expressed by the agents for those points. The variables *post* and *antic* function as guards to check to what extent it is possible to delay/anticipate departure without violating any time constraint. Finally, we also define *diffs* which contains the difference between the actual and the ideal time, for every point in L. Lines 8–11 set these variables. After this, at line 12 we check whether it is possible to satisfy all the time constraints. If the conditional statement is true, then at least two points are outside of their time window, one is late and one is ahead of time, or it may be necessary to anticipate τ of a given amount, but such action would result in the violation of another constraint. In such cases we return a null solution. If no constraints are violated, we improve τ in the cycle at lines lines 13–26 so that $\sum_{i=1}^{|L|} |\mathit{diffs}[i]|$ is minimised and does not invalidate any constraint. Notice that, to achieve this result, it is enough to check the direction of the points of the path. On the one hand, if the majority of the points are positive (the car is late) we anticipate τ. On the other hand, if the majority of the points are negative (the car is early) we delay τ.

Once we have a method to compute the optimal τ given a tuple $L \in \mathcal{VT}(S)$ and a driver $a_j \in S \cap D$, we can finally compute the optimal path L_S^* and the

Algorithm 3 OptimalDepTime(L, a_j)

$\tau \leftarrow \tau_j^{\sigma}$ {Initialise current best solution with driver's ideal departure time}
$p \leftarrow 0$ {Positive points counter (i.e., points where L is late)}
$n \leftarrow 0$ {Negative points counter (i.e., points where L is early)}
$z \leftarrow 0$ {Zero points counter (i.e., points where L is on time)}
$post \leftarrow +\infty$ {Maximum delay without constraint violation}
$antic \leftarrow -\infty$ {Maximum anticipation without constraint violation}
$diffs \leftarrow \langle$ {Differences among ideal times and actual times}
for all $i \in \{1, \ldots, |L|\}$ **do** {For all tuple points}
    $diffs[i] \leftarrow$ difference between $time_L(i, \tau)$ and ideal arriving time at $L[i]$
    Increment p or n or z based on the sign of $diffs[i]$
    Update $post$ and $antic$
if $post < antic$ **then return** $\emptyset$ {Conflict between two constraints}
repeat
    $shift \leftarrow 0$
    if $post < 0$ **then** $shift \leftarrow post$
    else if $antic > 0$ **then** $shift \leftarrow antic$
    else if $p > n + z$ **and** $antic < 0$ **then** {Majority of points are late}
        $lwp \leftarrow$ lowest positive in $diffs$
        $shift \leftarrow -\min\{lwp, -antic\}$
    else if $n > p + z$ **and** $post > 0$ **then** {Majority of points are early}
        $grn \leftarrow$ greatest negative in $diffs$
        $shift \leftarrow \min\{-grn, post\}$
    if $shift \neq 0$ **then**
        $\tau \leftarrow \tau + shift$
        Update $diffs$, recompute p, n, and z, and update $antic$ and $post$
until $shift = 0$
return τ

optimal driver a_S^* by modifying Equation 16 in the following way:

$$(L_S^*, a_S^*) = \underset{\substack{L \in \mathcal{VT}(S) \\ a_j \in S \cap D}}{\arg\max}\, value\left(concat\left(L\right), \textsc{OptimalDepTime}\left(L, a_j\right)\right). \tag{17}$$

L_S^* and a_S^* are computed by selecting the best combination over the possible valid tuples in $\mathcal{VT}(S)$ and drivers in S. For each of these combinations, we consider the corresponding optimal departure time provided by Algorithm 3. If such algorithm returns a null solution, the corresponding $value(\cdot)$ is $-\infty$, and hence, that particular combination is discarded. Equation 17 implicitly provides τ_S^*, which is the optimal departure time for the maximising L_S^* and a_S^*. Notice

that, following the same discussion at the end of Section 3.2, the search space of Equation 17 is at most $2520 \cdot 5 = 12600$ combinations of valid tuples and drivers for $|S| = 5$, and thus, it can be exhausted with a manageable effort.

For some coalitions it is impossible to satisfy all time constraints. Such coalitions are given a value of $-\infty$ by Equations 12 and 13, and hence, they can never be part of the optimal solution. However, the techniques discussed so far detect such infeasibility only after the execution of Algorithm 3. By contrast, we would identify such coalitions in advance and avoid their formation within SR-CFSS. By doing so, we could reduce the search space, hence improving the performance of our approach. We now show how we achieve this objective.

4.2. Detecting temporally infeasible coalitions

In this section we propose a method that allows us to prune parts of the search space that are guaranteed to always contain *temporally infeasible* coalitions (i.e., coalitions characterised by a set of time constraints that is not satisfiable), which cannot therefore appear in any feasible solution.

One simple approach would be to check, for each solution CS computed during the traversal of the search tree, if CS contains a temporally infeasible coalition, and in such case, discard CS and the corresponding subtree $ST(CS)$. Unfortunately, such a technique can lead to the exclusion of valid solutions. Specifically, given a coalition structure CS that contains a temporally infeasible coalition, $ST(CS)$ (i.e., the portion of the search space rooted at CS) can indeed contain valid solutions, and, hence, $ST(CS)$ cannot be entirely pruned. We provide the following example to better explain this concept. Let $S = \{a_1, a_2\}$ with $a_1 \in D$ (i.e., a_1 is the driver), and let $\theta_1^\sigma = [09{:}00 - 15', 09{:}00 + 15']$ and $\theta_2^\sigma = [09{:}45 - 15', 09{:}45 + 15']$. Assuming that the path that links p_1^σ and p_2^σ (i.e., the starting points of a_1 and a_2 respectively) corresponds to a travel time of 10 minutes, it is not possible to find a departure time τ such that a_1 does not arrive too early at p_2^σ. Thus, θ_2^σ will always be violated, and hence, S is temporally infeasible. Now, assume that, as the result of an edge contraction, $S' = S \cup \{a_3\}$ is formed, with $\theta_3^\sigma = [09{:}20 - 15', 09{:}20 + 15']$. If the paths from

p_1^σ to p_3^σ and from p_3^σ to p_2^σ both require 10 minutes, it is possible to satisfy the time constraints of all the members of S'. Hence, S' is not temporally infeasible.

Nevertheless, under certain conditions it is possible to identify a particular type of temporally infeasible coalitions that will always result in other temporally infeasible coalitions as a result of an edge contraction. Such coalitions can be safely discarded from $\mathcal{FC}(G)$, pruning a significant portion of the search space. Intuitively, if there exists a passenger a_i whose temporal preferences induce a time window *outside* the driver's time window (e.g., a_i latest departure time is earlier that the driver's earliest departure time), any coalition involving these two agents will always be temporally infeasible.

Proposition 3. *Let $a_i, a_j \in A$ with $a_i \in D$ and $a_j \notin D$. If we consider Constraint 2 (i.e., one driver per car) and $[\tau_j^\sigma + \beta_j^\sigma, \tau_j^\omega - \alpha_j^\omega] \not\subseteq [\tau_i^\sigma - \alpha_i^\sigma, \tau_i^\omega + \beta_i^\omega]$, then a_i and a_j can never be in a time feasible coalition together, i.e., $\forall S \in \mathcal{FC}(G) : \{a_i, a_j\} \subseteq S$, S is a temporally infeasible coalition.*

Proof. See Appendix A. □

If we consider a scenario that enforces Constraint 2, then Proposition 3 can be used to identify couples of agents (a_i, a_j) that can never be part of the same coalition, effectively introducing some additional hard constraints on the formation of coalitions. Such constraints can be easily expressed by marking each edge (a_i, a_j) (if existent) as red in the initial graph G, so to avoid the formation of a coalition in which a_i and a_j are together. On the other hand, if a_i and a_j are not connected by an edge in G, we *introduce* a new red edge, since if we do not do so, then a_i and a_j will be part of the same coalition for at least one coalition structure in the search tree. As an example, consider Figure 6.

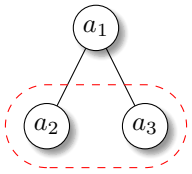

Figure 6: Example of a social network with 3 agents.

Assume that a_2 and a_3's time constraints satisfy Proposition 3. If we do not introduce a new red edge between a_2 and a_3, the grand coalition will be evaluated during the traversal of the search tree, even if such coalition is guaranteed to be temporally infeasible. On the other hand, the introduction of such red edge avoids such inefficiency in our approach. Against this background, we can exploit time constraints to restrict the formation of coalitions. We can also employ the upper bound computation techniques discussed in Section 3.3.1, as we motivate hereafter.

4.3. Bound computation

The upper bound methods proposed in Section 3.3.1 can also be applied when we introduce time constraint, as shown by the following proposition.

Proposition 4. *Propositions 1 and 2 are valid even if we substitute the definition of $v(S)$ in Equation 2 with the definition in Equation 10.*

Proof. See Appendix A. □

The techniques discussed so far (i.e., Sections 3 and 4) constitute our approach to compute the optimal arrangement of cars among a set of agents with given spatial, temporal, and social preferences. Formally, we discussed how we solve the CSG problem associated to SR. In the next section, we tackle the problem of dividing the cost associated to each car among its passengers in a fair and stable way, i.e., we solve the payment computation aspect of the CF problem. We now propose the PK algorithm, which exploits the structural properties of the SR scenario to improve upon the approach by Shehory and Kraus [39].

5. Payments for SR

Payment computation represents a key challenge in the CF process and it is of utmost importance when offering ridesharing services, especially when considering commuters with rational behaviours. One key aspect of payment distribution in CF is the game-theoretic concept of *stability*, which measures how agents are

Algorithm 4 SHEHORYKRAUSKERNEL(x, CS, ϵ)

1: **repeat**
2: **for all** $S \in CS$ **do**
3: **for all** $a_i \in S$ **do**
4: **for all** $a_j \in S - \{a_i\}$ **do**
5: $s_{ij} \leftarrow \max_{\{S' \in 2^A \mid a_i \in S', a_j \notin S'\}} e(S', x)$
6: $\{a_{i^*}$ and a_{j^*} have the maximum surplus difference $\delta\}$
7: $\delta \leftarrow \max_{(a_i, a_j) \in A^2} (s_{ij} - s_{ji})$
8: $(a_{i^*}, a_{j^*}) \leftarrow \arg\max_{(a_i, a_j) \in A^2} (s_{ij} - s_{ji})$
9: **if** $x[j^*] - v(\{a_{j^*}\}) < \delta/2$ **then** {Payments are individually rational}
10: $d \leftarrow x[j^*] - v(\{a_{j^*}\})$
11: **else**
12: $d \leftarrow \delta/2$
13: $x[j^*] \leftarrow x[j^*] - d$ {Transfer payment from a_{j^*}...}
14: $x[i^*] \leftarrow x[i^*] + d$ {... to a_{i^*}}
15: **until** $\delta / V(CS) \leq \epsilon$

keen to maintain the provided payments instead of deviating to a configuration deemed to be more rewarding from their individual point of view. Here, we induce stable payments in the context of the SR problem, employing *the kernel* [14] stability concept. Shehory and Kraus [39] adopt a transfer scheme (Algorithm 4) that represents the state of the art approach to compute kernel-stable payments. Such an algorithm has been designed to compute payments for CF scenarios in which the set of coalitions is *not* restricted by a graph. Such an approach can be readily applied also when the size of coalitions is limited to k members, as it happens in a SR scenario in which all cars have k seats [48].

