DIRECT: A Scalable Approach for Route Guidance in Selfish Orienteering Problems

Pradeep Varakantham, Hala Mostafa, Na Fu, Hoong Chuin Lau
School of Information Systems
Singapore Management University
{pradeepv,halamostafa,nafu,hclau}@smu.edu.sg

ABSTRACT

We address the problem of crowd congestion at venues like theme parks, museums and world expos by providing route guidance to multiple selfish users (with budget constraints) moving through the venue simultaneously. To represent these settings, we introduce the Selfish Orienteering Problem (SeOP) that combines two well studied problems from literature, namely Orienteering Problem (OP) and Selfish Routing (SR). OP is a single agent routing problem where the goal is to minimize latency (or maximize reward) in traversing a subset of nodes while respecting budget constraints. SR is a game between selfish agents looking for minimum latency routes from source to destination along edges of a network available to all agents. Thus, SeOP is a multi-agent planning problem where agents have selfish interests and individual budget constraints. As with Selfish Routing, we employ Nash Equilibrium as the solution concept in solving SeOP. A direct mathematical program formulation to find a Nash equilibrium in SeOP cannot scale because the number of constraints is quadratic in the number of paths, which itself is an exponential quantity. To address scalability issues, we make two key contributions. First, we provide a compact non-pairwise formulation with linear number of constraints in the number of paths to enforce the equilibrium condition. Second, we introduce DIRECT, an incremental and iterative master-slave decomposition approach to compute an approximate equilibrium solution. Similar to existing flow based approaches, DIRECT is scale invariant in the number of agents. We also provide a theoretical discussion of our approximation quality and present extensive empirical results on synthetic and real-world graphs demonstrating the scalability of combining DIRECT with our non-pairwise formulation.

Categories and Subject Descriptors

J.m [**Computing Application**]: Game Theory

General Terms

Leisure and Entertainment, Decision Support

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

Keywords

Game Theory; Selfish Routing; Orienteering; Network Congestion Games, Constraint Generation

1. INTRODUCTION

A fundamental problem in the management of large-scale transportation and communication networks is that of routing traffic to optimize network performance, given that performance degrades when network users act greedily and myopically. Game-theoretic equilibrium strategies are shown to provide good performance [10], achieving more than 75% of optimal social welfare in environments where the latency on an edge is linear in the number of agents using it (also referred to as *flow*). We therefore focus on the computation of game-theoretic equilibrium strategies.

We consider the routing of selfish individuals each of whom is interested in a set of nodes and is operating under a resource or time budget (minimum number of attractions to visit or maximum time to spend). This problem arises in crowd coordination for theme parks, museums and world expos. For the operators of such facilities, one important concern is the real-time coordination of visitors to provide customized route guidance that reduces visitor wait times while respecting their budgets. With the prevalence of smart devices, this coordination is realizable by delivering guidance to visitors' mobile phones.

The computation of routes (sequence through all or a subset of nodes of interest) in such scenarios has traditionally been studied in the context of the Orienteering Problem (OP). OP is a generalization of the Traveling Salesman Problem where the goal is to find a route that maximizes utility (or minimizes latency) while respecting budget constraints on resources or time [13]. OP has been extensively studied by the Operations Research and AI communities (a survey of OP and its variants is given in [15]). While efficient algorithms for OP exist, the model and its solution approaches are inapplicable in our setting because they do not address or represent the effects of interactions among selfish agents traversing common edges whose latencies depend on the number of agents using them, thereby linking the decision problems of individual agents.

On the other hand, there are related multi agent models such as Network Congestion Games (NCG) and Selfish Routing (SR) [10, 9, 3] which consider the interactions among selfish agents on a network. The goal for each agent is to compute the minimum latency path from a designated start node to a designated end node, where the latency of an edge depends on the number of agents traversing it. How-

ever, neither NCG nor SR considers each agent's resource or time budget constraints that limit the number of nodes this agent can visit.

To fill this gap, we introduce the Selfish Orienteering Problem (SeOP) where we represent the problem of finding a path (sequence of nodes) for each selfish agent to satisfy their resource and budget constraints as a game. An assignment of paths to agents results in certain flows on the edges of the graph. As in NCG and SR, the goal is to compute the set of paths that give rise to flows describing an (approximate) Nash equilibrium where no agent can achieve lower latency/higher reward by deviating from the prescribed (possibly non-deterministic) path strategies.

To that end, we first present a formulation for finding a Nash equilibrium in SeOP as an optimization problem. However, the resulting Mixed Integer Linear Program has limited scalability because it enforces the equilibrium condition using a number of constraints quadratic in the number of available paths. To address this, we make the following contributions:

1. We provide a compact optimization formulation for SeOP that enforces the equilibrium condition using a number of constraints linear in the size of the path set.
2. We present DIRECT, an iterative master-slave decomposition approach that computes an approximate equilibrium in a scalable manner by incrementally growing the set of paths that constitute the support set of mixed strategies for each agent type. Since SeOP is a generalization of SR problems, our approach can also be used to solve SR problems efficiently.

We provide a theoretical discussion of our approximation quality and present extensive empirical results demonstrating the scalability of combining DIRECT with our compact formulation. We report results from both synthetic and real-world graphs.

2. SELFISH ORIENTEERING PROBLEM

In the Selfish Orienteering Problem (SeOP), as with Selfish Routing (SR), each agent optimizes their individual objective (maximizing utility or minimizing latency). Orienteering problems of different agents are interdependent since the reward or latency of an edge is a function of the number of agents using that edge.

