Recommending Fair Payments for Large-Scale Social Ridesharing

Filippo Bistaffa,
Alessandro Farinelli
University of Verona
filippo.bistaffa@univr.it
alessandro.farinelli@univr.it

Georgios Chalkiadakis
Technical University of Crete
gehalk@intelligence.tuc.gr

Sarvapali D. Ramchurn
University of Southampton
sdr@ecs.soton.ac.uk

ABSTRACT

We perform recommendations for the Social Ridesharing scenario, in which a set of commuters, connected through a social network, arrange one-time rides at short notice. In particular, we focus on how much one should pay for taking a ride with friends. More formally, we propose the first approach that can compute fair coalitional payments that are also stable according to the game-theoretic concept of the *kernel* for systems with thousands of agents in real-world scenarios. Our tests, based on real datasets for both spatial (GeoLife) and social data (Twitter), show that our approach is significantly faster than the state-of-the-art (up to 84 times), allowing us to compute stable payments for 2000 agents in 50 minutes. We also develop a parallel version of our approach, which achieves a near-optimal speed-up in the number of processors used. Finally, our empirical analysis reveals 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).

Categories and Subject Descriptors

H.4.0 [**Information Systems Applications**]: General; I.2.11 [**Distributed Artificial Intelligence**]: Intelligent Agents, Multiagent Systems

General Terms

Algorithms

Keywords

Algorithm Scalability; Innovative Applications; Coalition Formation; Social Networks; Ridesharing

1. INTRODUCTION

Real-time ridesharing, in which people arrange one-time rides at short notice, is rapidly changing the way people commute for their daily activities. Online services such as *Uber* or *Lyft* compete with standard transportation systems (such as taxis or public transport), allowing users to quickly share their positions and arrange rides with other people they know or trust. A clear tendency in such services is to embed a social network in their framework, favouring the formation of groups of users that are connected in such network. In fact, Uber[1] and Lyft[2] incentivise users to share rides with their friends, showing that social relationships play a fundamental role in the ridesharing scenario, which is consequently referred as *Social Ridesharing* (SR). Several works [20, 21] have focused on building recommendation mechanisms in the book-a-taxi domain, while recent research has shown that recommendation accuracy can be greatly improved by incorporating trust and geo-spatial information embedded in social networks that also encompass geographical data [14, 19]. However, recommendation systems research has never focused on the SR problem, as we do in this paper.

In more detail, the SR problem is, by and large, a *Coalition Formation* (CF) problem [13], requiring that the set of rides is partitioned in disjoint groups (i.e., *coalitions*) that minimise the overall transportation costs for the entire system. In addition, such a division must be computed assessing the constraints imposed by the social network (naturally modelled as a graph), in order to ensure that passengers are connected by *friends-of-friends* relationships. The most successful approach that tackles the SR problem from a real-world perspective is proposed by Bistaffa et al. [3], which define it as a *Graph-Constrained Coalition Formation* (GCCF) problem, and also provide a solution that scales to thousands of agents. That work, however, remains silent about the problem of distributing the travel expenses of each car among its passengers. Such a task, namely the *payoff distribution* task [15], 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. In fact, payoffs (corresponding to cash payments for sharing trip costs) to the commuters need to be computed given their 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 distribution in 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 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* [6]

RecSys'15, September 16–20, 2015, Vienna, Austria.
© 2015 ACM. ISBN 978-1-4503-3692-5/15/09 ...$15.00.
DOI: http://dx.doi.org/10.1145/2792838.2800177.

[1]http://blog.uber.com/2013/07/15/faresplit.
[2]http://www.lyft.com/help/article/1637045.

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 (cf. Section 2 below). Kernel stability has been widely studied in cooperative game theory, and certain approaches have been proposed to compute kernel-stable payments [10, 16]. Specifically, Shehory and Kraus [16] adopt a transfer scheme that represents the state-of-the-art approach to compute kernel-stable payments. 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 one. First, it is designed for classic CF, failing to exploit the graph-constrained nature of this problem. Second, this algorithm assumes that coalitional values can be assessed at no computational cost (e.g., stored in memory or provided by an oracle). This hypothesis does not apply to SR, in which the value of a coalition is the solution of a routing problem and it cannot be stored in memory without limiting the scalability. These shortcomings lead to inefficiencies that prevent the application of the method proposed by Shehory and Kraus [16] in our case. Moreover, neither Klusch and Shehory [10] nor Shehory and Kraus [16] provide parallel solution algorithms, failing to take advantage of modern multi-core hardware. Finally, these papers have never been tested in realistic large-scale environments.