Definition 18 (k-CF)**.** *A CF problem is said to be a k-CF problem if the size of coalitions is limited to k members.*

In k-CF, the maximisation at line 5 has to be assessed among the coalitions of size up to k which include a_i but exclude a_j. This set, denoted as $\mathcal{R}$, can be easily obtained as $\mathcal{R} = \{\{a_i\} \cup S \mid S \text{ is a } h\text{-combination of } A - \{a_i, a_j\}, \forall h \in \{1, \ldots, k-1\}\}$. Unfortunately, in GCCF scenarios such as SR this simple approach would iterate over several infeasible coalitions (i.e., which do not induce a connected subgraph of the social network), leading to inefficiency and

reducing the scalability of the entire algorithm. In contrast, a better way to tackle this problem is to exploit the structure of the graph in order to consider *only* the coalitions that are indeed feasible. In addition, Algorithm 4 considers many coalitions more than once at the maximisation in the loop at lines 2–5. We provide the following example to clarify why this redundancy exists. Consider the set of agent $A = D = \{a_1, a_2, a_3, a_4\}$ and the graph G shown in Figure 7. Such a graph induces the set of feasible coalitions $\mathcal{FC}(G) = \{\{a_1\}, \{a_2\}, \{a_3\}, \{a_4\}, \{a_1, a_2\}, \{a_1, a_3\}, \{a_1, a_4\}, \{a_1, a_2, a_3\}, \{a_1, a_2, a_4\}, \{a_1, a_3, a_4\}, \{a_1, a_2, a_3, a_4\}\}$, and assumes a coalition structure $CS = \{\{a_1, a_2, a_3, a_4\}\}$.

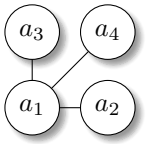

Figure 7: Example of a social network with 4 agents.

The loop requires 12 iterations, each looking at the coalitions reported in Table 1. Note that 23 (marked in bold) out of 33 coalitions (i.e., 70%) are evaluated more than once. This process substantially reduces the efficiency and the scalability of the algorithm in SR scenarios, where the computation cost required to assess coalitional values is not negligible and caching is not an option. In fact, storing all these values in memory is not affordable even for systems with hundreds of agents: since $\mathcal{FC}(G)$ can contain up to $O\left(n^k\right)$ coalitions, for $k = 5$ and $n = 100$, storing all coalitional values requires tens of GB of memory. Thus, each coalitional value must be computed only when needed, since computing them more than once reduces efficiency and scalability, as shown in Section 7.2.

To overcome these issues, in the next section we present the PK algorithm, our payment scheme that scales up to systems with thousands of agents.

5.1. The PK algorithm

We now present the PK (Payments in the Kernel) algorithm, our method to compute an ϵ-kernel payoff allocation given a coalition structure, and we apply it to the SR scenario. Our contribution improves on the k-CF version of

a_i	a_j	Coalitions
a_1	a_2	$\{a_1\}$ $\{a_1, a_3\}$ $\{a_1, a_4\}$ $\{a_1, a_3, a_4\}$
a_1	a_3	$\boldsymbol{\{a_1\}}$ $\{a_1, a_2\}$ $\boldsymbol{\{a_1, a_4\}}$ $\{a_1, a_2, a_4\}$
a_1	a_4	$\boldsymbol{\{a_1\}}$ $\boldsymbol{\{a_1, a_2\}}$ $\boldsymbol{\{a_1, a_3\}}$ $\{a_1, a_2, a_3\}$
a_2	a_1	$\{a_2\}$
a_2	a_3	$\boldsymbol{\{a_2\}}$ $\boldsymbol{\{a_1, a_2\}}$ $\boldsymbol{\{a_1, a_2, a_4\}}$
a_2	a_4	$\boldsymbol{\{a_2\}}$ $\boldsymbol{\{a_1, a_2\}}$ $\boldsymbol{\{a_1, a_2, a_3\}}$
a_3	a_1	$\{a_3\}$
a_3	a_2	$\boldsymbol{\{a_3\}}$ $\boldsymbol{\{a_1, a_3\}}$ $\boldsymbol{\{a_1, a_3, a_4\}}$
a_3	a_4	$\boldsymbol{\{a_3\}}$ $\boldsymbol{\{a_1, a_3\}}$ $\boldsymbol{\{a_1, a_2, a_3\}}$
a_4	a_1	$\{a_4\}$
a_4	a_2	$\boldsymbol{\{a_4\}}$ $\boldsymbol{\{a_1, a_4\}}$ $\boldsymbol{\{a_1, a_3, a_4\}}$
a_4	a_3	$\boldsymbol{\{a_4\}}$ $\boldsymbol{\{a_1, a_4\}}$ $\boldsymbol{\{a_1, a_2, a_4\}}$

Table 1: Coalitions computed by the loop at lines 2–5 of Algorithm 4.

Algorithm 4 by adopting a novel approach to calculate the surplus matrix s. Instead of computing each value s_{ij} using the maximisation at line 5 for each pair of agents in each $S \in CS$, we iterate over the set of feasible coalitions (as specified in Definition 5) induced by G, and we update the appropriate values of the surplus matrix for each of such coalitions. Specifically, this is achieved by iterating over the set of $\hat{k}$-subgraphs of G, i.e., the set of connected subgraphs of G with *at most* k nodes, and then executing the update by means of the UpdateMax routine only for those $\hat{k}$-subgraphs that actually correspond to feasible coalitions. This additional check is mandatory since not all $\hat{k}$-subgraphs necessarily satisfy Constraint 1, and hence, represent feasible coalitions. By so doing, we ensure the exact coverage of $\mathcal{FC}(G)$, as proved by Proposition 6.

PK is detailed in Algorithm 5. After having initialised the payoff vector x by equally splitting each coalitional value among the members of the coalition, ComputeMatrix computes the surplus matrix in each iteration of the main loop. In such a routine, UpdateMax is executed for each coalition that induces a $\hat{k}$-subgraph of G. These coalitions are computed with the EnumerateCsg algorithm proposed by Moerkotte and Neumann [35], which can list all the subgraphs of a given graph without redundancy (i.e., each subgraph is computed only once). Then, UpdateMax only considers the coalitions that satisfy Constraint 1 of the SR problem (line 1). For every S of such coalitions, lines 3–8

Algorithm 5 PK(CS, ϵ)

for all $S \in CS$ **do**
  for all $a_i \in S$ **do**
    $x_i \leftarrow {}^{v(S)}/_{|S|}$ {Equally split coalitional value}
repeat
  {Compute surplus matrix}
  $s \leftarrow$ ComputeMatrix (CS, x)
  {a_{i^*} and a_{j^*} have the maximum surplus difference δ}
  $\delta \leftarrow \max_{(a_i,a_j)\in A^2} (s_{ij} - s_{ji})$
  $(a_{i^*}, a_{j^*}) \leftarrow \arg\max_{(a_i,a_j)\in A^2} (s_{ij} - s_{ji})$
  {Ensure that payments are individually rational}
  if $x[j^*] - v(\{a_{j^*}\}) < {}^{\delta}/_{2}$ **then**
    $d \leftarrow x[j^*] - v(\{a_{j^*}\})$
  else
    $d \leftarrow {}^{\delta}/_{2}$
  $x[j^*] \leftarrow x[j^*] - d$ {Transfer payment from a_{j^*}...}
  $x[i^*] \leftarrow x[i^*] + d$ {... to a_{i^*}}
until ${}^{\delta}/_{v(CS)} \leq \epsilon$

update all the values s_{ij} for which a_i is a member of S and a_j is part of S' (i.e., the coalition in CS that contains a_i) but is not part of S. The correctness of our approach is ensured by Proposition 5.

Proposition 5. *Algorithm 6 computes each s_{ij} correctly.*

Proof. See Appendix A. □

Our surplus matrix-calculating method has polynomial time complexity, while computing all feasible coalitions only once, as shown by Proposition 6.

Proposition 6. *Algorithm 6 lists all feasible coalitions only once and it has a worst-case time complexity of $O\left(n^k\right)$.*

Proof. See Appendix A. □

Algorithm 6 ComputeMatrix(CS, x)

$s \leftarrow -\infty$ {Initialise the entire matrix with $-\infty$}
for all S that induce a $\hat{k}$-subgraph of G **do**
  $s \leftarrow$ UpdateMax (S, CS, s, x)
return s

Algorithm 7 UPDATEMAX(S, CS, s, x)

if S satisfies Constraint 1 **then**
  $e_S \leftarrow e(S, x)$ {Compute the excess of coalition S}
  for all $a_i \in S$ **do** {For each agent a_i in coalition S}
    $S' \leftarrow$ the coalition in CS that contains a_i
    for all $a_j \in S' - S$ **do** {For each $a_j \in S'$ but $\notin S$}
      {s_{ij} is updated with the maximum between}
      {its old value and the excess of coalition S}
      $s_{ij} \leftarrow \max(s_{ij}, e_S)$
return s

In the next proposition, we prove that PK has a polynomial time complexity.

Proposition 7. *Algorithm 5 has a polynomial worst-case time complexity with respect to n, i.e., $O\left(-\log_2(\epsilon) \cdot n^{k+1}\right)$.*

Proof. See Appendix A. □

PK provides a polynomial method to compute kernel-stable payments. Nonetheless, the $O\left(n^k\right)$ operations required for surplus matrix calculation may not be affordable in real-world scenarios with thousands of agents and $k = 5$ (i.e., the number of seats of an average sized car). Hence, we next propose a parallel version of PK, which allows us to distribute the computational burden among different threads, taking advantage of modern multi-core hardware.

5.2. P-PK

We now detail P-PK, the parallel version of our approach, in which the most computation-intensive task, i.e., the computation of the matrix s, is distributed among T available threads. In particular, Algorithm 8 details our parallel version of the COMPUTEMATRIX routine, obtained by having each thread t to compute a separate matrix s^t. Such a matrix is constructed considering the coalitions in $\mathcal{DIV}(G, t, k)$, i.e., the t^{th} fraction of the set of all $\hat{k}$-subgraphs of G, computed using the D-SLYCE algorithm [45].[17] Specifically, this fraction is obtained by

[17] Notice that nor ENUMERATECSG neither D-SLYCE solve the payment computation problem, as they address the enumeration of the $\hat{k}$-subgraphs of G.

splitting the first generation of children nodes in the search tree generated by the EnumerateCsg algorithm among the available threads, allowing a fair division of the set of the $\hat{k}$-subgraphs while ensuring that all feasible coalitions are computed exactly once. Thus, it also distributes the computation of the coalitional values.

