Multi-Agent Path Finding for Self Interested Agents

Zahy Bnaya
Information Systems Engineering
Ben-Gurion University
Beer-Sheva, Israel
zahy@bgu.ac.il

Roni Stern
SEAS
Harvard University
Cambridge MA, USA
roni.stern@gmail.com

Ariel Felner
Information Systems Engineering
Ben-Gurion University
Beer-Sheva, Israel
felner@bgu.ac.il

Roie Zivan and **Steven Okamoto**
Industrial Engineering and Management
Ben-Gurion University
Beer-Sheva, Israel
{zivanr,okamotos}@bgu.ac.il

Abstract

Multi-agent pathfinding (MAPF) deals with planning paths for individual agents such that a global cost function (e.g., the sum of costs) is minimized while avoiding collisions between agents. Previous work proposed centralized or fully cooperative decentralized algorithms assuming that agents will follow paths assigned to them. When agents are *self-interested*, however, they are expected to follow a path only if they consider that path to be their most beneficial option. In this paper we propose the use of a taxation scheme to implicitly coordinate self-interested agents in MAPF. We propose several taxation schemes and compare them experimentally. We show that intelligent taxation schemes can result in a lower total cost than the non coordinated scheme even if we take into consideration both travel cost and the taxes paid by agents.

1 Introduction

In a *multi-agent path finding* (MAPF) setting, agents situated in a graph must move to their goal vertices without colliding with each other. This setting exists in robotics, digital entertainment, and other fields. Most previous work on MAPF focused on algorithms that compute paths for individual agents (in a centralized or decentralized manner) in order to minimize a global cost function (e.g., time or fuel expenditure) while avoiding collisions between agents (Sharon et al. 2011; Standley 2010; Ryan 2008; 2010; Silver 2005; Dresner and Stone 2008; Sharon et al. 2012).

These studies make the fundamental assumption that agents are *fully cooperative* and share a global cost function. Cooperative agents will follow paths that are planned for them or obey constraints on their movements if doing so leads to lower global cost (Wang and Botea 2009; Jansen and Sturtevant 2008). In particular, agents may be willing to take longer paths to their goals in order to avoid colliding with other agents.

By contrast, in this work we assume that agents are *self-interested*, seeking only to minimize their individual costs. As a motivating example, consider human drivers. Each driver tries to reach his destination as quickly as possible, regardless of the effect on other drivers. The suboptimal result is a traffic jam, where drivers must make local decisions (slowing down and waiting for other drivers to move) in order to avoid collisions. In this paper we address the problem of incentivizing self-interested navigating agents to act in a way that minimizes a global cost function.

We propose a taxation mechanism for coordinating self-interested navigating agents in which additional *taxes* are imposed on agents that pass through specified locations (vertices or edges). These taxes provide *implicit coordination* by incentivizing agents to avoid locations where a traffic jam would otherwise occur, thus resulting in better navigation.

In this work we are interested in maximizing the total *social welfare* of the agents, defined as the negative combination of the travel cost and the taxation costs incurred by the agents for traversing their paths.

The combinatorial search challenge is in determining where, when and how much taxes should be imposed in order to maximize the total *social welfare* for all agents. To this end we introduce a general taxation framework called the Iterative Taxing Framework (ITF). ITF assigns taxes in an iterative manner. First, it predicts the selfish path selections made by agents and imposes taxes where agents would collide. ITF then predicts the impact of the newly-imposed taxes on agents' path selection and identifies new conflicts that may arise, imposing additional taxes to mitigate them. This process continues until a conflict-free set of paths is found for the self-interested agents or the process is halted externally.

Two implementations of ITF are proposed: Exhaustive Iterative Taxing Algorithm (EITA) and Monte-Carlo Iterative Taxing Algorithm (MC-ITA). EITA computes all possible paths that self-interested agents may consider while MC-ITA randomly chooses a single path per agent but repeats the process multiple times.

We evaluated EITA and MC-ITA on several domains. Our results show that both implementations lead to higher so-

cial welfare when compared to self-interested agents without taxation.

2 Problem Definition

In this work, we address the problem of MAPF with *self-interested* agents which is fundamentally different than the standard MAPF (also called cooperative MAPF).

Definition: A *self-interested* agent in MAPF always chooses to follow the path with the best individual *social welfare* (minimum sum of tax and travel costs).

First, we describe a standard cooperative MAPF setting and then define our problem of selfish navigation in that MAPF setting.

Cooperative MAPF: Our MAPF setting consists of k agents $a_1, a_2, \ldots, a_k$ situated on vertices in an undirected graph $G = (V, E)$ with non-negative edge costs. Each a_i begins at a start vertex $start_i$ and wishes to travel to a goal vertex $goal_i$. Time is discretized into steps, and during each time step each agent either moves to an adjacent vertex along an edge or remains at its current vertex. A *path* P_i for a_i is a sequence of vertices $start_i = v_0, \ldots, v_t = goal_i$ with consecutive vertices connected by an edge. The cost of traversing path P_i is denoted by $c(P_i)$ and defined as the sum of the costs of the edges in P_i.