Against this background, we propose Paying for Rides with Friends (PRF), the first approach to compute kernel-stable payments for SR. In particular, we address the shortcomings of the state-of-the-art algorithm in real-world scenarios, we design a solution that scales up to systems of thousands of agents, and we also provide an efficient parallel version. PRF allows us to provide the first recommendation system for SR scenarios, as it can be used in conjunction with Bistaffa et al.'s approach to provide recommendations for the optimal groups to form, and also to define fair, stable payments that commuters should make as a compensation for their rides.

In more detail, this paper advances the state-of-the-art in the following ways: i) we present the first approach able to compute kernel-stable payments for systems with thousands of agents, ii) we benchmark our approach on real data (i.e., GeoLife from Microsoft Research for the geospatial data and Twitter for social networks), 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, iii) 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), and iv) we provide a parallel version of our method, providing a practical solution technique for large-scale systems thanks to a speed-up of 10.6 on a 12-core machine w.r.t. the serial approach.

2. BACKGROUND

In the following sections we provide the necessary background on Social Ridesharing and on the kernel, i.e., the stability concept we use to compute agents' payments.

2.1 The Social Ridesharing Problem

The *Social Ridesharing* (SR) problem [3] considers a set of n agents (or *commuters*) $\mathcal{A} = \{a_1, \dots, a_n\}$ connected through a social network G, which arrange one-time rides at short notice in order to minimise the travel cost of the overall system. In the SR scenario, each commuter (which has a desired starting and destination point in the geographic map) can share a ride in one of the cars provided by the set of drivers $\mathcal{D} \subseteq \mathcal{A}$. We assume that each car has k seats, hence the maximum size of each group is at most k. To be part of a solution, each group of commuters must constitute a *feasible* coalition, i.e., it must satisfy the following constraints:

Constraint 1. *Its members must induce a $\hat{k}$-subgraph of G, where a $\hat{k}$-subgraph denotes a connected subgraph with* at most *k nodes.*

Constraint 2. *It must contain at least one driver.*[3]

The set of all feasible coalitions is referred as $\mathcal{FC}(G)$. Bistaffa et al. [3] tackle the SR problem casting it as a GCCF problem, in which each car represents a feasible coalition $C \in \mathcal{FC}(G)$, whose coalitional value $v(C)$, quantifying the total transportation cost for the group represented by C, is:

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

where P_C is the optimal path for C, $t(\cdot)$, $c(\cdot)$ and $f(\cdot)$ are negative functions representing time, cognitive (i.e., the fatigue incurred by the driver during the trip) and fuel costs of driving through the path. The computation of $v(C)$ (and of P_C in particular) requires solving a routing problem. In general, this is a hard problem [12], which can be solved in this setting thanks to the limited[4] number of points in each path. Bistaffa et al. [3] only solve the optimisation problem of the SR scenario, which involves computing the optimal feasible *coalition structure* CS^* (i.e., a partition of $\mathcal{A}$ into disjoint feasible coalitions) that maximises the social welfare:

$$CS^* = \underset{CS \in \mathcal{CS}(G)}{\arg\max} \sum\nolimits_{C \in CS} v(C),$$

where $\mathcal{CS}(G)$ refers to the set of all the feasible coalition structures. Such a problem is solved using a modified version of the state-of-the-art algorithm for the GCCF problem[5] (i.e., the CFSS algorithm [2]), providing optimal and approximate solutions with good quality guarantees.

Bistaffa et al.'s approach can be used to provide recommendations about the optimal groups to arrange from a joint cost perspective. However, such an approach does not address the payment computation problem for SR, as we do in this paper. In this context, a number of works have focused on computing incentives for ridesharing using mechanism design [8, 9] in order to promote *truthfulness* in the commuters. Nonetheless, in addition to such property, they do not address the stability of payments, which is crucial in selfish settings like ridesharing, but it has never been studied before. In addition, these works do not consider the role of the social network. Against this background, we introduce our payoff distribution scheme for SR, detailing the game-theoretic stability concept we employ, i.e., the kernel.

[3]Agents without a car can be in a singleton, which is considered a feasible coalition. If the total number of available seats is less than the total number of commuters in the system, an agent a_i might need to resort to public transport paying a cost $\kappa(\{a_i\})$ for the ticket.

[4]The points are at most $2 \cdot k$ (i.e., a starting and a destination point for each commuter). As an example, for k=5 only 2520 paths must be considered to compute the optimal one [3].

[5]Notice that, following the recent literature on optimisation in ridesharing [8], Bistaffa et al. maintain the assumption that all the request by the agents arrive at the same time, hence the solution can be computed offline in a static system.