Algorithm 8 P-ComputeMatrix(CS, x, T)

1: $s \leftarrow -\infty$ {Initialise all matrix elements with $-\infty$}
2: **for all** $t \in \{1, \ldots, T\}$ **do in parallel**
3: **for all** $S \in \mathcal{DIV}(G, t, k)$ **do**
4: $s^t \leftarrow \textsc{UpdateMax}(S, CS, s^t, x)$
5: **for all** $i \in \{1, \ldots, n\}$ **do in parallel**
6: **for all** $j \in \{1, \ldots, n\}$ **do in parallel**
7: $s_{ij} \leftarrow \max_{t \in \{1,\ldots,T\}} s^t_{ij}$
8: **return** s

We provide the following example to clarify how this division is realised. Consider the same $\mathcal{FC}(G)$ of the example in Section 5, and assume $T = 4$. Then, the necessary coalitions are distributed by doing the following partitioning:

1. $\mathcal{DIV}(G, 1, k) = \{\{a_1\}, \{a_2\}, \{a_3\}\}$
2. $\mathcal{DIV}(G, 2, k) = \{\{a_4\}, \{a_1, a_2\}, \{a_1, a_3\}\}$
3. $\mathcal{DIV}(G, 3, k) = \{\{a_1, a_4\}, \{a_1, a_2, a_3\}\}$
4. $\mathcal{DIV}(G, 4, k) = \{\{a_1, a_2, a_4\}, \{a_1, a_3, a_4\}\}$

Note that, since each matrix s^t is modified only by thread t, Algorithm 8 contains only one synchronisation point (i.e., before line 5), hence providing a full parallelisation. After that, the final surplus matrix s is computed with a maximisation on all the above matrices (lines 5–7), ensuring that the output of P-ComputeMatrix is equal to the one of ComputeMatrix, since each feasible coalition in $\mathcal{FC}(G)$ has been computed by a thread. Notice that P-PK requires storing t separate surplus matrices, one per thread. Hence, its memory requirements are $O\left(t \cdot n^2\right)$, i.e., still polynomial in the number of agents.

Having discussed our CF approach for the SR scenario, we now present our experimental evaluation. In particular, in the next section we benchmark SR-

CFSS, our algorithm that solves the GCCF aspect of SR. Then, in Section 7 we empirically evaluate PK, our payment allocation algorithm.

6. Evaluating the SR-CFSS algorithm

The main goals of the empirical analysis are i) to estimate the social welfare improvement when our SR model is used, ii) to evaluate the performance of the optimal version of SR-CFSS in terms of runtime and scalability, iii) to evaluate the approximate performance and guarantees that SR-CFSS can provide on a large number of agents, i.e., up to 2000 agents, and iv) to investigate the impact of time constraints on the above properties.

Since there are no publicly available datasets which include *both* spatial and social data for the same users, in our empirical evaluation we consider two separate real-world datasets and we superimpose the first on the second one. In particular, our map $M = (P, Q)$ is a realistic representation of the city of Beijing (Figure 8), with $|P| = 8330$ points and $|Q| = 13290$ edges, equivalent to an average resolution of a point every $\sim$10 meters. This map has been derived from the GeoLife dataset [34, 50] provided by Microsoft Research, which comprises 17621 trajectories with a total distance of about 1.2 million km, recorded by different GPS loggers with a variety of sampling rates. These trajectories are adopted to sample random paths used to provide starting points and destinations. Moreover, such a dataset also includes the timestamp of each trajectory, allowing us to create a distribution of the departure and arrival times (Figure 9), which is used to sample such parameters for each agent in all our experiments, unless otherwise stated (i.e., in all experiments considering time constraints except Section 6.2). As expected, this distribution exhibit two peaks, one in the morning from 7:00 to 9:00 and one in the evening from 17:00 to 19:00.

Figure 8: The map of Beijing derived from the GeoLife dataset.

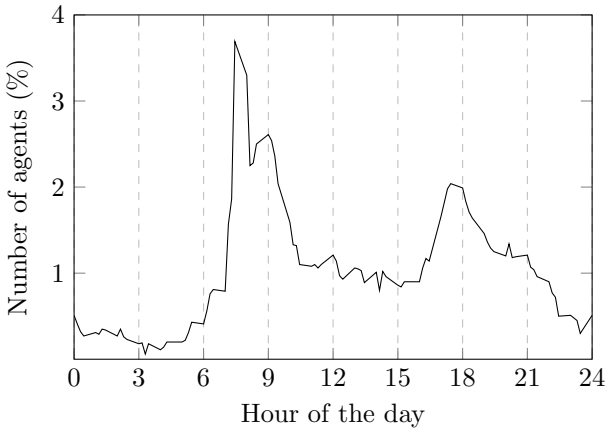

Figure 9: Default distribution of departure/arrival times (obtained from GeoLife).

In each experiment, the graph G is a subgraph of a large crawl of the Twitter social graph. Specifically, such dataset is a graph with 41.6 million nodes and 1.4 billion edges published as part of the work by Kwak et al. [30]. In particular, G is obtained by means of a standard algorithm [37] to extract a subgraph from a larger graph, i.e., a breadth-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 empirical evaluation there is no mapping between the trajectory data and the social graph, since they belong to independent projects. In all our experiments, the default number of agents n is

50. We adopt a cost model that considers fuel expenses, i.e., $v(C) = K_{fuel} \cdot \overline{L_C^*}$, where $\overline{L_C^*}$ represents the length of L_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 $k(\{a_i\}) = -3$ € $\forall a_i \in A$, 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., $seats(a_i) = 5\,\forall a_i \in D$. When time constraints are considered, we define $\gamma = -2$ €/h and a time window (i.e., the duration of θ_i) of 30', unless otherwise stated. All our tests account for Constraint 2 (drivers always drive their cars). Hence, since both bounding techniques detailed in Section 3.3.1 are valid and, in general, one does not dominate on the other, we take the minimum one at each step of the algorithm, providing a more effective pruning. Each test is repeated on 20 random instances, and we report the average and the standard error of the mean. SR-CFSS is implemented in C[18] and executed on a machine with a 3.40GHz CPU and 16 GB of memory.

6.1. Social welfare improvement without time constraints

In our first experiment we consider the improvement of the social welfare (i.e., the cost reduction for the overall system) when using our SR model without time constraints, 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. Formally, we define the social welfare improvement as $100 \cdot \left| \frac{V(CS^*) - V(A_{single})}{V(A_{single})} \right|$. Such an improvement is influenced by the percentage of drivers in the system (Figure 10), which determines the number of available seats and the number of riders that can share a ride without having to resort to public transport. Moreover, with more drivers it 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

[18] Our implementation is available at `https://github.com/filippobistaffa/SR-CFSS`.

only the 10% of the total riders own a car, the average cost reduction 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 infeasible coalitions. As Figure 10 shows, our method allows superior cost reductions with respect to such a greedy approach, which can provide a maximum improvement of -19.55% for $|D| = 20\%$. Notice that, when the majority of the riders owns a car, the greedy approach cannot improve upon the value of the baseline (i.e., no ridesharing).

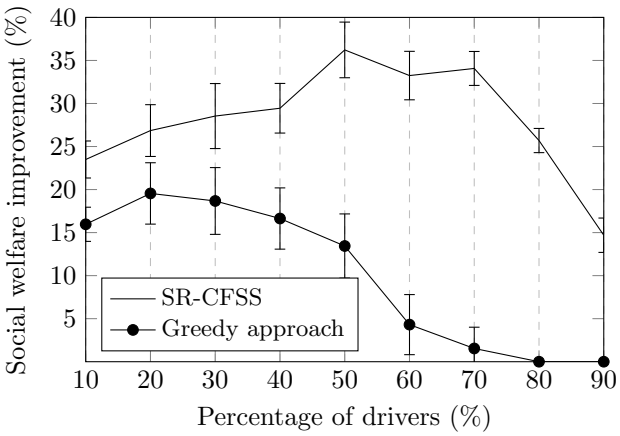

Figure 10: Social welfare improvement.

6.2. Social welfare improvement with time constraints

We now investigate how the social welfare improvement varies when we introduce time constraints. Specifically, we now study the influence of the duration of θ_i (i.e., the width of the time window) and the distribution of the agents' departure times on the social welfare. To this end, we vary these two parameters as follows. On the one hand, we sample the departure times of the agents within a time window of 6 hours according to 3 probability distributions (Figure 11). Specifically, we consider a uniform distribution (i.e., the departure times are

distributed uniformly in the time window) and two Gaussian distributions, in which the agents who desire to leave in the two central hours of the time window are the 30% (soft peak) and the 40% (hard peak) of the total respectively. On the other hand, we vary the width of the time window θ_i for each agent. For simplicity, we assume that $\alpha_i^\sigma = \beta_i^\sigma = \alpha_i^\omega = \beta_i^\omega$ are all equal for all agents, and we vary such value, namely θ_i's radius, within $[5', 60']$. Following the result of the experiments in the previous section, we only consider $D = 50\%$, i.e., the scenario that results in the highest social welfare improvement. Figure 12 shows that, in general, the social welfare improvement increases when we increase the θ_i's radius. In fact, with larger time windows it is easier to satisfy time constraints and, hence, to form coalitions to reduce the overall travel cost. Notice that such an improvement saturates when the radius exceeds 30 minutes, since larger θ_i's radii are associated to larger costs by the θ_C component, which contributes to reduce the social welfare improvement. In addition, Figure 12 also shows that the hard peak distribution provides the highest social welfare improvement (8.79%) with respect to the soft peak (6.62%) and the uniform (3.62%) ones. In fact, if the departure times of more agents are concentrated in a shorter time period, the cost provided by the θ_C component is lower. Moreover, SR-CFSS can evaluate a larger number of feasible solutions, since less temporally infeasible coalition structures have to be discarded. Finally, notice that, since time constraints result in additional costs and, more important, a reduced solution space, they cause a reduction of the social welfare improvement, as confirmed by the results in Section 6.4.

We further investigate the behaviour of the social welfare improvement by increasing the θ_i's radius only of a particular class of commuters, in order to identify which classes are more sensitive to the variation of such parameter in terms of overall cost reduction. Specifically, we observe 3 interesting classes of agents, i.e., drivers, riders, and hubs (i.e., agents whose connectivity in the social graph is significantly above the average), and we vary the θ_i's radius within $[15', 60']$ only for the considered class, while setting such parameter equal to $15'$ for the other classes.

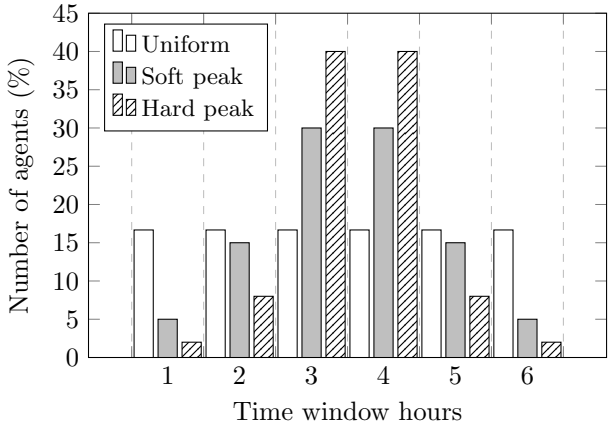

Figure 11: Probability distributions in a time window of 6 hours.