Formally, SeOP is a tuple: $< \Gamma, \mathbf{N}, \mathcal{V}, \mathcal{E}, \mathbf{L}, \mathbf{D}, \mathbf{B} >$. Γ is the set of agent types and $\mathbf{N} = \{\mathcal{N}^\tau\}_{\tau\in\Gamma}$ is a vector representing the number of agents of each type τ. $\langle \mathcal{V}, \mathcal{E}\rangle$ represents the underlying network graph with vertices/nodes $\mathcal{V}$ and edges $\mathcal{E}$. $\mathbf{L} = \{\mathcal{L}^\tau_{ij}\}_{\tau\in\Gamma, ij\in\mathcal{E}}$ is a vector of latency functions, one per agent type and edge. The latency experienced by type τ on path p is the sum of latencies experienced on each edge in the path:

$$\mathcal{L}^\tau_p = \sum_{ij\in p} \mathcal{L}^\tau_{ij}(x_{ij})$$

where each edge latency is a function of the flow x_{ij} on it. In the context of guidance at theme parks, the latency function for an agent type τ on an edge captures the tolerance of τ for congestion on this edge; the higher the latency the lower the tolerance. $\mathbf{D} = \{\mathcal{D}^\tau\}_{\tau\in\Gamma}$ is a vector where each $\mathcal{D}^\tau$ is the set of nodes that agents of type τ want to visit and $\mathcal{B}^\tau \in \mathbf{B}$ is the budget for each agent type. For example, $\mathcal{B}^\tau$ can be the minimum number of nodes in $\mathcal{D}^\tau$ that type τ wants to visit.

The goal is for each agent of type τ to find one or more paths that have minimum total latency from a designated start node to a designated end node and cover at least $\mathcal{B}^\tau$ nodes in $\mathcal{D}^\tau$. SeOP differs from network congestion games in multiple respects, namely budget on nodes to be visited and the path should cover a subset or all nodes in the list of interested nodes.

A solution of SeOP is an assignment of flows of every agent type to edges and, consequently, paths. We focus on computing equilibrium solutions, i.e. the calculated agent flows are such that each agent is prescribed a path with minimum latency and thus has no incentive to deviate from this path given the paths of other agents.

EXAMPLE 1. *To give a concrete example, we present SeOP in the context of the problem of providing route guidance to visitors at a theme park:*

- *Visitors to the theme park are the agents and examples of agent types, Γ, are adults, families with kids, families with senior citizens, etc.*
- *The nodes, $\mathcal{V}$, are theme park attractions.*
- *Latencies, $\boldsymbol{L}$, represent the wait times on edges*[1].
- $\boldsymbol{B}$ *is the resource (maximum number of nodes in the final path) or time (total time available to visit attractions) budget for each visitor type.*
- $\boldsymbol{D}$ *is the set of interesting attractions for each visitor type. For example, a family with children prefers child rides and moderately thrill rides.*

The goal is to compute a sequence of nodes (selected from $\mathcal{D}^\tau$) for each agent type τ such that no agent of any type has incentive to deviate from the prescribed sequence, i.e. the prescribed sequence has minimum latency in the context of sequences prescribed to other types.

3. SEOP AS AN OPTIMIZATION PROBLEM

To formulate SeOP as an optimization problem, we use the following observation, due to Wardrop [16], on equilibrium flows in traffic routing.

OBSERVATION 1. *At equilibrium, if a path p has positive flow, then no other path can have a strictly lower latency than the latency on path p [16] .*

3.1 Pairwise Formulation of SeOP

Based on Observation 1, we propose the optimization formulation for calculating equilibrium flows for SeOPs[2] given in Table 1. x^τ_p and x^τ_{ij} are the flows of agents of type τ on path p and edge ij, respectively. x_{ij} is the total flow of all agent types on edge ij. Note that if we generate the path set $\mathcal{P}^\tau$ for each agent type τ to only contain paths that meet the budget $\mathcal{B}^\tau$, we do not have to include explicit budget constraints.

Since our goal is to compute any equilibrium solution, our optimization problem is essentially a feasibility problem, with an option to include an objective function that

[1]At a theme park wait times are at attractions. While we consider wait times or latencies as being associated with edges, our model and approach can easily be modified to account for wait times at nodes as shown in Section 7.2.

[2]This is an extension of the SR formulation [10] to account for budget constraints and agent types.

Table 1: Pairwise SeOP formulation

$$\min \quad \mathbf{0} \qquad \text{subject to}$$

$$x_{ij} = \sum_{\tau} x_{ij}^{\tau} \quad \forall (i,j) \in \mathcal{E} \tag{1}$$

$$\forall \tau \in \Gamma;$$

$$x_{ij}^{\tau} = \sum_{p \in \mathcal{P}^{\tau} | ij \in p} x_p^{\tau} \quad \forall (i,j) \in \mathcal{E} \tag{2}$$

$$\sum_j x_{0j}^{\tau} = \sum_j x_{j|\mathcal{V}|}^{\tau} = \mathcal{N}^{\tau} \tag{3}$$

$$\sum_i x_{ij}^{\tau} = \sum_k x_{jk}^{\tau} \quad \forall j \in \mathcal{V} \tag{4}$$

$$\forall p \in \mathcal{P}^{\tau} \textbf{ if } x_p^{\tau} > 0 :$$

$$\sum_{ij \in p} \mathcal{L}_{ij}^{\tau}(x_{ij}) \leq \sum_{kl \in p'} \mathcal{L}_{kl}^{\tau}(x_{kl}) \quad \forall p' \in \mathcal{P}^{\tau} \tag{5}$$

$$x_p^{\tau}, x_{ij}^{\tau}, x_{ij} \geq 0 \tag{6}$$

Table 2: Non-pairwise SeOP formulation

$$\min \quad \mathbf{0} \qquad \text{subject to}$$

$$\forall \tau \in \Gamma, p \in \mathcal{P}^{\tau}$$

flow conservation and identity constraints

$$y_p^{\tau} \cdot M \geq x_p^{\tau} \tag{7}$$

$$y_p^{\tau} \leq x_p^{\tau} \cdot M \tag{8}$$

$$\sum_{ij \in p} \mathcal{L}_{ij}^{\tau}(x_{ij}) \geq d^{\tau} \tag{9}$$

$$\sum_{ij \in p} \mathcal{L}_{ij}^{\tau}(x_{ij}) - d^{\tau} \leq u_p^{\tau} \cdot M \tag{10}$$