2.2 The Kernel

Formally, the *payment computation* problem involves the computation of a *payoff allocation vector* x, which specifies a payoff x_i for each agent $a_i \in \mathcal{A}$ as a compensation of their contributions [15, 5]. As introduced above, computing payments that are *stable* (i.e., they incentivise agents to maintain the provided configuration) and *individually rational* (i.e., they guarantee that each agent receives at least the value of its singleton coalition) is of utmost importance in systems with selfish rational agents. As such, payoffs have to be distributed among agents to ensure that members get rewarded according to their bargaining power [5].

A number of recent papers have examined the computational complexity of reaching stable solutions in graph-restricted coalition formation scenarios [4, 7]. Nevertheless, these papers focus on the stability concept of *the core*, which denotes the set of payoff configurations that provide no incentives to players to deviate from the solution coalition structures. Unfortunately, the core might be empty, and its complexity makes it next to impossible to compute in real-world scenarios involving thousands of agents.

In this paper, we focus on *the kernel*, a stability concept introduced by Davis and Maschler [6] that is always non-empty and can be approximated in polynomial time when the size of coalitions is limited. 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. In order to define the kernel, we first define the *excess* of a coalition C as $e(C, x) = v(C) - x(C)$, where $x(C)$ refers to the sum of the payments of the members of C, i.e., $x(C) = \sum_{a_i \in C} x_i$. In the kernel, 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. In more detail, the *surplus* s_{ij} of agent a_i over agent a_j with respect to a given payoff configuration x, is defined by $s_{ij} = \max_{C \in 2^{\mathcal{A}} | a_i \in C, a_j \notin C} e(C, x)$, where $2^{\mathcal{A}}$ denotes the powerset of $\mathcal{A}$. In other words, s_{ij} is the maximum of the excesses of all coalitions C that include a_i and exclude a_j, with C not in the current coalition structure (since under the current coalition structure agents a_i and a_j belong in the same coalition). 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. Importantly, the set of kernel-stable payoff vectors is always non-empty. Stearns [17] 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, the ϵ-kernel [10] has been introduced, in order to represent an allocation whose payoffs do not differ from an element in the kernel by more than ϵ. Note that an ϵ-kernel-stable payoff allocation can be computed in $O(n)$ iterations.[6]

[6]The number of iterations is also affected by ϵ. A more detailed discussion is provided by Shehory and Kraus [16].

2.3 Computing Payments in the ϵ-Kernel

The current state-of-the-art approach to compute an ϵ-kernel payoff allocation for classic CF is presented in Algorithm 1 [16]. 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 (lines 2–5), which iterates over the entire set of coalitions to assess the maximum excess (line 5) 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 excess difference (i.e., $s_{ij} - s_{ji}$) is set up, while ensuring that each payment is individually rational. On the one hand, the maximisation at line 5 is a key bottleneck for classic CF, since it involves enumerating an exponential number of coalitions, i.e., $O(2^n)$. On the other hand, when the size of the coalitions is limited to k members as in our scenario, (denoted as k-CF in the remainder of the paper), such an algorithm has polynomial time complexity, since the coalitions are $O(n^k)$.

Algorithm 1 ShehoryKrausKernel(x, CS, ϵ)

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

Next, we discuss how this approach could be used to compute an ϵ-kernel payoff allocation for the SR problem.

3. COMPUTING PAYMENTS FOR SR

Algorithm 1 has been designed to compute payments for CF scenarios in which the set of coalition is *not* restricted by a graph. Such an approach can be readily applied also when the size of coalitions is limited to k members, in which 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 $\mathcal{R} = \{\{a_i\} \cup R \mid R$ is a h-combination of $\mathcal{A} - \{a_i, a_j\}, \forall h \in \{1, \ldots, k-1\}\}$.

On the other hand, in GCCF scenarios like SR this simple approach would iterate over several unfeasible 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, so to avoid any unnecessary computation.

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	**$\{a_1\}$** $\{a_1, a_2\}$ **$\{a_1, a_4\}$** $\{a_1, a_2, a_4\}$
a_1	a_4	**$\{a_1\}$** **$\{a_1, a_2\}$** **$\{a_1, a_3\}$** $\{a_1, a_2, a_3\}$
a_2	a_1	$\{a_2\}$
a_2	a_3	**$\{a_2\}$ $\{a_1, a_2\}$ $\{a_1, a_2, a_4\}$**
a_2	a_4	**$\{a_2\}$ $\{a_1, a_2\}$ $\{a_1, a_2, a_3\}$**
a_3	a_1	$\{a_3\}$
a_3	a_2	**$\{a_3\}$ $\{a_1, a_3\}$ $\{a_1, a_3, a_4\}$**
a_3	a_4	**$\{a_3\}$ $\{a_1, a_3\}$ $\{a_1, a_2, a_3\}$**
a_4	a_1	$\{a_4\}$
a_4	a_2	**$\{a_4\}$ $\{a_1, a_4\}$ $\{a_1, a_3, a_4\}$**
a_4	a_3	**$\{a_4\}$ $\{a_1, a_4\}$ $\{a_1, a_2, a_4\}$**

Table 1: Coalitions considered at each iteration.