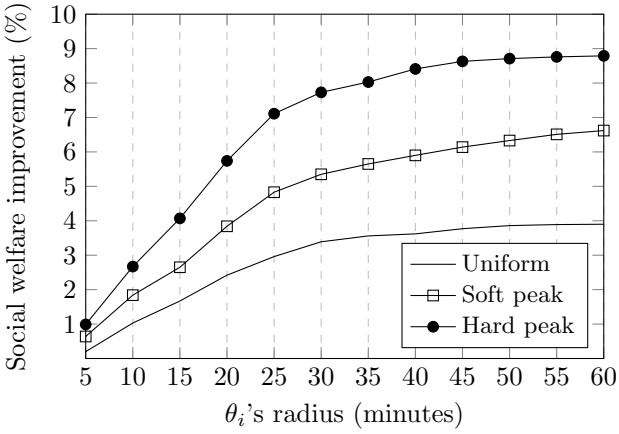

Figure 12: Social welfare improvement with respect to θ_i's radius.

Figure 13 shows that the social welfare improvement has the biggest increase for the drivers (+6.28%), reaching a final maximum of 14.24%. Such increase is slightly lower for hubs (+5.1%), while it is only +1.27% for riders. These results prove the impact of a larger θ_i's radius for drivers and hubs, which results in a larger number of potential coalitions, and, hence, a larger social welfare improvement.

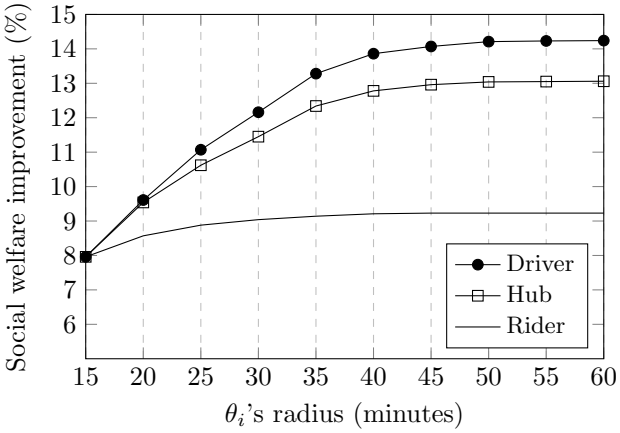

Figure 13: Social welfare improvement with respect to θ_i's radius.

6.3. Runtime performance without time constraints

In this section we discuss the performance of our approach in terms of runtime needed to compute the optimal solution of a SR problem without time constraints. Figure 14 shows the runtime with respect to the number of agents adopting our SR model without time constraints. 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 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 $|D| = 50\%$. 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 3.3.1) that 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

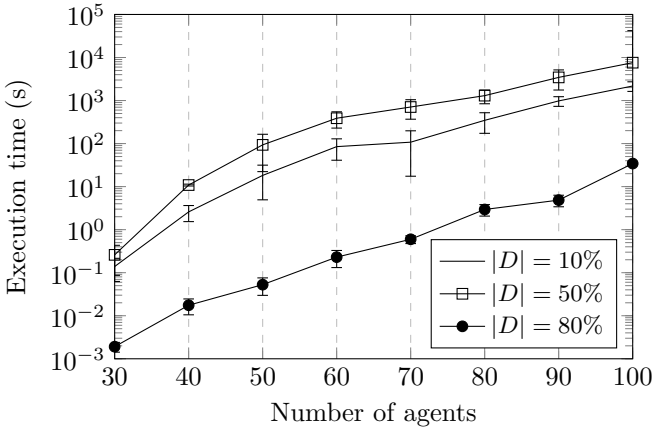

Figure 14: Runtime without time constraints.

instances with $n = 60$ and $|D| = 50\%$.

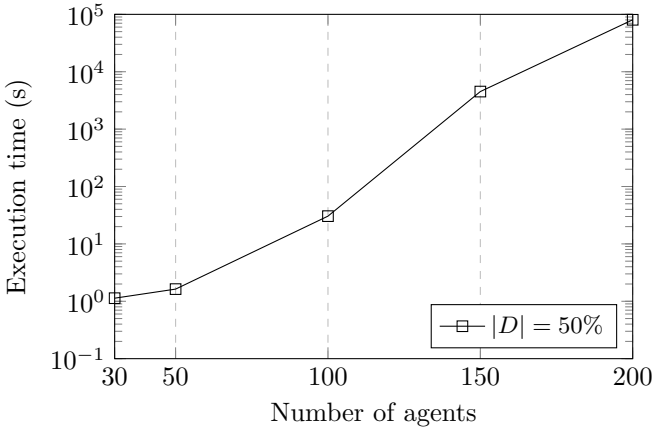

Figure 15: Runtime with time constraints.

6.4. Runtime performance with time constraints

When we consider time constraints (Figure 15), we notice a significant performance improvement of SR-CFSS, which can compute the optimal solution for 100 agents in 30 seconds, i.e., over two orders of magnitude faster than the above case. This improved performance also results in an increased scalability, as SR-CFSS can solve systems with 150 agents, i.e., 50 additional agents with respect

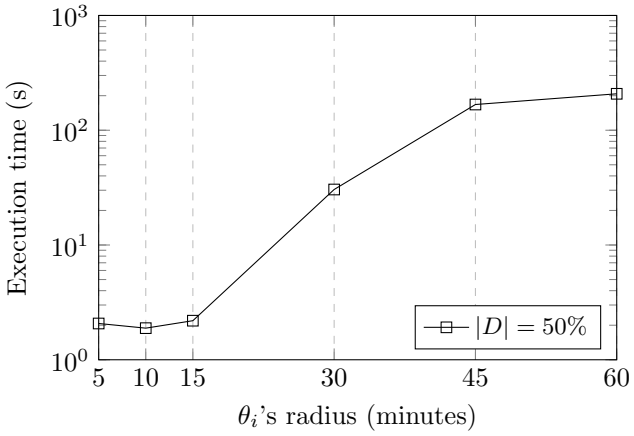

Figure 16: Runtime with respect to θ_i's radius.

to 100 agents in the previous experiment, in the same amount of time, and 200 in less than a day. We further investigate the impact of time constraints on the performance of SR-CFSS by varying the θ_i's radius, as discussed in Section 6.2. Figure 16 shows that larger radii correspond to harder SR problems. As an example, instances with a θ_i's radius equal to 15 minutes are solved by SR-CFSS more than two orders of magnitude faster with respect to when we consider 45 minutes. As discussed in Section 6.2, larger radii correspond to a larger number of feasible solutions, since fewer temporally infeasible coalition structures have to be discarded. These results confirm the impact of time constraints on the dimension of the solution space, which results in two main outcomes. On the one hand, scenarios with time constraints are easier to solve, since the number of solutions is lower. On the other hand, the reduced number of solutions allows a lower social welfare improvement in such scenarios (see Section 6.2).

6.5. Approximate performance

In this section we evaluate the quality of the solutions computed by the approximate version of SR-CFSS on a very large set of agents (i.e., 2000). A standard measure to evaluate the quality of the solutions of approximate algorithms is the *Performance Ratio* (PR) [3].

Definition 19 (PR)**.** *Given an instance I of an optimisation problem, its optimal solution $Optim(I)$ and an approximate solution $Approx(I)$, the* performance ratio $PR(I) = \max\left(\frac{Approx(I)}{Optim(I)}, \frac{Optim(I)}{Approx(I)}\right)$.

Both in the case of minimisation and maximisation problems, the PR is equal to 1 in the case of an optimal solution, and can assume arbitrarily large values in the case of poor approximate solutions. In our case, computing the optimal solution $Optim(I)$ for large-scale GCCF problems is not possible, hence the PR is not an applicable measure of quality. Thus, we define the *Maximum Performance Ratio* (MPR) following the above definition, and considering the upper bound on the optimal solution provided by Propositions 1 and 2.

Definition 20 (MPR)**.** *Given a GCCF instance I, we denote the approximate solution computed by CFSS as $Approx(I)$ and the upper bound on the optimal solution as $Bound(I)$. Then, we define the Maximum Performance Ratio $MPR(I) = \max\left(\frac{Approx(I)}{Bound(I)}, \frac{Bound(I)}{Approx(I)}\right)$.*

Since $|Bound(I)| \leq |V(CS_I^*)|$, where CS_I^* is the optimal solution of I, $MPR(I)$ represents an upper bound for $PR(I)$. The MPR provides an important quality guarantee for the approximate solution $Approx(I)$, since $Approx(I)$ always lies within a factor of $MPR(I)$ with respect to the optimal solution.[19] We run SR-CFSS on instances adopting the model without time constraints with $n \in \{500, 1000, 2000\}$ and we stop the execution after a time budget of 100 seconds. Then, we compute $Bound(I)$ as defined in Propositions 1 and 2. Specifically, since both propositions are applicable, we compute the upper bound using both techniques and then we consider the lowest one. Figure 17 shows that, on average, $Bound(I)$ is only 6.65% higher than $Approx(I)$ (i.e., the solution found within the time limit) for $n = 500$ and $|D| = 80\%$, reaching a maximum of +29.92% when $n = 2000$ and $|D| = 50\%$. In the worst case, SR-CFSS provides a maximum performance ratio of 1.41 and thus solutions whose values are at

[19] PR and MPR provide a *per-instance* quality guarantee, in contrast with the *approximation ratio* [12] that provide a quality guarantee valid for all possible instances.

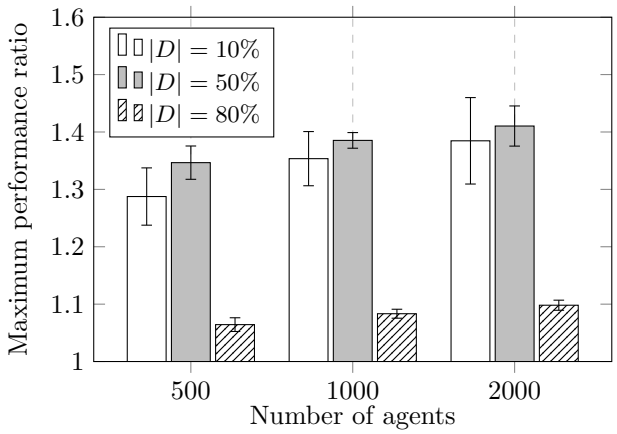

Figure 17: Maximum performance ratio of approximate solutions.

least 71% of the optimal. We obtained very similar results also when we consider time constraints, and hence, we do not report them here. Such a behaviour is reasonable since the maximum performance ratio is heavily influenced by the value of $Bound(I)$ and, as detailed in Section 4.3, we apply the same technique whether or not we consider time constraints.

6.6. SR-CFSS vs. C-Link: solution quality comparison

In our final experiment we further evaluate the approximate performance of SR-CFSS by comparing it against C-Link [21], one of the most recent CSG heuristic approaches. Specifically, we generate random SR instances with $n \in \{1000, 1200, \ldots, 2000\}$, considering 20 repetitions for each n. Then, we solve each instance with C-Link (adopting the best heuristic proposed by Farinelli et al. [21], i.e., Gain-Link) and then we run SR-CFSS on the same instance with a time budget equal to C-Link's runtime.