The key constraints in MAPF are that each vertex can be occupied by at most one agent at each time step, and two agents cannot traverse the same edge in the same time step. A common objective in cooperative MAPF is to find a path $P_1, .., P_k$, one for each agent, that does not violate these key constraints and minimizes $\sum_{i=1}^{k} c(P_i)$.

MAPF with Self-Interested Navigating Agents:

In our model, agents are self-interested and do not coordinate with one another. We assume that self-interested agents compute and follow a lowest-cost path to their destination, minimizing their individual cost without considering the paths planned by other agents. A self-interested agent might follow a path that causes other agents to delay or even to not be able to reach their destinations. None of the algorithms for cooperative MAPF can be used under this assumption, as they usually require some level of coordination. Therefore, a new solution scheme is needed.

In self-interested MAPF, multiple agents may plan to occupy a vertex v at the same time t; we call such a pair (v, t) a *conflict*. The same constraints defined in cooperative MAPF still hold. Therefore, if a conflict exists, the paths planned by the agents involved in the conflict cannot be executed.

Inspired by real-life situation, we assume that conflicts are detected locally and resolved by the conflicting agents as follows. One of the conflicting agents is promoted to be the *master* of the conflict and uses a conflict-dependent *reservation table* for reserving its n next steps. This is reminiscent of the global *reservation table* used by Cooperative A* and its many variants (Silver 2005) to solve MAPF. The other agents involved in the conflict must replan while obeying the reservation table. Unable to communicate, conflicts may occur, causing the agents to replan and resulting in increased cost.

In practice, the decision of which agent gets to be the master of the conflict is both subtle and complex. It can be based on personality, body language, or sense of urgency. In our work, we select the master for each conflict *randomly*. Other models of how self-interested navigating agents behave are possible, and most of the work presented in this paper is applicable to other models of behavior as well.

Assuming the above model of how self-interested navigating agents behave, the problem addressed in this paper is to minimize the total cost, given self-interested navigating agents. We do this by proposing the following taxation scheme.

3 Taxation Schemes for Manipulating Navigating Agents

To motivate the agents to avoid collisions and improve the total cost, we assume the possibility of a *taxation scheme* in which agents traveling through an edge or a vertex at a specific time may have an additional cost to pay. We use the term "penalty" to refer to these additional costs. Such time-dependent penalties exists in the real world in advanced toll roads.

Importantly, we assume that a penalty can be directly translated to the travel cost from the agent's perspective. Thus, a self-interested agent would consider to avoid using a penalized path, potentially causing less conflicts. We demonstrate experimentally that our mechanism is able to do so and results in a total cost that is substantially lower than without using this mechanism.

In detail, the proposed taxation mechanism works as follows. A penalty can be assigned to any vertex v at any time step t. We label this by $p(v, t)$. An agent located at v at time t must pay $p(v, t)$. Similarly, a penalty can also be assigned to an edge, using the notation $p(e, t)$. The values of all penalties are stored in a *penalty table* denoted by $\mathcal{T}$ and assumed to be known by all agents.

For a given path P_i, we define the *penalized cost* of P_i, denoted by $c_{\mathcal{T}}(P_i)$, as the sum of the penalties stored in $\mathcal{T}$ incurred by a_i traversing P_i. To differentiate from the penalized cost, we use the term *travel cost* of P_i to refer to the cost of following P_i with no penalties (this was denoted above as $c(P_i)$). The cost of traversing a path P_i is the sum of the travel cost and penalized cost of P_i. This sum is referred to as *penalized travel cost*, denoted by

$$\hat{c}_{\mathcal{T}}(P_i) = c(P_i) + c_{\mathcal{T}}(P_i)$$

Every self-interested agent is assumed to consider only *lowest-cost* paths, in terms of the penalized travel cost.

Figure 1 demonstrates the potential benefit of the proposed taxation mechanism. $S1$, $S2$, and $S3$ are the start locations and $G1$, $G2$, and $G3$ are the goals of a_1, a_2, and a_3, respectively. The travel cost of traversing every edge is one. Assume that the agents operate in the self-interested manner described in Section 2. Since the lowest-cost paths of all agents pass through C, they will all plan a path via C. As a result, they will conflict, and the conflict will be resolved by one of them waiting one time step and another waiting

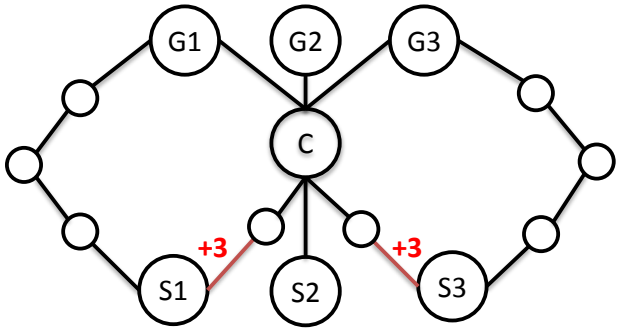


Figure 1: Example of the taxation mechanism

two time steps. This results in a total cost of 8+1+2=11. By contrast, assume that a taxation mechanism is used, and a penalty of 3 is added for using the edges marked in red in Figure 1 at time step 1. If a_1 and a_2 are aware of this penalty, each of them will choose the alternative path of reaching the goal by going around and as a result the total cost for all agents is 10.

