Abstract

Theoretical and empirical work in behavioral economics has shown that time inconsistency may lead to irrational and sub-optimal behavior, such as procrastination, abandonment of tasks, or inefficient changing of plans mid-task. These phenomena can be modeled as a planning problem on a task graph whose vertices represent states and whose weighted edges represent actions with costs. Without proper motivation, agents often abandon their task partway through the graph, or if they reach their destination node, they do so inefficiently, by a path other than the minimum-cost path. This paper proposes methods for motivating agents to complete their tasks efficiently successfully by assigning intermediate rewards to nodes of the graph. We demonstrate a simple, fail-safe motivational technique that motivates the agents at the actual cost of all their actions, but we show that in many cases a graph designer can motivate agents at less than the actual cost of all their actions. To develop our method we introduce new concepts of graph following that reflect agent intention as well as behavior. We formalize our methods algorithmically and make them more robust to practical considerations, such as when the graph designer is uncertain as to the agent’s true level of present bias.

I would like to thank Professor Elizabeth Penn for advising my thesis, and more generally for the part she has played in inspiring my interest in computational models of social science phenomena. I also thank Professors Bruce and Dorothy Petersen for their patience and helpful feedback throughout the whole thesis-writing process. Finally, I must acknowledge the debt of gratitude I owe to the many friends and family members who have supported me throughout the entire process of capping off my undergraduate education (and throughout the rest of college and life).
1 Introduction

Traditional economic theory often assumes that agents behave rationally, taking the mathematically correct approach to maximizing their utility, however it may be defined. In practice, however, this is often not the case. The field of behavioral economics attempts to explain the gap between the predictions of traditional economic theory and empirical findings. This involves challenging the traditional assumption of perfect rationality, instead making other assumptions. Behavioral economics typically constructs and verifies models using mathematical or experimental methods, but recent research has also introduced computational and algorithmic paradigms and methods.

Kleinberg and Oren [1] pioneer the latter approach, introducing a novel mathematical behavioral model along with algorithmic techniques to analyze it. To represent agent behavior more rigorously, they model it as a directed graph, whose vertices represent states of the world and whose edges are that take agents from one state to another. An agent begins at a starting node, an initial state where the task has not been completed, and seeks to travel through the graph by taking intermediate actions toward completing the task and reaching a goal node. The edges may be weighted by the cost of performing an action, in which case agents incur a nonnegative cost representing the effort an action takes.

Finding an efficient way to complete the tasks—a minimum cost path through the graph—can be done with any of several graph search algorithms from the standard computer science literature. Traditional economic theory would assume that a rational agent, capable of planning via graph search algorithms, would correctly find and consistently follow such a path. Kleinberg and Oren change by making a key assumption as alternative to rationality: their agents can still do graph search, but they are time-inconsistent, their relative evaluation of options possibly changing over time. Thus, though the costs of actions never changes, the agents may discount costs using a scheme that leads them to prefer options that they did not prefer earlier, or to reject options that they had earlier planned to take. Indeed, Kleinberg and Oren show that in some cases, this effect may lead agents to behave in inefficient ways paralleling common real-world phenomena: procrastination, changing to sub-optimal courses of action partway through a task, and beginning an undertaking and failing to complete it.

This begs the question: although left to their own devices, agents under these assumptions may behave suboptimally, can they be induced to behave optimally? Kleinberg and Oren propose as open directions for future work different possible means of motivating agents to complete tasks efficiently. This thesis follows up on one such direction by allowing an overseer to administer intermediate rewards. In their own work, Kleinberg and Oren sometimes allow an agent to earn a final reward upon completing the entire task, but to our knowledge we are the first to propose intermediate reward schemes. We study ways in which a benevolent (and wealthy) overseer can tailor the reward structure more finely, offer more intermediate rewards to compensate the agents more promptly for taking desirable actions. The goal is for the overseer to induce optimal behavior, motivating the agent to complete the task and to do so as efficiently as possible, while paying out as little in rewards as possible.

Methods for efficiently inducing efficient behavior have many applications. For example, a government may be willing to administer aid to its own states or to foreign nations, but it wants them to take beneficial but initially expensive steps toward self-improvement. Other examples featuring the public sector might include retirement savings plans and applications
in educational policy and public health. Conceivably, people would benefit by investing money into saving for the future or time into schooling, or by making less indulgent but healthier choices that may lead to an improved quality of life in the future. Kleinberg and Oren also use as a running example, pertinent to education and economic considerations thereof, a professor trying to motivate students to complete assignments (instead of turning them in late or dropping the class). Such an example becomes of practical importance on a much larger scale when we consider massive online open courses, whose potential benefits at providing widespread education at low cost are enormous, but where attrition is a well known phenomenon. Fundamentally, Kleinberg and Oren’s model and our reward methods could be applied to any scenario in which time-inconsistent agents face multi-step tasks where the costs begin to be incurred long before the payoff, but an overseer has a desire and the resources to help them do so efficiently.

2 Background and Related Work

The problem that we consider can be grounded in a larger history of work that combines economics with computer science. We also review particular ideas from economics that are relevant to the problem at hand. Background material on fundamentals of computer science, including techniques for analysis of algorithms and specific graph search techniques, can be found in the appendix.

2.1 Economics and Computation

Many areas of economic theory lend themselves naturally to algorithmic methods, and as such recent years have seen an explosion in the amount of literature at the intersection of economics and computation. We list a few examples:

Algorithmic Game Theory: Game theory studies behavior in strategic environments. Algorithmic game theory considers a variety of aspects of game theory from an algorithmic perspective. For instance, techniques from theoretical computer science may be used to compute Nash equilibria of games, or agents may be allowed to use algorithms to specify their strategies. Algorithmic game theory also has applications to many different areas of computer science, including network routing, auctions for online advertisements, and security. A more thorough treatment of techniques and applications of algorithmic game theory can be found in [13].

Algorithmic Mechanism Design: The inverse problem of game theory, mechanism design considers techniques for designing games so that they have desirable properties. In particular, a good mechanism should be somewhat resilient to adverse consequences of suboptimal agent behavior, for example from strategic manipulation. Algorithmic techniques, first proposed by [6] can also be used here: for example, algorithmic game theory as applied to computer security may consider how an intruder would try to break into a computer system, and algorithmic mechanism design would try to design a system to withstand such attacks. Algorithmic methods may be used both to design the system and verify its soundness, ideally that it makes the intruder’s job impossible or computationally infeasible.

Computational Social Choice: Social choice theory focuses on making decisions based on
the aggregation of heterogeneous preferences. Voting theory, for example, is one particular branch of social choice theory with clear applications to politics, but also many other applications. Common issues in voting theory and social choice more broadly include: making consistently good choices, giving all agents’ preferences a fair share of power to influence the outcome, being strategy-proof (or robust to strategic manipulation by submitting false preferences), and dealing with the possible impracticality of accurately obtaining or working with complete preference lists over all candidates. Computational social choice may try to design a protocol that, despite these issues, selects reasonable candidates given agents’ interests. Such a voting system would have to be responsive to agents’ preferences and also robust to quirky (or dishonest) preferences. Matching theory is similar, but instead of the end result being a general set of outcomes for everyone, each agent is matched with an item or another agent. Many of the same considerations apply, but the stability of the matchings (no two parties both have an incentive to switch with each other) is also important.

Other active areas of interest include: computational fair division (the problem of assigning parts of a divisible item or subsets of a collection of indivisible item to agents with heterogeneous preferences), incentives in crowdsourcing and human computation, prediction markets and reputation systems, and so on.

The problem that we consider, motivating agents with rewards, is fundamentally a mechanism design problem: we want to design a system to have desirable properties (induce good behavior). However, we do not consider the problem of strategic manipulation, where agents are so cunning as to know their own preferences perfectly but possibly submit false preferences if they think they can do even better by doing so. Agents in our model have behavioral biases in their preferences (more specifically, their means of discounting costs and rewards) that could lead them to make inefficient decisions: in essence, they are their own worst enemy. In this way, our work also shares the goal of social choice, to analyze the response of a system to agents’ preferences and design a robust, responsive system.

2.2 Time Inconsistency

Kleinberg and Oren account for inefficient ”irrational” behavior in their agents by assuming that the agents are time-inconsistent. Importantly, agents’ options at a given stage $X$ remain the same as they thought they would when they were at an earlier stage $Y$ trying to reason about $X$. Time inconsistency says that agents’ relative preferences between their options may change over time. That is, if an agent at stage $X$ has options $x_1$ and $x_2$, the agent may prefer $x_1$ to $x_2$ at an earlier stage $Y$ when it is looking ahead to $X$, but when it actually reaches $X$, it may change its mind and prefer option $x_2$ to $x_1$. To expand Kleinberg and Oren’s university course example, students enrolled in a class may, during the first week of the class considering about their final exam at the end of the semester, think it would be a good idea to spend reading period diligently studying each day for their exam instead of watching a favorite television show. When reading period actually comes, however, the immediate gratification of watching TV may be more attractive than having to actual study, and so the student might elect to watch TV instead.

