# Towards disrupting teams' plans

Andrada Voinitchi*, Elizabeth Black, Michael Luck

Department of Informatics
King's College London
United Kingdom
andrada.voinitchi@kcl.ac.uk

**Abstract.** In order for an agent or a team of agents to achieve a goal, a sequence of state transitions need to be performed. These transitions constitute a plan. On some occasions multiple ways of achieving the goal may exist. In competitive settings or sabotage scenarios, one may want to prevent or delay an agent or team of agents from achieving a goal; currently, there is little work addressing this issue in the context of agent teams. We argue that plans can be disrupted by preventing particular state transitions from being performed. We propose four algorithms to identify which state transitions should be thwarted such that the achievement of the goal is prevented (total disruption) or delayed (partial disruption). In order to evaluate the performance of our algorithms we define disruption (partial and total) and also provide metrics for its measurement. We do acknowledge that the disruptor may not always have an accurate representation of the disruptee's plans. Thus, we perform an experimental analysis to examine the performance of the algorithms when some of the state transitions available to the disruptee are unknown to the disruptor.

## 1 Introduction

In order to expose and motivate the issue of plan disruption we consider a real-life scenario: a team of terrorists is planning to place and detonate a bomb in a tube station. The bomb needs to be smuggled in part by part in order for the station staff not to become suspicious. There is more than one way that the team can achieve its target: the bomb can be smuggled a part at a time, while keeping the parts hidden in the tube station or it can be smuggled in different combinations of two or three parts. Then it needs to be assembled and detonated. The terrorists must coordinate in order to assemble the bomb on the premises, and leave the station before the bomb is detonated. In order to ease coordination the terrorists can act according to a common plan, a sequence of state transitions performed in order to achieve a desired outcome. The team has more than one way of achieving its goal (i.e. smuggling the bomb in different ways, either part by part or in combination), hence, it has a set of plans.

The terrorists do not know that the security services have discovered their intentions. The security services want to prevent achievement of the goal of detonating the bomb and destroying the tube station, but only have partial information about the plans. A

* An unpublished version of this paper is being presented informally at PlanSIG2013.

question of particular interest to us arises from this example: is there a way to prevent or at least delay the terrorists from detonating the bomb? If so, how can it be done?

We believe that the disruption of plans plays an important role in both defensive and competitive settings such as the terrorist attack scenario, where the secret services have to work in a time race against the terrorists in order to prevent a disaster. We also believe that the disruption of plans is applicable both in the context of teamwork (teams use plans [1]) as well as the context of single agents [2, 3]. If we analyze our scenario (where the terrorists act as a team with a set of plans), we can identify the state transitions that need to be prevented in order to make the attacks fail, providing valuable information about counter-acting malicious behavior. With this motivation, we address the questions of disrupting plans by determining sets of state transitions that, if prevented, render the agent's or team's goal unachievable. In order to determine such sets, we propose four algorithms that identify state transitions to be prevented in order to make the agent's or team's goal unachievable. We also provide an experimental analysis of the performance of the proposed algorithms on a set of different plans, varying parameters such as a plan's number of states, number of goal states, the number of state transitions that are unknown to the disruptor and the number of state transitions existing as part of the plans. Our general research focuses on identifying agents, abilities of agents, communication links between agents and resources required for these transitions to happen and finding ways of severing each of these aspects in order to prevent a set of transitions, but this is outside the scope of this paper.

The main contribution of this paper consists of the four algorithms that can be used to determine state transitions that must be disrupted in a set of plans. We also provide an experimental analysis of our algorithms' performance, discussing how the disruption value obtained from each algorithm is influenced by the number of states, goals, the density and the number of transitions unknown to the disruptor in a set of plans.

We start by defining the plan base (a graph that captures all possible plans to achieve a goal) in Section 3. We further discuss and define two types of disruption: full and partial disruption. We also provide a metric for measuring partial disruption in Section 4. Section 5 presents four algorithms that each identify from a plan base, a set of state transitions for which, once prevented, there is no longer a plan to achieve the goal. Since a disruptor may not always have an accurate representation of the plan base it wishes to disrupt, in Section 6 we discuss disruption under uncertainty. Sections 7 and 8 present an experiment: set-up and analysis of experimental data for the performance of the proposed algorithms on a set of different plans and under uncertainty.

## 2 Related Work

There is currently little work, that we are aware of, addressing the specific problem of disrupting plans. Existing research places the disruption of plans in the context of disrupting teams of agents. Voinitchi et al. propose that a team's plan is disrupted in order to prevent the team from achieving its goal. They suggest the use of norms and incentives in order to prevent agents from performing state transitions in team's plans, thus causing disruption [4]. This work is preliminary and does not provide a way of

identifying state transitions that should be prevented. We address this issue here, starting from the initial idea of finding critical state transitions in a plan.

When considering how a disruptor can determine a set of state transitions that, if prevented, would mean that there is no longer a plan to achieve the goal, we need to be able to represent plans. Work has been done specifying how single agents represent, reason and act about their plans, in order to achieve goals [5, 6], and agent architectures such as PRS and dMARS rely on plan libraries in order for such agents to function [2, 3]. We use concepts such as sets of plans, plans, states, state transitions and goals as referred to in existing work. We also use the idea of identifying state transitions that are part of more than one plan in a set of plans as a way of minimizing the set of state transitions that need to be thwarted in order to obtain disruption of the said set of plans. This idea is inspired by work to identify and order landmarks [7] in which, a landmark is a variable (fact) that is true at one point in all of the solution plans for reaching a goal. We do not use landmarks or state variables in our algorithms, but we adopted the idea of finding shared features (in our case, state transitions) among plans and using them to identify sets of state transitions we call critical for disruption.