4 Related work

The problem of optimizing routing traffic was widely discussed before in the context of *flow*-networks (Pigou 1952; Roughgarden and Tardos 2002). A *flow*-network includes traffic rates (or flows) between the nodes and a latency-function. The *flows* model large quantities of individual elements (cars, data packets, etc.) that are divided between alternate paths. The latency-function is a continuous function of the amount of flow on a given edge, that represent a "delay" incurred by traversing an edge.

Unlike the problem discussed in this paper, in flow-networks all agents that traverse a flow-edge incurs the same cost. In this way delay costs are a soft constraint. The latency in flow networks is a similar idea to when an agent remains in a given vertex until its next vertex is free in our setting.

Selfish routing in flow networks is known to be non-optimal (Roughgarden and Tardos 2002). Several works also suggest the use of toll mechanisms, similar to the one suggested in our work (Ferrari 2002; Cole, Dodis, and Roughgarden 2006; Dial 1999). A taxation mechanism was also introduced in the context of Boolean games (Endriss et al. 2011) and it is also related to congestion games (Rosenthal 1973).

Our model differs from flow networks in several important aspects. First, we consider discrete environments as opposed to the continuous quantities modeled in flow networks. Our model directly models the discrete elements (agents) and also the way in which they have to deal with the hard capacity constraints (only a single agent can be positioned in a give location).

Our model for latency is fundamentally different from the one used by flow networks. Agents who want to use a "congested" edge do not all receive the same cost. One agent gets to go through immediately (with no delay cost), the next agent suffers a delay of 1, the next agent a delay of 2, etc. The latency in our model also has a "cascading effect". Agents that encounter a congested edge, necessarily add constraints since they need to spend time in other locations or choose a different path. Thus, causing other congestions.

Another difference is the assumed observability of the agents. Works on *Flow*-networks assume that agents selfishly plan their paths based on the travel costs considering also traffic-based delays.

In contrast, we make the assumption that agents plan their paths while completely ignoring the effects (or even the existence) of other agents. The only thing that matters to the agents is the distance traveled and the tax at each time.

Usually in optimization of traffic networks, the task is to minimize latency and ignore the taxation. We believe that this is not reasonable. (Cole, Dodis, and Roughgarden 2006) discusses the problem of minimising the combined sum of travel and tax for selfish routing in flow networks. They showed that linear latency networks cannot be improved by using taxation and that in any case it is less beneficial than just removing an edge. However, in non linear latency networks, taxation can dramatically improve total latency.

We are not aware of any other attempt to address this problem with hard constraints for self-interested agents. We believe that using our model has its merits and demonstrates interesting observations.

The challenge we address next is how to construct a penalty table $\mathcal{T}$, such that the sum of the penalized travel cost incurred by all agents is minimized. This is expected to maximize he social welfare of the group of agents. Next, we describe a general framework for constructing efficient penalty tables for a given MAPF instance.

5 The Iterative Taxation Framework (ITF)

Next, we describe a high-level algorithmic framework for generating a penalty table. This framework, which we call the *Iterative Taxation Framework* (ITF), is composed of the following components.

- **Path planning (PLAN).** Generates for every agent a set of paths with lowest $\hat{c}_{\mathcal{T}}$.
- **Conflict detection (DETECT).** Detects a set of conflicts between the plans of the agents.
- **Conflict resolution (RESOLVE).** Attempts to resolve these conflicts by adding penalties to $\mathcal{T}$.

Algorithm 1 describes the roles of PLAN, DETECT and RESOLVE in ITF in generating a penalty table. First, an empty penalty table $\mathcal{T}_0$ is constructed (line 2). Then, for every agent, PLAN finds a set of paths having minimal penalized travel cost with respect to $\mathcal{T}_0$ (line 4). The DETECT component analyzes these paths and outputs a set of conflicts (line 5). If DETECT does not return any conflict, the algorithm halts and returns the current penalty table (line 7). Otherwise, some conflicts were detected and RESOLVE generates a new penalty table $\mathcal{T}_{i+1}$ in an effort to resolve these conflicts (line 8). This sequence of PLAN, DETECT and RESOLVE is called an *ITF iteration*. ITF

Algorithm 1: ITF

```
1 i ← 0
2 T_i ← ∅
3 while True do
4     P ← PLAN(T_i)
5     Conflicts ← DETECT(P, T_i)
6     if Conflicts is empty then
7         return T_i
8     T_{i+1}=RESOLVE(Conflicts, P,T_i)
9     i ← i + 1
```

runs these ITF iterations until DETECT does not find any conflict. Note that in every ITF iteration PLAN may return a different set of paths, since it considers the penalty table that was generated by RESOLVE in the previous ITF iteration. As a result DETECT and RESOLVE may also detect different conflicts and add new penalties to the penalty table.

One can also implement ITF with other stopping conditions, e.g., halting after a predefined number of iterations or according to a timeout. The penalty table generated by the last ITF iteration is returned.