$$y_p^{\tau} \leq w_p^{\tau} \cdot M \tag{11}$$

$$u_p^{\tau} + w_p^{\tau} \leq 1 \tag{12}$$

$$x_{ij}^{\tau}, x_{ij}, x_p^{\tau}, d^{\tau} \geq 0$$

$$y_p^{\tau}, u_p^{\tau}, w_p^{\tau} \in \{0, 1\}$$

measures the quality of an equilibrium solution (e.g. the social welfare of a solution). In Table 1, constraints 1 and 2 are identity constraints to ensure that agent flows on edges and paths are consistent. They specify that the flow on an edge is the sum of flows on path that include this edge and define edge flow as the sum of agents of all types that traverse this edge.

Constraints 3 and 4 enforce flow conservation for each type:

- Total flow out of the start node 0 equals total flow into the end node $|\mathcal{V}|$ equals the number of agents of this type.
- Total flow into a node equals total flow out of it.

Constraint 5 implements Observation 1 by enforcing a pairwise relation between the latencies of every pair of paths. It states that the latency of a path with positive flow should be no larger than the latency of any other path. Although constraint 5 is a logical constraint, it can easily be represented as a set of linear constraints (similar to the complementarity constraint in the next subsection).

3.2 Non-pairwise Formulation of SeOP

The formulation in Table 1 suffers from limited scalability due to $\mathcal{O}(|\mathcal{P}|^2)$ latency equilibrium constraints (5). We now propose a formulation that does not perform pairwise comparisons of path latencies and uses only $\mathcal{O}(|\mathcal{P}|)$ constraints. d^{τ} represents the minimum latency achieved by type τ.

First, we rephrase the equilibrium condition: at equilibrium, if the minimum attainable latency for type τ on any path is d^{τ}, then each path with positive flow of type τ agents must have latency d^{τ}. Table 2 shows our **non-pairwise (NPW)** SeOP formulation as a feasibility problem.

For each path p, let the binary variable y_p^{τ} indicate whether there is positive flow of agents of type τ on path p:

$$y_p^{\tau} = 1 \quad \text{i ff } \; x_p^{\tau} > 0$$

The above relation can be enforced using the linear constraints (7) and (8), where M is a large positive number.

The constraint that each path with positive flow must have minimum latency can be seen as a complementarity constraint between the indicator y_p^{τ} on the one hand and the gap between a path's latency $\mathcal{L}_p^{\tau}$ and the minimal latency d^{τ} on the other hand. If there is a gap, the indicator must be 0 (path must have zero flow), and if the path has positive flow, the gap must be zero.

$$y_p^{\tau} \cdot (\mathcal{L}_p^{\tau} - d_p^{\tau}) = 0$$

A complementarity constraint $pq = 0$ can be expressed using the linear constraints

$$\begin{aligned} p &\leq wM \\ q &\leq uM \\ u + w &\leq 1 \end{aligned}$$

where M is again a large positive number and u and w are binary variables. Constraints (10), (11) and (12) apply the same linearization to enforce complementarity between y_p^{τ} and the gap $\mathcal{L}_p^{\tau} - d_p^{\tau}$. Constraint (9) ensures that d^{τ} is indeed the minimum latency.

4. ITERATIVE DECOMPOSITION

Even though NPW reduces the number of constraints from quadratic to linear in the number of paths, it is not enough to ensure scalability, since the number of paths is itself exponential in the number of nodes in $\mathcal{D}^{\tau}$ and the budget $\mathcal{B}^{\tau}$. To overcome this problem, we do not compute equilibria using the entire support set $\mathcal{P}^{\tau}$ of each type. Instead, we propose Deviation Incentive based Restricted dEComposiTion (DIRECT), an iterative master-slave decomposition approach where the master computes equilibria over a relatively small support set and the slave searches for better paths to be included in the support set in the next iteration. DIRECT computes ϵ-equilibrium solutions where ϵ is controlled using the input parameter $\delta \in [0, 1]$.

As shown in Table 3, we start by initializing a seed path set for each type, for example by randomly choosing a predefined number of paths that meet the budget, or incorporating domain knowledge to generate an initial set of promising

paths. In each iteration and for each type τ, DIRECT performs the following steps:

1. The master computes equilibrium flows for all agents except for a **fraction δ of an agent** of type τ. Note that δ is not a fraction of all agents of type τ. Instead, it is a fraction of only one agent of type τ.
2. The slave finds the minimum latency path for the remaining δ of an agent given the other flows computed by the master.
3. If the slave path has lower latency than the minimum latency for type τ calculated by the master, the agent has incentive to deviate and we need to add this path to the support.

The algorithm converges when no more paths can be added to the support. At convergence, we solve the master problem with the final support set of paths for all $\mathcal{N}$ agents. The resulting approximate equilibrium flows are such no agent has incentive to deviate for more than fraction δ of the time.