Moreover, Algorithm 1 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 $\mathcal{A}=\mathcal{D}=\{a_1, a_2, a_3, a_4\}$ and a graph G that induces a 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 assume a coalition structure $CS = \{\{a_1, a_2, a_3, a_4\}\}$. In this case, such a 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 fact can substantially reduce the efficiency and the scalability of the computation of the surplus matrix 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 the number of feasible coalitions is $O\left(n^k\right)$, 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 significantly reduces efficiency and scalability, as shown in Section 4.1.

To overcome these issues, in the next section we present the PRF algorithm, an improved technique to calculate the surplus matrices in the SR scenario, allowing our payment scheme to scale up to systems with thousands of agents.

3.1 The PRF algorithm

We now present the PRF (Paying for Rides with Friends) algorithm, our method to compute an ϵ-kernel payoff allocation, given a coalition structure CS that is a solution to the SR problem. Our contribution improves on the k-CF version of Algorithm 1 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 $C \in CS$, we iterate over the set of $\hat{k}$-subgraphs of G (i.e., those satisfying Constraint 1 of the SR problem). Then, we update the appropriate values of the surplus matrix for each coalition with at least one driver (i.e., satisfying Constraint 1). By so doing, we ensure the exact coverage of the entire set of feasible coalitions $\mathcal{FC}(G)$, as ensured by Proposition 2.

PRF is detailed in Algorithm 2. 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 SlyCE algorithm [18], which can list all the subgraphs of a given graph without redundancy (i.e., each subgraph is computed only once).

Algorithm 2 PRF(CS, ϵ)

```
 1: for all $C \in CS$ do
 2:     for all $a_i \in C$ do
 3:         $x_i \leftarrow v(C)/|C|$ {Equally split coalitional value}
 4: repeat
 5:     {Compute surplus matrix}
 6:     $s \leftarrow$ ComputeMatrix $(CS, x)$
 7:     {$a_{i^*}$ and $a_{j^*}$ have the maximum surplus difference $\delta$}
 8:     $\delta \leftarrow \max_{(a_i,a_j)\in\mathcal{A}^2} (s_{ij} - s_{ji})$
 9:     $(a_{i^*}, a_{j^*}) \leftarrow \arg\max_{(a_i,a_j)\in\mathcal{A}^2} (s_{ij} - s_{ji})$
10:     {Ensure that payments are individually rational}
11:     if $x_{j^*} - v(\{a_{j^*}\}) < \delta/2$ then
12:         $d \leftarrow x_{j^*} - v(\{a_{j^*}\})$
13:     else
14:         $d \leftarrow \delta/2$
15:     $x_{j^*} \leftarrow x_{j^*} - d$ {Transfer payment from $a_{j^*}$...}
16:     $x_{i^*} \leftarrow x_{i^*} + d$ {... to $a_{i^*}$}
17: until $\delta/v(CS) \leq \epsilon$
```

Algorithm 3 ComputeMatrix(CS, x)

```
1: $s \leftarrow -\infty$ {Initialise the entire matrix with $-\infty$}
2: for all $C$ that induce a $\hat{k}$-subgraph of $G$ do
3:     $s \leftarrow$ UpdateMax $(C, CS, s, x)$
4: return $s$
```

Algorithm 4 UpdateMax(C, CS, s, x)

```
1: if $|C| = 1 \vee C \cap \mathcal{D} \neq \emptyset$ then {Constraint 2 of SR problem}
2:     $e_C \leftarrow e(C, x)$ {Compute the excess of coalition $C$}
3:     for all $a_i \in C$ do {For each agent $a_i$ in coalition $C$}
4:         $C' \leftarrow$ the coalition in $CS$ that contains $a_i$
5:         for all $a_j \in C' - C$ do {For each $a_j \in C'$ but $\notin C$}
6:             {$s_{ij}$ is updated with the maximum between}
7:             {its old value and the excess of coalition $C$}
8:             $s_{ij} \leftarrow \max(s_{ij}, e_C)$
9: return $s$
```

UpdateMax only considers the coalitions that satisfy Constraint 2 of the SR problem (line 1), i.e., singletons or coalitions that contain at least one driver. For every coalition C that satisfies this property, lines 3–8 update all the values s_{ij} for which a_i is a member of C and a_j is part of C' (i.e., the coalition in CS that contains a_i) but is not part of C. The correctness of our approach is ensured by Proposition 1.