The properties and performance of ITF greatly depend on the implementation of the PLAN, DETECT and RESOLVE components. There are many possible ways to implement these components, and we describe two types of implementations that performed best in our experimental evaluation. The first ITF implementaiton, which is described next, is based on enumerating all the paths that a self-interested agent would consider. The second ITF implementation described in this paper is more tractable, and is based on sampling the paths that each self-interested agent might choose.

5.1 Exhaustive ITA

The first ITF implementation we present is called the *Exhaustive Iterative Taxing Algorithm* (EITA). The high-level concept of EITA is to consider all the possible paths that each agent may consider. Then, detect all possible conflicts between these paths and attempt to add the minimum amount of penalties such that the agents involved in each conflict will consider alternative paths that will allow the conflict to be resolved. Next, we describe the PLAN, DETECT and RESOLVE components of EITA.

PLAN The purpose of the PLAN component in EITA is to find for each agent all the path that it might consider. Following our model of how self-interested agents behave (describe earlier in this paper), each agent considers to use only paths that have the lowest penalized cost.

Let $PLAN_{\mathcal{T}}(a_i)$ denote these set of paths for agent a_i considering penalty table $\mathcal{T}$. Let $best_{i,\hat{c}_{\mathcal{T}}}$ be the penalized cost of these paths. PLAN in EITA returns for every agent a_i the paths in $PLAN_{\mathcal{T}}(a_i)$.

As a running example consider the scenario in Figure 1 and assume that $\mathcal{T}$ contains no penalties. In that case, $PLAN_{\mathcal{T}}(a_1)$ contains a single path passing from S_1 to G_1 via C, and $best_{1,\hat{c}_{\mathcal{T}}} = 3$.

DETECT The purpose of the DETECT component in EITA is to find all possible conflicts that exist between plans that the agents may considered. This is done by going over the cross product of $PLAN_{\mathcal{T}}(a_j)$ and $PLAN_{\mathcal{T}}(a_i)$ for every $i \neq j$. A conflict is found when $p_i \in PLAN_{\mathcal{T}}(a_i)$ and $p_j \in PLAN_{\mathcal{T}}(a_j)$ contain the same time-location pair. EITA exhausitively reutrns all such conflicts.

We use a 4-tuple (T, L, a_i, a_j) to denote a conflict occuring in time T at location L between agents a_i and a_j. In our running example, DETECT in EITA will return the conflicts: $(2, C, a_1, a_2)$,$(2, C, a_1, a_3)$, and $(2, C, a_2, a_3)$.

RESOLVE The RESOLVE component in EITA attempts to add minimum penalties to the penalty table such that agents involved in a conflict (as detected by the DETECT component) are motivated to consider additional paths. The motivation behind this is that these additional path will cause the selfish agents to avoid conflicts. Next, we describe how we implemented this approach.

We define $\overline{PLAN_{\mathcal{T}}(a_i)}$ to be the set of paths from the start location of agent a_i to its goal that are not in $PLAN_t(a_i)$. Let Δ_i to be the difference between $best_{i,\hat{c}_{\mathcal{T}}}$ and the penalized cost of the path with the lowest penalized cost in $\overline{PLAN_{\mathcal{T}}(a_i)}$:

$$\Delta_i = best_{i,\hat{c}_{\mathcal{T}}} - min\{\hat{c}_{\mathcal{T}}(P) | P \in \overline{PLAN_{\mathcal{T}}(a_i)}\}$$

Clearly, adding penalties smaller than Δ_i to the paths in $PLAN_{\mathcal{T}}(a_i)$ will not trigger a_i to consider more paths than those in $PLAN_{\mathcal{T}}(a_i)$. On the other hand, adding penalties larger than Δ_i to the paths in $PLAN_{\mathcal{T}}(a_i)$ excludes these paths from the next PLAN call. Thus, DETECT might find conflicts and add penalties when a conflict-free combination exists. These paths will be excluded from the next DETECT procedure. Therefore, RESOLVE in EITA attempts to assign a penalty of exactly Δ_i to each of the paths in $PLAN(a_i)$.

RESOLVE in EITA requires agents to consider new paths by having their Δ_i added as a penalty to all paths in $PLAN_t(a_i)$ otherwise the next call to DETECT will be identical. However, it is not necessary for all agents to include additional paths. Actually, having at least one of the agents include new paths is a sufficient condition without getting a weaker solution.

EITA prefers minimal penalization, therefore for every set of conflicts on any of the paths, EITA penalizes according to the agent that has the minimal Δ_i. Although this can make one of the agents to produce the same path set on the next PLAN call, it avoids over penalization.

Choosing how to resolve all the detected conflicts while adding a minimal amount of penalty can be formulated as a linear program.

First, we define the set of variables $X = \{x_1, x_2...x_n\}$ that represents the penalties to be added to each of the conflicts returned on current iteration.

We also add slack variables $S^i = \{s_1, s_2, ...s_k\}$ for each agent. The role of these variables is to allow EITA to solve the linear equations such that penalization is done according to the agents with the minimal Δ.