EXAMPLE 2. *Consider a problem with 10 agents all of one type and $\delta = 0.2$. Consider an iteration where $\hat{\mathcal{P}}^\tau = \{p_1, p_2\}$ and the equilibrium flows returned by the master in this iteration are 6.3 agents on p_1 and 3.5 agents on p_2 for a total of 9.8 $(= 10 - \delta)$ agents. Let the latency of each of these paths (which must be equal since they have positive flow) be 20. The slave then checks if δ (= 0.2) fraction of one agent has incentive to use a path other than p_1 and p_2. The slave computes the shortest non-divisible flow path for δ given the flows computed for the other 9.8 agents. If this new shortest path has latency 18, it is added to the support set and the next iteration's master uses the larger support to compute an equilibrium with lower latency than the current iteration.*

The example illustrates the following two key approximations we make in DIRECT:

1. Every call to the master computes equilibria using a *restricted* set of paths $\hat{\mathcal{P}}$ rather than the full set $\mathcal{P}$. As we show empirically, $|\hat{\mathcal{P}}|$ is typically orders of magnitude smaller than $|\mathcal{P}|$, leading to significant reductions in run times.

2. The second approximation allows the slave to check if an agent has an incentive to deviate from the equilibrium computed by the master in a scalable manner. This is done by restricting the slave to only use binary flow variables x_{ij}, thereby finding the minimum latency path among the set of deterministic paths only.

The master problem MASTER$(\hat{\mathcal{P}}, \delta, \tau)$ has the same form as Table 2 with two key differences; the path sets are $\hat{\mathcal{P}}^\tau$ rather than $\mathcal{P}^\tau$; and the flow conservation constraints (3) are modified such that the total flow for type τ is only $\mathcal{N}^\tau - \delta$. Table 4 provides the slave as an integer linear program. The objective function is the latency over all edges given current flows $\mathbf{x}^{\hat{\mathcal{P}}}$ from the master in addition to the planned flow for δ fraction of an agent of type τ. Constraints (13) and (14) ensure indivisible flow conservation for the δ fraction of an agent. Constraints (15) and (16) ensure that the returned path only visits desirable nodes and satisfies the budget.

Ideally, we want to know if there is incentive for any one agent to switch to any *set* of paths; i.e., search the space of non-deterministic strategies. But this would require the slave to allow fractional flows and the branching of flow at some nodes. Expressing the overall latency experienced by the agent would then require an indicator variable for each possible path and the slave problem would have the same complexity as the original master over the full set of paths. Therefore, instead of computing fractional flows for one agent, we compute non-divisible flow for a fraction δ of an agent, thereby avoiding the exponential enumeration of paths at the expense of introducing the approximation error discussed in the next section.

While we consider latency (reward) functions to be linear, as long as they are convex, our decomposition approach can use off-the-shelf solvers to obtain a solution. For non-convex and non-linear latency (or reward) functions, we can employ piecewise constant or piecewise linear approximations. Extending DIRECT to these function classes is a topic for future research.

Table 3: DIRECT (δ)

```
//Initialize subset of paths for each type
P̂ = {P̂^1, P̂^2, .., P̂^|Γ|}
converged ← false
while !converged do
  converged ← true
  for all τ ∈ Γ do
    < x^P̂, d^τ > ← MASTER(P̂, δ, τ)
    p^τ ← SLAVE(x^P̂, δ, τ)
    if p^τ ∉ P̂^τ and L^τ_{p^τ} < d^τ then
      P̂^τ ← P̂^τ ∪ {p^τ}
      converged ← false
return MASTER(P̂, 0, −1)
```

Table 4: Slave($\mathbf{x}^{\hat{\mathcal{P}}}, \delta, \tau$)

$$\min_{x \in \{0,1\}} \sum_{(ij)} x_{ij} \cdot \mathcal{L}^\tau_{ij}(\mathbf{x}^{\hat{\mathcal{P}}}_{ij} + \delta)$$

$$\sum_j x_{0j} = \sum_i x_{i|\mathcal{V}|} = 1 \tag{13}$$

$$\sum_i x_{ij} = \sum_k x_{jk} \qquad \forall j \tag{14}$$

$$\sum_i x^\tau_{ij} = 0 \quad \forall j \notin \mathcal{D}^\tau \tag{15}$$

$$\sum_{i,j \in \mathcal{D}^\tau} x^\tau_{ij} \geq \mathcal{B}^\tau \tag{16}$$

5. THEORETICAL RESULTS

In this section, we provide an *a posteriori* (after computing the solution) bound on the approximation error, ϵ for a given value of δ with our DIRECT algorithm.

PROPOSITION 1. *DIRECT(δ) converges to an ϵ equilibrium with*

$$\epsilon \leq \delta \cdot \max_\tau \left(\mathcal{L}^{\tau,\#}(\delta) - \mathcal{L}^{\tau,\#}(0) \right) \tag{17}$$

where $\mathcal{L}^{\tau,\#}(\delta)$ and $\mathcal{L}^{\tau,\#}(0)$ are the latency values obtained by solving SLAVE$(\boldsymbol{x}^{\hat{\mathcal{P}}}, \delta, \tau)$ *and* SLAVE$(\boldsymbol{x}^{\hat{\mathcal{P}}}, 0, \tau)$ *respectively at convergence. $\boldsymbol{x}^{\hat{\mathcal{P}}}$ is the flow obtained by solving the master at the last iteration for $\mathcal{N} - \delta$ agents, with δ representing a fraction of type τ agent.*