Furthermore, the idea of representing an opponent's plans has also been presented in the context of adversarial planning [8, 9]. The perspective often invoked is that of an actor in a scenario (for example in Go [10] and Capture the Flag [11] game settings) that plans the sequence of actions to achieve a goal in response to the actions that it believes its adversary, or adversaries [9], are likely to perform. Our work is different as we focus on disrupting all of an adversary's plans, rather than plan our actions to reach our goal based on what an adversary might do. Furthermore, we view disruption from a global perspective rather than the perspective of an actor in a scenario. We assume a partial representation of all of a malicious team's (or agent's) plans and identify the state transitions to be disrupted in order to prevent or delay the achievement of goals, as a starting step in the disruption of plans.

Teams of agents also use plans to achieve a shared goal. A team of agents is a group of agents that have a joint goal [12, 13]. Currently existing teamwork theories all share a common feature: agents in teams use plans in order to achieve the goal. However, the way that the teams' plans are constructed differs from theory to theory. For example, Shared Plans specifies the use of complete plans in which agents know the steps that need to be taken to achieve the goal: plans in which agents are assigned to actions (an agent causes a state transition by performing an action). Shared Plans also allows partial shared plans which represent a specification of the minimal requirements that agents need to have in order to take part in the collaboration [1]. Joint Intentions differs in that a plan is built as agents commit to performing actions towards achieving the goal: if one agent performs an action, another agent is also committed to doing its share [14]. Joint Responsibility theory involves a joint commitment to a common plan, once it is established that the agents involved fulfill a set of pre-imposed collaboration requirements. A plan specifies how agents should behave in order to achieve a goal, thus answering the question of how the joint goal will be achieved [15]. The fact that agent teams also use plans in order to achieve their goals extends the applicability of our proposed algorithms to disrupting teams' plans.

We choose to represent sets of plans as sequences of states that are brought about by uni-directional state transitions. Each plan has one starting state and one goal state. The starting state is unique in a set of plans, but, multiple different goal states are allowed. Based on this description, a set of plans can be represented as a set of paths in a directed, acyclical graph. Thus, the problem of finding a set of state transitions that need to be disrupted such that no plans to achieve a goal remain, is translated into the problem of finding a set of edges to cut in a graph such that one node (the start state) is part of a different sub-graph to a set of other nodes (the goal states).

One approach to finding cuts that ensure a disconnection between two nodes in a graph is using the Maximum Flow Minimum Cut theorem in conjunction with a maximum flow algorithm such as Ford-Fulkerson [16] or Goldberg and Tarjan's [17] approaches. The Maximum Flow Minimum Cut theorem specifies that the maximum flow directed from a source node to a sink node in a flow network (a directed, weighted graph) is equal to the minimum capacity that, if removed in a specific way from the network, results in a situation where no flow can be directed from the source node to the sink node. In other words, if we find the edges that can no longer be used to direct flows, those edges constitute the minimum cut. Ford-Fulkerson and Goldberg-Tarjan algorithms can be used to determine the maximum flow in a network and proceed towards finding a minimum cut. However, currently our model does not use any edge weights for the graph used to represent a set of plans. Using this approach and assigning flow values for each edge ends up indirectly prioritizing state transitions to be cut based on the values assigned. To avoid this, we leave this approach for further work, once edge weights are introduced to account for resources needed to prevent transitions.

A second approach to get the minimum cut in a graph is the non-flow based approach. Algorithms such as Stoer-Wagner [18] and Karger's [19] algorithm can be used to determine cuts of edges between a start node and an end node such that no path between the nodes is available. These algorithms are meant to provide the cut of minimum weight from a graph and work on weighted graphs. We devise our own approach inspired by Karger's algorithm and adapted for unweighted graphs because our model does not involve edge weights at this time. Furthermore, rather than obtaining just one cut, we aim to provide more options, to address scenarios where a specific cut cannot be applied (i.e. one of the state transitions cannot be prevented for whatever reason).

## 3 Representing Plans

We represent a plan as a sequence of transitions between states. We denote the set of all states as $\mathcal{S}$ and assume a set of possible state transitions, where a state transition is simply a pair of states. A *plan* is then a sequence of state transitions, the application of which causes the overall transition from the start state to the goal state of the plan.

**Definition 1.** *A* **plan** *with* **start state** $s_0 \in \mathcal{S}$ *to achieve* **goal state** $s_g \in \mathcal{S}$ *given* **possible state transitions** $T \subseteq \mathcal{S} \times \mathcal{S}$ *is a sequence of state transitions*

$$[(s_0, s_1), (s_1, s_2), \ldots, (s_{n-1}, s_n), (s_n, s_g)]$$

*where* $n \geq 1$ *and for all* $i$ *such that* $0 \leq i \leq n-1$, $(s_i, s_{i+1})$ *and* $(s_n, s_g) \in T$. *The set of all plans with start state* $s_0$ *to achieve goal state* $s_g$ *given possible state transitions* $T$ *is denoted* $\mathsf{Plans}(s_0, s_g, T)$.

We are interested in preventing a goal from being achieved; if there were only one state that achieves the goal and only one plan to achieve that goal state, it would be sufficient to prevent a state transition from that plan. However, often there are several states in which the goal is achieved and multiple plans to achieve those goal states from a particular start state. We define a *plan base* for a particular start state, set of goal states and set of possible state transitions as a graph that captures all possible plans that can be used to achieve one of the goal states.