We add constraints to the system as follows. For every path p in $PLAN(a_i)$ and q in $\overline{PLAN(a_i)}$ and for every agent i we add the following constraint:

$$\sum_{x_i \in X} I(p, x_i)x_i + s_i = \sum_{x_i \in X} I(q, x_i)x_i + \Delta_i$$

Where $I(p, x)$ is an indication of whether conflict x exists in path p:

$$I(p, c) = \begin{cases} 1 & c \in p \\ 0 & otherwise \end{cases}$$

Since we also want the penalization to be always positive, we add the following condition to the system of equations:

$$\forall x_i \in X : x_i \geq 0$$

We then minimize the following function:

$$\sum_{x_i \in X} x_i - \sum_{s \in S} s_i$$

EITA continuously adds penalties to the penalty-table until there is a combination of paths, one for each agent, such that none of the agents are colliding. Since EITA considers all paths on each iteration, none of the agents has a path with a smaller $\hat{c}_{\mathcal{T}}$ than the found combination. On most cases, this path combination also has the minimal sum of travel costs ($c(p_i)$).

Computational Complexity While appealing, the most major shortcoming of EITA is its computational complexity. PLAN can be performed in time that is polynomial in the size of the graph and linear in the number of agents, by simply performing an A* search from every agent. DETECT requires matching all the paths found by PLAN for every agent. This operation is exponential in the number of agents, requiring $O(\prod_{a \in A} |PLAN(a)|)$. RESOLVE involves solving a set of linear equations, which can be done in polynomial time in the number of equations. However, the number of such equations is linear in the number of conflicts, which can theoretically be as large as the cross product of all paths returned by PLAN for each agent.

One approach to handle the complexity of EITA is to limit the number of lowest-cost paths returned by PLAN for every agent. This approximation version of EITA is denoted by EITA(k), where k is the maximum number of paths returned by PLAN for every agent. If an agent has more than k lowest-cost paths, k of them are selected randomly. Next, we propose an alternative approach that is based on Monte-Carlo simulations.

6 Simulation-based Taxation

Consider the behavior of EITA(1). The PLAN component returns a single lowest-cost path for every agent. Then, DETECT matches these paths, which can be done trivially by comparing the location of every agent in every step along its path. Finally, RESOLVE adds the minimum amount of penalties to resolve each of the conflicts along these paths.

Naturally, the complexity of EITA(1) is substantially smaller than EITA. However, the number of lowest-cost paths that exist for a given agent might be larger than 1, which are all equally preferable by the agent. EITA(1) considers only a single path for every agent, thus, detecting and resolving conflicts that are relevant only for the case where that agent chose that specific path. However, if an agent chooses one of the other lowest-cost paths, other conflicts may occur, which are not resolved. Following, we propose the *Monte-Carlo Iterative Taxing Algorithm* (MC-ITA) that performs a large set of quick EITA(1)-like iterations in an effort to cover a wider range of conflicts and resolve them. Then, the RESOLVE component of MC-ITA uses multiple quick simulations to decide how to distribute penalties among the detected conflicts. Next, we describe the details of MC-ITA.

6.1 PLAN and DETECT in MC-ITA

The PLAN and DETECT components in MC-ITA are the same as these components in EITA(1). For every agent a_i, PLAN returns a single lowest-cost path that is randomly chosen from all the lowest-cost pathS of a_i. Then, the DETECT component finds conflicts by simulating the movement of every agent along the path found for it by PLAN, detecting a conflict when two agents collide.

6.2 RESOLVE in MC-ITA

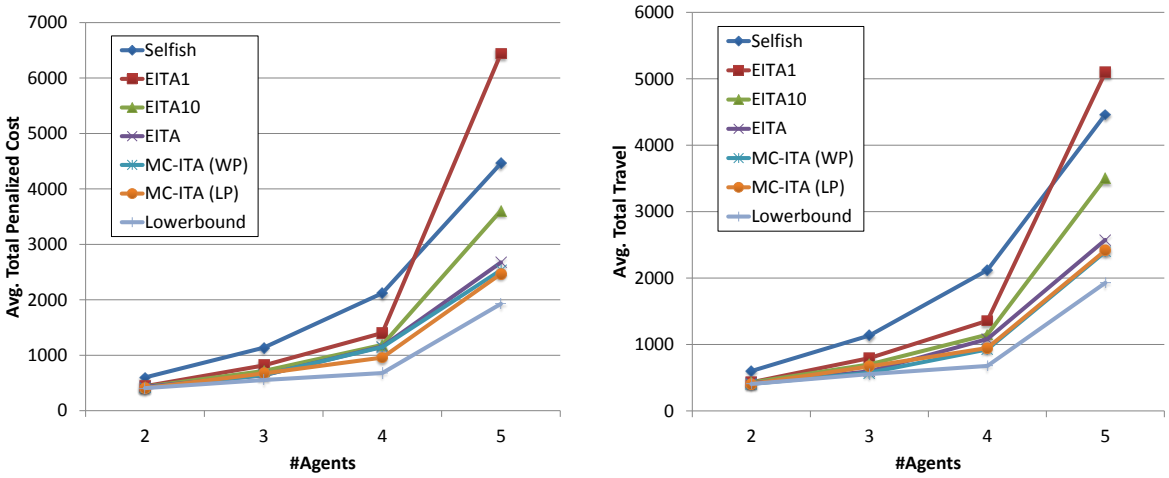
In every ITF iteration of MC-ITA, the PLAN component may choose a different lowest-cost path potentially finding new conflicts. To take advance of this, the RESOLVE in MC-ITA considers all the conflicts returned by DETECT, including conflicts found in previous ITF iterations. This is done to allow information passing between iterations.