Figure 18 shows the average and the standard error of the mean of the ratio between the value of the solution computed by C-Link and the one computed by SR-CFSS. Since we consider solutions with negative values, when such ratio is > 1 the solution computed by C-Link is better (i.e., corresponds to a lower cost) than the one computed by SR-CFSS. Our results show that, for $n < 1600$, the quality of C-Link's solutions is better than SR-CFSS. Then, for $n \geq 1600$

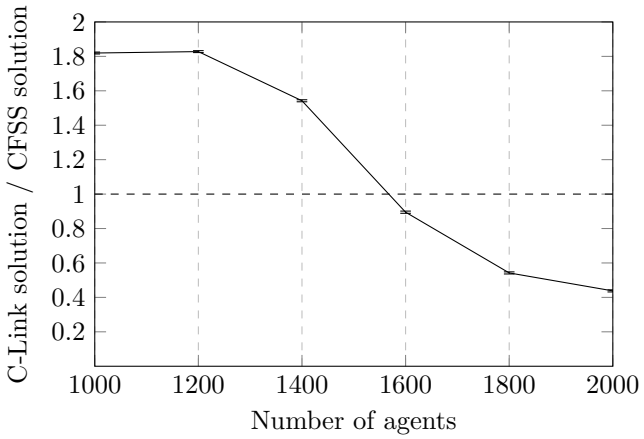

Figure 18: Ratio between C-Link and SR-CFSS approximate solutions.

SR-CFSS outperforms C-Link in terms of solution quality. In particular, for $n = 2000$ the solutions provided by our approach correspond to costs that are, on average, 2.28 times lower than the counterpart ones.

6.7. Summary of results

Our empirical evaluation demonstrates that our approach results in a significant cost reduction when applied to SR scenarios, reaching -36.22% when half of the agents in the system owns a car. Such a scenario, resulting in the best cost improvement, is characterised by the largest search space, i.e., it is the most computationally intensive to solve with respect to other percentages of drivers. As expected, the introduction of time constraints reduces the cost improvement for the system, since it limits the number of possible solutions, reducing the size of the search space. As a consequence, computing SR solutions with time constraints is less computationally demanding. Crucially, results show that SR-CFSS is a viable method for the computation of SR solutions, especially in large-scale scenarios (i.e., with 2000 agents), where our approach provides good approximate solutions whose quality is at least 71% of the optimal.

After the empirical evaluation of SR-CFSS, we now benchmark PK.

7. Evaluating the PK algorithm

In this section, we focus on the evaluation of our approach to the computation of kernel-stable payments for SR. The main goals of the empirical analysis are i) to test the performance of PK when computing payments for systems of thousands of agents, ii) to perform an analysis of the features that influence the distribution of payments among the agents, iii) to investigate the impact of time constraints on the above properties, iii) to compare the efficiency of PK with respect to the state of the art approach proposed by Shehory and Kraus, and iv) to estimate the speed-up obtainable by using P-PK with respect to PK.

In all our tests, we adopt the same methodology and datasets discussed in Section 6 (i.e., we adopt the GeoLife and Twitter datasets), unless otherwise stated. In the experiments looking at the performance of PK (i.e., Sections 7.1, 7.2 and 7.3) we only consider the SR model without time constraints, since the performance of PK is negligibly affected by them.[20] PK is implemented in C[21] and executed on a machine with a 3.40GHz CPU and 16 GB of memory.

7.1. Runtime performance

In our first experiment, we evaluate the performance of our approach when computing payments in large-scale instances. Figure 19 shows the runtime needed to execute P-PK on systems with $n \in \{100, 500, 1000, 1500, 2000\}$. In each test, the coalition structure has been computed using the approximate version of SR-CFSS using our SR model without time constraints.

Our results show that P-PK is able to compute payments for 2000 agents with a runtime ranging from 13 to 50 minutes, hence it can successfully scale to large systems. In particular, for each value of n, we consider $|D| \in \{10\%, 50\%, 80\%\}$. Our results also show the influence of the percentage of drivers on the complexity of the problem. On average, computing payments on an instance with $|D| = 80\%$ is easier with respect to $|D| = 10\%$ and $|D| = 50\%$. Our findings are consistent

[20] The complexity of computing each coalitional value is comparable whether or not we consider time constraints.

[21] Our implementation is available at `https://github.com/filippobistaffa/PK`.

with the results in Section 6.1, showing that the scenario with $|D| = 50\%$ is more difficult to solve since more drivers are available, hence it is possible to form more cars, resulting in a larger search space. In fact, the number of feasible coalitions is determined by 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).

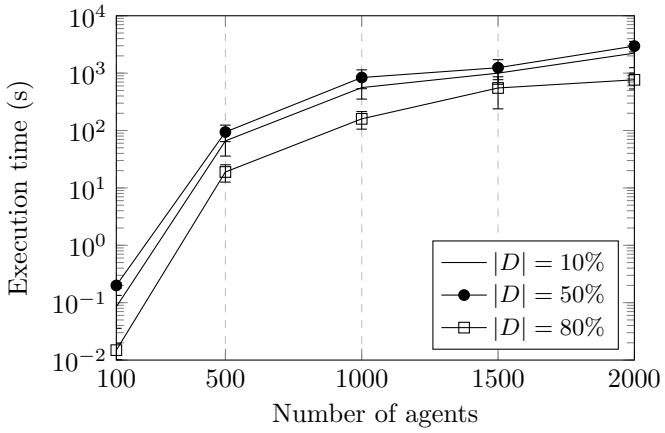

Figure 19: Runtime needed to compute payments.

7.2. Benchmarking PK

Figure 20 shows the runtime needed by our approach to compute a kernel-stable payoff vector, comparing it with the state of the art approach by Shehory and Kraus [39], i.e., Algorithm 4. In particular, we consider the runtime needed to solve SR instances with $n \in \{30, 40, 50, 60, 70, 80, 90, 100\}$ and $|D| = 50\%$. We employ the sequential version of PK, since Algorithm 4 is also sequential.

Our results show that PK is at least one order of magnitude faster than the counterpart approach, outperforming the state of the art by 27 times in the worst case, with an average improvement of 53 times, and a best case improvement of 84 times. Thus, our comparison has been run only up to $n = 100$, since the counterpart approach becomes impractical for instances with thousands of agents. In fact, with 1000 agents it requires over one day of computation, compared to a runtime of 2 hours required by PK, and 14 minutes required by P-PK. In

particular, the approach in [39] is slower due to several redundant computations of many coalitional values, with a significant impact on the runtime.

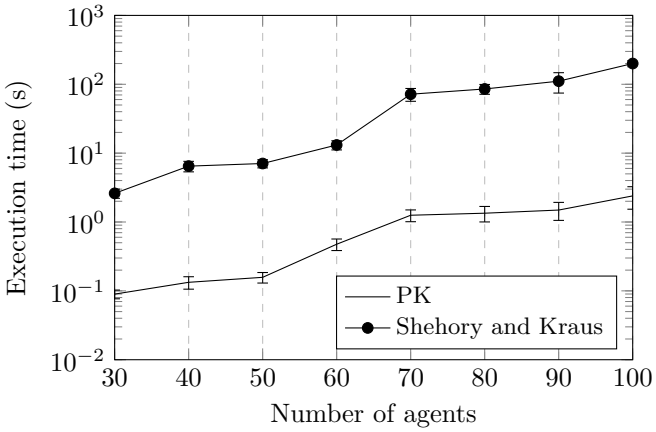

Figure 20: Runtime needed to compute payments.

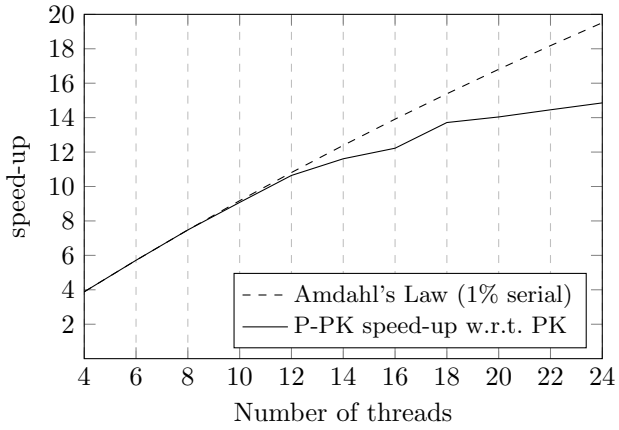

Figure 21: Multi-threading speed-up.

7.3. Parallel performance

Here we analyse the speed-up that can be achieved by using P-PK with respect to PK, i.e., its sequential version. We ran the algorithms on instances with 500 agents and $|D| = 50\%$, using a machine with 2 Intel® Xeon® E5-2420.

The speed-up measured during these tests has been compared with the maximum theoretical one provided by the Amdahl's Law [2], considering an estimated

non-parallelisable part of 1%, due to memory allocation and thread initialisation. Figure 21 shows that the actual speed-up follows the theoretical one for up to 12 threads (i.e., the number of physical cores for this machine), reaching a final speed-up of 14.85× with all 24 threads active.

7.4. Costs and network centrality

PK computes a cost allocation that is guaranteed to be kernel-stable. A priori, such an allocation does not have any particular property linked with structure of the problem. The purpose of this section is to analyse the relationship between the cost incurred by a commuter and the properties that determine its importance in the environment, i.e., being a node with a high degree in the social network, or being driver or rider. To this end, we first compute the optimal solution of a SR problem without time constraints on random instances with $n \in \{30, 40, 50, 60, 70, 80, 90, 100\}$ and $|D| \in \{10\%, 50\%, 80\%\}$, and we use our algorithm to compute a kernel-stable payoff vector. Then, to assess this correlation in a quantified manner, we define the *normalised cost* $\overline{c}_i$ and the *normalised degree* $\overline{d}_i$ for each agent a_i as follows:

- For any a_i in a coalition S with $|S| > 1$, we define its *normalised cost* $\overline{c}_i$ as
$$\overline{c}_i = \frac{-x[i] - \min_x^S}{\max_x^S - \min_x^S},$$
where $\min_x^S$ and $\max_x^S$ are the minimum and the maximum values of the negative values of x among the members of S, i.e., $\min_x^S = \min_{a_i \in S} -x[i]$ and $\max_x^S = \max_{a_i \in S} -x[i]$. Note that we consider negative values since in our model, costs are represented by negative values for $x[i]$.

- For any a_i in a coalition S with $|S| > 1$, we define its *normalised degree* $\overline{d}_i$ as
$$\overline{d}_i = \frac{deg\,(a_i) - \min_d^S}{\max_d^S - \min_d^S},$$
where $deg\,(a_i)$ is the degree of a_i in the social network, and $\min_d^S$ and $\max_d^S$ are the minimum and the maximum degrees of the members of S.

When the denominator of $\overline{c}_i$ is 0, i.e, when $\max_x^S = \min_x^S$, it means that all the agents in C have the same payoff. In these cases, $\overline{c}_i$ is defined to be 0.5 as a middle point between 0 and 1 (the same discussion applies to $\overline{d}_i$).

Notice that, a direct comparison of two agents that are not part of the same coalition would not be appropriate for determining their overall power or benefits derived from participation in the SR setting, since payments computed according to the kernel do not consider agents belonging to different coalitions. Nonetheless, it would definitely be interesting to have a way to measure and compare the power of the agents, regardless of the coalition to which each one belongs. To allow this comparison, both $\overline{c}_i$ and $\overline{d}_i$ are normalised between 0 (for the agents having the minimum costs/degrees in their coalitions) and 1 (similarly for the agents with maximum costs/degrees). The normalisation is done with respect to the coalition the agent belongs to, because to reach kernel-stability, payment transfers only take place among agents within the same coalition. As an example, if an agent's normalised cost is 0.4, it means that its incurred cost is a value placed at the 40% of the range between the minimum and the maximum costs incurred by the agents in its coalition. Finally, note that agents in singletons have been excluded, as they do not have to split their value.

