Web Intelligence and Agent Systems: An International Journal 12 (2014) 347–358
DOI 10.3233/WIA-140304
IOS Press

Multi-agent orienteering problem with time-dependent capacity constraints

Cen Chen, Shih-Fen Cheng * and Hoong Chuin Lau
School of Information Systems, Singapore Management University, Singapore
E-mail: {cenchen.2012,sfcheng,hclau}@smu.edu.sg

Abstract. In this paper, we formulate and study the Multi-agent Orienteering Problem with Time-dependent Capacity Constraints (MOPTCC). MOPTCC is similar to the classical orienteering problem at the single-agent level: given a limited time budget, an agent travels around the network and collects rewards by visiting different nodes, with the objective of maximizing the sum of his collected rewards. The most important feature we introduce in MOPTCC is the inclusion of multiple competing and interacting agents. All agents in MOPTCC are assumed to be self-interested, and they interact with each other when arrive at the same nodes simultaneously. As all nodes are capacitated, if a particular node receives more agents than its capacity, all agents at that node will be made to wait and agents suffer collectively as a result (in terms of extra time needed for queueing). Due to the decentralized nature of the problem, MOPTCC cannot be solved in a centralized manner; instead, we need to seek out equilibrium solutions; and if this is not possible, at least approximated equilibrium solutions. The major contribution of this paper is the formulation of the problem, and our first attempt in identifying an efficient and effective equilibrium-seeking procedure for MOPTCC.

Keywords: Multi-agent orienteering problem, sampled fictitious play

1. Introduction

The orienteering problem is a generalization of the traveling salesman problem that can be used to model a wide variety of real-world problems like tour planning, route planning for facility inspection and patrolling of security forces in a network. A large number of variants and corresponding algorithms for solving them have been introduced. The most common variants of the orienteering problems include: (1) The team orienteering problem, in which a group of centrally controlled agents are sent to collect rewards by visiting check points [2,5], (2) The orienteering problem with time windows, in which service time windows are specified for each node [12], and (3) the combination of the above two variants (the team orienteering problem with time windows) [18].

In this paper, we introduce a multi-agent version of the problem, which we believe to be the first of its kind. The application domain that motivates our research is the crowd control problem for a collection of interconnected service providers (real-world examples include the MICE[1] industry, amusement parks, and museums). Individual agents (visitors) in this environment aim to visit a sequence of selected service providers with the objective of maximizing their utilities obtained by receiving services, while observing their individual time budget limitations and service provider's time-dependent capacity constraints.

For the operators of such facilities, one important task they need to perform on a daily basis is to provide proper guidance to visitors, such that visitors can obtain as much values from selected service providers as possible. Such guidance may be passive, which is delivered via signboards or staff on the ground. With mobile devices becoming popular, we see the opportunity that such guidance can also be personalized and dynamic, in which case individual visitors may be guided by following instructions delivered in real time to their mobile devices. As the operator needs to watch closely

*Corresponding author.

[1]Meetings, incentives, conferences, and exhibitions.

1570-1263/14/$27.50

how queues build up at different service providers throughout the day, it needs to generate recommendations for individual visitors following their respective preferences on one hand, while satisfying queue capacity constraints for different service providers throughout the day on the other.

More precisely, we introduce the Multi-agent Orienteering Problem with Time-dependent Capacity Constraints (MOPTCC). Our contributions are as follows:

1. We introduce and formulate this new problem. In our problem, the nodes are subject to time-dependent capacity constraints and time-dependent rewards. The rewards allow us to model individual visitors' preferences, while the capacity constraints enable the operator to manage and control crowds.
2. We solve the MOPTCC involving multiple *self-interested* agents. Since agents are maximizing their respective utilities (and not the global objective function in a standard optimization problem), the challenge is to seek what is known as an *equilibrium* solution rather than a centralized optimal solution.
3. We propose two solution approaches: (1) a centralized approach with integer linear program (ILP) that computes the exact global solution (which in most cases are not equilibrium solutions); and (2) a variant of the sampled fictitious play algorithm [15] that can efficiently identify equilibrium solutions. We focus on the second approach, and use a wide variety of computational experiments to demonstrate its effectiveness.

This paper proceeds as follows. After literature review in Section 2, an integer linear programming model of the MOPTCC is presented in Section 3. An efficient Nash equilibrium seeking algorithm based on sampled fictitious play is then proposed in Section 4. The experimental results are presented in Section 5. Finally, Section 6 concludes the paper.

2. Literature review

The *orienteering problem* (OP), originally defined by Tsiligirides [25], was motivated by scheduling a cross-country sport in which participants get rewards from visiting a predefined set of checkpoints. As a generalization of the Traveling Salesman problem, it is a notoriously challenging NP-hard problem that has long been studied since 1980s. Tsiligirides [25] presented an early survey of heuristic methods for OP; this was followed more recently by a survey of Vansteenwegen et al. [28] which provided formulations and solution approaches for the OP and its related variants. Essentially, our problem is a variant of OP which can be characterized as the team orienteering problem (TOP) with time windows (TOPTW). TOP is an extension of OP where the goal is to plan a set of routes for all the members of the team that maximizes the total rewards collected by the team within the time limit $T_{\max}$ [5]. TOP is well-studied, and many researchers have proposed either exact solution approach (e.g., Butt and Cavalier [4], Tang and Miller-Hooks [23], and Boussier et al. [2]) or heuristic approach (e.g., Chao et al. [5] and Tang and Miller-Hooks [23]). Most recent research efforts on TOP have been on the development of efficient and effective heuristics, such as tabu-search-based heuristic by Archetti et al. [1], ant colony optimization approach by Ke et al. [13], guided local search approach by Vansteenwegen et al. [26], and greedy randomized adaptive search procedure by Souffriau et al. [22]. Compared to TOP, TOPTW received much less attention, and majority of the research efforts are solely on the development of heuristics, e.g., the ant colony optimization by Montemanni and Gambardella [18], the iterated local search by Vansteenwegen et al. [27], and the hybridized evolutionary local search algorithm by Labadie et al. [14].

In this paper, we depart from the classical setting of TOP and TOPTW, which is concerned with the route planning for a team of agents in a centralized fashion. Instead we treat the problem as a multi-agent planning problem where individual agents are self-interested and will scrutinize their given plans carefully. Therefore, instead of seeking for a *globally optimal plan*, we focus on identifying a Nash equilibrium where individual agents cannot improve their current utilities by deviation.