One might consider the same linear programming approach described above for the RESOLVE component of EITA. However, although solving a linear program is polynomial, using it in MC-ITA would incur a significant computational overhead, since MC-ITA is designed to run many quick ITF iterations, calling the RESOLVE component many times. Thus, we propose two simpler alternative implementations for RESOLVE in MC-ITA. One may consider using these implementations for EITA as well.

Penalty Location and Time

The two main questions in a RESOVLE component are, first, where and when to add penalties and second, what penalties to add to resolve the detected conflicts. An intuitive answer to the first question, where to add a penalty to resolve a conflict in time t at location l, is to add a penalty exactly in time t at location l. As a result, agents that have alternative paths with lower cost than their currently given path after the added penalty, will use those paths in the next call to PLAN, and thus, this conflict will be avoided in the next iteration. However, if one of the agents gets to use location l at time t, it must pay the assigned penalty. We call this method for selecting where and when to add the penalty the *winner-penalization* (WP) method.

An alternative method for selecting where and when to add a penalty to attempt to resolve a conflict (at l, t) is to add a penalty to one of the edges leading to l at time $t-1$ in one of the conflicting paths. The agent that planned to use that edge on its way to arrive at the conflicting location will choose an alternative path by PLAN, if such an alternative path exists and has a lower cost than the cost of the con-



(a) Avg. total cost for 3x3 grid

(b) Avg. total travel cost for 3x3 grid

Figure 2: Results on the 3X3 grids.

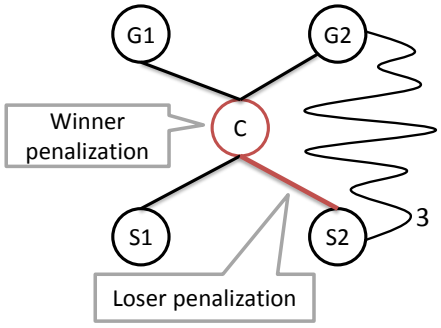


Figure 3: WP vs. LP methods.

flicting path plus the added penalty. Choosing which edge that leads to the conflict will be penalized is done randomly. We call this method for selecting where and when to add a penalty the *loser-penalization* (LP) method.

To demonstrate the difference between WP and LP, consider the simple conflict in Figure 3. S1, S2, G1 and G2 are the start vertices of agents 1 and 2, and the goal vertices of agents 1 and 2, respectively. Assume that all the edges have unit cost, except for the edge between S2 and G2 whose cost is 3. Clearly, (C,2) is a conflict between agent 1 and agent 2. WP would add a penalty to (C,2) while LP may add a penalty to either the edge from $S2$ to C or to the edge from $S1$ to C. This demonstrates both the benefit and the drawback of the LP over WP. If the conflict point is important to one of the agents, in WP it will have to pay the penalty for using that conflict point, while in LP this can be avoided (for example, if the edge between S2 and C is penalized). On the other hand, the conflict might not be resolved at all with LP, requiring more iterations of MC-ITA.

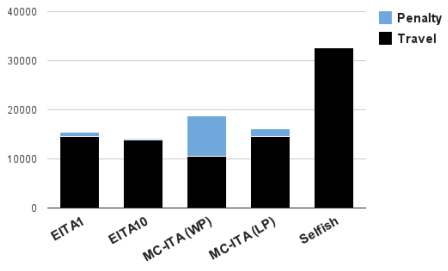
Once the time-location pair to which the penalty should be assigned has been selected, using either WP or LP, we add it to the set of conflicts found so far and proceed to the task of determining the magnitude of the penalty. This is done by creating multiple penalty tables, as candidates for becoming the next penalty table. In our implementation, this was done by generating a random real-valued number in the range of [0,1] for each of the conflicts. The motivation behind this approach is that it is both simple and quick. Each penalty table is then evaluated by simulating the behavior of the different agents with this penalty table, where each agents behaves according to our model for self-interested agents (Section 2). This random assignment of penalties and evaluation by simulation is repeated a predefined set of times, and the assignment with the highest overall utility is used. The number of simulations performed is a parameter of the algorithms, which was set to be 50 in our experimental results. Parameter tuning mechanisms may be implemented to choose this number more efficiently.

7 Experimental Results

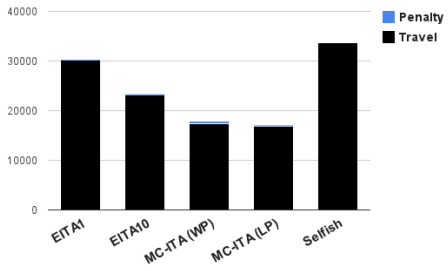
We compare the performance of EITA, MC-ITA with WP and MC-ITA with LP on a number of pathfinding settings. In every experiment, each of the algorithms was used to generate a penalty table. The performance of a penalty table was measured by simulating the behavior of the agents given that penalty table, following the self-interested navigation model described in Section 2. As a baseline approach, we also evaluated the performance on an "empty" table, where no penalties are given. This is denoted as "Selfish".