Time inconsistency is also observed and studied in experimental psychology. More generally, people may prefer ”virtues”, or actions that have a higher immediate cost but greater long-term benefits, to ”vices”, or actions that have a lower immediate cost or a higher reward
but lower long-term benefits, when they are considering their options in advance, but reverse their preferences in the heat of the moment [15]. In an empirical study, participants chose to watch a “highbrow” movie such as Schindler’s List (i.e. a virtue, in the context of this study) statistically significantly more often when they had to choose the movie to watch in advance of watching it. In contrast, they chose “lowbrow” movies such as The Breakfast Club far more often when they chose the movie they day they watched it [15].

The emphasis on gains to be had in the present over whatever may come in the future leads to one mathematical way of modeling time-inconsistency: quasi-hyperbolic discounting [3]. Agents with a quasi-hyperbolic discounting scheme discount all future periods by the same parameter $\beta \leq 1$, possibly in addition to a standard exponential discount factor $\delta \leq 1$. In this case, a reward gained $D$ periods into the future would be discounted by a factor of $\beta \delta^D$. This is also known as present bias: an agent automatically values the present as $\frac{1}{\beta}$ times more valuable than any future period, regardless of how far it is into the future or what the traditional exponential discount factor $\delta$ is.

To demonstrate how a quasi-hyperbolic scheme can capture time-inconsistency, consider an agent with a present bias factor $\beta = \frac{1}{2}$ in period 1 registering for a class (an action that has cost zero, though its cost is irrelevant in this case). In period 2, the agent is assigned work: the agent can either do the work diligently and incur cost of 100, or do the work less diligently (watching more TV instead) and incur cost of 50. In period 3, the agent takes the examination: an agent who did the work will incur only a minor cost of 20 from the mild fatigue of taking the exam, but a less diligent agent will incur cost of 100 from doing poorly. Of course, in period 1 when registering for the course, the agent discounts both the timelines where it studies and where it doesn’t, and thus it prefers the timeline where it studies (incurring cost of $\frac{1}{2}(100 + 20) = 60 < \frac{1}{2}(50 + 100) = 75$). In period 2, when deciding which timeline to follow (studying or not studying), the agent actually changes its mind and prefers to not study. This is because it no longer discounts the second period, and so it perceives the cost of not studying as $50 + \frac{1}{2}(100) = 100 < 100 + \frac{1}{2}(20) = 110$. Thus, the agent ends up changing its relative, not just its absolute, evaluation of its lifestyle choices (studying versus not studying) depending on in which time period it is considering them. In this case, as in many of Kleinberg and Oren’s examples, this in fact leads to sub-optimal (more costly) choices.

Note that the traditional exponential discounting scheme, where a cost or reward incurred $D$ periods into the future is discounted by a factor $\delta^D$, is not time-inconsistent. In exponential discounting, which is standard in microeconomic theory, agents’ absolute valuation of a reward or cost changes depending on how many periods into the future it is incurred. However, their relative evaluation of options they could choose in the same time period will not change: for options $A$ and $B$ and utility function $\mu$, it is obvious that if $\mu(A) \geq \mu(B)$, then $\delta^D \mu(A) \geq \delta^D \mu(B)$ for positive $\delta$. This is true for any value of $D$: whether they are considering the options a single period in advance or thousands.

Quasi-hyperbolic discounting has been used to model retirement savings [4] and behavior in social security systems [5]. For theoretical models involving quasi-hyperbolic discounting, it is often customary to let $\delta = 1$ to isolate the effects of the present bias [1, 15].
3 The Reward Problem

Kleinberg and Oren [1] propose a task graph framework for present-biased agents as a modeling tool, but they leave most of the concerns of design—how to design a graph or accessorize it with features such as rewards so as to induce efficient behavior from agents—as open questions. This section builds on their work in the direction of one of their open questions: how to best outfit a graph with intermediate rewards. The section begins by introducing new concepts and extending existing ones: in particular, we allow Kleinberg and Oren’s framework to include intermediate rewards. We then introduce new concepts of agent planning and behavior that are sensitive to the more extensive reward structure before using these new concepts to demonstrate results that are applicable to the graph designer’s problem.

3.1 Terminology

Consider a problem where an agent with present bias parameter $\beta \leq 1$, navigating a task graph $G$, incurs cost $c(i, j)$ from traveling from any node $i$ to an adjacent node $j$. However, nodes (including the destination) also have rewards for reaching them: the reward for reaching any node $k$ is given by $r(k)$.

An agent’s goal is to find a path through the graph that obtains the desired reward, hopefully as profitably as possible. A path $P$ can be written as a totally set of $N_P$ (sometimes abbreviated as $N$ if the context makes clear which path is being discussed) nodes: $\{n_1, \ldots, n_{N_P}\}$. A $k$-prefix of $P$ is the first $k$ nodes in $P$: $\{n_1, \ldots, n_k\}$. A $k$-suffix of $P$ is the last $k$ nodes of $P$: $\{n_{P_{N-N}}, \ldots, n_{P_N}\}$.

An agent at a starting node $s$ considers it worthwhile to set out for a goal node $t$ if and only if it finds a path where the total rewards are at least as great the total costs (both discounted as appropriate). Such a path has length $N$ for some $N \geq 1$. For notational convenience, we can regard $s$ as $n_1$ and $t$ as $n_N$ in the path’s formulation. Consider paths of length $N \geq 2$, such that the start and goal nodes are distinct, or else the problem of motivating an agent to reach the goal is solved before it has begun. Say that a path $P$ motivates an agent to move if the cost it incurs by moving forward is no greater than the reward it immediately obtains at the next node plus the discounted difference between all future rewards and costs it anticipates incurring. Formally, this is when $c(n_1, n_2) \leq r(n_2) + \beta \sum_{i=3}^{N} (r(n_i) - c(n_{i-1}, n_i))$.

A reward motivates an agent to move if it turns a previously non-motivating path into a motivating one. An agent is motivated to move if there exists some path $P$ from $s$ to $t$ that motivates the agent.

If more than one motivating path exists, the agent has more than one profitable way to reach the goal, and so it plans to follow the most profitable path. Let $P'$ be this most profitable path, which maximizes $r(n_2) - c(n_1, n_2) + \beta \sum_{i=3}^{N} (r(n_i) - c(n_{i-1}, n_i))$ over all choices of $n_2, \ldots, n_{N-1}$ for some $N$ that is the length of a path to the destination node $t$. We say that the agent plans this path $P'$, since at the agent’s current state it appears to be the most attractive option.

Note that the agent’s first action will be to take edge $(n_1, n_2)$ to end up at $n_2$, where $n_1, n_2 \in P'$, but after that it will re-evaluate: now $c(n_2, n_3)$ and $r(n_3)$ are “the present” and...
will not be discounted by $\beta$, so high costs or rewards at these nodes could seem even more prohibitive or appealing, respectively. This may in some cases cause dramatic changes in the agent’s plans at different stages and lead to the ”irrational” phenomena that are of interest in behavioral economics.

Because time inconsistency introduces a new layer of complexity to an agent’s behavior, we introduce new concepts for the notion of path following that can capture more of the nuances of a time-inconsistent agent’s plans as well as behavior. Let $\mathcal{P}$ be the path the agent ends up taking, where by definition the agent completed the task if and only if the last node in $\mathcal{P}$ is the destination. Finally, an agent \textit{strongly follows} a path $P$ if $P = \mathcal{P}$ and for all $i < N - 1$, the agent plans to take $(n_j, \ldots, n_N) \subseteq P$ for all $j > i$. By contrast, an agent \textit{fails to follow} a path $P = (n_1, \ldots, n_N)$ if, for every node to the agent actually travels, it did not find itself at $n_i$ planning to take $(n_i, \ldots, n_N)$ for some $i$. These are likely familiar concepts of path following; the agent has the same intention to take or not take the path, respectively, at every step.

We introduce two more concepts of path following that capture not only the agent’s actual behavior, but also its intentions. An agent \textit{partially follows} a path $P$ if for some $n_i \in P$, the agent found itself at $n_i$ planning at that point to take $(n_i, \ldots, n_N)$, but note that it $P$ may or may not equal $\mathcal{P}$. An agent \textit{weakly follows} a path $P$ if $P = \mathcal{P}$ but note that there may or may not exist $i < N - 1$ such that, for some $j > i$, the agent was not planning to take $(n_j, \ldots, n_N) \subseteq P$. Note that partial following may have the same outcome as failing to follow a path (the agent does not take the path), and weak following may have the same outcome as strong following (the agent takes a path). However, as will become clear, it can be very useful to understand and try to appeal to the agent’s intentions at critical points in time, instead of focusing only on its ultimate behavior.