Formally speaking, this multi-agent TOP is modeled as a game, where players are agents, player's strategy space is the set containing all possible routes, and the payoff function is the mapping from a joint strategy (routes from all players) to a vector of *payoff values* for all players. If a particular joint strategy is infeasible (e.g., if queue lengths at some service providers violate the capacity constraints), all players will receive the value of $-\infty$.

For any normal-form game, the existence of mixed strategy Nash equilibrium is guaranteed, but pure strategy Nash equilibrium does not always exist (see for instance [19]). While the complexity of finding a mixed Nash equilibrium in an n-player game is still unknown, computing a mixed Nash equilibrium in a 2-player game is PPAD-complete [7]. For the case of pure strategy Nash equilibrium, determining its existence in a graphical game (a special case of normal-form game) is NP-complete [10]. From the computational perspective, it has been shown that finding pure strategy Nash equilibrium is only possible in fairly small games (e.g., even for 5-player, 5-strategy games, it may takes hours and sometimes days to solve). The classical approach for finding Nash equilibrium in a 2-player game is the pivot-based Lemke-Howson algorithm [16]. More recently, a mixed integer programming formulation is also proposed for solving 2-player normal-form games [21]. In cases where payoff matrix is large and complete characterization is computationally intractable (e.g., each payoff value can only be estimated by running multiple time-consuming simulations), the focus has been on computationally tractable approaches in approximately finding equilibria (e.g., see Wellman et al. [29] and Jordan et al. [11]) without complete payoff matrix.

Finally, most previous works on OP are static in that the network parameters (such as travel times and node delays) remain constant over time. This is another major feature that distinguishes our work from the literature. In the problem we are about to describe, we allow queueing times at service providers to be dependent on the number of visitors showing up in the same time period. As such, the time required to receive service from a particular provider would depend on not just this agent's strategy, but also other agents' strategies.

3. Problem formulation

3.1. A motivating example

To understand why a game-theoretic framework would be necessary for MOPTCC, let's start with a simple example to illustrate the inadequacy of global optimum in a multi-agent environment. Consider the following two-agent, two-provider problem: Let n_0 be the designated starting and ending nodes, and let n_1 and n_2 represent two providers. We assume that both agents start their trips from n_0 at time 1 and they have to return to n_0 again on or before time 5. The travel time between any two nodes is 1. For each provider, it can only serve one agent at a time, and its service time is 1 time unit. If multiple agents request service from the same provider simultaneously, we assume that agents are to be served one after another according to their ID numbers. We further assume that the queueing policy is set to allow provider n_2 to handle at most one agent at a time (i.e., no queueing allowed) and no limit for provider n_1. Finally, we assume that agents collect their utilities when they finish their services at the provider.

Agents' time-dependent utilities for receiving services from the two providers are listed in Table 1.

Table 1

Agents' time-dependent utilities at different providers

t	Agent 1		Agent 2	
	n_1	n_2	n_1	n_2
1	1	2	3	1
2	1	2	3	1
3	2	3	3	1
4	2	3	5	2
5	2	3	5	2

Table 2

Payoff matrix for all joint decisions

		Agent 2	
		n_1	n_2
Agent 1	n_1	2, 5	2, 1
	n_2	3, 3	$-\infty, -\infty$

Based on the above setup, we can see that due to the time limit constraint, each agent can choose at most one provider before returning to n_0. The outcomes resulting from agents' joint decisions are summarized as the payoff matrix in Table 2. Note that for the joint decision (n_1, n_1), both agents would arrive at n_1 in time 2, with Agent 1 receiving service first, followed by Agent 2. Agent 1 would leave n_1 in time 3 and receives the value of 2 (according to Table 1); for Agent 2, he begins his service in time 3, and leaves n_1 in time 4, receiving the value of 5. For (n_1, n_2) and (n_2, n_1), since there are no conflict, the corresponding payoff values can be directly found in row ($t = 3$) of Table 1. (n_2, n_2) is infeasible since provider n_2 can handle at most 1 agent (i.e., no queueing is allowed for n_2).

From Table 2, we can see that the global optimum is (n_1, n_1), with combined value 7. However, this solution is not stable, as Agent 1 would be better off by deviating from n_1 to n_2. In fact, the joint strategy (n_2, n_1), with combined value 6, is a Nash equilibrium.

This is a classical demonstration where selfish agents would deviate from the globally optimal solution and opt for Nash equilibria with lower combined payoff. In this instance, there are two major factors contributing to such phenomenon: (1) agents have their respective time-dependent payoffs, and (2) providers handle agents sequentially, and individual providers might be given different limits on queue lengths.

3.2. *Centralized formulation*

Although global optimum is not very meaningful for MOPTCC, as argued earlier, we should still present the centralized formulation first. This centralized formula can serve as the comparison baseline, and it is also an important subproblem to be solved repetitively when we introduce the game-theoretic formulation.

The MOPTCC is derived from the classical single-agent orienteering problem, where n providers (nodes) are assumed to be fully connected and can be represented as a complete graph with t_{ij} denoting travel time from i to j. We assume that there are m independent agents, and let s_{ik}^t be the utility agent k receives when visiting node i in time t. The service time at provider d is a constant v_d, and the number of agents allowed to simultaneously visit provider d is capped at $Q_d^{\max}$. The horizon of the problem is set to be T time periods. Without loss of generality, we assume that each agent k starts his trip at node 1 in time T_1^k and should end his trip at node n before time T_n^k (nodes 1 and n can either be real or dummy nodes).

3.2.1. *Integer linear programming formulation*

With these notations and assumptions, we can then formulate a centralized optimization problem as an integer linear program.