**Definition 2.** *The* **plan base** *with* **start state** $s_0 \in \mathcal{S}$, **goal states** $G \neq \emptyset \subseteq \mathcal{S}$ *and* **possible state transitions** $T \subseteq \mathcal{S} \times \mathcal{S}$, *denoted* $\mathsf{PlanBase}(s_0, G, T)$, *is the graph* $(N, E)$ *such that* $N \subseteq \mathcal{S}$, $E \subseteq T$ *and the following conditions hold.*

1. *For all* $s_g \in G$, *if there exists* $[(s_0, s_1), \ldots, (s_{n-1}, s_n), (s_n, s_g)] \in \mathsf{Plans}(s_0, s_g, T)$, *then for all* $i$ *such that* $0 \leq i \leq n$, $(s_i, s_{i+1}) \in E$ *and* $\{s_i, s_{i+1}\} \subseteq N$.
2. *If there exists* $(s_x, s_y) \in E$, *then there exists* $s_g \in G$ *such that* $[(s_0, s_1), \ldots, (s_{n-1}, s_n), (s_n, s_g)] \in \mathsf{Plans}(s_0, s_g, T)$ *and there exists* $i$ *such that* $0 \leq i \leq n$, $s_x = s_i$ *and* $s_y = s_{i+1}$.
3. *If there exists* $s_x \in N$, *then there exists* $s_g \in G$ *such that* $[(s_0, s_1), \ldots, (s_{n-1}, s_n), (s_n, s_g)] \in \mathsf{Plans}(s_0, s_g, T)$ *and either* $s_x = s_g$ *or there exists* $i$ *such that* $0 \leq i \leq n$ *and* $s_x = s_i$.

Given a particular plan base, in order to try to prevent the achievement of a goal, a disruptor can make certain state transitions impossible. This results in a *disrupted plan base*. If the disrupted plan base is empty, *full disruption* has been achieved. Otherwise, if there does not exist a plan of shorter or equal length than the shortest plan pre-disruption, *partial disruption* has been achieved. *Partial disruption* translates into a delay in the achievement of the goal.

**Definition 3.** *The* **disrupted plan base** *that results from* **applying the prevention of state transitions** $T' \subseteq \mathcal{S} \times \mathcal{S}$ *with regard to* **start state** $s_0 \in \mathcal{S}$, *goal states* $G \subseteq \mathcal{S}$ *and* **possible state transitions** $T \subseteq \mathcal{S} \times \mathcal{S}$, *is denoted* $\mathsf{DisruptedPlanBase}(s_0, G, T, T')$ *such that* $\mathsf{DisruptedPlanBase}(s_0, G, T, T') = \mathsf{PlanBase}(s_0, G, T \setminus T')$. *A disrupted plan base* $\mathsf{DisruptedPlanBase}(s_0, G, T, T')$ *can then be said to be* **fully disrupted** *iff* $\mathsf{DisruptedPlanBase}(s_0, G, T, T') = (\emptyset, \emptyset)$.

In order to try to bring about a fully disrupted plan base, a disruptor must identify which state transitions it could prevent. If preventing a particular set of state transitions produces a fully disrupted plan base, that set of state transitions is a *critical set*.

**Definition 4.** *Let* $s_0 \in \mathcal{S}$, $G \neq \emptyset \subseteq \mathcal{S}$ *and* $T' \subseteq T \subseteq \mathcal{S} \times \mathcal{S}$. *The set of state transitions* $T'$ *is a* **critical set** *with regards to* **start state** $s_0$, *goal states* $G$ *and* **possible state transitions** $T$, *denoted* $T' \in \mathsf{CriticalSets}(s_0, G, T)$, *iff* $\mathsf{DisruptedPlanBase}(s_0, G, T, T')$ *is fully disrupted.*

Preventing a critical set of transitions of a plan base results in full disruption; however, in some cases this may not be possible, either because some of the critical transitions cannot be successfully prevented or because there are some possible state transitions that are unknown to the disruptor. Nevertheless, some disruption may occur if certain plans to reach the goal are no longer possible. In the following section we define a metric for measuring disruption.

## 4 Measuring Disruption

In order to fully disrupt a plan base, it is not enough to disrupt only one of its plans. All plans need to be disrupted such that there is no possibility of bringing about any of the goal states. However, a plan base can also be considered disrupted if the length (i.e. the number of transitions) of the shortest plan post-disruption is larger than the length of the shortest plan pre-disruption (we call this *partial disruption*). We define a *disruption metric* (DM) to measure the disruption of a plan base. DM is calculated as a function of the shortest plan lengths in a plan base pre and post-disruption.

**Definition 5.** *We denote the* **length** *of a plan* $p = [(s_0, s_1), \ldots, (s_n, s_g)]$ *as* $\mathsf{Length}(p)$ *such that* $\mathsf{Length}(p) = n$.
*We denote the* **minimum length** *of a plan base* $P = \mathsf{PlanBase}(s_0, G, T)$ *as* $\mathsf{Min}(P)$ *such that* $\mathsf{Min}(P) = \mathsf{Length}(p)$ *where* $p = \underset{p \in \{\mathsf{Plans}(s_0, s_g, T) | s_g \in G\}}{\operatorname{argmin}} \mathsf{Length}(p)$.
*The* **disruption metric** *that results from* **applying the prevention of state transitions** $T'$ *with regards to* **start state** $s_0$*, goal states* $G$ *and* **possible state transitions** $T$ *is denoted* $\mathsf{DM}(s_0, G, T, T') \in [0, 1]$ *such that:*

- $\mathsf{DM}(s_0, G, T, T') = 1$ *iff* $\mathsf{Plans}(s_0, G, T \setminus T') = \emptyset$*, else*
- $\mathsf{DM}(s_0, G, T, T') = \frac{\mathsf{Min}(P') - \mathsf{Min}(P)}{\mathsf{Min}(P')}$ *where* $P' = \mathsf{PlanBase}(s_0, G, T \setminus T')$ *and* $P = \mathsf{PlanBase}(s_0, G, T)$.