We present an example graph in Figure 3.1 that demonstrates the various path following concepts and their usefulness in analyzing time-inconsistent planning.

Suppose an agent starting at $s$ whose goal node is $t$ has present bias parameter $\beta = \frac{1}{2}$ and is presented with this graph. We note that it evaluates the cost of path $(s, a, t)$ as $6 + \beta(2) = 6 + \frac{1}{2}(2) = 7$. If the agent chooses to move toward $a$, it will have no choice but to continue on to $t$. Thus, it will strongly follow the path $(s, a, t)$ and fail to follow all the rest of the paths. Note that if an agent strongly follows one path, it must fail to follow all the other paths, as if it ever had any intention of following another path, this would preclude
the definition of strong following.

In fact, the path \((s,a,t)\) is clearly the minimum cost path from a non-discounted standpoint. To the present-biased agent, however, the path \((s,b,c,t)\) looks equally good; it evaluates the cost of the path \((s,b,c,t)\) as \(2 + \beta(8 + 2) = 2 + \frac{1}{2}(10) = 7\), the same cost as \((s,a,t)\). Thus, the agent has its choice of shortest paths. If it instead chooses to continue to \(b\), intending to follow \((s,b,c,t)\), it reaches another crossroads. Here the agent can continue to \(c\) as planned, incurring a discounted cost of \(8 + 2\beta = 8 + 1 = 9\), or it can deviate to \(d\), incurring a discounted cost of \(4 + 8\beta = 4 + 4 = 8\). At \(b\), then, the agent finds it cheaper to switch to \(d\), so it does so. It therefore only partially follows the path \((s,b,c,t)\), intending to take it at node \(s\) but abandoning it before the goal. Once at \(d\), of course, its only option is to continue on to \(t\). Thus, the agent ends up taking the path \((s,b,d,t)\). Note that the agent weakly followed this path, since at \(s\) it intended to take the path \((s,b,c,t)\) instead and only changed its mind later.

As we can see, the agent’s choice of \((s,b,d,t)\) leads it to incur a cost of \(2 + 4 + 8 = 14\), which is in fact the highest cost path from \(s\) to \(t\). Its present bias opened up the possibility of it making one suboptimal choice to start (moving to \(b\)), and led it to change its mind and make a subsequent suboptimal switch later on as well (switching to \(d\) instead of continuing to \(c\)).

Note that in this case, the agent had its choice of shortest paths. In the event of a tie, we assume that the graph designer can break ties however it wants. It could do so, for example, by putting a reward of any \(\epsilon > 0\) on any node along the path it wants the agent to follow. In this case, however, we left out any discussion of rewards for simplicity of illustration; we assumed that the agent had no choice but to continue in some way to \(t\).

To conclude this section, we make two quick but helpful observations. The first follows immediately from the definitions and the fact that the only way for an agent to respond to an increased reward at a node is to plan to visit that node. Either the agent will plan to visit that node, or it will not change its previous plan because even the increased reward is not high enough to induce it to change its mind.

**Proposition 1.** If placing a reward \(r(i)\) at a node \(i\) leads an agent to reach the destination node \(t\), then the agent must partially follow a path \(P\) such that \(i \in P\).

Next, we observe that for the subgoal of motivating an agent to move from node \(n_i\) to \(n_j\), where \(i < j \leq N\), putting the reward farther in the future than \(j\) never motivates the agent more. This is straightforward because an agent is biased toward the present to not discount rewards immediately received (so if a reward is put very close in the future it may not discount it), but discounts future period costs and rewards equally.

**Proposition 2.** Let \(1 \leq i < j < k \leq N\) be such that \(n_i, n_j, n_k\) are all on the path \(P\) from \(n_1\) to \(n_N\). If \(r(n_k)\) makes \(P\) motivate an agent to move from \(n_i\) to \(n_j\), then placing the same amount of reward as \(r(n_k)\) at \(n_j\) would also make \(P\) motivate the agent to move.

Of course, after the agent claimed the reward at \(n_j\), it would need new motivation to continue moving forward from \(n_j\), which the designer would have to take into account. Nevertheless, from the standpoint just of getting the agent to move forward one more step, it makes sense to issue a reward sooner rather than later.
3.2 Motivating Strong Following

Suppose a designer wants to use rewards to induce agents to follow a path and reach a goal node, perhaps the lowest cost path, while having to spend as little a reward as possible. The strongest and most intuitive concept of path following is strong following, and this section concentrates on inducing such behavior.

By Proposition 1, we know that a reward can only motivate an agent to reach a node if the agent plans to take a path through the node on which the reward was placed. By definition, an agent strongly follows a path if and only if at each node it plans to take that path. The next result for strong following follows immediately.

**Lemma 3.1.** To motivate an agent to strongly follow a path \( P(n_1, \ldots, n_N) \) by placing rewards, the only rewards that can make \( P \) motivating, if it is not already, are \( r(n_i) \) for some \( i \) such that \( 2 \leq i \leq N \).

**Theorem 3.2.** The most efficient way to motivate an agent to strongly follow a path \( P = \{n_1, \ldots, n_N\} \) is to assign \( r(n_i) = c(n_{i-1}, n_i) \forall i, 2 \leq i \leq N \). The total payout in rewards is thus \( R_{\text{strong}} = \sum_{i=1}^{N-1} c(n_i, n_{i+1}) \).

**Proof.** This follows directly from Lemma 3.1 and Proposition 2.

According to Theorem 3.2, the cheapest way to induce an agent to strongly follow a path is simply to reimburse the agent’s costs for taking each step, in full and immediately. Thus, if the graph designer insists on inducing strong following in an agent, their task would be simple: the graph designer finds the least-cost path using standard graph search techniques and sets a reward on each node of the path that is exactly equal to the weights of the edge on the path leading to the node. Note that this result does not depend on a particular value of \( \beta \).

It is easy to formulate this method algorithmically. A graph designer can motivate any agent, regardless of present bias parameter, to follow any path in the graph by simply setting the reward at each node to be the cost of the edge in the path leading to that node.

**Algorithm 1** STRONG-FOLLOW-REWARD(G,path)

```plaintext
for node in 1 to LENGTH(path) - 1 do
    r(path[node+1]) = c(path[node],path[node+1])
```

Strong following is important in many cases: for example, if at some point the agent has only one path to the goal, it must be strongly following that path the rest of the way to the goal: there is no other path it could intend to follow. Very simple graphs may well leave only the possibility of strong following. Furthermore, strong following is important as a benchmark in our subsequent work, because it is a very conservative requirement. In our analysis of a graph designer’s ability to induce weak following, we want to see what happens if we relax the requirement that agents must always intend to follow the path that they actually take.

Essentially, the difference between weak following and strong following is a question of the agent’s cognitive states. In both cases, the agents ultimately take the same path, but
while an agent that strongly follows a path is resolute in doing what it set out to do, an agent that weakly follows a path must have changed its mind partway through, abandoning a different path that started out the same way. The question of allowing for weak following is: with less stringent criteria of the agents’ cognitive states, can we ever achieve the same efficient behavior with lower total reward payouts?

### 3.3 Weak Following

This section contains our most important result: a graph designer can often encourage an agent to follow a path by paying out less than the cost to the agent of following the path. Specifically, the method we describe is applicable when the agent has multiple paths to the goal node that share the same $k$-prefix for $k \geq 1$: they start out the same for the first $k$ steps, up till the $k$-th node which is a fork in the road, a place where multiple paths that were the same now diverge. One of these paths to the goal node is the “target” path, the path that the graph designer actually wants the agent to follow. Because the graph designer is trying to induce efficient behavior, the target path should be the least-cost path from the start node to the goal node.

Our key finding is that the graph designer, instead of inducing an agent to strongly follow this target path, is better served by placing a large reward on another “decoy” path. The idea is that if the decoy reward is large enough, the agent will take the first $k$ steps without requiring payment, but if it is not too large, the agent can be induced to switch at node $k$, the fork. Of course, this is not completely a free lunch. To get the agent to forgo the reward it was originally pursuing, the graph designer must pay out a higher reward on the target path right after the fork, a switching bonus as it were, than it otherwise would have to (if only the cost of that action were considered). Nevertheless, we show that with a judicious choice of decoy reward and switching bonus, the graph designer still saves on reward payments.

#### 3.3.1 Conditions for Using The Weak Following Method

We note first of all that weak following only makes sense when the agent’s present-bias parameter $\beta \in (0, 1)$. If $\beta = 1$, the agent has no preference between the present and the future and so the problem reduces to a least-cost path finding problem. On the other hand, if $\beta = 0$, the agent cares nothing for the future and only values rewards and costs received the very next period. Thus, only an immediate and full reward will induce it to incur any costs at all, and the graph designer is forced into the strong following strategy denoted by Theorem 3.2: reimbursing costs immediately and completely. Thus, in our analysis we assume that $\beta \in (0, 1)$.