Let x_{ijk}^t be the binary decision variable which is set to 1 if agent k leaves node i at time t and goes to node j, and 0 otherwise. Let Q_d^t denotes the number of agents visiting node d at time t. We first define the objective function to maximize the combined utility received by all agents:

$$\max \sum_{t=1}^{T} \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} s_{ik}^t \, x_{ijk}^t. \tag{1}$$

The first set of constraints ensure that for each agent k, he starts at node 1 and ends at node n:

$$\sum_{t=1}^{T} \sum_{j=1}^{n} x_{1jk}^t = \sum_{t=1}^{T} \sum_{i=1}^{n} x_{ink}^t = 1, \quad \forall k. \tag{2}$$

Equation (3) guarantees that flows are conserved at all nodes except the origin (node 1) and the destination (node n):

$$\sum_{t=1}^{T} \sum_{i=1}^{n} x_{idk}^t = \sum_{t=1}^{T} \sum_{j=1}^{n} x_{djk}^t, \quad \forall k,\ d \neq 1 \text{ or } n. \tag{3}$$

As in all classical OP, we assume that for each agent k, each node d is visited at most once:

$$\sum_{t=1}^{T} \sum_{j=1}^{n} x_{djk}^t \leq 1. \tag{4}$$

Equation (5) defines the queue length for each node d at time t. For simplicity, we assume that service rate is 1 at all nodes. Thus Q_d^t equals the queue length from time $t-1$ plus the inflow and minus the outflow of current time t for this node. Equation (6) ensures that the queue length Q_d^t should not exceed its corresponding threshold $Q_d^{\max}$ at all times. Both Eqs (5) and (6) are defined for all d and t.

$$Q_d^t = Q_d^{t-1} + \sum_{k=1}^{m} \left(\sum_{i=1}^{n} x_{idk}^{t-t_{id}} - \sum_{j=1}^{n} x_{djk}^t \right), \tag{5}$$

$$Q_d^t \leq Q_d^{\max}. \tag{6}$$

In Eq. (7), the arrival and departure times for agent k at node d are constrained by taking into account all potential delays such as service time, queue length at arrival, and travel time.

$$\sum_{t=1}^{T} \sum_{i=1}^{n} \big(t + t_{id} + Q_d^{t+t_{id}} + v_d\big) x_{idk}^t = \sum_{t=1}^{T} t \left(\sum_{j=1}^{n} x_{djk}^t \right). \tag{7}$$

Finally, Eqs (8) and (9) ensure that for each agent k, the schedule starts at T_1^k and ends before T_n^k.

$$\sum_{t=1}^{T} \sum_{j=1}^{n} t \cdot x_{1jk}^t = T_1^k, \tag{8}$$

$$\sum_{t=1}^{T} \sum_{j=1}^{n} t \cdot x_{njk}^t \leq T_n^k. \tag{9}$$

By expanding Eq. (5) recursively, it can be rewritten as Eq. (10).

$$Q_d^t = Q_d^1 + \sum_{k=1}^{m}\sum_{i=1}^{n}\sum_{s=2-t_{id}}^{t-t_{id}} x_{idk}^s - \sum_{k=1}^{m}\sum_{j=1}^{n}\sum_{s=2}^{t} x_{djk}^s, \quad \forall d, t. \tag{10}$$

When substituting Q_d^t in Eq. (7) with (10), there are non-linear terms. To linearize these non-linear constraints, we introduce α_{ijdlk}^{st} to represent $x_{jdl}^s \cdot x_{idk}^t$ and β_{ijdlk}^{st} to replace $x_{djl}^s \cdot x_{idk}^t$. After the transformation, Eq. (7) is replaced by Eqs (11)–(13).

$$\sum_{t=1}^{T}\sum_{j=1}^{n} t \cdot x_{djk}^t = \sum_{t=1}^{T}\sum_{i=1}^{n}(t + v_d + t_{id})x_{idk}^t + \sum_{t=1}^{T}\sum_{i=1}^{n}\sum_{l=1}^{m}\sum_{j=1}^{n}\sum_{s=2-t_{id}}^{t} \alpha_{ijdlk}^{st} - \sum_{t=1}^{T}\sum_{i=1}^{n}\sum_{l=1}^{m}\sum_{j=1}^{n}\sum_{s=2}^{t+t_{id}} \beta_{ijdlk}^{st}, \quad \forall d, k, \tag{11}$$

$$\begin{cases} \alpha_{ijdlk}^{st} \le x_{jdl}^s, \\ \alpha_{ijdlk}^{st} \le x_{idk}^t, \\ \alpha_{ijdlk}^{st} \ge x_{jdl}^s + x_{idk}^t - 1, \end{cases} \quad \forall i, d, j, k, l, s, t, \tag{12}$$

$$\begin{cases} \beta_{ijdlk}^{st} \le x_{djl}^s, \\ \beta_{ijdlk}^{st} \le x_{idk}^t, \\ \beta_{ijdlk}^{st} \ge x_{djl}^s + x_{idk}^t - 1, \end{cases} \quad \forall i, d, j, k, l, s, t. \tag{13}$$

After linearization, the above mathematical programming model can then be solved by using standard integer linear programming solver such as CPLEX. However, such formulation does not scale well and only very small instance can be solved [6]. In this work, our focus is to solve MOPTCC as a game, and the above formulation can be revised to solve a single-agent version of the problem. Before introducing the equilibrium-seeking algorithm, we will first model the problem using game-theoretic framework.

3.3. A game-theoretic formulation for MOPTCC

The MOPTCC game is defined as the tuple $\Gamma = \langle \mathcal{K}, \mathcal{S}, u \rangle$, where $\mathcal{K} = \{1, \ldots, m\}$ is the set of all players (agents), $\mathcal{S} = \mathcal{S}_1 \times \ldots \times \mathcal{S}_m$ is the joint strategy space, and $u : \mathcal{S} \to R^m$ is the payoff function. When not considering $\mathcal{S}_{-k}$, player k's strategy space is defined as:

$$\mathcal{S}_k = \big\{ (s_k^1, \ldots, s_k^n) | s_k^i \in \{1, \ldots, n\}, \quad \forall i; \\ s_k^1 = 1; \quad \exists d,\ s_k^d = n, \\ \text{for } 1 < i < d,\ s_k^i \notin \{s_k^1, \ldots, s_k^{i-1}\}, \\ \text{for } d < i \le n,\ s_k^i = 0 \big\}. \tag{14}$$

In other words, a player's strategy must always begin with node 1, end with node n, never repeat, and if the visit sequence is shorter than n, all visits after node n must be no-op, which is denoted as 0.

Given any joint strategy profile s, we can straightforwardly compute the corresponding Q_d^t for all pairs of (t, d). We say that a joint strategy $s \in \mathcal{S}$ produces *feasible* joint orienteering plan if the resulting Q_d^t does not exceed $Q_d^{\max}$ for all pairs of (t, d). The utility function u is only defined for strategies that produce feasible joint orienteering plans. If a joint strategy s produces infeasible plan, we defined $u_k(s)$ to be $-\infty$ for all players.