As mentioned in Definition 5, DM is a value in the interval $[0, 1]$. A DM value of 0 indicates no disruption of the plan base, while a DM value of 1 indicates full disruption. A DM value in the interval $(0, 1)$ indicates partial disruption of the plan base: there exists a longer plan from the start state to a goal, post-disruption.

Both partial and full disruption can be measured using DM. In the next section we propose four algorithms that each determine a critical set of state transitions to be prevented in order to cause the disruption of a plan base.

## 5 Algorithms for the Disruption of Plans

Identifying a set of critical state transitions in a plan base $PlanBase(s_0, G, T)$ can be abstracted to finding one cut of edges in a directed, acyclical graph such that there is no path between a node $s_0$ and any nodes in the set $G$.

Cutting algorithms such as Ford-Fulkerson [16], Stoer-Wagner [18], Karger's [19] or Goldberg and Tarjan's approach [17] have been considered as a starting point for

developing a solution. In the case where one always needs to determine the minimum set of state transitions that need to be prevented (the minimum cut) in order to cause disruption, adapted versions of these algorithms can be used. However, as previously mentioned, we do not always need to find the minimum set of state transitions to be prevented and because using some of the above-mentioned algorithms adds unnecessary complexity to our problem, we have chosen to determine any cut that may cause disruption rather than finding just the minimum cut, for the time being. We propose four algorithms that can be used to obtain cuts: the *Start Cut*, *Goal Cut*, *Random Cut* and *Approximate Minimum Cut*. The performance comparison for all algorithms with regard to DM is presented in detail in Section 8.

### 5.1 Start Cut

The *Start Cut* algorithm can be used to determine the set of all edges that are directed out of a given start node of a directed unweighted acyclical graph representing a plan base. It takes a directed unweighted acyclical graph $PlanBase(s_0, G, T) = (N, E)$ as an input and adds all of the edges directed out of the start node to the cut set, as shown in Algorithm 1. It then returns a set of edges that can be cut out of the graph (the cut set) such that the start node and goal nodes are always part of two different sub-graphs.

**Algorithm 1:** StartCut: returns the set of all edges directed out of a start node in a graph (a start node has no inward edges)

**Data**: $PlanBase(s_0, G, T) = (N, E)$
**Result**: startCut

```
startCut = ∅;
for i ∈ 0,..,(size of N)-1 do
    if (s0, si) ∈ E then
        startCut = startCut ⋃ {(s0, si)};
    end
end
return startCut;
```

### 5.2 Goal Cut

The *Goal Cut* algorithm can be used to determine the set of all edges that are directed into any node of a given set of goal nodes of a directed unweighted acyclical graph representing a plan base. It takes a directed unweighted acyclical graph $PlanBase(s_0, G, T) = (N, E)$ as an input and adds all of the edges that are directed into any of the goal nodes to the cut set, as shown in Algorithm 2. It then returns a set of edges that can be cut out of the graph (the cut set) such that the start node and goal nodes are always part of two different sub-graphs.

**Algorithm 2:** GoalCut: returns the set of all edges directed into the goal nodes of a graph (a goal node has no outward edges).

```
Data: PlanBase(s0, G, T) = (N, E)
Result: goalCut
goalCut = ∅;
for i ∈ 0,..,(size of N)-1 do
    for every s_j ∈ G do
        if (s_i,s_j) ∈ E then
            goalCut = goalCut ⋃ {(s_i,s_j)};
        end
    end
end
return goalCut;
```

### 5.3 Random Cut

The *Random Cut* algorithm is used to determine a random set of edges to be removed from a directed, unweighted, acyclical graph (representing a plan base) such that no path exists between a start node and a set of goal nodes. As shown in Algorithm 3, it takes an input consisting of a directed, unweighted, acyclical graph (the $PlanBase(s_0, G, T) = (N, E)$), and is inspired by Karger's algorithm, where a single edge is randomly chosen for contraction at every step, until only one edge is left in the graph. However, our method does not use edge contraction. Instead, we split the nodes in the graph into two sets. The first set, $S_1$ initially contains the start node and the second set, $S_2$, initially contains all of the goal nodes. At every step a node (that is not part of any set) is chosen at random and added to one of the sets (randomly chosen). When there are no more nodes to be added to a set, all of the edges $(s_i, s_j)$, where $s_i \in S_1$ and $s_j \in S_2$ are added to the Random Cut set. The algorithm returns the set of edges representing the Random Cut, as shown in Algorithm 3.