When placing the decoy reward, the graph designer must make sure that the decoy reward must not be claimable after the agent switches away from the path on which it was placed. This means first of all that the decoy reward cannot be placed on the goal node itself, as the only paths the agent ever takes are ones that reach the goal. Additionally, it must not be possible for the agent to switch away from the decoy path at one fork, but switch back to it at a later fork. It is easy to verify that the agent will be unable to do this if the underlying undirected graph, the task graph where all directed edges between states become undirected, is acyclic.
Many important classes of graph satisfy this desirable property (that their underlying undirected graph is acyclic). For simplicity, we will only analyze graphs that do not have edges going from any of the nodes on the target path to any of the nodes on the decoy path after the fork. This is the case in many commonly seen classes of graphs, such as trees. A tree-shaped task graph has the form of a tree up to penultimate nodes along paths to the goal, which have the goal node as a child. These would be leaves in a tree, but here of course they have the goal node as a child. In trees, the root node (which could be the start node in our case) has indegree zero, and every node has indegree one, meaning that there is only one path from the start node to any other node (except, in our case, the goal node, which can be reached from any of the leaves). Indeed, our example graphs will be tree-shaped task graphs.

In practice, we note that a graph designer can abide by this condition for a task graph of arbitrary topology by making rewards path-dependent, not just state-dependent. That is, a graph designer can issue a reward conditional upon the agent not just reaching a certain node, but also reaching that node by a specific path. In this way, the decoy reward can be made conditional on (weakly or strongly) following the decoy path; thus, when the agent switches away from the decoy path to the target path, it ipso facto forfeits the decoy reward. Essentially, for the purposes of issuing rewards this makes the task graph tree-shaped; just as in a tree-shaped task graph there is only one path from the start node to a non-goal node, so also with path-dependent rewards there is only one path to a given node that an agent can follow to earn a reward at that node.

Thus, the approach that we will describe permits a simple state-based reward scheme for many graphs and is applicable using a more complicated path-based reward scheme to any graph.

Additionally, it is worth noting that with two exceptions, the decoy reward can be placed anywhere on the decoy path. Its precise placement does not matter in the sense that the graph designer never intends for the agent to claim it; the reward is just there so that the agent will see it and evaluate the path as worth following for a time. The precise placement does matter, however, in two ways. First, unless the graph designer is making rewards path-dependent, the decoy reward should not be placed on the goal node; otherwise, it could be claimed from any other valid path to the goal node, including the target path. Second, the decoy reward should not be placed right after the fork. Otherwise, the agent will not discount it on the fork, anticipating that it will receive it in the next period, and so the agent will not switch away from the decoy path so easily. Thus, we also require that the decoy path have at least two other nodes other than the goal node after the fork with state-dependent rewards, and at least one other node with path-dependent rewards (so that the reward will still be discounted by the agent at the fork, but cannot be claimed from the target path).

We summarize the conditions for the weak following method below:

(1) \( \beta \in (0, 1) \).

(2) An agent can only claim rewards placed on a given path by weakly or strongly following that path. This can be achieved by one of two ways:

(a) Making the rewards path dependent.
(b) Considering only graphs whose underlying undirected graph is acyclic.

(3) A reward can be placed two nodes after the fork along a decoy path and not be claimable from the target path.

### 3.3.2 Weak Following at a Fork

In this section we derive appropriate values for a decoy reward and a switching bonus by considering an arbitrary graph of the most basic form that still permits weak following. That is, we consider a graph with two paths to the goal: the target path that the graph designer wants the agent to follow, and the decoy path that the graph designer will have the agent partially follow so that it will eventually weakly follow the target path. During the part of the path that the agent follows the decoy path, the target path and the decoy path must of course be the same since the agent ends up taking the target path. Thus, the graph has a fork (a node with multiple children, or outdegree greater than one) where the target branch and the decoy path diverge. The continuations of the decoy path and the target path can be thought of as branches (paths beginning at the fork).

Abstractly, such a graph has exactly two paths from a start node $s$ to a goal node $t$ that share the same first $k$ nodes and diverge from the $k + 1$-node until the goal node. We illustrate this abstract form in Figure 3.3.2.

A cost function $c(i,j)$ is a weighting scheme for the edges such that $c(i,j)$ is the cost of taking the action represented by the edge $(i,j)$. Additionally, for consistency of notation, we denote $s$ by $n_1$ and $t$ by $n_{k+1}$.

In this case, there is a least-cost path from $s$ to $t$, which will be the target path. Denote the path by $P$ and the decoy by $P'$. The nodes of $P$ after the fork $n_k$ are denoted without primes and the nodes of $P'$ after the fork are denoted with primes. Since the nodes $n_1, \ldots, n_k$ are part of both $P$ and $P'$, we can equivalently denote them as $n_1, \ldots, n_k$ or $n'_1, \ldots, n'_k$, depending on which path we are referring to.

By Theorem 3.2, the graph designer would have to spend at least $\sum_{i=1}^{k+l-1} c(n_i, n_{i+1})$ to guarantee strong following of $P$. For a graph satisfying the conditions in the previous section, we now show a way for the graph designer to induce weak following of $P$ by choosing a suitable decoy reward.

The decoy reward should be large enough to entice the agent, at every step up to the fork, to take another step. Mathematically speaking, if a total reward of $r'$ is to be put on
the decoy path, we write this as
\[ r' \geq c(n_j, n_{j+1}) + \beta \sum_{i=j+1}^{k+l'-1} c(n'_i, n'_{i+1}) \]
for all \( j, 1 \leq j < k \). This expression represents the total cost the agent evaluates the remainder of the path to the goal as having at a given step. Of course, if the reward on the decoy path motivate the agent to move forward even at the step at which it perceives the highest cost, it will motivate the agent to move forward at every other step up to the fork. Thus, the graph designer should place a reward that the agent will value at
\[ \max_{0 \leq j < k} c(n_j, n_{j+1}) + \beta \sum_{i=j+1}^{k+l'-1} c(n'_i, n'_{i+1}) \]
on the decoy path. Since this decoy reward is in the future, the agent will discount it; thus, the real value of the reward should be
\[ r' = \frac{\max_{0 \leq j < k} c(n_j, n_{j+1}) + \beta \sum_{i=j+1}^{k+l'-1} c(n'_i, n'_{i+1})}{\beta} \]
(1)
The graph designer can place this decoy reward on node \( n'_{k+2} \) to be comfortably after the fork.

Note that if the graph designer thought that the agent would actually claim this decoy reward, it would probably not be a good idea to offer this decoy reward. First of all, it is not placed on the shortest path, unless the decoy path is tied in cost with the target path. Second of all, since the agent originally intends to follow the decoy path, it is at least partially following it; if it claims the reward, it will be strongly following the decoy path, and the most efficient reward solution for that, as characterized in Theorem 3.2, is not a lump sum amount like the decoy reward. Essentially, the graph designer is bluffing with the promise of a too-grand reward that it hopes to never have to pay out. The graph designer can do this thanks to the agent’s present bias: the graph designer knows that while the too-large-for-comfort decoy reward is still a discounted dream, the agent can instead be lured away with a smaller switching bonus in the present.

At node \( n_k \), however, the graph designer wants the agent to switch back to \( P \), which means she must make it more profitable for the agent to follow \( P \) than \( P' \). If the agent follows \( P' \) the rest of the way, it would incur cost \( c' \) of
\[ c' = c(n'_k, n_{k+1}) + \beta \sum_{i=1}^{l'-1} c(n'_{k+i}, n'_{k+i+1}) \]
Of course, it also expects to receive \( r' \) as a decoy reward, which it discounts and values as \( \beta r' \). It thus stands to gain a profit \( p \) given by
\[ p = \beta r' - c' \]
\[
= \max_{0 \leq j < k} (c(n_j, n_{j+1}) + \beta \sum_{i=j+1}^{k} c(n_i', n_i'+1)) - (c(n_k', n_{k+1}) + \beta \sum_{i=k+1}^{k+l'} c(n_i', n_i'+1))
\]
\[
= \max_{0 \leq j < k} (c(n_j, n_{j+1}) + \beta \sum_{i=j+1}^{k} c(n_i', n_i'+1)) - c(n_k', n_{k+1})
\]
\[
= \max_{0 \leq j < k} (c(n_j, n_{j+1}) + \beta \sum_{i=j+1}^{k-1} c(n_i', n_i'+1)) - (1 - \beta)c(n_k', n_{k+1}) \tag{2}
\]

Note that if the lower limit is less than the lower limit of a sum, (e.g. if \( j = k - 1 \) so that \( i \) starts at \( k \) and ends at \( k - 1 \), then we say the sum evaluates to zero.