As the MOPTCC game is defined as a normal-form game, all joint strategies can be played. However, due to the feasibility condition defined above, only a small fraction of strategies should ever be considered. As such, the next challenge we have to address would be to devise an algorithm that can effectively and efficiently identify *feasible* equilibria of the MOPTCC game.

4. A fictitious play-based algorithm for finding pure Nash equilibria

As reviewed in Section 2, even for a very simple game that contains only two players, it can be very computationally challenging to compute equilibrium solutions. As the number of players and the size of strategy space increase, the complexity of the equilibrium-seeking would increase quickly. In the MOPTCC game, the critical challenge is the size of the strategy space. In fact, as in the usual orienteering problem, the size of the strategy space grows exponentially as the number of destinations increases (e.g., for problem with n destinations, the size of the strategy space is in the order of n!). Because of this, most traditional enumeration-based equilibrium seeking techniques (e.g., the well-known Lemke-Howson [16] algorithm) will not be effective, and we have to find alternatives.

One computational approach that shows promise in dealing with the strategy space explosion is the fictitious play algorithm, which is originally proposed by Brown [3] and later adopted by researchers in operations research and computer science in dealing with either centralized or decentralized planning problems. Without going into technical details, we can view fictitious play algorithm as a way for players to learn about how to anticipate other players' responses, so that proper strategy can be selected. The strength of the fictitious play is its simplicity, and it's known that if potential function can be defined for the game in interest, the fictitious play algorithm will converge [17]. Important classes of games that possess such property include games with identical interests (the team game) and a wide variety of congestion games.

Unfortunately, the original fictitious play algorithm has a number of undesirable properties, both theoretically and computationally. First, the equilibrium that the fictitious play algorithm could converge to (if the convergence is possible) is in mixed form, since the convergence results are all established on the belief distribution (which is probabilistic in nature). Second, in each iteration of the fictitious play algorithm, all players need to compute their *best responses* against the current belief distribution, which potentially may contain a big chunk of the original joint strategy space. This implies that the evaluation of best responses (which is based on expected) might be exponential as well.

To address the second issue, Lambert III et al. [15] have introduced the idea of sampling to the evaluation of best responses: instead of evaluating against all possible combinations from the history in the belief distribution, a small number (in most cases, only *one* sample is needed) of joint strategies will be sampled, and the best response will be computed against these samples. Lambert III et al. [15] proved that this *sampled fictitious play* (SFP) will converge in belief to equilibrium for games of identical interests. They then use this result to solve large-scale unconstrained discrete optimization problems as games using SFP.

We will adopt the similar sampling idea in our first attempt to solve the MOPTCC game. However, as we are looking to generate recommendations for agents with heterogeneous preferences (represented in the form of payoff function), we will focus on finding *pure strategy equilibria* instead. As the existence of pure strategy equilibrium is not guaranteed in general, and we cannot prove it analytically due to the complexity of the formulation, we will use a wide variety of computational experiments to explore whether this is something that is achievable for the MOPTCC game.

Algorithm 1 Sampled fictitious play algorithm for MOPTCC games.

1: **Input:** $(\Gamma, k_{\max})$
2: **Output:** $\mathbf{B}_{\mathrm{NE}}$
3: $\mathbf{B} \leftarrow$ INITIALSOLUTIONS()
4: $\mathbf{H} \leftarrow$ UPDATEHISTORY($\{\}, \mathbf{B}$)
5: $k \leftarrow 1$
6: **while** $k <= k_{\max}$ **do**
7:   $\mathbf{D} \leftarrow$ SAMPLE($\mathbf{H}, k$)
8:   **for** each agent i **do**
9:     $\mathbf{Q}_{-i} \leftarrow$ AGGREGATEQUEUES($\mathbf{D}_{-i}$)
10:     $(\mathbf{B}_i, \delta_i) \leftarrow$ BESTRESPONSE($\Gamma, \mathbf{Q}_{-i}$)
11:   **end for**
12:   $\mathbf{H} \leftarrow$ UPDATEHISTORY($\mathbf{H}, \mathbf{B}$)
13:   **if** $\max_i \delta_i = 0$ **then**
14:     $\mathbf{B}_{\mathrm{NE}} \leftarrow$ APPEND($\mathbf{B}_{\mathrm{NE}}, \mathbf{B}$)
15:     $k \leftarrow k + 1$
16:   **end if**
17: **end while**
18: **Return:** $\mathbf{B}_{\mathrm{NE}}$

4.1. Sampled fictitious play algorithm

In Algorithm 1, we define a variant of the SFP algorithm used in solving MOPTCC game. The major new features we implement are: (1) the handling of *infeasible* samples, which based on our earlier definition refer to joint strategies that would result in over-capacitated destination; and (2) focus on identifying pure strategy equilibrium when executing the SFP algorithm.

Algorithm 1 is a simplified skeleton that hides most implementation complexity. To start the algorithm, we first randomly generate joint strategy that is feasible by calling INITIALSOLUTIONS() in line 3. This initial solution is then used to initiate the history (i.e., the belief distribution). The iteration then begins, in which a feasible joint strategy is to be sampled at the beginning of the iteration in line 7. The tighter the capacity constraint, the more difficult it is in sampling a feasible joint strategy. However, as the initial joint strategy is feasible, we can always find such sample. With a feasible sample joint strategy, we then solve each agent's best response problem as a mathematical program, which is to be defined later. Note that when the best response is computed in line 10, congestions at all destinations (Q_{-i}) are determined by other players' joint strategy (D_{-i}) in line 9. When computing the best response, another information we get is the individual deviation δ_i, which refers to the improvement made by choosing the best response. If δ_i is 0, it implies that player i cannot benefit from unilaterally deviating from the sampled strategy. If $\max_i \delta_i$ is 0, no player can benefit from their deviations, and D is a pure strategy equilibrium. Whenever we find such solution, we will store it in the output set (note that in practice we will store all relevant information such as utility and congestion besides just the equilibrium strategy).

In the next two subsections, we will explain how we generate random initial solutions, how do we compute best responses using mathematical programming approach.