### 5.4 Approximate Minimum Cut

The *Approximate Minimum Cut* algorithm is an alternative to the *Random Cut* algorithm. It can be used to determine an approximation of the minimum set of edges to be removed from a directed, unweighted, acyclical graph (representing a plan base) such that no path exists between a start node and a set of goal nodes. As shown in Algorithm 4, it is inspired by Karger's algorithm and makes use of the fact that, by generating multiple random cuts of a graph, the minimum cut is eventually obtained [19]. The method is named $ApproximateMinimumCut$ as we do not implement Karger's approach regarding the estimated number of iterations of random cutting before a minimum is obtained. We stop after 1000 iterations of the random cut. The method can be improved by adding a convergence test in order to get a more accurate approximation of the minimum cut. The input consists of a directed, unweighted, acyclical graph representing the $PlanBase(s_0, G, T) = (N, E)$. It then runs 1000 iterations of the Random Cut

**Algorithm 3:** RandomCut: finds a cut of randomly chosen edges so that there is no path between a start node $s_0$ and any of the end nodes $G$ in a graph.

**Data**: $PlanBase(s_0, G, T) = (N, E)$
**Result**: randomCut

randomCut = $\emptyset$; $S_1$ = $\emptyset$; $S_2$ = $\emptyset$; $S_1$ = $S_1 \bigcup \{s_0\}$;
**for** $s_g \in G$ **do**
  $S_2$ = $S_2 \bigcup \{s_g\}$;
**end**
stateSet = $N \setminus (G \bigcup \{s_0\})$;
**while** $stateSet \neq \emptyset$ **do**
  index = random of $\{1, 2\}$;
  randomly select $s_j$ from stateSet;
  $S_{index}$ = $S_{index} \bigcup \{s_j\}$;
  stateSet = stateSet $\setminus \{s_j\}$;
**end**
**for** $(s_i, sj) \in E$ **do**
  **if** $s_i \in S_1$ *and* $s_j \in S_2$ **then**
    randomCut = randomCut $\bigcup \{(s_i, sj)\}$;
  **end**
**end**
return randomCut;

algorithm and determines the minimum cut obtained this way. Upon termination, the algorithm returns a set of edges representing the approximate minimum cut.

**Algorithm 4:** ApproximateMinimumCut: finds an approximate minimum set of edges so that no path exists between a start node $s_0$ and any of the end nodes $G$ in a graph.

**Data**: $PlanBase(s_0, G, T)$ = $(N, E)$
**Result**: approximateMinimumCut

approximateMinimumCut = randomCut($PlanBase(s_0, G, T)$);
minCutSize = $\mid approximateMinimumCut \mid$;
**for** *i* $\in$ *1..1000* **do**
  temp = randomCut($PlanBase(s_0, G, T)$);
  **if** $\mid temp \mid < minCutSize$ **then**
    $minCutSize = \mid temp \mid$;
    $approximateMinimumCut = temp$;
  **end**
**end**
return approximateMinimumCut;

The four algorithms presented in this section return a set of edges that represent state transitions to be prevented in order to obtain full disruption of a plan base. Full

disruption is guaranteed under the assumption that the disruptor has an accurate representation of the plan base they want to disrupt. In the following sections we investigate the performance of the proposed algorithms when this is not the case.

## 6 Disruption Under Uncertainty

A disruptor may not always have an accurate view of the plan base it is aiming to disrupt. We consider a particular type of uncertainty, where the disruptor has an accurate view of the states that appear in a plan base but may be unaware of the existence of some state transitions. We acknowledge that there may exist false positive state transitions, that are present in the disruptor's view of the plan base they wish to disrupt but are not part of the disruptee's plan base. Even if included in the set to be prevented, these will not affect the disruption produced, only increasing the number of paths in the graph and as a consequence, the size of the sets of cuts returned by the algorithms. False positive state transitions are not discussed further.

A *disruption scenario* is defined by its start state, the goal states, the possible state transitions, and the possible state transitions that are unknown to the disruptor.

**Definition 6.** *A* **disruption scenario** *is a tuple* $(s_0, G, T, T')$ *where* $s_0 \in \mathcal{S}$ *is the* **start state**, $G \neq \emptyset \subseteq \mathcal{S}$ *is the set of* **goal states**, $T \subseteq \mathcal{S} \times \mathcal{S}$ *are the* **possible state transitions**, *and* $T' \subseteq T$ *is the set of* **unknown state transitions**.

For a particular disruption scenario, we can consider the *true plan base* the disruptor wishes to disrupt and the (possibly inaccurate) *view of the plan base* the disruptor holds.

**Definition 7.** *Let* $ds = (s_0, G, T, T')$ *be a disruption scenario. The* **true plan base** *of* $ds$*, denoted* $\mathsf{TruePlanBase}(ds)$*, is the plan base* $\mathsf{PlanBase}(s_0, G, T)$*. The* **disruptor's view of the plan base** *of* $ds$*, denoted* $\mathsf{DisruptorView}(ds)$*, is a plan base* $\mathsf{PlanBase}(s_0, G, T \setminus T')$.

In order to evaluate the performance of our proposed algorithms under uncertainty, we have run a set of experiments, which we describe in the following section.

## 7 Experimental Set-up

In order to find a set of state transitions to prevent, the disruptor uses the proposed algorithms, passing its view of the plan base as a parameter. In order to determine the effectiveness of each algorithm, we measure the disruption caused when preventing the state transitions returned by the algorithm in the true plan base.

To investigate performance under uncertainty, we also vary the following parameters: number of states $nr^s$, number of goal states $nr^g$, percentage of unknown state transitions $perc^{ukn}$ and density of the plan base $density$. For each run of the experiment, we first generate a true plan base with $nr^s$ states and $nr^g$ goal states. A true plan base is generated in two steps, as described below.