The first set of experiments were performed on a small 3x3 grid. Figure 2(a) and 2(b) depicts the total cost ($\hat{c}_{\mathcal{T}}(\cdot)$) and the travel cost ($c(\cdot)$) of each of the algorithms on a 3x3 four-connected grid with no obstacles. The x-axis is the number of agents, and the y-axis is the total penalized travel cost and total travel cost, respectively. We used 50 instances and randomized the source and target of each of the agents. Figure 2(a) shows the total travel costs of the same problem set, as function of the number of agents.

As can be seen, all algorithm except EITA(1) are always



(a) 50x50 grid with 20% for 20 agents



(b) Dragon age's den520 for 10 agents

(c) Dragon age's den520

Figure 4: Results on large grids.

Agents	EITA	EITA(1)	EITA(10)	MC-ITA	
				WP	LP
2	11.6	9.8	2	8	0
3	41.6	20.5	16	82	4.7
4	75.8	45	37	145.8	10.4
5	100.2	1345.9	107.6	90.9	53.6

Table 1: Sum of penalties in 3x3 grids.

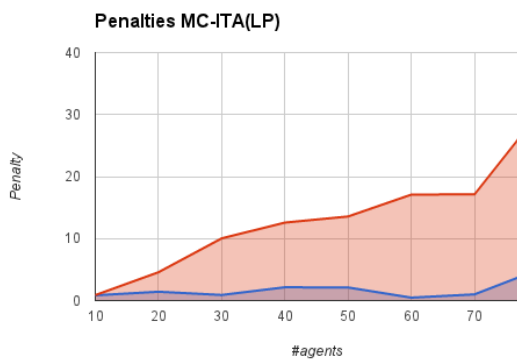
able to find a solution of lower total cost than the "Selfish" case. Furthermore, we added a "Lower bound" line to these results, which are the optimal solution to this MAPF problem when agents are coordinated. Naturally, no penalty table can achieve a better total cost. As can be seen, the total cost obtained by both MC-ITA variants (LP and WP) and EITA obtains superior results over EITA(1), EITA(10) and Selfish, and perform very close to the lower bound. The sum of all the penalties in the penalty table are shown in Table 1. Notably, the WP method in MC-ITA adds substantially more penalty than the LP method. This supports the intuition behind LP, that penalizing adjacent edges instead of the conflict itself can lead agents to avoid some penalties.

The second set of experiments were performed for 20 agents on a four-connected 50x50 grid with 20% randomly selected blocked cells. The sources and destinations of the agents were set by generating multiple random points and selecting only the points that are positioned in narrow places and in the proximity of the source or destination of other agents. This is done to prefer challenging cases where agents collide and need to replan frequently. Running EITA and computing the lower bound was impractical even for this relatively small grid and thus we only experimented with EITA(1), EITA(10), MC-ITA (LP and WP), and Selfish. The MC-ITA algorithms were run for 1000 iterations, which still faster than EITA(10).

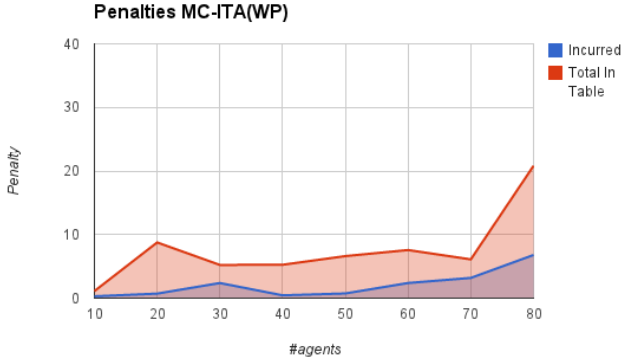
Figure 4(a) shows the total cost achieved by each of the algorithms, averaged over 50 instances. The y-axis is the total cost ($\hat{c}_{\mathcal{T}}$), where and the black portion of every bar shows the travel cost and the blue portion shows the penalized cost ($c_{\mathcal{T}}$). These results demonstrate the great benefit of using a penalty table, as all of the proposed algorithms were able to generate a penalty table that considerably improves the performance of not having a penalty table, shown by the Selfish results. The most notable results are obtained by using the MC-ITA algorithms, EITA(10) and EITA. Between the two variants of MC-ITA, the one using LP and the one using WP, the results clearly show that the LP method achieves a more efficient total utility. A deeper observation reveals that WP caused the agents to travel through locations with penalties than LP. This is reasonable, since in WP the penalty is added on the conflict point, thus if any agent decides to pass through the conflict point, it will have to pay the penalty of the conflict. By contrast, in LP the penalty is assigned only on one of the agents' path, and thus there is less chance that the penalty will be actually incurred. Interestingly, the WP method resulted in solutions with smaller travel cost.