If an agent were to switch to \( P' \) away from \( P' \) by taking edge \((n_k, n_{k+1})\), it would incur not just cost \( c(n_k, n_{k+1}) \) but also the opportunity cost of forgoing this profit \( p \). Note that if this quantity \( p \) is negative (which might happen if the decoy path has an extremely high cost right after the switch relative to the costs of all the previous actions), then we treat \( p \) as zero. Of course the agent will not perceive it to be profitable to go down the decoy path, and so it will require no switching bonus to switch back to the start path. However, note that if the graph designer does not motivate an alternative path, the agent could just drop out; it will never have to incur negative profit, so the graph designer can never get away with giving it a negative switching bonus, as it were. Thus, the total cost to the agent of taking edge \((n_k, n_{k+1})\), which we can denote by \( C_k \), is

\[
C_k = c(n_k, n_{k+1}) + \max(0, p)
\]

\[
= c(n_k, n_{k+1}) + \max\{0, \max_{0 \leq j < k} (c(n_j, n_{j+1}) + \beta \sum_{i=j+1}^{k-1} c(n_i', n_i'+1)) - (1 - \beta)c(n_k', n_{k+1})\}
\]

Now, in the graph in Figure 3.3.2, the agent has no other path to choose from, so it will be strongly following \( P \) the rest of the way. Thus, we are in a simple case where Theorem 3.2 gives the graph designer’s least cost strategy, so the graph designer should strictly reimburse the costs of all steps taken after node \( n_k \), including the opportunity cost of the step \((n_k, n_{k+1})\).

Note, however, that these are the only rewards that the graph designer has to pay out; in particular, she paid nothing to get the agent to take the first \( k \) steps, since the agent was pursuing a reward that it never claimed.

Thus, the total payout in rewards when is

\[
R_{\text{weak}} = C_k + \sum_{i=k+1}^{k+l-1} c(n_i, n_{i+1})
\]

When \( p \leq 0 \), we have that

\[
R_{\text{weak}} = \sum_{i=k}^{k+l-1} c(n_i, n_{i+1})
\]
Given that as shown in Theorem 3.2, the payout required to motivate strong following is

$$R_{\text{strong}} = \sum_{i=1}^{k+l-1} c(n_i, n_{i+1})$$

we have that the savings from motivating weak following are

$$R_{\text{strong}} - R_{\text{weak}} = \sum_{i=1}^{k-1} c(n_i, n_{i+1})$$

If instead \( p > 0 \), we have that

$$R_{\text{weak}} = c(n_k, n_{k+1}) + \max_{0 \leq j < k} (c(n_j, n_{j+1}) + \beta \sum_{i=j+1}^{k-1} c(n_i', n_{i+1}')) - (1 - \beta)c(n_k', n_{k+1}') + \sum_{i=k+1}^{k+l-1} c(n_i, n_{i+1})$$

$$= \max_{0 \leq j < k} (c(n_j, n_{j+1}) + \beta \sum_{i=j+1}^{k-1} c(n_i, n_{i+1})) - (1 - \beta)c(n_k', n_{k+1}') + \sum_{i=k}^{k+l-1} c(n_i, n_{i+1})$$

Note that this is no higher than, and when edges have positive costs, generally much lower than, the amount that the graph designer would have to spend to induce completion of \( P \) via strong following: \( R_{\text{strong}} = \sum_{i=1}^{k+l-1} c(n_i, n_{i+1}) \).

We can make it easy to compare the two, showing the savings with weak following, by rewriting \( R_{\text{strong}} \). Then we have

$$R_{\text{strong}} = \sum_{i=1}^{j-1} c(n_i, n_{i+1}) + c(n_j, n_{j+1}) + \sum_{i=j+1}^{k-1} c(n_i, n_{i+1}) + \sum_{i=k}^{k+l-1} c(n_i, n_{i+1})$$

$$R_{\text{weak}} = (c(n_j, n_{j+1}) + \beta \sum_{i=j+1}^{k-1} c(n_i, n_{i+1})) - (1 - \beta)c(n_k', n_{k+1}') + \sum_{i=k}^{k+l-1} c(n_i, n_{i+1})$$

for \( j, 0 \leq j < k \), such that the quantity \( c(n_j, n_{j+1}) + \beta \sum_{i=j+1}^{k-1} c(n_i, n_{i+1}) \) is maximized. Then we have the savings of weak following, the amount extra that strong following costs. We break down the savings into three components that all have interesting interpretations.

$$R_{\text{strong}} - R_{\text{weak}} = A + B + C$$

where

$$A = \sum_{i=1}^{j-1} c(n_i, n_{i+1})$$

$$B = (1 - \beta) \sum_{i=j+1}^{k-1} c(n_i, n_{i+1})$$

$$C = (1 - \beta)c(n_k', n_{k+1}')$$
The quantity $A$ is a result of the fact that the agent at each step ignores sunk costs: it considers only rewards and costs it has yet to incur en route to the goal from its current state. Thus, if the agent perceives the cost as being highest at step $j > 1$, then at this step the agent no longer even considers the costs it has incurred from previous steps. Since the decoy reward is set to just balance out the cost at the step the agent perceives as most expensive (since then it will certainly motivate the agent at states where the path to the goal seems less expensive), the costs of the first steps before node $n_j$ no longer need factor into the decoy reward. The lower the decoy reward can be made, the lower the actual reward paid out need be, since a lower switching bonus is needed to nudge the agent back to the target path.

The quantity $B$ at first seems counterintuitive, as we find that the graph designer saves on reward payouts when it makes the agent operate in a “future-thinking” state, looking ahead toward a large future reward instead of demanding smaller intermediate rewards. Proposition 2 tells us that this is not the case in strong following: all else (e.g. reward size) being equal, rewards further in the future never motivate the agent’s next step more effectively than rewards issued sooner. Nevertheless, the mathematical derivation lends itself to a neat intuition. The large decoy reward motivates the agent’s next step again and again—at every step from the step of highest perceived cost $j$ up till the fork node $n_k$. In particular, at step $j$ (where the agent perceives the cost as highest, and whose perceived cost the decoy reward is calibrated to), it discounts the costs of all steps starting at node $n_{j+1}$ onward. Thus, only the discounted costs of the $j + 1 - th$ through $k - 1 - th$ steps feature into the decoy reward, which in turn features into the switching bonus that is actually paid out. In contrast, of course strong following takings into account the full costs of these steps. A weak following approach saves the difference.

The quantity $C$ results from the fact that at the step of highest perceived cost $j$, the agent discounts the cost it will have to take right after the fork along the decoy path. At the fork, however, the agent no longer discounts the cost of the step to $n_{k+1}$, which makes it view the decoy path as somewhat more expensive. The difference can be docked from the switching bonus, as the decoy path being more expensive means a smaller switching bonus will get the agent to forsake it.

This decomposition allows us to choose the appropriate path to make the decoy path, if multiple paths branch off from the fork in addition to the target path (which the graph designer should always make the lowest cost path in accordance with the goal of inducing efficient behavior). Note that in the expression for the savings from our weak following method, $A$ and $B$ depend only on the costs up to the fork node $n_k$, which are the same for all paths. However, $C$ depends on the cost of the next action after the path, $c(n'_k, n'_{k+1})$ for some path $P'$. Thus, to maximize $C$, we should choose the path with the largest cost of the action immediately after the fork: formally, the path $P'$ such that $c(n'_k, n'_{k+1})$ is maximized.

### 3.3.3 Many Branches, Many Forks

The work in the previous section describes a means for a graph designer to motivate an agent. Of course, it only applies to graphs with exactly one fork, whereas task graphs may have many forks. Furthermore, while it works in some cases, we know that weak following does not always perform better than strong following. For example, in a simple task graph
with no forks, an agent can only plan to follow the path that it actually is following: it may have no alternatives. In this section we present an algorithm that handles very simple graphs such as ones that have no forks (whereupon it reverts to strong following), as well as more complex graphs with many forks.

For clarity, we assume that the value of $\beta$ is strictly between 0 and 1. We know that if the agent has a present bias of 0, it is completely myopic and responds only to the complete reimbursements of strong following; alternatively, if it has a present it may have a present bias of 1, in which case the problem reduces to a simple shortest path algorithm. Thus, in practice we could first check to see if the agent has these extreme values of $\beta$, and if so use the simpler methods that we already know.

We furthermore assume that we have methods to find the minimum cost path; see the appendix. Many of these methods (e.g. Dijkstra’s Algorithm) also find all paths from a start node to a destination node, so it is reasonable of us to assume a method that can do this. We also make the very simple assumptions that we have methods to find the length of a path or the size of a set (of decoy paths). These are all commonly accepted computer science techniques, so we do not define them separately. However, we do define separately our methods for setting the decoy and switching bonuses, which we do at each fork. Pulling these out of the main algorithm makes it more readable, and also allows us to change the decoy and switching bonuses more easily if desired (as we will consider later).