4.2. *Generating feasible initial solution*

The initial solutions are generated using a simple greedy approach in INITIALSOLUTIONS(). We detail the used greedy approach as follows:

1. Initialize the congestion $\{Q_d^t\}$ to be 0 for all (t, d) pairs.
2. Choose any player k who doesn't have an itinerary yet.
3. For this agent, randomly choose one destination at a time, assuming that the current congestion is $\{Q_d^t\}$. We use rejection-base sampling: choosing all unvisited destinations with equal chance, and if the chosen destination d is at its capacity at the estimated arrival time t, another destination will be drawn.
4. For each agent, we would artificially reduce its time budget by 50%. This is to approximately factor in the potential impact of this agent's decision on other agents' increased wait time. Based on our computational experience, this damping factor can significantly improve the likelihood of us getting feasible decisions. Depending on the number of players and the problem parameters, different damping factors might be appropriate. Our computational study on 5-agent instances is summarized in Fig. 5 in the Appendix.
5. After we have exhausted player k's time budget, we fix player k's itinerary and update $\{Q_d^t\}$.
6. If the set of free players is not empty, go to Step 2 and repeat.

We first generate initial solutions without artificially reducing player's time budget, but we soon find out that for problems with tight capacity constraints, initial solutions generated with all players spending most of their time budgets will result in joint solutions that are almost impossible to improve upon. In these cases, time budget reduction in Step 4 is shown to be very effective in improving the efficiency of the algorithm (intuitively speaking, Step 4 allows us to reserve buffer time in the schedule generated).

4.3. *Computing best responses*

An agent's best response in the MOPTCC game can be computed using an ILP model very similar to the one introduced in Section 3.2, Eqs (1)–(9). There are two major differences:

- The k index which represents different agent identities can be dropped. E.g., the decision variable will become only x_{ij}^t.
- All other agents' chosen strategies, which are taken from the sampled joint strategy, will collectively decide the background queue length. We define Q_{dt}^{input} to be the background queue length built up by other agents at node d in time t.

Most constraints will stay the same except Eqs (5) and (7). These two sets of constraints will be rewritten as:

$$Q_d^t = Q_{dt}^{\text{input}} + \sum_{i=1}^{n} x_{id}^{t-t_{id}}, \quad \forall d, t, \tag{15}$$

$$\sum_{t=1}^{T} \sum_{i=1}^{n} \big(t + t_{id} + Q_{d,t+t_{id}}^{\text{input}} + v_d\big) x_{id}^t = \sum_{t=1}^{T} t \sum_{j=1}^{n} x_{dj}^t, \quad \forall d. \tag{16}$$

Since Q_{dt}^{input} is provided as problem data, the constraint is already linear and requires no linearization. Together with the fact that index k is dropped, the problem becomes much more tractable, and can be solved reasonably fast in our computational study.

5. Computational experiments

In this section, we evaluate how effective our SFP variant is in finding pure strategy equilibrium. All the instances used in this section are generated randomly using our MOPTCC instance generator. The run times reported below are measured on machines running 3.16 GHz Intel Xeon CPU X5460 with 16 GB RAM.

Table 3

Results for equilibria found on *per-instance bases*. For smaller instances (2-agent and 5-agent cases), each instance is solved by executing SFP algorithm for *20 iterations*. For larger instances (8-agent), we execute SFP algorithm for *50 iterations*. Note that all mentions of *equilibria* refer to *pure strategy equilibria*

(m, n, T)	$(2, 10, 10)$	$(2, 10, 10)$	$(5, 10, 10)$	$(5, 10, 10)$	$(8, 10, 10)$	$(8, 10, 10)$
Instance Type	tight	loose	tight	loose	tight	loose
Total instances	25	25	25	25	25	25
Success rate of finding equilibria	100%	100%	100%	100%	100%	100%
Avg. # of equilibria found	267.88	256.08	79.12	66.92	44.96	20.35
Avg. # of distinct equilibria found	17.88	17.76	61.4	52.64	31.68	14.00
Best equilibrium / Best feasible	100%	100%	99.94%	99.97%	98.71%	98.98%
Avg. computational time(s)	114.24	172.43	337.08	439.14	3466.80	9913.22

5.1. Data generation

Numerical instances in our computational study are characterized by the tuple: $(m, n, T, \textit{type})$, where m, n, T denote the number of agents, the number of nodes, and the number of time periods respectively. The final parameter, *type*, refers to the tightness of the instance, which can be either *loose* or *tight*. The tightness of the instance affects how node capacities ($Q_d^{\max}$) are drawn. For *loose* instances, capacities are drawn from discrete uniform distribution between $(p \cdot m)$ and m; for *tight* instances, capacities are drawn from discrete uniform distribution between 1 and $(p \cdot m)$. In both instances, p is set to a constant between 0 and 1. In all our experiments, p is set to 0.5.

The utility value for agent k to visit node i in time t, s_{ik}^t, is assumed to be uniformly distributed between 1 to 5. The only exceptions are the start and end nodes, s_{1k}^t and s_{nk}^t, whose values are both set to be 5. Finally, all travel times (t_{ij} between nodes i and j) and service times (v_d for node d) are set to 1 for simplicity.

For our computational experiments presented in this section, we generate six categories of instances, with parameters $m \in \{2, 5, 8\}$, $n = 10$, $T = 10$, and $type \in \{loose, tight\}$. 25 random instances are generated for each category. To avoid being trapped in unpromising joint strategy subspace and increase the likelihood of finding pure strategy equilibrium, each instance is independently solved for 20 times, each time with a randomly generated initial solution (following steps introduced in Section 4.2).

5.2. Numerical results

For smaller instances (2-agent and 5-agent cases), each instance is solved by executing SFP algorithm for 20 iterations. For larger instances (8-agent), we execute SFP algorithm for 50 iterations so that better solution might be found. The performance of our SFP variant in finding pure strategy equilibria is summarized in Table 3. As we can see from Table 3, SFP can identify large amount of high-quality pure strategy equilibria for smaller instances (2-agent and 5-agent cases) very quickly. When the problem expands to have 8 agents, we can see that it's drastically more difficult to find pure strategy equilibria (measured by both the computational time and the number of pure strategy equilibria found). Another interesting finding is that *loose* instances are much more difficult to solve than *tight* instances for all numbers of agents, probably because of greater degree of freedom we have (number of feasible joint strategies would be much larger when the capacity constraints are loose, and agents would have a more difficult time in producing coordinated actions as a result). We also compute the ratio between the best pure strategy equilibrium and the best feasible solution found for each instance. We can see that the ratio is reasonably high for all instances. This is an encouraging computational result, as it's well known that the adhering to Nash equilibria might cause significant drop in social welfare (e.g., see Roughgarden and Tardos [20]'s work on quantifying the *price of anarchy* in the routing domain, i.e., the sacrifice one needs to endure for implementing Nash equilibrium solution).