The third set of experiments were performed on the *den520d* map (shown in Figure 4(c)) of the game *Dragon Age:Origins*, made publicly available by Stutervant (2012). Figure 4(b) shows the average total penalized travel cost for 10 agents. Similar trends are observed. All algorithms are more efficient than Selfish, again demonstrating the effectiveness of using a penalty table to motivate self-interested agents to avoid conflicts. In this large map, MC-ITA is substantially better than EITA(1) and EITA(10), and the LP method of MC-ITA is slightly better than the WP method. Very interestingly, while the performance of the algorithms is substantially better than Selfish, the actual penalties paid by the agents were very small (these are the blue portions of each of the bars in Figure 4(b)). Thus, adding penalties to certain time-location pairs caused the agents to plan to traverse other paths, that were not penalized but also not congested, resulting in lower travel cost as well as lower total penalized travel cost. Note that the penalty paid by the agents is smaller than the sum of penalties in the penalty table, i.e., the penalty table may contain penalties set for certain time-location pairs that were not used by the agent.

We display the trends achieved by our MC-ITA algorithms in Figure 5. The y-axis shows the total costs ($\hat{c}_{\mathcal{T}}$) and the x-axis shows the number of agents. Our two different MC-ITA penalization schemes are shown in Figure 6. The red lines display the total penalization assigned to $\mathcal{T}$. The blue lines show the total penalty incurred by the agents. While MC-ITA(LP) assigns more penalties on the table, the actual penalty incurred by the agents is still very small.



(a) Penalization MC-ITA(LP)



(b) Penalization MC-ITA(WP)

Figure 6: MC-ITA penalization schemes for 50x50 with 20% obstacles

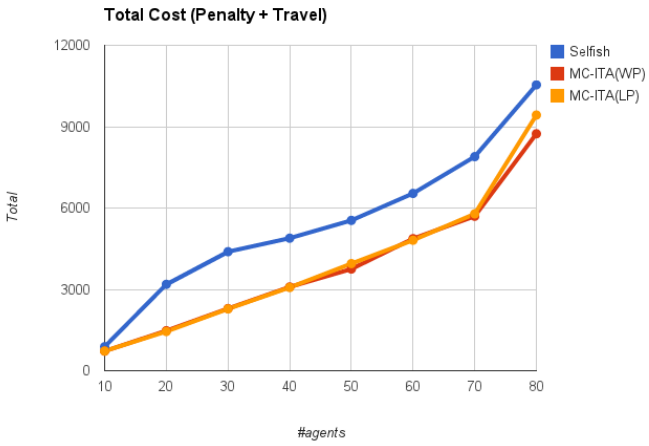


Figure 5: 50x50 with 20% obstacles

8 Conclusion

In this paper we addressed the scenario of multiple self-interested agents navigating in a graph. We proposed a taxation scheme for coordinating these agents implicitly, applying additional costs (penalties) to agents that pass through specified locations at specific times. The Iterative Taxing Framework (ITF) was presented for determining where, when and how much penalty should be given. Two specific implementations of ITF are described. EITA, which detects conflicts from all possible paths that the agents might choose, and MC-ITA, which samples the set of possible paths. Experimental results show that the taxation schemes produced by both of these algorithms are able to improve the total cost of the agents substantially, even when adding the cost penalties costs to the traveling costs.

There are many exciting future research directions. The possible implementations of the ITF components should be studied deeply; identifying how much effort should be invested in detecting conflicts and studying the most fitting way to resolve these conflicts. Another interesting future work is to incorporate more complex behavior of the self-interested agents, going beyond the model described in Section 2 and including direct communication between the agents.

9 Acknowledgements

This research was supported by the Israeli Science Foundation (ISF) under grant #305/09 to Ariel Felner.

References

Cole, R.; Dodis, Y.; and Roughgarden, T. 2006. How much can taxes help selfish routing? *Journal of Computer and System Sciences* 72(3):444–467.

Dial, R. B. 1999. Network-optimized road pricing: Part ii: Algorithms and examples. *Operations Research* 47(2):327–336.

Dresner, K., and Stone, P. 2008. A multiagent approach to autonomous intersection management. *JAIR* 31:591–656.

Endriss, U.; Kraus, S.; Lang, J.; and Wooldridge, M. 2011. Incentive engineering for boolean games. In *IJCAI*, 2602–2607.

Ferrari, P. 2002. Road network toll pricing and social welfare. *Transportation Research Part B: Methodological* 36(5):471–483.

Jansen, M., and Sturtevant, N. 2008. Direction maps for cooperative pathfinding. In *AIIDE*.

Pigou, A. C. 1952. *The economics of welfare*. Transaction Pub.

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

Roughgarden, T., and Tardos, É. 2002. How bad is selfish routing? *Journal of the ACM (JACM)* 49(2):236–259.

Ryan, M. 2008. Exploiting subgraph structure in multi-robot path planning. *JAIR* 31:497–542.

Ryan, M. 2010. Constraint-based multi-robot path planning. In *ICRA*, 922–928.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2011. The increasing cost tree search for optimal multi-agent pathfinding. In *IJCAI*, 662–667.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2012. Conflict-based search for optimal multi-agent path finding. In *AAAI*.

Silver, D. 2005. Cooperative pathfinding. In *AIIDE*, 117–122.

Standley, T. 2010. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, 173–178.

Sturtevant, N. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games* 4(2):144 – 148.

Wang, K.-H. C., and Botea, A. 2009. Tractable multi-agent path planning on grid maps. In *IJCAI*, 1870–1875.