Proposition 1. *Algorithm 3 computes each s_{ij} correctly.*

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 1. ☐

Moreover, our surplus matrix-calculating method has polynomial time complexity, while allowing to compute all feasible coalitions only once, as shown by Proposition 2.

Proposition 2. *Algorithm 3 lists all feasible coalitions only once and it has a worst-case time complexity of $O\left(n^k\right)$.*

Proof. Algorithm 3 lists all $\hat{k}$-subgraph of G exactly once [18]. Note that the number of $\hat{k}$-subgraphs is $O\left(n^k\right)$, since we only consider coalitions with up to k members [16]. Hence, Algorithm 3 makes at most $O\left(n^k\right)$ calls to UPDATEMAX. Finally, note that the time complexity of UPDATEMAX is constant w.r.t. n, since computing $e\left(C, x\right)$ requires the computation of $v\left(C\right)$ (which has constant time complexity [3]), and the loop at lines 3–8 requires $O\left(k^2\right)$ iterations. Moreover, UPDATEMAX only considers coalitions that satisfy Constraint 2 and it computes each coalitional value only once at line 2. Thus, Algorithm 3 computes all feasible coalitions only once and its worst-case time complexity is $O\left(n^k\right)$. $\square$

Proposition 3. *Algorithm 2 has a polynomial worst-case time complexity w.r.t. n, i.e.,* $O\left(-\log_2\left(\epsilon\right) \cdot n^{k+1}\right)$.

Proof. All the equations and lemmas referred in the following proof are provided by Stearns [17]. Each iteration of Algorithm 2 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, 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 2), $d \leq \delta/2$, 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. Lemma 2 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 approximately $\lambda = \log_2([\delta_0/v(CS)]/\epsilon)$ 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 approximately $\lambda \cdot n = O\left(-\log_2\left(\epsilon\right) \cdot n\right)$ 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 $O\left(-\log_2\left(\epsilon\right) \cdot n^{k+1}\right)$. $\square$

Given this, PRF 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 PRF, which allows us to distribute the computational burden among different threads, taking advantage of modern multi-core hardware.

3.2 P-PRF

We now detail P-PRF, the parallel version of our approach, in which the most computation-intensive task, i.e., the computation of s, is distributed among T available threads. In particular, Algorithm 5 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 $\mathcal{DIV}\left(G, t, k\right)$, i.e., the tth fraction of the set of all $\hat{k}$-subgraphs of G, computed using the D-SlyCE algorithm [18]. Specifically, this fraction is obtained by splitting the first generation of children nodes in the search tree generated by the SlyCE algorithm [18] 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 (a more detailed discussion is provided by Voice et al. [18]). As such, it also distributes the computation of the coalitional values.

Algorithm 5 P-COMPUTEMATRIX(CS, x, T)

```
1: $s \leftarrow -\infty$ {Initialise all matrices with $-\infty$}
2: for all $t \in \{1, \dots, T\}$ do in parallel
3:     for all $C \in \mathcal{DIV}(G, t, k)$ do
4:         $s^t \leftarrow$ UPDATEMAX$(C, CS, s^t, x)$
5: for all $i \in \{1, \dots, n\}$ do in parallel
6:     for all $j \in \{1, \dots, n\}$ do in parallel
7:         $s_{ij} \leftarrow \max_{t \in \{1,\dots,T\}} s^t_{ij}$
8: return $s$
```

We provide the following example to clarify how this division is realised. Consider the same $\mathcal{FC}\left(G\right)$ of the example in Section 3, and assume $T = 4$. Then, the necessary coalitions are distributed by doing the following partitioning:

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

Note that, since each matrix s^t is modified only by thread t,[7] Algorithm 5 contains only one synchronisation point (i.e., at 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 $\mathcal{FC}\left(G\right)$ has been computed by a thread.

The effectiveness of our parallel approach will be demonstrated through the empirical evaluation, detailed below.

4. EMPIRICAL EVALUATION

Having described and analysed our approach to compute kernel-stable payments for the SR problem, we now benchmark our approach on a real-world dataset. We first present our evaluation methodology, then we discuss the achieved results. The main goals of the empirical analysis are:

1. To test the performance of our approach when computing payments for systems of thousands of agents.
2. To compare the efficiency of our algorithm w.r.t. the state-of-the-art approach proposed Shehory and Kraus.
3. To perform an analysis of the features that influence the payoff allocation.
4. To estimate the speed-up obtainable by using P-PRF w.r.t. PRF.

[7]Our parallel approach 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.

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. This approach does not affect realism and, in our view, provides a far better experimental setting than using synthetic data. Moreover, since we are interested in evaluating the algorithmic performance of our approach, the specific datasets adopted are not crucial for the purposes of our analysis.