Our helper routines for calculating and placing the decoy reward and the switching bonus are simply algorithmic calculations of the decoy reward and switching bonus quantities, given by Equations 1 and 2. Note that although our analysis was for a graph with one fork, our decoy reward and switching bonus can be calculated myopically, without regard for past or upcoming choices at other forks in the graph. This is because rewards offered at previous forks are sunk costs and do not affect the current decision, and similarly after the switch the reward offered at the current fork will also be a sunk cost.

To demonstrate the correctness of this algorithm, or its ability to motivate agents to take the target path, it suffices to show that when rewards are distributed as this algorithm dictates, the agent’s best option is always to take another step along the target path. If the agent is not at a fork, then it will be enticed by a decoy reward chosen (by the arguments of the previous section) to be sufficiently large as to cover the largest cost it might perceive up until the fork. Thus, it will want to move forward toward the decoy reward, and since the target and the current decoy path are the same up until the fork, this step will also take it further along the target path. On the other hand, if the agent is at a fork, Algorithm 4 explicitly matches the profit offered by the decoy reward, so the agent maximizes its perceived utility by switching to the target path for its next step forward.

Note that this algorithm only pays out a switching bonus at each fork, as well as strict reimbursements for all actions between the last fork the the goal (when only strong following works). We know that the switching bonus takes into account the discounted costs of all actions except the first action taken after the start of the current segment; furthermore, we subtract from it the difference between the undiscounted and discounted costs of the first action of the next segment along the decoy path. Thus, as in the previous section this algorithm does no worse than strong following, and in many cases considerably better.

We demonstrate Algorithm 2 on an example task graph given in Figure 3.3.3. Here costs are given as weights on the edges, and we assume that an agent facing this task
Algorithm 2 WEAK-FOLLOW-REWARD(G,s,t,β)

```plaintext
target = FIND-MIN-COST-PATH(G,s,t)
fork = 0
decoys = NULL
for node in 2 to LENGTH(target)-2 do
    possibleDecoys = FIND-ALL-PATHS(G,node,t)
    for path in possibleDecoys do
        if path[node + 1] == target[node + 1] or LENGTH(path) <= node + 2 then
            possibleDecoys = possibleDecoys - path
    if SIZE(possibleDecoys) > 0) then
        fork = node
decoys = possibleDecoys
        break
if fork ≠ s then
    decoy = NULL
decoyCostAfterFork = 0
for possibleDecoy in decoys do
    if c(possibleDecoy[fork],possibleDecoy[fork+1]) > decoyCostAfterFork then
        decoy = possibleDecoy
decoyCostAfterFork = c(possibleDecoy[fork],possibleDecoy[fork+1])
PLACE-DECOY(G,target,decoy,fork,β)
PLACE-SWITCHING-BONUS(G,target,decoy,fork,β)
WEAK-FOLLOW-REWARD(G,target[fork+1],t,β)
else
    STRONG-FOLLOW-REWARD(G,target)
```

Algorithm 3 PLACE-DECOY(G,target,decoy,fork,β)

```plaintext
maxPerceivedCost = 0
for nodeIndex in 1 to fork do
    perceivedCost = c(decoy[nodeIndex], decoy[nodeIndex+1])
    for decoyIndex in nodeIndex+1 to LENGTH(decoy - 1) do
        perceivedCost = perceivedCost + β c(decoy[decoyIndex], decoy[decoyIndex + 1])
    if perceivedCost > maxCost then
        maxPerceivedCost = perceivedCost
decoyReward = \frac{maxPerceivedCost}{β}
r(decoy[fork+2], decoy[1:fork+2]) = decoyReward
```
Algorithm 4 PLACE-SWITCHING-BONUS(G, target, decoy, decoyReward, fork, β)

\[
\text{decoyCost} = c(\text{decoy}[\text{fork}], \text{decoy}[\text{fork}+1])
\]

\[
\text{for decoyIndex in fork+1 to LENGTH(decoy - 1) do}
\]

\[
\text{decoyCost} = \text{decoyCost} + \beta c(\text{decoy}[\text{decoyIndex}], \text{decoy}[\text{decoyIndex} + 1])
\]

\[
\text{decoyProfit} = \text{decoyReward} - \text{decoyCost}
\]

\[
\text{if decoyProfit} < 0 \text{ then}
\]

\[
\text{decoyProfit} = 0
\]

\[
\text{switchingBonus} = \text{decoyProfit}
\]

\[
\text{r}(\text{target}[\text{fork}+1], \text{target}[1:\text{fork}+1]) = c(\text{target}[\text{fork}], \text{target}[\text{fork}+1]) + \text{switchingBonus}
\]

Graph has present bias parameter \(\beta = \frac{1}{3}\). It is simple to verify that the minimum cost path through the graph runs along the top edge of the graph, denoted without any kind of superscripts on the nodes. This is the target path, which we will denote \(P\), and it has cost \(6 + 9 + 9 + 3 + 6 + 6 + 3 = 42\). Note that using Algorithm 1 to make the agent strongly follow \(P\) costs 42. However, we find that the agent can do much better by placing intermediate rewards using Algorithm 2. Note that for clarity of analysis we omit discussion of contingent paths, the specific path that the agent must follow to claim a specific reward. It is understood that decoy rewards’ contingent paths are the decoy paths up to where the reward is placed, and switching bonus’ contingent paths are the target path up to where the switching bonus is placed.

Running Algorithm 2, however, we find the first valid fork at \(n_2\). In addition to the target path, which continues to node \(n_3\), we have two possible choices for a decoy path: \(P^*\) continuing with node \(n_2^*\), and \(P'\) continuing with node \(n_3'\). Of these two choices, we note that \(P^*\) has the higher cost after the fork: edge \((n_2, n_2^*)\) has cost 6, while edge \((n_2, n_3')\) has cost 3, so we choose to motivate \(P^*\) with a decoy reward. Note that this is more expensive than \(P'\) and certainly much more costly than \(P\), so motivating it seems counterintuitive. However, the intention is not for the agent to actually take it to the goal and claim the reward, so the fact that it is a costly path to take is actually advantageous as it means the agent is more easily persuaded to switch away from it, back to the target. Running Algorithm 3, we calculate this decoy reward to be \(\frac{6 + 6(6 + 6 + 3 + 6 + 18)}{\beta} = 3(6 + \frac{1}{3}(42)) = 3(6 + 14) = 3(20) = 60\), so we place this reward an additional step after the fork along \(P^*\): \(r(n_4^*) = 60\).

Motivated by this decoy reward, the agent will move forward until it finds itself at \(n_2\). When it gets to \(n_2\), however, the graph designer wants it to move to \(n_3\). Running Algorithm 4 and recalling that the decoy reward was 60, the graph designer finds that it must place a switching bonus of \(60\beta - (6 + \beta(3 + 6 + 18)) = 60(\frac{1}{3}) - (6 + \frac{1}{3}(27)) = 20 - 15 = 5\) right after the fork along the target path: \(r(n_3) = 5\). Assuming that the agent will have a motivating path to the goal from here, the extra switching bonus will be enough to persuade the agent to continue along the target path, forgoing the decoy \(r(n_4^*)\).

To motivate the agent the rest of the way, we realize that the problem of motivating the agent forward from \(n_2\) can be solved with the same approach as the original problem, where the agent started at \(s\). Observe that if the target path is the minimum cost path from \(s\) to \(t\), then it is also the minimum cost path of all the paths from any subsequent node \(n_i\) to \(t\); otherwise if some other path \(\hat{P}\) were shorter, then the path to \(n_i\) completed from \(n_i\) to \(t\) by \(\hat{P}\) would have been the shortest path. Thus, the continuation of \(P\) from \(n_2\) remains the
target path when Algorithm 2 is called again. Furthermore, all costs incurred up to $n_2$ are sunk and are thus ignored by the agent. Thus, we can call Algorithm 2 recursively to place rewards on the rest of the graph.

We use Algorithm 3 to set a decoy reward $r(n_6) = \frac{1}{\beta}(9 + \beta(3 + 6 + 9 + 3)) = 48$. This requires a switching bonus, as Algorithm 4 finds of $r(n_5) = 48\beta - (6 + \beta(9 + 3)) = 48\left(\frac{1}{3}\right) - (6 + \left(\frac{1}{3}\right)(4)) = 16 - 10 = 6$. Furthermore, since there are no more forks after $n_4$, the graph designer must convince the agent to strongly follow the target path from $n_4$. This requires an additional reward of 6 to be placed on $n_5$, bringing the total reward $r(n_5)$ up to $6 + 6 = 12$. Then also $r(n_6) = 6$ and $r(t) = 3$. Note that the reward on $t$ cannot be claimed from any of the decoy paths; as we have said, we have omitted for the sake of clarity the contingent paths for rewards, but to be perfectly clear here we note that $r(t)$ can only be claimed contingent on the agent reaching $t$ via $P$.