Another interesting way to visualize the growing of computational complexity in finding pure strategy equilibrium is to plot the histogram of agents' maximal deviations in all iterations. Intuitively speaking, if we have lots of cases with 0 deviation, it implies that it's easy to identify pure strategy equilibrium (when the maximal deviation is 0 among all players in any iteration, it implies that a pure strategy equilibrium has been found, since no agent can benefit from deviating unilaterally). Not surprisingly, with the number of agents increasing, the performance of the algorithm takes a hit and the frequency of zero deviation should

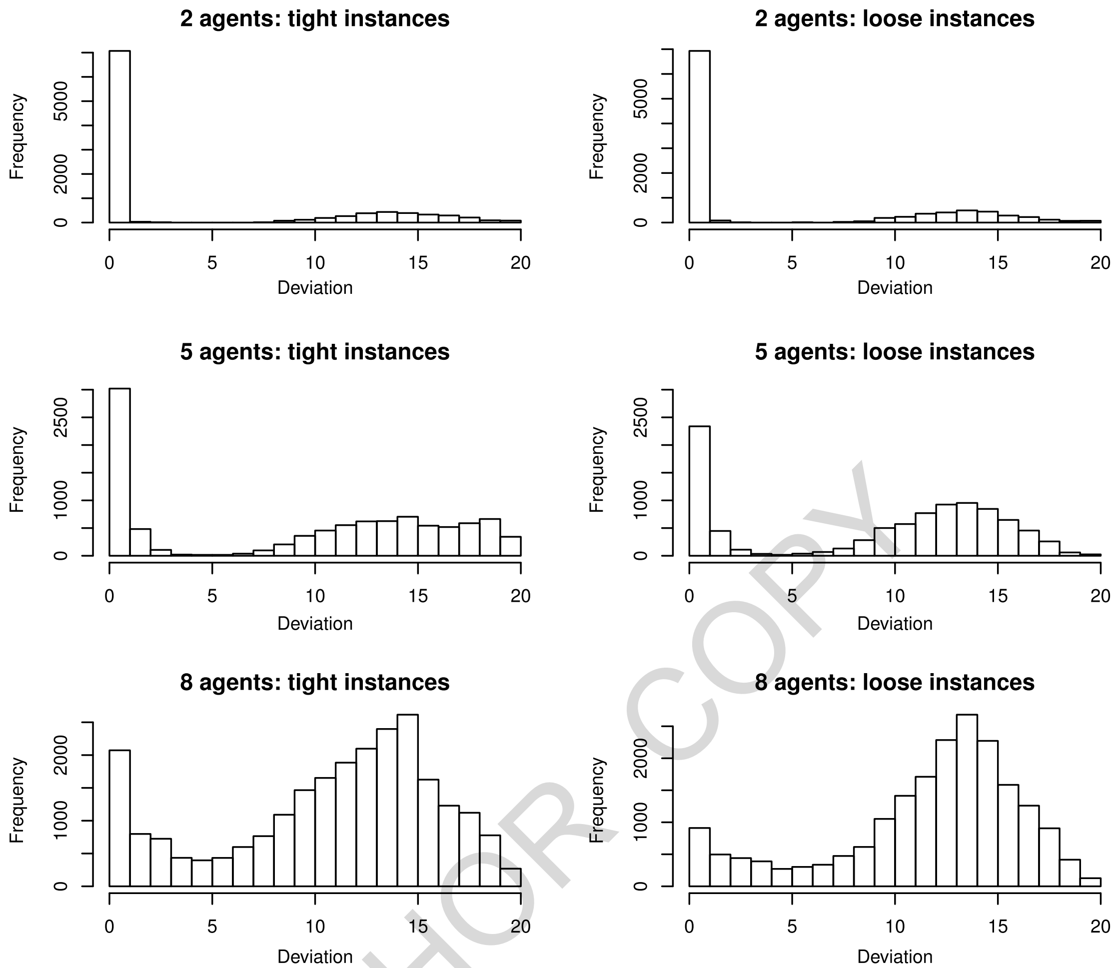

Fig. 1. Max deviations of 2-agent, 5-agent, and 8-agent instances.

decrease. The plots in Fig. 1 confirm our speculation. Besides steady decrease in zero-deviation cases, the distribution of deviations also gradually shifts to the right hand side, indicating greater difficulty in reaching coordinated outcomes. The instances with *loose* capacity constraints also consistently have *fatter* tails to the right, indicating that it's more difficult to identify equilibrium in general for *loose* instances.

In terms of utility value improvement, the SFP algorithm progresses very quickly. In Fig. 2 we can observe the average progress of the SFP algorithm over the iteration, with error bars at $+1$ and -1 standard deviation. As illustrated in Fig. 2, we can see that most of the progress is made at the early iterations, after which the algorithm settles down quickly, with stable values and very low standard deviations. This execution pattern is also consistent with prior research in using SFP algorithm for large-scale discrete optimization

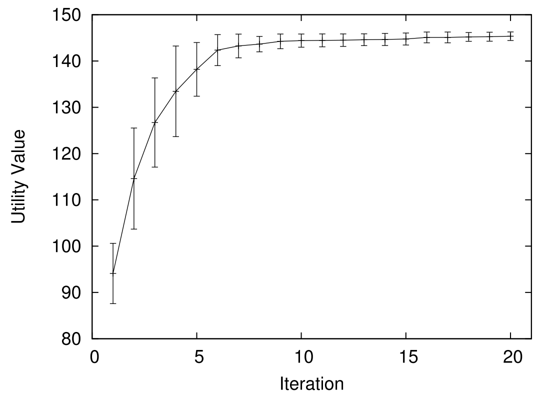

Fig. 2. The average progress of SFP algorithm over iterations for a sample instance with random restarts (the error bar is one standard deviation over all random restarts of this instance).