In particular, the spatial map is a realistic representation of the city of Beijing, derived from the GeoLife [22] dataset provided by Microsoft. These trajectories are also adopted to sample random paths used to provide the start and destination points to the riders in our tests. Moreover, in each instance the graph G is a subgraph of a large crawl of the Twitter network completed in 2010 by Kwak et al. [11].

In all our experiments we use $\epsilon = 0.05$. Since the coalition structure considered to compute the payments is obtained using the CFSS algorithm [3], we consider the same parameters with them in the definition of the cost model in the SR scenario. Specifically, we adopt a cost model that only considers fuel expenses (considering a fuel cost of 1 € per litre and an average consumption of 1 litre of fuel every 15 km). Moreover, we assume that each car has a capacity of 5 seats, i.e., $k = 5$. Our approach is executed on a machine with dual 8-core 2.60GHz processors and 32 GB of memory.

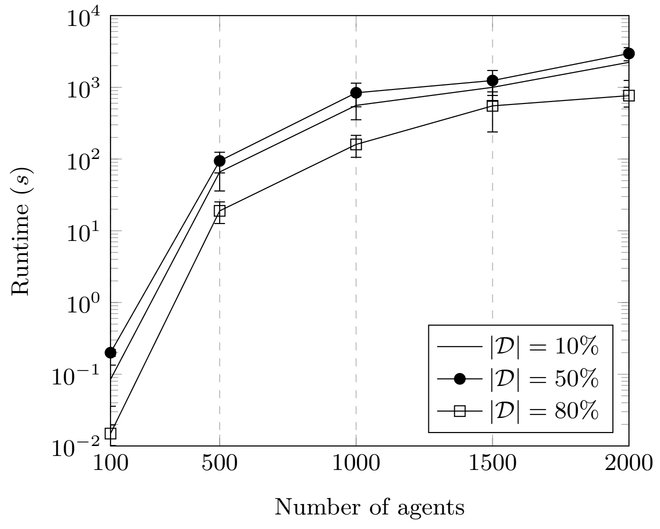

Figure 1: Runtime needed to compute payments.

Figure 1 shows the runtime needed to execute P-PRF on systems with a large number of agents, i.e., $n \in \{100, 500, 1000, 1500, 2000\}$. Our results show that P-PRF 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 $|\mathcal{D}| \in \{10\%, 50\%, 80\%\}$ with 20 repetitions for each n and $|\mathcal{D}|$, reporting the average and the standard error of the mean. In each test, the coalition structure has been computed using the approximate version of CFSS [3].

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 $|\mathcal{D}| = 80\%$ is easier w.r.t. $|\mathcal{D}| = 10\%$ and $|\mathcal{D}| = 50\%$. Our findings are consistent with the results obtained by Bistaffa et al. [3], showing that the scenario with $|\mathcal{D}| = 50\%$ is more difficult to solve (i.e., requires a greater runtime), 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).

4.1 Comparison with State-of-the-Art

Figure 2 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 [16], i.e., Algorithm 1. In particular, we consider the runtime needed to solve random instances with $n \in \{30, 40, 50, 60, 70, 80, 90, 100\}$ and $|\mathcal{D}| = 50\%$,[8] with 20 repetitions for each n. To ensure a fair comparison, both algorithms have been run on the same set of instances. Moreover, for this comparison we employ the serial version of PRF, since Algorithm 1 is serial.

Our results show that PRF is at least one order of magnitude faster, 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 latter 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 PRF, and 14 minutes required by P-PRF. In particular, the approach proposed by Shehory and Kraus [16] is slower since it makes several redundant computations of many coalitional values, resulting in a significant impact on its runtime.

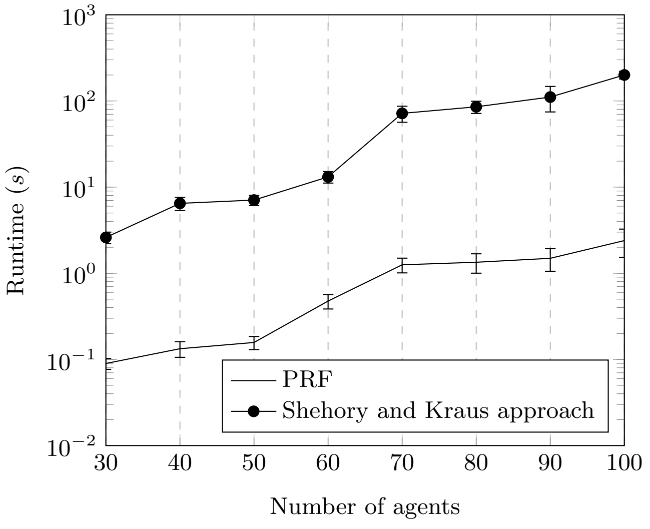

Figure 2: Runtime needed to compute payments.