Thus, the total payout is $r(n_3) + r(n_5) + r(n_6) + r(t) = 5 + 12 + 6 + 3 = 26$, which is considerably less than the cost of 42 that would be obtained by motivating strong following. Note that if there were no present bias assumption, then the problem would also relax to a minimum cost path finding problem, in which case the graph designer would again have to pay 42 to motivate the agent. So we find that the agent's present bias allows it to be more easily motivated to efficiently complete its task.

### 3.4 Designing A System When $\beta$ Is Not Fixed

In the previous section, we assumed that the graph designer knew the agent’s type, determined by the extent to which they are present-biased as given by $\beta$, exactly. In practice, however, we may not always be able to make this assumption. For example, the value of $\beta$ that the graph designer uses to calculate the amount of rewards to set may be a noisy estimate of the agent’s present bias parameter $\beta$ instead of its exact value. That is, if the true value of the agent’s present bias parameter is $\beta^*$, we could say that $\beta = \beta^* + \epsilon$, where the noise parameter $\epsilon$ is a random variable with hopefully zero mean (if $\beta$ is an unbiased
estimator of $\beta^*$) and small variance.

Alternatively, there may exist many agents who may have a range of types. While the graph designer might like to customize the task graph to each agent depending on its type, this may not be possible. Perhaps this is because the graph designer needs to avoid accusations of unfairness, or perhaps the graph designer simply cannot reliably tell which agents have which types. Either way, the graph designer is faced with the problem of having to place one set of rewards that motivates agents of many different types. Again, the graph designer wants to motivate efficient behavior insofar as it can, while spending no more than it must.

First, we note that the strong following approach described in Theorem 3.2 will motivate efficient behavior for all agents, since it does not depend on the value of $\beta$. Of course, this strategy may not be desirable, as we know that in many scenarios the graph designer can motivate weak following instead and do better than a rote immediate reimbursement of the agent’s costs.

Note, however, that motivating weak following with Algorithm 2 is a riskier strategy than motivating strong following when $\beta$ is not known for sure. In particular, note that the value of the decoy reward is set to just compensate the agent’s perceived cost at the point where the perceived cost is highest, assuming that the agent has present bias parameter $\beta^* = \beta$ for a value $\beta$ that the graph designer has in mind. Of course, the agent’s true present bias parameter is $\beta^*$, which may well not be equal to $\beta$.

So the agent is presented with decoy reward $r'$ that of course it discounts, valuing it at $\beta^* r'$. At step $j$, the step where it perceives the cost to the goal as highest, it values the decoy reward it gets as

$$\beta r^* = \beta^* \frac{c(n_j, n_{j+1}) \beta + \sum_{i=j+1}^{k+l'-1} c(n_i', n_{i+1}')} {\beta}$$

and the cost to the goal as

$$c^* = c(n_j, n_{j+1}) + \beta^* \sum_{i=j+1}^{k+l'-1} c(n_i', n_{i+1}')$$

It thus expects to make profit of

$$p^* = \beta r^* - c^*$$

$$= \beta^* \frac{c(n_j, n_{j+1}) \beta + \sum_{i=j+1}^{k+l'-1} c(n_i', n_{i+1}')} {\beta} - (c(n_j, n_{j+1}) + \beta^* \sum_{i=j+1}^{k+l'-1} c(n_i', n_{i+1}'))$$

$$= \frac{\beta^*}{\beta} (c(n_j, n_{j+1}) + \beta^* \sum_{i=j+1}^{k+l'-1} c(n_i', n_{i+1}')) - (c(n_j, n_{j+1}) + \beta^* \sum_{i=j+1}^{k+l'-1} c(n_i', n_{i+1}'))$$

$$= \frac{\beta^*}{\beta} (c(n_j, n_{j+1}) - c(n_j, n_{j+1}))$$
So we have that
\[ p^* = \left( \frac{\beta^*}{\beta} - 1 \right) c(n_j, n_{j+1}) \] (4)

. Note that assuming positive costs, this quantity is negative if \( \beta^* < \beta \). If the agent perceives that by moving forward it will obtain a negative profit, of course, it will drop out. So for the agent to continue, this quantity must be nonnegative, which requires that \( \beta^* \geq \beta \).

The agent’s other crucial decision point is at the fork node \( n_k \), where it is deciding between continuing on the decoy path and achieving the decoy reward, or switching to the target path and getting its switching bonus as well as discounted costs. Note that it after the fork, the agent strongly follows the target path, so regardless of its value of \( \beta \), the agent to continue, this quantity must be nonnegative, which requires that \( \beta^* \geq \beta \).

\( \beta \text{ nonnegative. Assuming positive costs, we see that this quantity is nonnegative when } \beta^* \text{, it expects to make only the switching bonus. When the switching bonus is nonnegative, we have:} \)

\[ p^*_\text{target} = \max \left[ 0, \max_{0 \leq j < k} \left( c(n_j, n_{j+1}) + \beta \sum_{i=j+1}^{k-1} c(n_i' \in n_{i+1}) \right) - (1 - \beta) c(n_k', n_{k+1}) \right] \]

. However, by following the decoy path, it also expects to make a profit, assuming the profit is nonnegative:

\[ p^*_\text{decoy} = \beta^* r' - (c(n_k', n_{k+1}) + \beta^* \sum_{i=k+1}^{k+l'-1} c(n_i', n_{i+1})) \]

\[ = \frac{\beta^*}{\beta} \left( c(n_j, n_{j+1}) + \beta^* \sum_{i=j+1}^{k+l'-1} c(n_i', n_{i+1}) \right) - (c(n_k', n_{k+1}) + \beta^* \sum_{i=k+1}^{k+l'-1} c(n_i', n_{i+1})) \]

\[ = \frac{\beta^*}{\beta} \left( c(n_j, n_{j+1}) + \beta^* \sum_{i=j+1}^{k} c(n_i', n_{i+1}) + \beta^* c(n_k', n_{k+1}) - \frac{\beta^*}{\beta} c(n_k', n_{k+1}) \right) \]

\[ = \frac{\beta^*}{\beta} \left( c(n_j, n_{j+1}) + \beta^* \sum_{i=j+1}^{k-1} c(n_i', n_{i+1}) - (\frac{\beta^*}{\beta} - \beta^*) c(n_k', n_{k+1}) \right) \]

Thus we have

\[ p^*_\text{target} - p^*_\text{decoy} = (1 - \frac{\beta^*}{\beta}) \left( c(n_j, n_{j+1}) + (\beta - \frac{\beta^*}{\beta}) \sum_{i=j+1}^{k} c(n_i', n_{i+1}) - (\beta - \frac{\beta^*}{\beta}) c(n_k', n_{k+1}) \right) \]

\[ p^*_\text{target} - p^*_\text{decoy} = (1 - \frac{\beta^*}{\beta}) \left( c(n_j, n_{j+1}) + \frac{\beta^2 - \beta^*}{\beta^2} \sum_{i=j+1}^{k-1} c(n_i', n_{i+1}) - \frac{\beta^2 - \beta^*}{\beta^2} c(n_k', n_{k+1}) \right) \] (5)

The agent will switch to the target path from the decoy path when this quantity is nonnegative. Assuming positive costs, we see that this quantity is nonnegative when \( \beta^* \leq \beta \) and negative otherwise.
A graph designer who is uncertain of the agent’s true level of present bias $\beta^*$, but instead regards the actual values of $\beta^*$ as being drawn from some distribution with probability distribution function $f$, thus encounters a tradeoff. By setting the decoy value using a small $\beta$ such that $\beta^*$ is with high probability greater than $\beta$, the graph designer can make completion (as opposed to abandonment) more likely, at the cost of having to set a higher decoy reward and thus pay out a higher switching bonus. Likewise, when choosing to set the switching bonus with a high value of $\beta$, the graph designer can make efficient completion via the target path (as opposed to the inefficient choice to stick with the decoy path) more likely, but again this requires paying out a higher switching bonus.

There is no prescription for how much extra in payouts a marginal increase of $\delta$ in (efficient) completion rates should be worth to the graph designer. Instead, we assume that the graph designer is exogenously given standards to meet: $\rho_1, \rho_2 \in [0, 1]$ such that the probability that a given agent chooses to move forward in pursuit of the decoy is $\rho_1$, and the probability that an agent would, if it were at the fork choosing between the decoy and the switching bonus, choose to switch to the target path is $\rho_2$. Let $\beta_{\text{high}}$ satisfy $\Pr(\beta^* \leq \beta_{\text{high}}) = \rho_1$, or alternatively $f(\beta_{\text{high}}) = \rho_1$. Likewise, let $\beta_{\text{low}}$ satisfy $\Pr(\beta^* \geq \beta_{\text{low}}) = \rho_2$: $1 - f(\beta_{\text{low}}) = \rho_2$. Then this analysis suggests that the graph designer can meet the standards parametrized by $\rho_1, \rho_2$ using Algorithm 2 by calling the subroutine for setting the decoy, Algorithm 3, with present bias parameter $\beta_{\text{high}}$ and calling the subroutine for setting the switching bonus, Algorithm 4, with present bias parameter $\beta_{\text{low}}$.