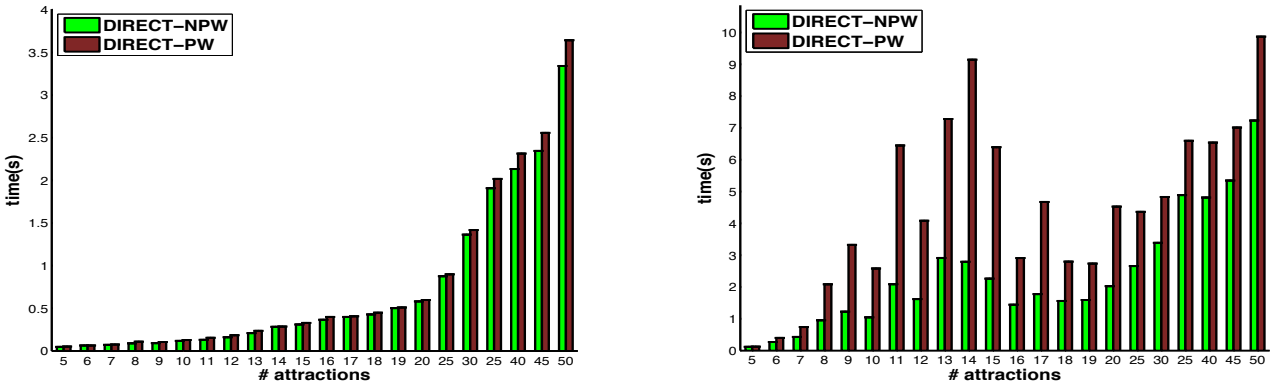


Figure 1: Effect of number of attractions $|\mathcal{V}|$ on runtime for 1 (L) and 2 (R) types. $\delta = 0.5$

Proof. There is an approximation error with DIRECT for an input parameter of δ, because potentially less than δ fraction of an agent can find a path that has lower latency value given the flows for the rest of the agents. A lower bound on the lowest latency value for a type τ agent is obtained by solving the shortest path problem given flow for $\mathcal{N} - \delta$ agents (equivalent to $\textsc{Slave}(\mathbf{x}^{\hat{\mathcal{P}}}, 0, \tau)$) and considering that δ agents do not contribute to the flow on paths. This is a lower bound because latency without adding δ fraction of the agent will only be smaller than the latency obtained by adding the δ fraction of the agent. Therefore for agent of type τ, the error would be given by:

$$\epsilon_\tau \leq \delta \cdot \left(\mathcal{L}^{\tau,\#}(\delta) - \mathcal{L}^{\tau,\#}(0) \right)$$

We multiply by the δ factor because $1 - \delta$ fraction of the agent remains the same in both components of the difference. Approximation error over all types is the maximum over the error for each individual type and hence the bound on error given in Equation 17. ■

6. EXPERIMENTAL RESULTS

We now compare 3 approaches for solving SeOP. **Full-NPW** formulates SeOP as the non-pairwise feasibility problem in Table 2 using the full path set and no decomposition[3]. **DIRECT-PW** uses master-slave decomposition with pairwise formulation in the master (Table 1). **DIRECT-NPW** combines decomposition with the compact non-pairwise formulation in Table 2[4].

We change 3 parameters: $\delta \in \{0.2, 0.5, 0.9\}$ changes the tightness of the approximate equilibrium while the number of attractions in the theme park $|\mathcal{V}|$ and the number of agent types $|\Gamma|$ change the size of the problem. For the different approaches, we compare 1) runtime; 2) latency of the (approximate) equilibria; and 3) number of paths in the support set.

6.1 Synthetic graphs

For each parameter combination, we generate 10 instances that vary in the latency functions of their edges, which have the form $ax + b$ where a, b are randomly drawn from the range 1 to 10. Unless stated otherwise, we set the budget $\mathcal{B}^\tau$ of each type to be $|\mathcal{V}|$, so the full set of possible paths is the set of permutations of all attractions. In DIRECT-PW and DIRECT-NPW, the set $\hat{\mathcal{P}}^\tau$ for each type is initialized with a single path traversing all nodes in order.

Impact of number of attractions $|\mathcal{V}|$

We first consider the effect of the number of attractions on runtime. Figure 1 shows run times for 1 and 2 agent types and $\delta = 0.5$. With 1 type, Full-NPW is faster only for very small problems (up to $|\mathcal{V}| = 8$) that do not need decomposition. For $|\mathcal{V}| = 9, 10$, Full-NPW takes 17 and 401 seconds, respectively and runs out of time (aborted after 1 hour) and/or memory while generating the full set of paths for larger problems. For 2 types, Full-NPW could only handle 5 and 6 attractions. For the rather large value of 0.5 for δ, both DIRECT approaches scale very well with $|\mathcal{V}|$, handling up to 50 attractions in a few seconds.

The figures clearly show the significant impact of the number of types on runtime. To understand the effects of $|\mathcal{V}|$ and $|\Gamma|$, note that DIRECT's gradual increase of the support size largely mitigates the main challenge posed by a large set of attractions, namely the exponential explosion of the set of paths. Increasing the number of types, however, increases the number of times the master and slave problems are solved in each iteration of DIRECT, in addition to each master problem being larger, since we have a set of variables per type per path. Moreover, a path introduced by 1 type's slave can give all other types incentives to deviate, upon which paths are added to their respective sets.

Impact of input parameter δ

Tables 5 and 6 show the effect of δ ($\delta = 0$ refers to Full-NPW) on the latency of the equilibrium paths $\mathcal{L}$ and the size of the support set $|\hat{\mathcal{P}}|$ (for DIRECT-*, this is also the number of iterations). As explained earlier, lower values of δ result in tighter approximations. Entries marked '-' indicate running out of time/memory. To have a meaningful comparison of these quantities, Table 5 is for $|\Gamma|$=1. DIRECT-PW and DIRECT-NPW have the same $\mathcal{L}$ and $|\hat{\mathcal{P}}|$, which confirms that the pairwise and non-pairwise constraints enforce the same semantics on flows. As expected, higher δ leads to fewer iterations (thus smaller path sets) because given

[3]We were only able to run **Full-PW** on very small problems and hence is not described in experimental results.

[4]Using a Mac with 3.4 GHz Intel Core i7, 32 GB 1600 MHz DDR3. All optimization problems were solved using IBM ILOG CPLEX Optimization Studio V12.4.