(e.g., see [9]). Do note that Fig. 2 is relative; when we have larger number of agents, it's expected that more

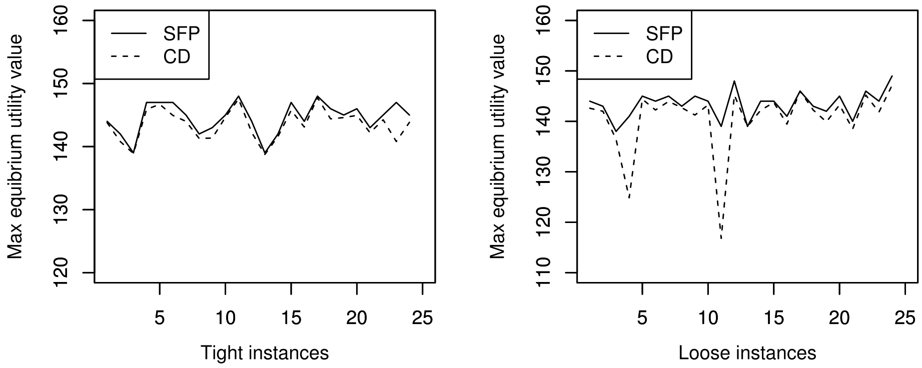

Fig. 3. Maximal equilibrium utility value for 5-agent instances: 25 tight instances (left) and 25 loose instances (right). X-axis denotes individual instances. Y-axis represents the maximum utility value of the equilibrium solutions discovered for that instance.

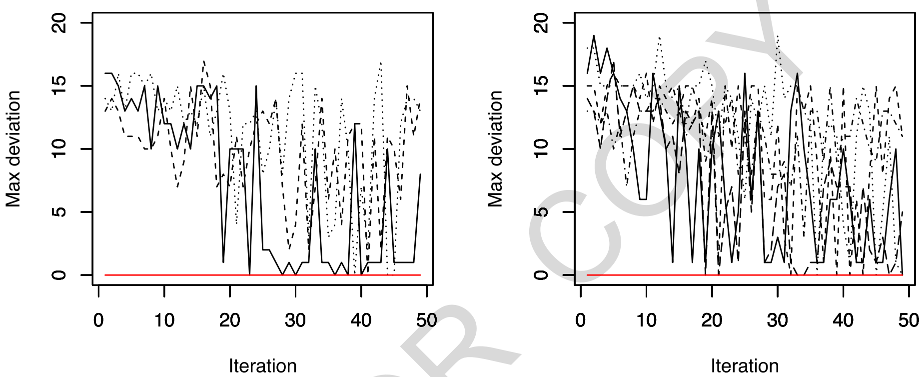

Fig. 4. Maximal deviations for selected 8-agent (left) and 10-agent (right) instances with random restarts. Each line represents a different random restart. X-axis denotes iterations, Y-axis denotes the maximal deviation $\max_i \delta_i$. A deviation δ_i for player i can be viewed as the utility improvement made by choosing the best response. If $\max_i \delta_i$ is 0, a pure strategy equilibrium is found, i.e., no player can benefit from unilateral deviation. Setting: $(m, n, T) = (m, 10, 10)$.

iterations would be necessary, however, the pattern improvement would resemble what is illustrated here.

5.3. Comparison against baseline algorithm: Coordinate descent

To understand whether the SFP algorithm we use for MOPTCC indeed has its merits, we compare it against a popular baseline algorithm called the Coordinate Descent (CD) algorithm (similar CD algorithm has also been compared to the SFP algorithm in other domains, such as the coordinated traffic signal control [8]). The CD algorithm is a simple yet effective algorithm for solving large-scale discrete optimization (despite its simplicity, it's shown to work well in practice and in theory [24]). In the MOPTCC domain, it executes as follows: the CD algorithm would start from any player, for whom a best response against the current solution is computed and this best response will then be used to replace this player's current strategy. The CD algorithm then move on to the next player and repeat the above procedures. The CD algorithm would terminate either after no further improvement can be made after we have iterated through all agents or the computational time is up.

The comparison between the CD algorithm and the SFP algorithm is conducted for our 5-agent instances, with results plotted in Fig. 3. From the figure we can see that the SFP algorithm is at least as good as the CD algorithm, and for some instances the SFP algorithm managed to greatly outperform the CD algorithm.

Finally, we extend our experiments to 8-agent and 10-agent instances, running 50 iterations of SFP algorithm. As shown in Fig. 4, the number of iterations needed to find a equilibrium increases with the number of agents. In the 2-agent and 5-agent cases, SFP is able to find equilibrium within 20 iterations. For the 8-agent and 10 agent cases, equilibrium is sel-

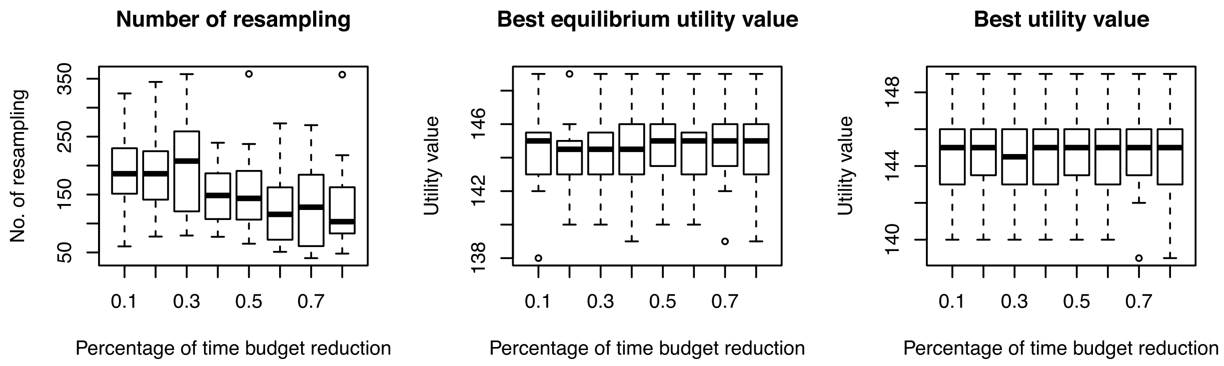

Fig. 5. Comparison for different time budget discount ratios for 5-agent instances: $(m, n, T) = (5, 10, 10)$.

dom reached within the first 20 iterations and most equilibria are found by the second half of 50 iterations.

6. Conclusions and future work

In this paper, we introduce a new variant of the OP, referred as the Multi-agent Orienteering Problem with Time-Dependent Capacity Constraints (MOPTCC). It can be used as the starting point for modeling many combinatorial optimization problems which involve time-dependent capacity constraints; e.g., it can be applied to crowd management in leisure settings, where controlling queue lengths for various attractions is of vital concern to the operator.