First, we generate a graph $(N, E)$ where $\mid N \mid = nr^s$, $G \subset N$ and $\mid G \mid = nr^g$ and all the following properties hold: there are no cycles: $\nexists (s_i, s_j) \in E$ such that $i \geq j$,

$0 \leq i, j \leq nr^s$, there are no edges leaving goal nodes: $\nexists(s_i, s_j) \in E$ such that $s_i \in G$ and the total number of edges is $\mid E \mid = \frac{(nr^g-1)(2nr^s-1)}{2}$ because each non-goal node $s_i$ has $nr^s - i - 1$ edges leaving it. The formula for the number of edges is derived through an addition of the number of edges leaving each node in the graph that is not a goal node: $(nr^s-1)+(nr^s-2)+\ldots+(nr^s-nr^g+1)$. In order to obtain a graph with these properties, we start with a set of nodes N. For each $s_i \in N$ $i \in [0, nr^s\text{-}nr^g - 1]$ we add outgoing edges $(s_i, s_j)$, where $j \in [i+1, nr^s]$ and $s_j \in N$.

The second step derives a random true plan base from the graph generated in the previous step by removing a number of edges determined by the parameter $density$, where $\mid E' \mid = \frac{|E| \times (100-density)}{100}$. Each edge to be removed $(s_i, s_j) \in E'$ is picked out randomly. If upon removal $\exists(s_i, s_{j'}) \in E$ and $\exists(s_{i'}, s_j) \in E$, where $s_{i'}$ and $s_{j'} \in N$ the edge is removed. If not, then a new edge is picked randomly to be removed. This process continues until $\mid E' \mid$ edges are removed from the graph. The true plan base generated this way has $nr^s$ nodes, $nr^g$ goals and each of the nodes in the graph is on a path from a starting node to a goal node.

From the resulting true plan base $(N, E \setminus E')$, we then generate a disruptor view of the plan base. The following conditions hold for the disruptor's view $(N', E'')$: $E'' \subseteq E$, $\frac{|E''|}{|E|} = perc^{ukn}$ and $N = N'$. The disruptor view is generated through edge elimination, in the same way that the random true plan base is derived from the complete plan base.

We apply each of the four algorithms to determine $T'$, the set of state transitions to prevent. We then remove these state transitions from the true plan base and compute $\mathsf{DM}(s_0, G, T, T')$.

We consider the following set of parameter combinations and perform 100 runs of the experiment for each combination in the set. Each run is the equivalent of a disruption scenario.

$nr^s \in \{10, 60, \ldots, 200\}$
$nr^g \in \{1, 2, \ldots, 0.3 \times nr^s\}$
$perc^{ukn} \in \{0, 10, \ldots, 80\}$
$density \in \{10, 20, 30, 40, 50, 60, 70, 79, 89, 99\}$.

For each combination of parameters we return an average of DM over the 100 runs. Experimental results are discussed in the following section.

## 8 Performance Analysis for Plan Disruption

In order to analyze the performance under uncertainty of the four algorithms for identifying state transitions to disrupt in a plan base we performed the experiment described in Section 7. In this section we present and discuss the results obtained.

### 8.1 DM decreases as number of states in a plan base increases

We averaged the data obtained for each set of plan bases of sizes from 10 to 200 nodes. Thus, one DM value in the plot symbolizes the average of all DM values obtained using a specific cut for all plan bases of a specific number of states and number of goals for all

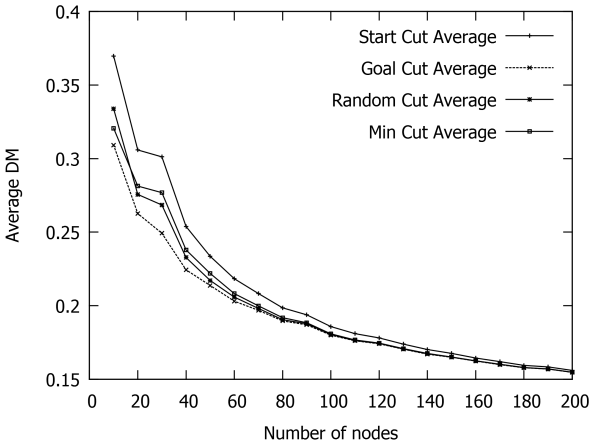


**Fig. 1.** Disruption (DM) vs. number of states (nodes) in a plan base ($nr^s$).

combinations of $perc^{ukn}$ and density covered in the parameter space. The same method applies for all of the following plots as well. For smaller plan bases of sizes 10 to 30 nodes, it can be observed in Figure 1 that the disruption obtained (DM) is slightly higher than for 30 to 200 nodes. However, DM stabilizes and becomes constant starting at 30 nodes. We conclude that the number of states in a plan base influences the performance of the proposed cutting algorithms. As can be observed, the higher the number of states, the lower the performance. However, the peak in disruption for smaller plan bases (up to 30 nodes) may have also been the result of an experimental bias due to the fact that, sometimes, for smaller plan bases the number of edges removed to create the disruptor's version may have been decreased, in order to avoid graph disconnection that would have resulted in new start states or new goal states in a plan base.

### 8.2 DM decreases as unknown state transitions increase

We averaged the data obtained for each set of plan bases grouped by the percentage of unknown state transitions, varied from 10 to 90 percent. An almost exponential decrease of DM can be observed in Figure 2 as the $perc^{ukn}$ increases. As expected, when there are no unknown state transitions in the disruptor's plan base ($perc^{ukn}$ = 0), all four algorithms guarantee full disruption (DM = 1). As the percentage of unknown state transitions increases, DM decreases. Some disruption is still obtained as $perc^{ukn}$ approaches 20. However, disruption becomes negligible as $perc^{ukn}$ approaches 70-80. The dramatic decrease of 80% caused by an increase in unknown state transitions by 20% shows that the disruption of a plan base is heavily influenced by this. The data also suggests that the algorithms for plan disruption yield significant results when the unknown state transitions in a disruptor's plan base is not higher than 20%.