Table 5: Effect of δ on latency at equilibrium and support size. $|\Gamma| = 1$

	$\|\mathcal{V}\|$=8		$\|\mathcal{V}\|$=10		$\|\mathcal{V}\|$=20	
δ	$\mathcal{L}$	$\|\hat{\mathcal{P}}\|$	$\mathcal{L}$	$\|\hat{\mathcal{P}}\|$	$\mathcal{L}$	$\|\hat{\mathcal{P}}\|$
0	91.8	720	100.9	40,320	-	-
0.2	92	12.7	101.2	15.7	143.6	20.6
0.5	93.3	8.6	103.7	10	153.1	12
0.9	97.8	6.2	110.5	7.5	169.3	7.8

Table 6: Effect of δ on runtime (seconds). $|\Gamma| = 2$

	$\|\mathcal{V}\|$=10		$\|\mathcal{V}\|$=20	
δ	NPW	PW	NPW	PW
0	-	-	-	-
0.2	25	108	90	496
0.5	1.0	2.6	2.0	4.5
0.9	0.3	0.5	0.8	0.9

Table 7: Tightness of error bound

$\|\mathcal{V}\|$	Upper bound on ϵ (Proposition 1)	Observed ϵ	Final Latency
8	1.212	0.15	91.84
9	1.416	0.37	95.58
10	1.802	0.31	100.89

flows for other agents, an agent can have no incentive to deviate 0.9 of the time, but have incentive to deviate 0.2 of the time, so the slave can find a path to add to the support with $\delta = 0.2$, but not with $\delta = 0.9$. Table 5 also shows the increasing approximation gap between $\mathcal{L}$ of Full-NPW and the DIRECT approaches as δ and $|\mathcal{V}|$ increase[5]. Noting how little the support size changes as $|\mathcal{V}|$ increases explains the scalability of DIRECT demonstrated in Figure 1. The timing results in Table 6 show that DIRECT-PW takes much longer to achieve the same result as DIRECT-NPW, with the difference in runtime growing with $|\mathcal{V}|$.

To investigate how the quest for tighter approximation affects the scalability of DIRECT approaches, we set $\delta = 0.2$ to generate Figure 2(a) which strongly suggests that if the goal is better approximate equilibria, the pairwise formulation can get prohibitively expensive while the non-pairwise formulation continues to scale very well with $|\mathcal{V}|$.

Impact of number of types $|\Gamma|$

We next investigate the scalability of our approaches with the number of types $|\Gamma|$. Based on our observations from a real life theme park, patrons of different types (e.g., age, nationality) are interested in different subsets of attractions. We experiment with 16 attractions and for each type, generate a set of 8 random attraction that agents of this type want to visit, so $|\mathcal{D}|^\tau = \mathcal{B}^\tau = 8\ \forall\tau$. Figure 2(b) again exhibits the observation from Figure 1 that the number of types has a strong impact on runtime, and shows the limitation of DIRECT-PW as $|\Gamma|$ increases.

Tightness of error bound

We now demonstrate the tightness of the error bound on ϵ by providing the difference in observed value of ϵ and upper bound on ϵ computed using Propostion 1. Since, we can only run Full-NPW for small problems, we consider $|\Gamma| = 1$ and $|\mathcal{V}| = \{8, 9, 10\}$. As can be noted from Table 7, the predicted upper bound on ϵ using Proposition 1 is a tight bound and the difference in predicted and observed is less than 1.5% of the final latency (last column).

[5]We assume the all 3 approaches are converging to the same equilibrium.

Table 8: Real Network: Runtime and support size

δ	DIRECT-NPW Runtime	DIRECT-PW Runtime	$\|\hat{\mathcal{P}}\|$ /type
$\|\Gamma\| = 1$			
0.2	22.1354	22.6016	13.1
0.5	9.545	10.0806	8.4
0.9	4.1876	4.3091	5.9
$\|\Gamma\| = 2$			
0.2	53.4626	80.0936	9.4
0.5	21.329	26.081	6.85
0.9	10.7274	11.7274	5.1

Table 9: Real Network: Impact of initial path set. $|\Gamma| = 1$

Initial Path Set	Final Latency	Support Set Size
1	121.31	9
2	121.16	9
3	121.16	10
4	121.22	10
5	120.04	9
6	119.18	10
7	120.31	10
8	122.26	8
9	120.50	9
10	119.99	11

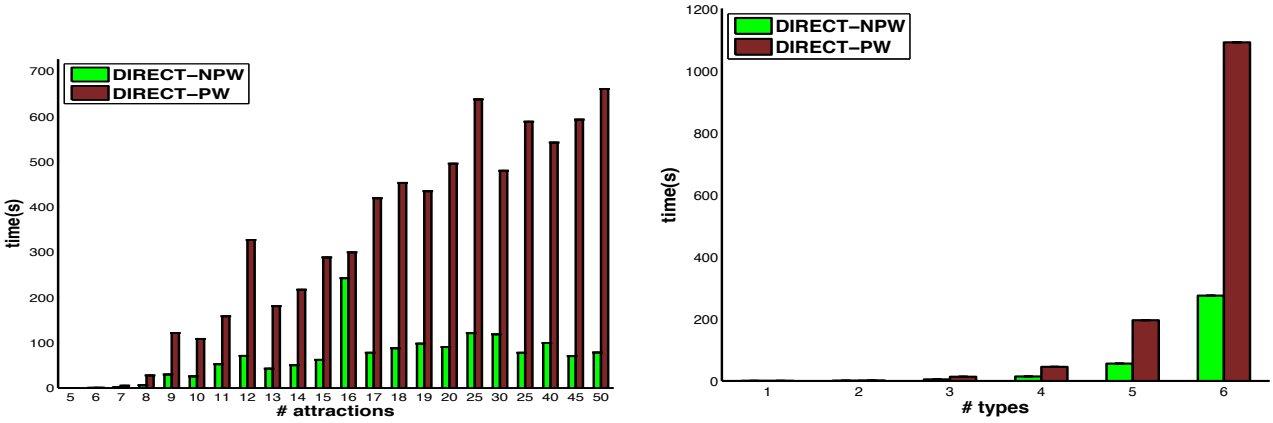
6.2 Graph of a real-world theme park

To demonstrate the applicability of our approach to real networks, we present results of running our formulations using graphs obtained from a major theme park in Singapore with 12 attractions/nodes. The graph is based on analysing movement dynamics of visitors among attractions over a one year period. Although this graph is not a complete graph, the comparison results between DIRECT-PW and DIRECT-NPW with respect to runtime are qualitatively the same as those obtained on complete synthetic graphs.