[8]We benchmark both approaches in the most difficult scenario, i.e., the one with the largest search space. This is an intrinsic property of the instances, ensuring the fairness of our comparison.

4.2 Parallel Performance

Here we analyse the speed-up that can be achieved by using P-PRF w.r.t. PRF, i.e., its serial version. We ran the algorithms on 20 random instances with 500 agents and $|\mathcal{D}| = 50\%$, using a machine with 2 Intel® Xeon® E5-2420. For a fair comparison, both algorithms have been run on the same set of instances. The speed-up measured during these tests has been compared with the maximum theoretical one provided by Amdahl's Law [1], considering an estimated non-parallelisable part of 1%, due to memory allocation and thread initialisation. Our experiments show 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.

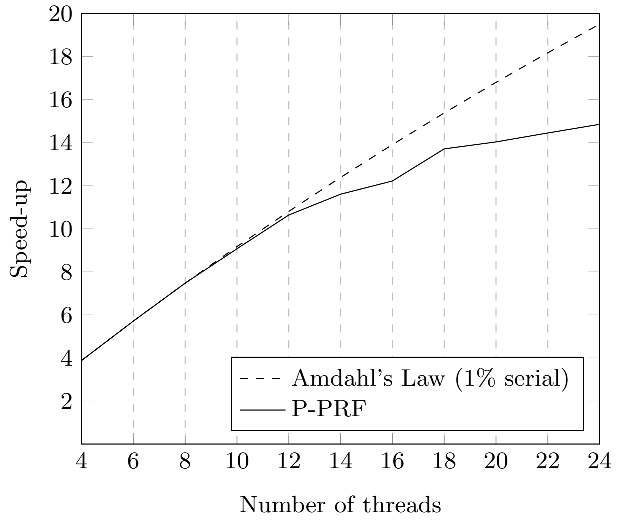

Figure 3: Multi-threading speed-up.

4.3 Costs and Network Centrality

The purpose of this section is to analyse the relationship between the cost incurred by a commuter and 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 the SR problem on random instances with $n \in \{30, 40, 50, 60, 70, 80, 90, 100\}$ and $|\mathcal{D}| \in \{10\%, 50\%, 80\%\}$, with 20 repetitions for each pair of parameters, 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 agent a_i in a coalition C with $|C| > 1$, we define its *normalised cost* $\overline{c}_i$ as

$$\overline{c}_i = \frac{|x_i| - \min^C_{|x|}}{\max^C_{|x|} - \min^C_{|x|}},$$

where $\min^C_{|x|}$ and $\max^C_{|x|}$ are the minimum and the maximum values of $|x_i|$ among the members of C. Note that we consider the absolute value of x_i since in our model, costs are represented by negative values for x_i.

- For any agent a_i in a coalition C with $|C| > 1$, we define its *normalised degree* $\overline{d}_i$ as

$$\overline{d}_i = \frac{deg(a_i) - \min^C_d}{\max^C_d - \min^C_d},$$

where $deg(a_i)$ represents the degree of a_i in the social network, and $\min^C_d$ and $\max^C_d$ are the minimum and the maximum degrees among the members of C.

When the denominator of $\overline{c}_i$ is 0, i.e, when $\max^C_{|x|} = \min^C_{|x|}$, 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 the agents with respect to payments would not be appropriate for determining their overall power or benefits derived from participation in the SR setting. 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. Finally, note that agents in singletons have been excluded from this analysis, as they do not have to split their coalitional value.

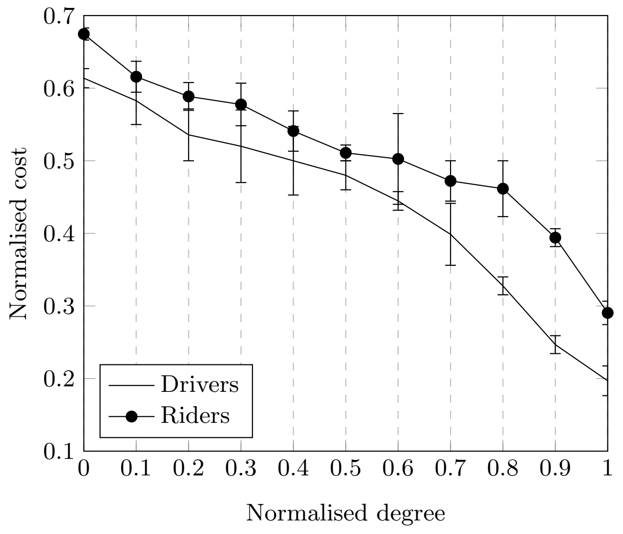

Figure 4: Normalised cost w.r.t. normalised degree.