In Figure 22 we report the average and the standard error of the mean for the normalised cost with respect to the normalised degree. Our results clearly show that costs are strongly influenced by the degree of the agents, and whether they are drivers or riders. Specifically, in our tests drivers had to pay costs that were on average 16% lower than riders. Moreover, agents with the minimum number of social connections in their coalition (i.e., with a normalised degree of 0) paid a cost 171% higher than the ones with the highest degree.

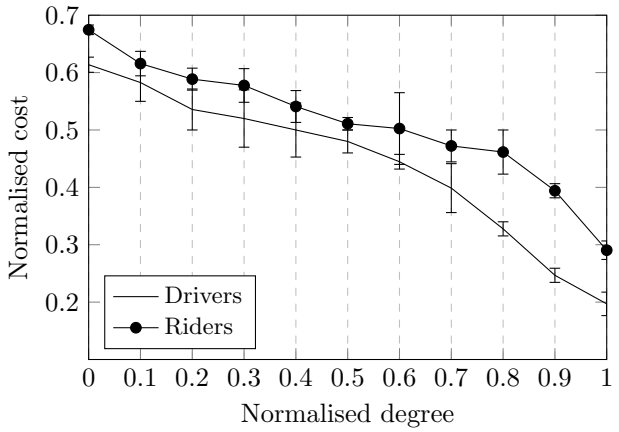

Figure 22: Normalised cost w.r.t. normalised degree without time constraints.

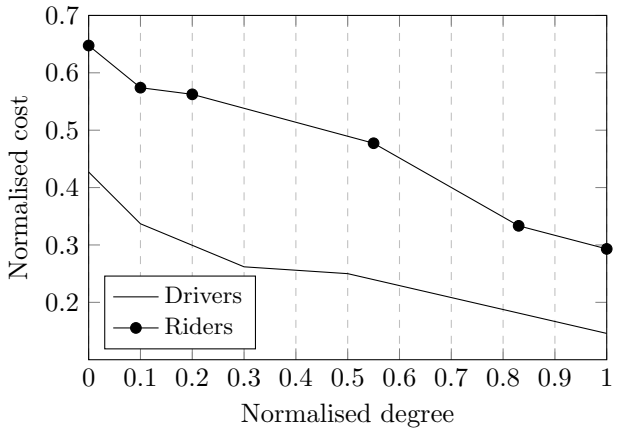

Figure 23: Normalised cost with respect to normalised degree with time constraints.

We now investigate how the features of the cost distributions studied above are affected by the introduction of time constraints. Figure 23 shows a behaviour similar to the one discussed in the above section. Moreover, we notice that the introduction of time constraints results in significantly lower costs for the drivers (i.e., drivers pay costs that are on average 35% lower than the previous experiment), while riders' costs are comparable in both scenarios. These results can be explained by recalling that time constraints significantly reduce the solution space (see Sections 6.2 and 6.4), and hence, the influence of drivers (who are

crucial to determine whether or not a coalition can be formed) is stronger if the pool of possible alternative coalitions is smaller.

We further investigate the role of time constraints in the payment distribution process by studying to what extent more tolerant agents are rewarded with lower costs. To this end, we assign a random θ_i's radius within $\{5', 10', 15', 20', 25', 30'\}$ to each agent and we look at the corresponding normalised cost. Figure 24 shows that the agents that are willing to tolerate more with respect to their ideal departure/leaving time are rewarded by the system with lower costs, as a consequence of the fact that, by having a larger θ_i's radius, they can choose among a larger pool of alternatives and hence, they achieve a higher bargaining power in the payment distribution process.

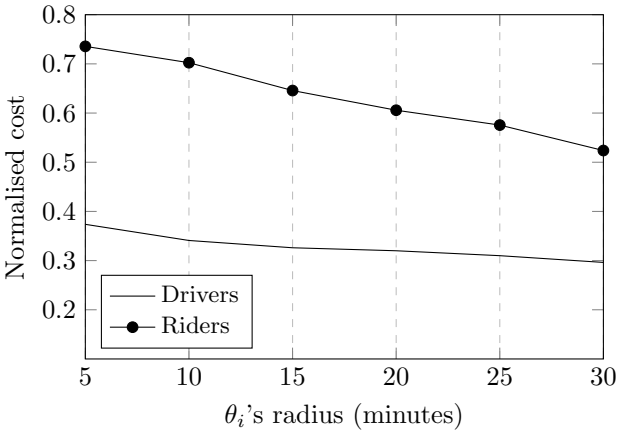

Figure 24: Normalised cost with respect to θ_i's radius

7.5. Summary of results

In general, our experimental results suggest that the kernel can be a valid stability concept in the context of SR. In fact, it induces a reasonable behaviour in the formation of groups, which can be directly correlated with some simple properties of the agents in the system (i.e., network centrality and being a driver or a rider). Moreover, the computation of kernel-stable payments has a tractable complexity and hence, it is a viable approach for large-scale environments.

8. Conclusions

In this work, we showed how the social ridesharing (SR) problem can be modelled as a GCCF problem 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 a MPR of 1.41 in the worst case). Furthermore, we tackled the payment computation aspect associated to SR, by proposing PK, the first approach able to compute kernel-stable payments for systems with thousands of agents. PK is able to compute payments for 2000 agents in less than an hour and it is 84 times faster than the state of the art in the best case. Finally, we identify a relationship between the ability of an agent to obtain a high payment and its degree in the social graph.

Future work will look at extending our approach by focusing on the development of an online SR system, motivated by the inherent dynamic nature of realistic ridesharing systems. In this perspective, we aim at the design of a SR model in which agents can join and leave the system over an extended amount of time. Such a scenario suggests a solution scheme that employs an offline method (e.g., SR-CFSS) at each time step, which possibly adopts heuristics to restrict solutions only to local areas. Myopic, short-sighted solutions are then avoided by estimating future mobility patterns for the agents, which could be inferred by the history of previous requests. A further research direction could consider multi-hop ridesharing [19] (not currently used by most existing ridesharing services) for journeys outside the urban scenario. Finally, in the context of payment computation, we aim at investigating whether recent tractability results on particular network structures [23] could also be applied to the SR scenario.

Acknowledgements

Bistaffa was supported by the H2020-MSCA-IF-2016 HPA4CF project. This work was also supported by the EPSRC-Funded ORCHID Project EP/I011587/1.

References

[1] N. Agatz, A. Erera, M. Savelsbergh, and X. Wang. "Optimization for dynamic ride-sharing: A review". In: *European Journal of Operational Research* 223.2 (2012), pp. 295–303.

[2] G. M. Amdahl. "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities". In: *Spring Joint Computer Conference*. 1967, pp. 483–485.

[3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer, 2012.

[4] Y. Bachrach, E. Markakis, E. Resnick, A. D. Procaccia, J. S. Rosenschein, and A. Saberi. "Approximating power indices: theoretical and empirical analysis". In: *Autonomous Agents and Multi-Agent Systems* 20.2 (2010), pp. 105–122.

[5] R. Baldacci, V. Maniezzo, and A. Mingozzi. "An Exact Method for the Car Pooling Problem Based on Lagrangean Column Generation". In: *Operations Research* 52.3 (2004), pp. 422–439.

[6] D. Berend and T. Tassa. "Improved bounds on Bell numbers and on moments of sums of random variables". In: *Probability and Mathematical Statistics* 30.2 (2010), pp. 185–205.

[7] F. Bistaffa, A. Farinelli, J. Cerquides, J. Rodríguez-Aguilar, and S. D. Ramchurn. "Anytime Coalition Structure Generation on Synergy Graphs". In: *International Conference on Autonomous Agents and Multi-Agent Systems*. 2014, pp. 13–20.

[8] F. Bistaffa, A. Farinelli, G. Chalkiadakis, and S. D. Ramchurn. "Recommending Fair Payments for Large-Scale Social Ridesharing". In: *ACM Conference on Recommender Systems*. 2015, pp. 139–146.

[9] F. Bistaffa, A. Farinelli, and S. D. Ramchurn. “Sharing Rides with Friends: a Coalition Formation Algorithm for Ridesharing”. In: *AAAI Conference on Artificial Intelligence*. 2015, pp. 608–614.

[10] G. Chalkiadakis, E. Elkind, and M. Wooldridge. *Computational Aspects of Cooperative Game Theory*. Synthesis Lectures on Artificial Intelligence and Machine Learning. 2011.

[11] G. Chalkiadakis, G. Greco, and E. Markakis. “Characteristic function games with restricted agent interactions: Core-stability and coalition structures”. In: *Artificial Intelligence* 232 (2016), pp. 76–113.

[12] T. H. Cormen. *Introduction to algorithms*. MIT press, 2009.

[13] G. B. Dantzig and J. H. Ramser. “The truck dispatching problem”. In: *Management science* 6.1 (1959), pp. 80–91.

[14] M. Davis and M. Maschler. “The kernel of a cooperative game”. In: *Naval Research Logistics Quarterly* 12.3 (1965), pp. 223–259.

[15] R. Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

[16] G. Demange. “On Group Stability in Hierarchies and Networks”. In: *Political Economy* 112.4 (2004), pp. 754–778.

[17] X. Deng and C. Papadimitriou. “On the complexity of cooperative solution concepts”. In: *Mathematics of Operations Research* 19.2 (1994), pp. 257–266.

[18] E. W. Dijkstra. “A note on two problems in connexion with graphs”. In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[19] F. Drews and D. Luxen. “Multi-hop ride sharing”. In: *Sixth Annual Symposium on Combinatorial Search*. 2013.

[20] A. Fanelli and G. Gianluigi. “Ride Sharing with a Vehicle of Unlimited Capacity”. In: *International Symposium on Mathematical Foundations of Computer Science*. Vol. 58. Leibniz International Proceedings in Informatics. 2016, 36:1–36:14.

[21] A. Farinelli, M. Bicego, S. Ramchurn, and M. Zucchelli. "C-link: A Hierarchical Clustering Approach to Large-scale Near-optimal Coalition Formation". In: *International Joint Conference on Artificial Intelligence*. 2013, pp. 106–112.

[22] K. Ghoseiri, A. E. Haghani, and M. Hamedi. *Real-time rideshare matching problem*. Mid-Atlantic Universities Transportation Center, 2011.

[23] G. Greco, E. Malizia, L. Palopoli, and F. Scarcello. "On the complexity of core, kernel, and bargaining set". In: *Artificial Intelligence* 175.12 (2011), pp. 1877–1910.

[24] P. Hart, N. Nilsson, and B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *Systems Science and Cybernetics, IEEE Transactions on* 4.2 (1968), pp. 100–107.

[25] B. Kallehauge, J. Larsen, O. B. Madsen, and M. M. Solomon. *Vehicle routing problem with time windows*. Springer, 2005.

[26] E. Kamar and E. Horvitz. "Collaboration and Shared Plans in the Open World: Studies of Ridesharing". In: *International Joint Conferences on Artificial Intelligence*. 2009, pp. 187–194.