As can be noted from Table 8, the runtime results are similar to the ones obtained for synthetic graphs. We have also considered benchmarks for single and team orienteering problems [14] from literature. All these problems are fully connected graphs with 21 and 32 nodes. Since we show results on problems with up to 50 nodes that have randomly generated latency functions, we do not explicitly provide results on these benchmark instances.

Impact of initial path set

As explained earlier, DIRECT-* approaches rely on an initial path set that is incrementally expanded at each iteration. Here, we experiment with DIRECT-NPW using 10 different initial path sets for $\delta = 0.5$ on both the real theme park graph introduced earlier and a fully connected syn-



(a) Effect of number of attractions $|\mathcal{V}|$ on runtime for $|\Gamma|$=2 $\delta = 0.2$

(b) Effect of number of types $|\Gamma|$ on runtime for $|\mathcal{V}|$=16 $\delta = 0.5$ and $\mathcal{B}^\tau$=8

Figure 2: DIRECT-NPW vs DIRECT-PW

Table 10: Real Network: Impact of Initial Path Set. $|\Gamma| = 2$

Initial Path Set	Final Latencies	Support Set Sizes
1	195.82 ; 155.82	6 ; 6
2	192.79 ; 158.47	7 ; 8
3	195.66 ; 155.61	7 ; 7
4	199.48 ; 152.52	5 ; 7
5	194.59 ; 152.85	6 ; 8
6	196.04 ; 157.08	7 ; 7
7	198.86 ; 151.65	8 ; 8
8	197.70 ; 151.37	7 ; 7
9	193.69 ; 156.99	6 ; 8
10	197.35 ; 155.29	7 ; 7

Table 11: Synthetic Network: Impact of initial path set. $|\Gamma| = 1$

Initial Path Set	Final Latency	Support Set Size
1	106.70	11
2	106.86	10
3	107.00	11
4	107.02	11
5	106.71	11
6	107.70	11
7	108.14	9
8	106.98	11
9	107.36	10
10	107.25	10

thetic graph, both of which have 12 nodes. Tables 9, 10, 11 and 12 show the results on one and two types of agents. As can be seen, differences in the final latencies for both real and synthetic examples are minimal. The run-time results are also very similar for different initial path sets, which is to be expected, since the final support set size are similar.

It is interesting to note that we converge to a very similar equilibrium even though we start from vastly different path sets. While we cannot make general claims from this result, it potentially suggests a low price of anarchy for SeOPs. We leave the study of price of anarchy in SeOPs for future work.

7. DISCUSSION

In this section, we discuss variants and extensions of our model and approach.

7.1 Maximizing Reward with Time Budget

It should be noted that while we have described SeOP in terms of minimizing latency constrained by a resource budget on nodes to be visited, our optimization approach and its enhancement are directly applicable to the following variants of SeOP:

- The budget for each agent type τ is on the maximum time available to visit the nodes in $\mathcal{D}^\tau$.
- Each agent maximizes the reward accumulated (instead of minimizing latency) in visiting a subset of attractions/nodes within the given time budget.

Table 1 can be modified as follows when each agent maximizes a reward function $\mathcal{R}$ instead of minimizing latency $\mathcal{L}$ and budget constraints are on the maximum time available instead of the minimum number of desirable nodes to visit:

- We replace the latency constraint 5 with the reward constraint

$$\textbf{if } x_p > 0 : \sum_{ij \in p} \mathcal{R}^\tau_{ij}(x_{ij}) \geq \sum_{(kl) \in p'} \mathcal{R}^\tau_{kl}(x_{kl}), \forall \tau \in \Gamma, p, p' \in \mathcal{P}^\tau$$

  Similar to the latency functions, $\mathcal{R}^\tau_{ij}(x_{ij})$ is the reward obtained by agents of type τ for travelling on edge ij when the flow on this edge is x_{ij}. The path sets $\mathcal{P}^\tau$ are no longer computed from the budget, and include all possible paths for agents of type τ.
- Unlike the budget constraint on resources which is implicit in the set of available paths, the budget constraint on time requires the following new constraint:

$$\textbf{if } x_p > 0 : \sum_{ij \in p} \mathcal{L}^\tau_{ij}(x_{ij}) \leq \mathcal{B}^\tau$$

Table 12: Synthetic Network: Impact of initial path set. $|\Gamma| = 2$

Initial Path Set	Final Latencies	Support Set Sizes
1	135.00 ; 118.02	10 ; 10
2	135.32 ; 117.32	9 ; 10
3	134.97 ; 119.11	10 ; 10
4	133.5 ; 118.78	9 ; 10
5	136.88 ; 118.89	9 ; 9
6	134.34 ; 117.54	10 ; 11
7	133.46 ; 121.88	9 ; 8
8	135.53 ; 118.46	10 ; 9
9	134.6 ; 118.8	10 ; 10
10	133.73 ; 118.72	10 ; 10

7.2 Wait Times at Nodes Instead of Edges

To accurately represent the theme park setting, we need to consider wait times at nodes instead of at edges. For this, the key change is in the computation of latency for any path, p. Instead of summing over all edges in the path, we sum over latencies at all nodes in the path, i.e., $\sum_{i \in p} \mathcal{L}_i^\tau(\sum_j x_{ij})$. It should be noted that the latency for a node, i will be calculated based on the incoming flow into that node i.e., $\sum_j x_{ij}$ and not on x_{ij}. All other constraints in the optimization problems (Table 1 and Table 2) remain the same. In fact, considering latency at nodes may be an easier problem than considering latency on edges.