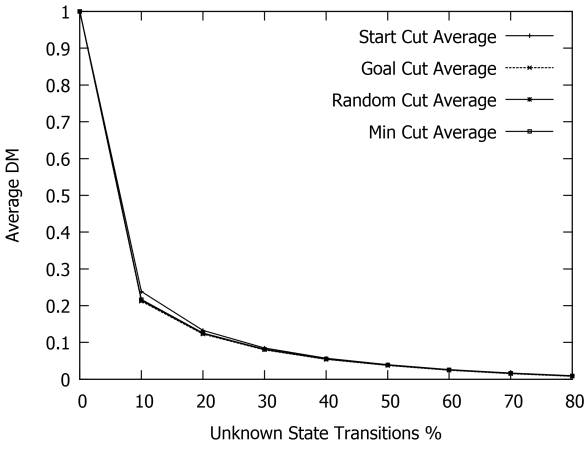


**Fig. 2.** Disruption vs. % of unknown state transitions in a plan base ($perc^{ukn}$).

### 8.3 DM decreases as the number of goals in a plan base increases

We averaged the data obtained for each set of plan bases grouped by the number of goals, varied from 1 to 60. A decrease of DM can be seen in Figure 3 as the $nr^g$ increases. As expected, when there are fewer goals in the plan base, a higher value of disruption is obtained. This may be explained by the fact that, when a single goal exists more state transitions in the plan base are shared by plans that lead to the goal, than in the case of multiple goals. In such cases, the removal of one transition does cause disruption of multiple plans. Based on the experimental data, we conclude that the probability of disruption decreases as the number of goals in a plan base increases.

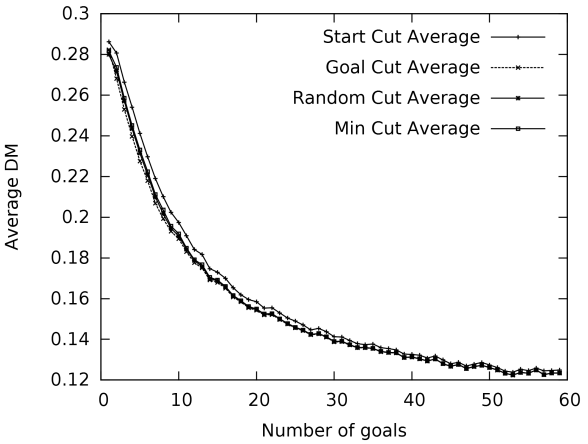


**Fig. 3.** Disruption vs. number of goals in a plan base ($nr^g$).

### 8.4 DM decreases as the density of a plan base increases

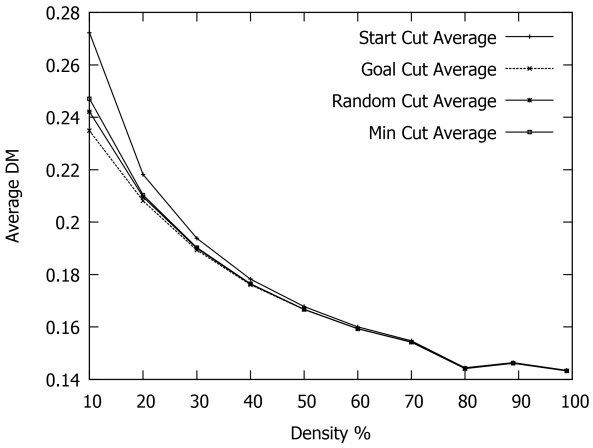


**Fig. 4.** Disruption vs. density of a plan base (density).

We averaged the data obtained for each set of plan bases varying from 10 to 99 percent density. The density of a plan base is represented by the percentage of the state transitions that can be performed in a plan base out of all possible state transitions that could be performed in a plan base. A significant decrease of DM can be observed in Figure 4 as the density of plan bases increases. Some disruption may still be obtained for denser plan bases, however, we notice a significant decrease in disruption around the density of 80 percent. This may be explained by the fact that, as the plan base density increases, the number of possible state transitions available also experiences an increase (the number of state transitions available in a plan base of size $nr^s$ that has $nr^g$ goals is $2^{(nr^s - nr^g)}$).

## 9 Conclusion and Future Work

In this paper we have discussed the concept of disruption of plans as a way to prevent or delay an agent or team of agents from achieving a goal. We argue that a set of plans can be disrupted by preventing particular state transitions. We have discussed two kinds of disruption (partial and total), provided metrics for the disruption of plans and proposed four algorithms to identify which state transitions should be stopped such that the achievement of goals is prevented. We have also considered the fact that the disruptor may not always have an accurate representation of the disruptee's plan base and provided an experimental analysis of algorithm performance under uncertainty.

Based on the experimental data we make the following observations.

1. The number of states in a plan base affects algorithm performance: DM decreases as the number of states increases.

2. The number of unknown state transitions in a plan base impacts algorithm performance: DM decreases as these unknown state transitions increase.
3. The number of goals in a plan base exhibits patterns of inverse exponential dependency: DM decreases as the number of goal states in the base increases.
4. DM obtained using the algorithms decreases as plan base density increases.
5. As observed in Figures 1–4, all four proposed cutting algorithms perform similarly, overall, apart from *Start Cut* which seems to have a slight advantage.
6. All of the proposed algorithms have a chance to cause disruption in any scenario covered in our experiment. This increases as the number of goals decreases, percentage of unknown state transitions decreases and density of a plan base decreases.