[27] L. Khatib, P. Morris, R. Morris, and F. Rossi. "Temporal Constraint Reasoning with Preferences". In: *International Joint Conference on Artificial Intelligence*. 2001, pp. 322–327.

[28] A. Kleiner, B Nebel, and V Ziparo. "A Mechanism for Dynamic Ride Sharing based on Parallel Auctions". In: *International Joint Conference on Artificial Intelligence*. 2011, pp. 266–272.

[29] M. Klusch and O. Shehory. "A Polynomial Kernel-Oriented Coalition Algorithm for Rational Information Agents". In: *International Conference on Multi-Agent Systems*. 1996, pp. 157–164.

[30] H. Kwak, C. Lee, H. Park, and S. Moon. "What is Twitter, a Social Network or a News Media?" In: *International Conference on World Wide Web*. 2010, pp. 591–600.

[31] J. K. Lenstra and A. Kan. “Complexity of vehicle routing and scheduling problems”. In: *Networks* 11.2 (1981), pp. 221–227.

[32] D. Liben-Nowell, A. Sharp, T. Wexler, and K. Woods. “Computing shapley value in supermodular coalitional games”. In: *International Computing and Combinatorics Conference*. 2012, pp. 568–579.

[33] Y. Matsui and T. Matsui. “NP-completeness for calculating power indices of weighted majority games”. In: *Theoretical Computer Science* 263.1 (2001), pp. 305–310.

[34] Microsoft Research. *GeoLife Dataset*. 2009. URL: `http://research.microsoft.com/en-us/projects/geolife`.

[35] G. Moerkotte and T. Neumann. “Analysis of two existing and one new dynamic programming algorithm for the generation of optimal bushy join trees without cross products”. In: *International Conference on Very Large Databases*. 2006, pp. 930–941.

[36] R. B. Myerson. “Graphs and Cooperation in Games”. In: *Mathematics of Operations Research* 2.3 (1977), pp. 225–229.

[37] M. A. Russell. *Mining the Social Web*. O’Reilly Media, 2013.

[38] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohmé. “Coalition structure generation with worst case guarantees”. In: *Artificial Intelligence* 111.1 (1999), pp. 209–238.

[39] O. Shehory and S. Kraus. “Feasible Formation of Coalitions Among Autonomous Agents in Non-Super-Additive Environments”. In: *Computational Intelligence* 15.3 (1999).

[40] O. Skibski, T. P. Michalak, T. Rahwan, and M. Wooldridge. “Algorithms for the shapley and myerson values in graph-restricted games”. In: *International Conference on Autonomous Agents and Multi-Agent Systems*. 2014, pp. 197–204.

[41] R. E. Stearns. “Convergent Transfer Schemes for N-Person Games”. In: *Transactions of the American Mathematical Society* 134.3 (1968), pp. 449–459.

[42] P. Toth and D. Vigo. *Vehicle routing: problems, methods, and applications*. Siam, 2014.

[43] M. Vinyals, F. Bistaffa, A. Farinelli, and A. Rogers. “Coalitional energy purchasing in the smart grid”. In: *IEEE International Energy Conference*. 2012, pp. 848–853.

[44] T. Voice, M. Polukarov, and N. R. Jennings. “Coalition structure generation over graphs”. In: *Journal of Artificial Intelligence Research* 45 (2012), pp. 165–196.

[45] T. Voice, S. Ramchurn, and N. Jennings. “On coalition formation with sparse synergies”. In: *International Conference on Autonomous Agents and Multi-Agent Systems*. 2012, pp. 223–230.

[46] S. Winter and S. Nittel. “Ad hoc shared-ride trip planning by mobile geosensor networks”. In: *International Journal of Geographical Information Science* 20.8 (2006), pp. 899–916.

[47] X. Xing, T. Warden, T. Nicolai, and O. Herzog. “Smize: a spontaneous ride-sharing system for individual urban transit”. In: *Multiagent System Technologies*. 2009, pp. 165–176.

[48] L. Yang, P. Luo, C. C. Loy, and X. Tang. “A Large-Scale Car Dataset for Fine-Grained Categorization and Verification”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3973–3981.

[49] D. Zhao, D. Zhang, E. H. Gerding, Y. Sakurai, and M. Yokoo. “Incentives in Ridesharing with Deficit Control”. In: *International Conference on Autonomous Agents and Multi-Agent Systems*. 2014, pp. 1021–1028.

[50] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma. “Understanding mobility based on GPS data”. In: *International Conference on Ubiquitous Computing*. 2008, pp. 312–321.

Appendix A. Proofs of propositions

Proposition 1. *If Constraint 2 holds, for any feasible coalition structure CS*

$$M_1(CS) = \sum\nolimits_{S \in A_d(CS)} v(S)$$

is an upper bound for the value of any CS' in $ST(CS)$, i.e., the subtree rooted in CS. Formally, $M_1(CS) \geq V(CS')$ for all $CS' \in ST(CS)$.

Proof. By contradiction. Suppose there exists a coalition structure $CS' \in ST(CS)$ such that $V(CS') > M_1(CS)$, i.e., CS' results in a cost *lower*[13] than $M_1(CS)$. Now, since $CS' \in ST(CS)$ and Constraint 2 holds, CS' must have been formed by adding single riders to already formed cars in CS. All such cars correspond to coalitions whose values are lower than the original ones, since the addition of a single rider cannot reduce the cost. This contradicts $V(CS') > M_1(CS)$. □

Lemma 1. *Given a feasible coalition structure CS and a coalition structure $CS' \in ST(CS)$ such that $V(CS') > M_2(CS)$, then*

$$\exists S' \in A_d(CS') : v(S') > \frac{1}{2} \cdot \sum\nolimits_{a_i \in S'} m(a_i). \tag{A.1}$$

Proof. By contradiction. First notice that

$$V(CS') = V(A_d(CS')) + V(CS' \setminus A_d(CS')),$$

i.e., $V(CS')$ is the sum of the values of all the cars in CS' plus the values of the singletons of riders that are not drivers. From $V(CS') > M_2(CS)$, it follows that

$$V(A_d(CS')) + V(CS' \setminus A_d(CS')) > \frac{1}{2} \cdot \sum\nolimits_{a_i \in U_d(CS)} m(a_i).$$

Since $V(CS' \setminus A_d(CS')) = \sum_{S \in A_d(CS')} k(S) \leq 0$ (Equation 2), it follows that

$$V(A_d(CS')) > \frac{1}{2} \cdot \sum\nolimits_{a_i \in U_d(CS)} m(a_i).$$

Since we only merge coalitions in the formation of new coalition structures in $ST(CS)$, it is impossible that a rider exits a car, i.e., $U_d(CS) \subseteq U_d(CS')$. Moreover, since the function $m(\cdot)$ is negative (see Definition 15), it is also true that

$$V(A_d(CS')) > \frac{1}{2} \cdot \sum\nolimits_{a_i \in U_d(CS')} m(a_i). \tag{A.2}$$

Now, suppose that (A.1) is not true, i.e., $v(S') \leq \frac{1}{2} \cdot \sum_{a_i \in S'} m(a_i), \quad \forall S' \in A_d(CS')$. If we apply such property to all the coalitions S' considered in the summation $\sum_{S' \in A_d(CS')} v(S') = V(CS')$, we obtain

$$V(A_d(CS')) \leq \frac{1}{2} \cdot \sum\nolimits_{a_i \in U_d(CS')} m(a_i),$$

which contradicts (A.2). □

Proposition 2. *If Assumption 1 holds, for any feasible coalition structure CS $M_2(CS)$ is an upper bound for the value of any CS' in $ST(CS)$, i.e., the subtree rooted in CS. Formally, $M_2(CS) \geq V(CS')$ for all $CS' \in ST(CS)$.*

Proof. By contradiction. Suppose there exists a coalition structure $CS' \in ST(CS)$ such that $V(CS') > M_2(CS)$. By applying Lemma 1, there exists $S' \in A_d(CS')$ such that $v(S') > \frac{1}{2} \cdot \sum_{a_i \in S'} m(a_i)$. Since Assumption 1 holds, it follows that

$$value(concat(L^*_{S'})) > \frac{1}{2} \cdot \sum\nolimits_{a_i \in S'} m(a_i), \tag{A.3}$$

for some $L^*_{S'} \in \mathcal{VT}(S')$. Now, $value(\cdot)$ is additive (Definition 7), thus it can be seen as the sum of the costs of all the subpaths that form $concat(L^*_{S'})$. Formally,

$$value(concat(L^*_{S'})) = \sum\nolimits_{k=1}^{|L^*_{S'}|-1} value(sp(L^*_{S'}[k], L^*_{S'}[k+1])). \tag{A.4}$$

By combining (A.3) and (A.4) we obtain

$$\sum\nolimits_{k=1}^{|L^*_{S'}|-1} value(sp(L^*_{S'}[k], L^*_{S'}[k+1])) > \frac{1}{2} \cdot \sum\nolimits_{a_i \in S'} m(a_i). \tag{A.5}$$

Now, it is easy to see that the cost provided by $\sum_{a_i \in S'} m(a_i)$ cannot be higher

than *twice*[22] the cost of any valid path that goes through the starting points and destinations of the members of S'. It follows that $\frac{1}{2} \cdot \sum_{a_i \in S'} m(a_i)$ cannot be *lower* than the corresponding $value(\cdot)$ for any of such valid paths, since we consider negative cost functions. This contradicts (A.5). □

Proposition 3. *Let $a_i, a_j \in A$ with $a_i \in D$ and $a_j \notin D$. If we consider Constraint 2 (i.e., one driver per car) and $[\tau_j^\sigma + \beta_j^\sigma, \tau_j^\omega - \alpha_j^\omega] \not\subseteq [\tau_i^\sigma - \alpha_i^\sigma, \tau_i^\omega + \beta_i^\omega]$, then a_i and a_j can never be in a time feasible coalition together, i.e., $\forall S \in \mathcal{FC}(G) : \{a_i, a_j\} \subseteq S$, S is a time infeasible coalition.*

Proof. If $[\tau_j^\sigma+\beta_i^\sigma, \tau_j^\omega-\alpha_j^\omega] \not\subseteq [\tau_i^\sigma-\alpha_i^\sigma, \tau_i^\omega+\beta_i^\omega]$, then $\tau_j^\sigma+\beta_i^\sigma<\tau_i^\sigma-\alpha_i^\sigma$ or $\tau_j^\omega-\alpha_j^\omega > \tau_i^\omega+\beta_i^\omega$. Intuitively, a_j's latest departure time is earlier than a_i's earliest departure time or a_j's earliest arriving time is later than a_i's latest arriving time. Since we consider Constraint 2, a_i can be the only driver of any coalition containing both a_i and a_j. Thus, it is trivial to verify that the above time constraint will always be violated, since travelling back in time is not (yet) possible. □

Proposition 4. *Propositions 1 and 2 are valid even if we substitute the definition of $v(S)$ in Equation 2 with the definition in Equation 10.*

Proof. Given a coalition $S \in \mathcal{FC}(G)$, the value provided by $v(S)$ in Equation 2 is necessarily greater than the one provided by Equation 10, since the latter is equal to the former with the addition of $\theta_S(L_S^*, \tau_S^*)$, which is negative by definition. Notice that the $t(L_S^*) + c(L_S^*) + f(L_S^*)$ is exactly the same, since we make Assumption 1 in both cases, and we assess L_S^* in the same way. As a consequence, given a feasible coalition structure CS, $V(CS)$ is greater if we consider Equation 2 with respect to Equation 10. Therefore, since Propositions 1 and 2 provide upper bounds and are valid considering Equation 2, they are also valid with Equation 10. □

Proposition 5. *Algorithm 6 computes each s_{ij} correctly.*

[22] If we sum all the values of the couples of edges incident to the points that form a given path, we consider each edge twice.