8. RELATED WORK

The Team Orienteering Problem (TOP) [1, 6, 14, 11] represents OP for teams of agents and is different from the work in this paper due to selfishness of our agents.

The optimization formulation for Selfish Routing by Roughgarden *et al.* [10] is the closest work to this paper. However, we address budget constraints (additional knapsack constraints) and use decomposition to handle larger SeOP instances. While decomposition approaches such as column generation for optimizing social welfare in traffic routing [12] exist, it is non-trivial to adapt these approaches to account for equilibrium constraints since a column refers to a path and two paths with overlapping edges are not completely independent of each other as these approaches assume.

Although the master-slave decomposition in DIRECT has an iterative best response (IBR) flavor, we cannot use IBR as a baseline since 1) at each iteration IBR needs to compute best response and as we have shown (with the performance of centralised optimization formulations) optimising even for a single agent over the entire path set becomes practically infeasible very quickly; and 2) iterating to compute pure strategies for individual agents is expensive (as we consider domains with thousands and thousands of agents), while iterating at the level of types can be easily shown to not converge to strategies that are stable at the individual agent level. DIRECT addresses both issues: 1) the master incrementally builds the support set and 2) the slave finds candidate paths by trying to route a fraction of an agent, guaranteeing at convergence that no agent prefers to deviate more than δ of the time.

Fictitious play [2] and other adaptive learning approaches can potentially be used to solve SeOPs. However, they also run into the same issues as IBR and furthermore, with multiple agent types, they do not guarantee convergence because the game becomes atomic. Hence there is no guarantee on the existence of a potential function [7]. Since there exists an equilibrium for every finite game and our approach is based in optimization, we do not have issues with convergence. In contrast to variational inequality models [4] commonly used for traffic equilibrium problems in transportation networks, our approach is much more scalable computationally as demonstrated in the experimental results.

While SeOP is similar to network congestion games (NCGs), it has the following complicating requirements: 1) an agent needs to visit a given set of nodes and 2) different agent types have different latency functions (cost vectors) for a given edge. As such, the proof of existence of pure NE that holds for NCGs [9, 3] does not necessarily hold in SeOP. Unlike symmetric NCGs where agents have the same strategy spaces and for which a polynomial time algorithm exists [3], asymmetric NCGs are PLS-complete. So even if SeOPs have pure NE, because each agent in SeOP must visit certain nodes (and thus has a different set of valid strategies), SeOP is likely to be PLS-complete or harder. Another complexity argument is that OP is NP-hard [5], so our multi-agent version is at least in this complexity class. Milchtaich [8] considered congestion games with player-specific utilities, but each player selects exactly one resource and the game is not networked.

DIRECT is similar to column generation in that we gradually increase the size of the support set. In order to compute reduced cost of adding a column (a path in this case), column generation needs to have columns that are independent of each other. However, because multiple paths (columns) can share edges, the independence assumption does not hold.

9. CONCLUSION

This paper proposes a new model for the Selfish Orienteering Problem (SeOP) to combine the Orienteering Problem and Selfish Routing/network congestion games. We first give a direct formulation of finding a Nash equilibrium in SeOP as an optimization problem and address its limited scalability through two main contributions: a compact formulation with a linear number of constraints to enforce the equilibrium condition without pairwise comparisons of path latencies and DIRECT, an iterative master-slave decomposition approach that incrementally grows the set of paths to compute an approximate equilibrium. We provide a theoretical discussion of our approximation quality and present experimental results clearly showing that 1) our non-pairwise formulation achieves the same solution quality as the pairwise one using a fraction of the number of constraints; and 2) our master-slave decomposition achieves solutions with adjustable approximation gap using a fraction of the full path set. Together, these two contributions form an approach to solving SeOP that is highly scalable in the number of attractions and types of agents, while also being scale invariant in number of agents of each type.

Acknowledgements 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, Media Development Authority (MDA).

REFERENCES

[1] C. Archetti, A. Hertz, and M. G. Speranza. Metaheuristics for the team orienteering problem. *Journal of Heuristics*, 13(1):49–76, 2007.

[2] G. W. Brown. Iterative solution of games by fictitious play. *Activity Analysis of Production and Allocation*, 13(1):374–376, 1951.

[3] A. Fabrikant, C. Papadimitriou, and K. Talwar. The complexity of pure nash equilibria. In *Proc. of STOC*, 2004.

[4] T. L. Friesz, D. Bernstein, T. E. Smith, R. L. Tobin, and B. W. Wie. A variational inequality formulation of the dynamic network user equilibrium problem. *Operations Research*, 41(1):179–191, 1993.

[5] B. L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.

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

[7] K. Kollias and T. Roughgarden. Restoring pure equilibria to weighted congestion games. In *Proceedings of the 38th international conference on Automata, languages and programming - Volume Part II*, ICALP'11, pages 539–551, Berlin, Heidelberg, 2011. Springer-Verlag.

[8] I. Milchtaich. Congestion games with player-specific payoff functions. *Games and Economic Behavior*, 13:111–124, 1996.

[9] R. W. Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

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

[11] 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):1853–1859, 2010.

[12] G. N. T. Leventhal and L. Trotter. A column generation algorithm for optimal traffic assignment. *Transportation Science*, 7(2):168–176, 1973.

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

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

[15] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1 – 10, 2011.

[16] J. G. Wardrop and J. I. Whitehead. Correspondence. some theoretical aspects of road traffic research. *ICE Proceedings: Engineering Divisions*, 1, 1952.