As an example, consider a graph designer trying to motivate an agent to move from $s$ to $t$ in the task graph of Figure 3.4. Once again, the decoy path is marked in primes where it diverges from the minimum-cost target path. Here, however, the graph designer does not know the exact value of $\beta$, but instead knows that $\beta \sim \mathcal{N}(0.5, 0.1^2)$. Using standard techniques from probability and statistics, we find that $\beta_{\text{low}} \equiv 0.336$, and $\beta_{\text{high}} = 0.664$. Thus, using Algorithm 3 the graph designer finds it should place a reward of $r_{\text{decoy}} = \frac{4 + \beta_{\text{low}}(4 + 4 + 6 + 6)}{\beta_{\text{low}}} \equiv 31.922$ on node $n'_4$ conditional on the agent following the decoy path up to the respective nodes. Having calculated this adjusted decoy reward, the graph designer uses Algorithm 4 to find the switching bonus it must place on $n_3$ conditional on following the target path. This comes out to be $\beta_{\text{high}}r_{\text{decoy}} - (4 + \beta_{\text{high}}(6 + 6)) \equiv 9.238$. The agent thus places this switching bonus on $n_3$ in addition to the actual cost of 2 for moving to $n_3$, and also places rewards of 8 on $n_4$ and 4 on $t$, all contingent on the agent taking the target path up to the respective nodes.

Notice that the agent has to pay 9.238 to get the agent to take the first three steps, when by inducing strong following the entire way it would have had to pay $4 + 4 + 2 = 10$ for those first three steps. Thus, its savings are not very large, considerably less than they would be if the agent knew $\beta$ for sure. In this case weak following still does yield the graph designer some savings. In some cases—if the variance were high, for example—it may not even be cheaper to use weak following instead of strong following at all. In such cases, the graph designer should of course motivate the target path using strong following, which as discussed previously works independent of the agent’s value of $\beta$.
4 Conclusion and Directions for Future Work

Much of the literature, including Kleinberg and Oren’s original paper [1], examines how time inconsistency makes agents more likely to act inefficiently. In this paper, we present an algorithmic approach to motivate time-inconsistent agents efficiently—at no higher and in many cases lower cost than it would take to motivate them if they were time-consistent. To do so, we introduced new concepts of path following that took into account agents’ intentions (partial following and weak following) as well as their actual choices. We extended our model to take into account practical considerations such as uncertainty over the agent’s true present bias parameter $\beta$.

We hope that our work inspires further productive research, which could go in a few possible directions. First, for our weak following algorithms to apply, we had to make the assumption that rewards, once forfeited, could not be claimed later. This required us to place restrictions on the graph topology or make the rewards contingent on specific paths taken to reach them. To make our method as generalizable as possible, it would be useful to develop an extension of our algorithm that applies to graphs of arbitrary topology and issues rewards simply for reaching a given state.

Further, while we show that our weak following algorithms in many cases do noticeably better than the simple approach based on strong following, this can be formalized. In particular, we have no proof that our algorithm is optimal—that it motivates the agent with as little reward payouts as possible. How close to optimal is our algorithm?

In our analysis, we assume that the target path is always the minimum cost path, and the graph designer will do whatever it takes to motivate the agent to take the most efficient actions. One can easily imagine a scenario in which the minimum cost path cost much more to motivate than another path that was almost as inexpensive. Further work could formalize the tradeoff between efficiency of the agents’ actions and the cost to the graph designer and explore its implications.

Finally, when we consider cases where the graph designer does not know the agent’s present bias, it is interesting to note that as the agent progresses through the graph, the graph designer gains more information about the agent’s present bias parameter. For example, it must be high enough that it was willing to set out in pursuit of the decoy reward in the first place. The probability that it then opts to switch at the fork could be considered as a conditional probability given the information that it made it to the fork in the first place. As the agent makes it to subsequent forks, it continues to reveal that its present bias parameter was not too low that it gave up earlier and not too high that it got sidetracked for good by a previous decoy reward. A graph designer may wish to take this information into account.
when placing decoy and target rewards throughout the graph.

Answering such open questions can strengthen the theoretical justification for our approach and/or make it more robust in many practical settings. We believe our model has many potential applications and also raises interesting questions about present bias itself and planning for present-biased agents.

5 Appendix: Basic Computer Science Concepts

This section covers some basic algorithms ideas to contextualize some of the techniques we discuss or assume in our analysis. A more detailed treatment of algorithm design and analysis can be found in [14].

5.1 Complexity Notation

When we design algorithms, it is important to know how efficiently they run. An algorithm that will run in longer than the age of the universe is not much better than no algorithm at all! Of course, the running time of an algorithm takes into account many things: the size of the input, the machine the algorithm is run on, the efficiency of the implementation, and so on. Generally, we want to abstract away from specific machine or programming details. An exact running time analysis depends on these factors, the exact number of operations the algorithm must perform, and so on. Usually such an analysis is difficult if not impossible, and it is usually not necessary for determining more generally how efficient an algorithm is. Instead, we can abstract away from all these details by expressing the number of operations the algorithm will perform as an approximate function of the input size.

If we have an input of size $N$ (for example, if we want to search a list of $N$ values), we know that our algorithm will take $f(N)$ steps for some function $f$. Maybe we take $2N$ steps to perform some algorithm, but then we also had to do 4 steps at the beginning before we manipulate the input, so that $f(N) = 2N + 4$. It would be tedious (and too dependent on the specific implementation of the algorithm) to count up most algorithm’s operations like this. Moreover, we don’t actually care too much about the exact constants 2 and 4 in this function, since for small $N$ the algorithm will run quickly and for large $N$ the growth of $N$ will dominate the influence of the other constants on the running time. We are concerned about the asymptotic behavior about $f(N)$ as $N$ may get very large, not the particular constants it may contain.

**Big O Notation**: Formally, we need a way to compare $f(N)$ asymptotically with another function $g(N)$. There are many ways to do this; we introduce the most common one, and the one that is relevant for our purposes, here. We say that $f(N) = O(g(N))$ if there exists $M, n_0$ such that $f(N) \leq Mg(N)$ for all $N \geq n_0$. That is, for sufficiently large $N$ (larger than some well-chosen $n_0$), $f(N)$ is bounded by $g(N)$ up to a constant factor $M$. Note that instead of worrying about details like constant factors or matters of implementation, we can quickly see using Big O notation how much the running time of our algorithm will grow.

As an example, note that $f(N) = 2N + 4$ is $O(N)$, since if we choose $M = 3$ and $n_0 = 4$, $3N \geq 2N + 4$. We could also say that $f(N) = O(N^2)$ as the relationship is even more clearly true, though this would not be *asymptotically tight*, in the sense that we found that a slower
growing function \((N)\) also does the job. Note, however, that \(N^2\) is not \(O(N)\), as if we let \(n_0 = M + 1\), \(N^2 \geq (M + 1)N > MN\) for any \(M\).

5.2 Graph Search Algorithms

The problem of finding the shortest weighted path in a graph is a well-known problem in computer science. Several algorithms exist for solving this problem, making different assumptions. We review a few well-known algorithms and the assumptions they make here.

For a graph with vertex set \(V\) and edge set \(E\), Dijkstra’s algorithm [7] finds the shortest path from a start node to a destination node and all other nodes if desired in \(O(|V| \log |V| + |E|)\) time (in time proportional to \(|V| \log |V| + |E|\) up to some constants) with suitable choice of data structure [8], but it requires that the weights on the edges (in this case, the costs of taking those edges) be nonnegative. Note that in some cases, such as the setting in Kleinberg and Oren’s original problem without intermediate rewards, this is a reasonable assumption, as it makes sense that most agents would view tasks as costly to perform and hence need a reward to motivate them.

If edges have negative weights, however, Dijkstra’s algorithm is not applicable and other algorithms should be preferred. One alternative is the Bellman-Ford algorithm [9] which is slower, running in \(O(|V| \cdot |E|)\) time, but allows for some of the weights to be negative.

Note that the Bellman-Ford algorithm will not handle well, and should terminate, if there is a negative-weight cycle in the graph, as traversing that cycle arbitrarily many times will always lower the cost of a path. In the context of our problem, such a cycle occurs when the graph designer rewards agents for making no progress toward the goal (taking a series of actions that have them ending up where they started). Under most reasonable sets of assumptions, this can be treated as a design flaw on the part of the designer, so the discovery of such a cycle should in fact indicate an error.

Other techniques for graph search include the Floyd-Warshall algorithm, which relies on dynamic programming, and A* search, which in practice often achieves faster results for finding the path to a single goal node by informing the search with heuristics. A large-scale industrial-grade implementation of our work might benefit from more investigation to see which graph search algorithm yields the best practical results. Nevertheless, all of them, as far as their assumptions apply to our problem, should work well.

References