The time performance of the algorithms depends on the number of states $(N)$, transitions $(E)$ and goals $(G)$ of a plan base. The Start Cut algorithm has a worst case performance of $O(N)$. The Goal Cut performs in $O(N \times G)$ worst case while the Random Cut's worst case performance is $O(max(N, E))$ and the Approximate Minimum Cut worst case performance is $1000 \times O(max(N, E))$ because the algorithm iterates Random Cut 1000 times. Depending on the known values of parameters $N$, $E$ and $G$ for a specific disruption scenario, a best performing algorithm in terms of execution time can be determined for the scenario.

The time performance discussion and observations of algorithm performance measured through DM are ultimately to help a disruptor decide whether using any of the proposed algorithms to identify a set of transitions to be prevented in order to disrupt a set of plans is sensible. For example, it would be pointless to use our algorithms when $90\%$ of the transitions in a plan base are unknown to the disruptor; also, if a minimum cut is needed and time constraints on obtaining such a result are loose, then the disruptor can use the Approximate Minimum Cut algorithm, which yields a more accurate minimum cut than the Random Cut algorithm but at a higher time cost.

Future work aims to provide a formal model of disruption and investigate whether it is possible to derive a formula of failure for each cutting algorithm including parameters such as number of states, number of goals, percentage of state transitions unknown to the disruptor and density of a plan base. In addition, the current plan base model is to be extended to incorporate resources required to prevent transitions. A further larger scale experimental analysis (for up to 1000 states in a plan base) and comparison of the performance of our proposed algorithms versus classical cutting algorithms such as Tarjan's and Ford-Fulkerson will be carried out once we introduce resources (e.g. time required to prevent a transition) as edge weights in the plan base. Since re-planning occurs in real-life, we will also extend our work to repeat the prevention of critical actions at every step in the execution of the plan to be disrupted.

Having determined what state transitions can be disrupted in a plan base, we need to address the question of how these state transitions can be disrupted. We propose the study of disruption using norms as the following step in our work, once the formal model of a disruption scenario has been provided. We also plan to address other ways of preventing state transitions in further research. These include hindering communication links between agents in a team and depleting resources required for specific transitions identified using the algorithms proposed in this paper.

## References


1. Grosz, B.J., Hunsberger, L., Kraus, S.: Planning and acting together. AI Magazine **20**(4) (1999) 23–34
2. Georgeff, M.P., Lansky, A.L.: Reactive reasoning and planning. In: Proceedings of the 6th National Conference on Artificial Intelligence. (1987) 677–682
3. d'Inverno, M., Kinny, D., Luck, M., Wooldridge, M.: A formal specification of dmars. In: Proceedings of the 4th International Workshop on Agent Theories, Architectures, and Languages. (1998) 155–176
4. Voinitchi, A., Black, E., Luck, M.: Introduction to team disruption mechanisms. In: Proceedings of the 2012 Imperial College Computing Student Workshop. (2012) 149–155
5. McDermott, D.: Robot planning. AI Magazine **13**(2) (1992) 55–79
6. Hanks, S., McDermott, D.: Modeling a dynamic and uncertain world: symbolic and probabilistic reasoning about change. Artificial Intelligence **66**(1) (1994) 1–55
7. Porteous, J., Sebastia, L.: Extracting and ordering landmarks for planning. Journal of Artificial Intelligence Research **22**(1) (2004) 215 – 278
8. Carbonell, J.G.: Counterplanning: A strategy-based model of adversary planning in real-world situations. Artificial Intelligence **16**(3) (1981) 295 – 329
9. de Cote, E.M., Chapman, A., Sykulski, A.M., Jennings, N.: Automated planning in repeated adversarial games. In: Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence. (2010) 376–383
10. Willmott, S., Richardson, J., Bundy, A., Levine, J.: Applying adversarial planning techniques to go. Theoretical Computer Science **252**(12) (2001) 45 – 82
11. Huang, H., Ding, J., Zhang, W., Tomlin, C.J.: A differential game approach to planning in adversarial scenarios: A case study on capture the flag. In: Proceedings of the 2011 IEEE International Conference on Robotics and Automation. (2011) 1451–1456
12. Tambe, M.: Towards flexible teamwork. Journal of Artificial Intelligence Research **7**(1) (1997) 83–124
13. Pynadath, D.V., Tambe, M.: Multiagent teamwork: analyzing the optimality and complexity of key theories and models. In: Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems. (2002) 873–880
14. Jennings, N.R.: Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. Artificial Intelligence **75**(2) (1995) 195–240
15. Jennings, N.R., Mamdani, E.H.: Using joint responsibility to coordinate collaborative problem solving in dynamic environments. In: Proceedings of the 10th National Conference on Artificial Intelligence. (1992) 269–275
16. Fulkerson, D.R., Ford, L.R.: Maximal flow through a network. Canadian Journal of Mathematics **8**(1) (1956) 399–404
17. Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum-flow problem. Journal of the ACM **25**(4) (1988) 921–940
18. Stoer, M., Wagner, F.: A simple min-cut algorithm. Journal of the ACM **44**(4) (1997) 585–591
19. Karger, D.R., Stein, C.: A new approach to the minimum cut problem. Journal of the ACM **43**(4) (1996) 601–640