To this end, we first propose a centralized model using integer linear programming formulation. Due to the distributed nature of the problem, we reformulate it in the game-theoretic framework, and we propose to use the sampled fictitious play algorithm (SFP) which is shown to be computational efficient in identifying pure strategy equilibrium. By introducing *rejection-base sampling* in the fictitious play iteration, we are able to deal with computational intractability and also the *feasibility* requirement we impose on the MOPTCC game, which is to require all capacity constraints be observed at all times.

Our initial computational experiments show great promises, as we are able to find pure strategy equilibrium in all the randomly generated instances for 2-agent, 5-agent, and 8-agent MOPTCC games. Our immediate next step is in trying to theoretically prove the existence of pure strategy equilibrium for MOPTCC games. We are also interested in further improving the computational approach so that it can scale to even larger instances.

Acknowledgment

This research is supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office.

Appendix

The impact of using different time budget discount ratios is illustrated in Fig. 5. We can clearly see that more aggressive time budget reduction directly leads to fewer number of re-samplings. The best utility value from discovered equilibrium solutions also improves a bit with higher time budget discount ratios.

References

[1] C. Archetti, A. Hertz, and M. Grazia Speranza, Metaheuristics for the team orienteering problem, *Journal of Heuristics* **13**(1) (2007), 49–76.
[2] S. Boussier, D. Feillet and M. Gendreau, An exact algorithm for team orienteering problems, *4OR: A Quarterly Journal of Operations Research* **5**(3) (2007), 211–230.
[3] G.W. Brown, Iterative solution of games by fictitious play, *Activity Analysis of Production and Allocation* **13**(1) (1951), 374–376.
[4] S.E. Butt and T.M. Cavalier, A heuristic for the multiple tour maximum collection problem, *Computers & Operations Research* **21**(1) (1994), 101–111.
[5] I.-M. Chao, B.L. Golden and E.A. Wasil, The team orienteering problem, *European Journal of Operational Research* **88**(3) (1996), 464–474.
[6] C. Chen, S.-F. Cheng and H.C. Lau, The multi-agent orienteering problem, in: *Tenth Metaheuristics International Conference*, Singapore, August 2013.
[7] X. Chen and X. Deng, Settling the complexity of two-player nash equilibrium, in: *Foundations of Computer Science (FOCS)*, 2006.

[8] S.-F. Cheng, M.A. Epelman and R.L. Smith, CoSIGN: A parallel algorithm for coordinated traffic signal control, *IEEE Transactions on Intelligent Transportation Systems* **7**(4) (2006), 551–564.

[9] A. Ghate, S.-F. Cheng, S. Baumert, D. Reaume, D. Sharma and R.L. Smith, Sampled fictitious play for multi-action stochastic dynamic programs, *IIE Transactions* **46**(7) (2014), 742–756.

[10] G. Gottlob, G. Greco and F. Scarcello, Pure nash equilibria: Hard and easy games, *Journal of Artificial Intelligence Research* (2003), 215–230.

[11] P.R. Jordan, Y. Vorobeychik and M.P. Wellman, Searching for approximate equilibria in empirical games, in: *Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, 2008, pp. 1063–1070.

[12] M.G. Kantor and M.B. Rosenwein, The orienteering problem with time windows, *The Journal of the Operational Research Society* **43**(6) (1992), 629–635.

[13] L. Ke, C. Archetti and Z. Feng, Ants can solve the team orienteering problem, *Computers & Industrial Engineering* **54**(3) (2008), 648–665.

[14] N. Labadie, J. Melechovskỳ and R. Wolfler Calvo, Hybridized evolutionary local search algorithm for the team orienteering problem with time windows, *Journal of Heuristics* **17**(6) (2011), 729–753.

[15] T.J. Lambert III, M.A. Epelman and R.L. Smith, A fictitious play approach to large-scale optimization, *Operations Research* **53**(3) (2005), 477–489.

[16] C.E. Lemke and J.T. Howson, Jr., Equilibrium points of bimatrix games, *Journal of the Society for Industrial & Applied Mathematics* **12**(2) (1964), 413–423.

[17] D. Monderer and L.S. Shapley, Potential games, *Games and Economic Behavior* **14**(1) (1996), 124–143.

[18] R. Montemanni and L. Gambardella, Ant colony system for team orienteering problems with time windows, *Foundations of Computing and Decision Sciences* **34**(4) (2009), 287–306.

[19] C. Papadimitriou, Algorithms, games, and the internet, in: *Symposim on Theory of Computing (STOC)*, 2001, pp. 749–753.

[20] T. Roughgarden and E. Tardos, How bad is selfish routing? *Journal of ACM* **49**(2) (2002), 236–259.

[21] T. Sandholm, A. Gilpin and V. Conitzer, Mixed-integer programming methods for finding nash equilibria, in: *National Conf. on Artificial Intelligence (AAAI))*, 2005.

[22] W. Souffriau, P. Vansteenwegen, G. Vanden Berghe and D. Van Oudheusden, A path relinking approach for the team orienteering problem, *Computers & Operations Research* **37**(11) (2010), 1853–1859.

[23] H. Tang and E. Miller-Hooks, A tabu search heuristic for the team orienteering problem, *Computers & Operations Research* **32**(6) (2005), 1379–1407.

[24] P. Tseng, Convergence of a block coordinate descent method for nondifferentiable minimization, *Journal of Optimization Theory and Applications* **109**(3) (2001), 475–494.

[25] T. Tsiligirides, Heuristic methods applied to orienteering, *The Journal of the Operational Research Society* **35**(9) (1984), 797–809.

[26] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe and D. Van Oudheusden, A guided local search metaheuristic for the team orienteering problem, *European Journal of Operational Research* **196**(1) (2009), 118–127.

[27] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe and D. Van Oudheusden, Iterated local search for the team orienteering problem with time windows, *Computers & Operations Research* **36**(12) (2009), 3281–3290.

[28] P. Vansteenwegen, W. Souffriau and D. Van Oudheusden, The orienteering problem: A survey, *European Journal of Operational Research* **209**(1) (2011), 1–10, ISSN 0377-2217.

[29] M.P. Wellman, D.M. Reeves, K.M. Lochner, S.-F. Cheng and R. Suri, Approximate strategic reasoning through hierarchical reduction of large symmetric games, in: *Twentieth National Conference on Artificial Intelligence*, 2005, pp. 502–508.