Proof. Once the loop has ended, each s_{ij} stores the maximum excess among *all* feasible coalitions with a_i but without a_j, with both a_i and a_j part of the same coalition in CS. This matches line 5 of Algorithm 4. □

Proposition 6. *Algorithm 6 lists all feasible coalitions only once and it has a worst-case time complexity of* $O\left(n^k\right)$.

Proof. Algorithm 6 lists all $\hat{k}$-subgraph of G exactly once [45]. Note that the number of $\hat{k}$-subgraphs is $O\left(n^k\right)$, since we only consider coalitions with up to k members [39]. Hence, Algorithm 6 makes at most $O\left(n^k\right)$ calls to UpdateMax. Finally, note that the time complexity of UpdateMax is constant with respect to n, since computing $e\left(S, x\right)$ requires the computation of $v\left(S\right)$ (which has constant time complexity), and the loop at lines 3–8 requires $O\left(k^2\right)$ iterations. Moreover, UpdateMax only considers coalitions that satisfy Constraint 1 (whose check is constant with respect to n) and it computes each coalitional value only once at line 2. Thus, Algorithm 6 computes all feasible coalitions only once and its worst-case time complexity is $O\left(n^k\right)$. □

Proposition 7. *Algorithm 5 has a polynomial worst-case time complexity with respect to* n*, i.e.,* $O\left(-\log_2\left(\epsilon\right) \cdot n^{k+1}\right)$.

Proof. Here we refer to equations and lemmas provided by Stearns [41]. Each iteration of Algorithm 5 identifies the agents a_i and a_j with the maximum surplus difference $\delta = s_{ij} - s_{ij}$, performing a transfer of size d from a_j to a_i. Thus, by Lemma 1 [41], in the following iteration these surpluses will be $s'_{ij} = s_{ij} - d$ and $s'_{ji} = s_{ji} + d$. Notice that $s'_{ij} - s'_{ji} = s_{ij} - s_{ji} - 2 \cdot d = \delta - 2 \cdot d$. Now, by definition of d (lines 11–14 of Algorithm 5), $d \leq \delta/2$, hence $s'_{ij} - s'_{ji} \geq 0$. Therefore, we can affirm that the transfer from a_j to a_i is indeed a *K-transfer*, since it satisfies Equation 4, 5, 6 and 7 [41]. Lemma 2 [41] ensures the convergence of Algorithm 2, by affirming that a K-transfer *cannot* increase the larger surpluses in the system. Specifically, in the next iteration the difference between the surpluses between a_j to a_i will be half of what was in the previous one. After λ iterations, its value will be $\frac{1}{2^\lambda}$ of the original one. Thus, it will take

$\lambda = \log_2(^{[\delta_0/v(CS)]}/_{\epsilon})$ iterations to ensure that $^{[\delta_0/v(CS)]}/_{2^\lambda} \leq \epsilon$, with δ_0 being the original maximum s_{ij} surplus. Since we have n agents into the setting, it will take $\lambda \cdot n = O\left(-\log_2(\epsilon) \cdot n\right)$ iterations to convergence. Then, we know by Proposition 2 that ComputeMatrix, which dominates the time complexity of each iteration, has a worst-case time complexity of $O\left(n^k\right)$. Given this, Algorithm 2 has a worst-case time complexity of $O\left(-\log_2(\epsilon) \cdot n^{k+1}\right)$. □

Appendix B. Existence of the core in the SR scenario

As introduced in Section 2.2, the *core* [10] is a very strong stability concept, whose computation has an exponential time complexity with respect to the number of agents. An in-depth discussion of the complexity aspects of core-related problems is provided by Chalkiadakis et al. [11].

Due to its strength, the core is not guaranteed to be always non-empty, i.e., it is not always possible to compute a core-stable payment allocation. Consider the following SR instance, which, for simplicity, does not take into account time constraints. Such instance has been generated from the datasets discussed in Section 6. Let G be the graph in Figure B.25. The only driver is agent a_5. Consider the following coalitional values (only feasible coalitions are reported):

$$v(\{a_0\}) = 3.00€,\ v(\{a_1\}) = 3.00€,\ v(\{a_2\}) = 3.00€,\ v(\{a_3\}) = 3.00€,$$
$$v(\{a_4\}) = 3.00€,\ v(\{a_5\}) = 2.02€,\ v(\{a_5, a_2\}) = 3.13€,$$
$$v(\{a_5, a_2, a_4\}) = 3.19€,\ v(\{a_5, a_0, a_2\}) = 3.84€,\ v(\{a_5, a_0, a_2, a_3\}) = 3.99€,$$
$$v(\{a_5, a_0, a_2, a_4\}) = 4.11€,\ v(\{a_5, a_0, a_2, a_3, a_4\}) = 4.41€,$$
$$v(\{a_5, a_1\}) = 3.76€,\ v(\{a_5, a_1, a_4\}) = 3.85€,\ v(\{a_5, a_0, a_1\}) = 5.01€,$$
$$v(\{a_5, a_1, a_3\}) = 4.68€,\ v(\{a_5, a_1, a_3, a_4\}) = 4.81€,$$
$$v(\{a_5, a_0, a_1, a_4\}) = 5.13€,\ v(\{a_5, a_0, a_1, a_3\}) = 5.18€,$$
$$v(\{a_5, a_0, a_1, a_3, a_4\}) = 5.30€,\ v(\{a_5, a_1, a_2\}) = 4.85€,$$
$$v(\{a_5, a_1, a_2, a_4\}) = 4.94€,\ v(\{a_5, a_0, a_1, a_2\}) = 5.61€,$$
$$v(\{a_5, a_1, a_2, a_3\}) = 5.46€,\ v(\{a_5, a_1, a_2, a_3, a_4\}) = 5.59€,$$
$$v(\{a_5, a_0, a_1, a_2, a_4\}) = 5.73€,\ v(\{a_5, a_0, a_1, a_2, a_3\}) = 5.78€.$$

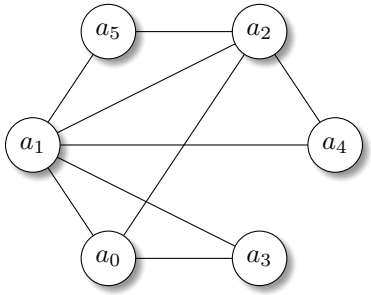

Figure B.25: Example of a social network with 6 agents.

The optimal coalition structure in the above instance is $CS^* = \{\{a_5, a_0, a_2, a_3, a_4\}, \{a_1\}\}$. We implemented a Linear Programming (LP) algorithm[23] that computes a core-stable payment allocation, if it exists. Using such an algorithm, we determined that the core is empty in the above instance.

We ran further experiments to evaluate the percentage of instances that have an empty core in the SR scenario. Figure B.26 shows such a percentage, considering 60 SR instances generated from our datasets for each n. Our results show that, *most of the times*, the core is empty in the SR scenario, in contrast with the kernel, which *always* exists. Specifically, the core is empty in the 75% of the instances with 13 agents or more. More important, our results confirm that, as expected, the number of instances with an empty core increases when we increase the number of agents. Henceforth, the core is not a viable stability concept for large-scale SR problems we are interested to tackle.

Appendix C. Pseudo-code of the CFSS algorithm

In this appendix we report the pseudo-code of the CFSS algorithm [7], which solves the GCCF problem corresponding to a given graph G. Notice that, being a branch and bound algorithm, CFSS requires a technique to compute an upper-bound $M(\cdot)$ for the characteristic function. A complete discussion about the techniques used to compute $M(\cdot)$ and, in general, about the CFSS algorithm,

[23] The implementation is available at `https://github.com/filippobistaffa/PK/tree/core`.

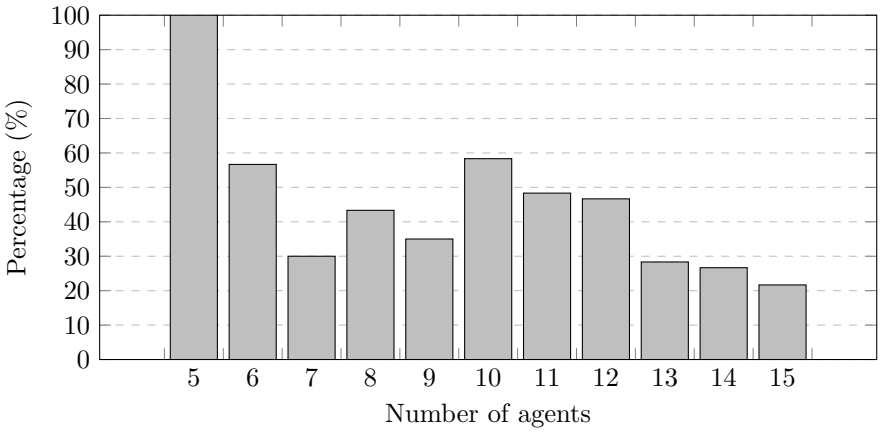

Figure B.26: Percentage of SR instances with a non-empty core.

is provided by Bistaffa et al. [7].

Algorithm 9 CFSS(G)

1: $\mathcal{G}_c \leftarrow G$ with all green edges
2: $best \leftarrow \mathcal{G}_c$ {Initialise current best solution with singletons}
3: $F \leftarrow \emptyset$ {Initialise search frontier F with empty stack}
4: F.PUSH $(\mathcal{G}_c)$ {Push $\mathcal{G}_c$ as the first node to visit}
5: **while** $F \neq \emptyset$ **do** {Branch and bound loop}
6: $\quad node \leftarrow F$.POP $()$ {Get current node}
7: $\quad$ **if** $M\,(node) > V\,(best)$ **then** {Check bound value}
8: $\quad\quad$ **if** $V\,(node) > V\,(best)$ **then**
9: $\quad\quad\quad best \leftarrow node$ {Update current best solution}
10: $\quad\quad F$.PUSH $($CHILDREN $(node))$ {Update frontier F}
11: **return** *best* {Return optimal solution}

Algorithm 10 CHILDREN($\mathcal{G}$)

1: $\mathcal{G}'_c \leftarrow \mathcal{G}_c = (\mathcal{A}, \mathcal{E}, colour)$ {Initialise graph G' with $\mathcal{G}_c$}

2: $Ch \leftarrow \emptyset$ {Initialise the set of children}

3: **for all** $e \in \mathcal{E} : colour\,(e) = green$ **do** {For all green edges}

4: $\quad Ch \leftarrow Ch \cup \{\text{GREENEDGECONTR}\,(\mathcal{G}'_c, e)\}$

5: $\quad$ Mark edge e with colour red in $\mathcal{G}'_c$

6: **return** Ch {Return the set of children}