In Figure 4 we report the average and the standard error of the mean for the normalised cost w.r.t. 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.

These findings allow us to discuss two interesting properties of our approach. On the one hand, our payment scheme *incentivises* commuters to be drivers, which is of utmost importance in SR scenarios with selfish rational agents. In fact, a rational user would not accept to join a ridesharing system as a driver (which requires her to deviate from the shortest path between its source and its destination, and to share its car with other people) without being appropriately rewarded for its service.

On the other hand, the social aspect of our ridesharing model introduces an additional degree of freedom that can be exploited by commuters to influence the distribution of payments. In fact, the more social connections they have, the more coalitions they can potentially join, the more bargaining power they obtain. Moreover, note that this fact is important for commuters that cannot choose to be drivers (e.g., they do not own a car), as it enables them to improve their payments by establishing new social relationships.

5. CONCLUSIONS

We perform recommendations for SR scenarios, developing a novel algorithm, PRF (Paying for Rides with Friends), to compute fair payments for large-scale systems. PRF can be used in conjunction with Bistaffa et al.'s approach to provide recommendations for the optimal groups to form, and also to define fair, stable costs that commuters should pay as a compensation for their rides. PRF avoids any redundancy by exploiting the structure of the social graph. Moreover, we parallelise PRF achieving a speed-up close to the maximum theoretical one. Our tests, based on real dataset for both spatial and social data, shows that our approach is up to 84 times faster than the state-of-the-art, allowing us to compute fair payments for 2000 agents in less than an hour. 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 focus on studying the quality guarantees of payments when using approximate solutions (which have not been studied yet) and will aim to extend our parallel approach to different multi-threading models (e.g., General-Purpose Graphics Processing Units).

6. ACKNOWLEDGMENTS

This work was carried out as part of the ORCHID project funded by EPSRC (EP/I011587/1). We also acknowledge funding from the EPSRC-funded International Centre for Infrastructure Futures (ICIF) (EP/K012347/1).

References

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

[2] F. Bistaffa, A. Farinelli, J. Cerquides, J. Rodríguez-Aguilar, and S. D. Ramchurn. "Anytime Coalition Structure Generation on Synergy Graphs". In: *AAMAS.* 2014, pp. 13–20.

[3] F. Bistaffa, A. Farinelli, and S. D. Ramchurn. "Sharing Rides with Friends: a Coalition Formation Algorithm for Ridesharing". In: *AAAI.* 2015, pp. 608–614.

[4] G. Chalkiadakis, E. Markakis, and N. R. Jennings. "Coalitional stability in structured environments". In: *AAMAS.* 2012, pp. 779–786.

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

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

[7] G. Greco, E. Malizia, L. Palopoli, and F. Scarcello. "On the complexity of the core over coalition structures". In: *IJCAI.* 2011, pp. 216–221.

[8] E. Kamar and E. Horvitz. "Collaboration and Shared Plans in the Open World: Studies of Ridesharing". In: *IJCAI.* 2009, pp. 187–194.

[9] A. Kleiner, B. Nebel, and V. A. Ziparo. "A mechanism for dynamic ride sharing based on parallel auctions". In: *IJCAI.* Vol. 11. 2011, pp. 266–272.

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

[11] H. Kwak, C. Lee, H. Park, and S. Moon. "What is Twitter, a Social Network or a News Media?" In: *WWW.* 2010, pp. 591–600.

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

[13] R. B. Myerson. *Game Theory: Analysis of Conflict.* Harvard University Press, 1997.

[14] A. Papadimitriou, P. Symeonidis, and Y. Manolopoulos. "Geo-social recommendations". In: *RecSys 2011 Workshop on PeMA.* 2011.

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

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

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

[18] T. Voice, S. Ramchurn, and N. Jennings. "On coalition formation with sparse synergies". In: *AAMAS.* 2012, pp. 223–230.

[19] X. Yang, H. Steck, Y. Guo, and Y. Liu. "On top-k recommendation using social networks". In: *RecSys.* 2012, pp. 67–74.

[20] N. J. Yuan, Y. Zheng, L. Zhang, and X. Xie. "T-finder: A recommender system for finding passengers and vacant taxis". In: *TKDE* 25.10 (2013), pp. 2390–2403.

[21] X. Zheng, X. Liang, and K. Xu. "Where to Wait for a Taxi?" In: *ACM SIGKDD International Workshop on Urban Computing.* 2012, pp. 149–156.

[22] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. "Mining interesting locations and travel sequences from GPS trajectories". In: *WWW.* 2009, pp. 791–800.