Abstract

A point H lies on a network Q according to some unknown distribution. A Searcher starts at a given point O of Q and moves to find H at speeds which depend on his location and direction. He seeks the randomized search algorithm which minimizes the expected search time. This is equivalent to modeling the problem as a zero-sum hide-and-seek game whose value V is called the search value of (Q, O).

We make a new and direct derivation of an explicit formula $V = (1/2)(\tau + \Delta)$ for the search value of a tree, where τ is the minimum tour time of Q and Δ (called the incline of Q) is an average over the leaf nodes i of the difference $\delta(i) = d(O, i) - d(i, O)$, where $d(x, y)$ is the time to go from x to y. The function δ can be interpreted as height, assuming uphill is slower than downhill. We then apply this formula to obtain numerous results for general networks. We also introduce a new general method of comparing the search value of networks which differ in a single arc. Some simple networks have very complicated optimal strategies which require mixing of a continuum of pure strategies. Many of our results generalize analogous ones obtained for constant velocity (in both directions) by S. Gal, but not all of those results can be extended.
Keywords: search, network, game, minimax, search game, immobile hider

Searching a Variable Speed Network

Steve Alpern and Thomas Lidbetter
Department of Mathematics
The London School of Economics
Houghton Street, London WC2A 2AE, UK

November 2, 2012

1 Introduction

This paper analyzes the search game played between an immobile Hider and a mobile Searcher on a finite network Q with a distinguished root node O. The Hider simply picks a point H in Q (the hiding place). Q is considered as a subspace of Euclidean space, so that the Hider is not restricted to the nodes of Q, but can also choose a point on the arcs of Q. The Searcher chooses a path $S = S(t)$ in Q which starts at O, covers Q, and satisfies a time constraint

$$d(S(t_1), S(t_2)) \leq t_2 - t_1, \text{ for } t_1 < t_2,$$

where $d(x, y)$ is a given quasimetric (satisfying all axioms for a metric except possibly symmetry in x and y) denoting the minimum time required to go *from x to y*. As usual, $d_W(x, y)$ denotes the travel time within the subset W. If d is symmetric ($d(x, y) = d(y, x)$), and thus a metric, then we say that Q is *time-symmetric*. The payoff of this zero-sum game (to the minimizing Searcher) is the *capture time* T given by

$$T(S, H) = \min\{t : S(T) = H\}.$$

Although both players have infinite pure strategy sets, the value V of this game $\Gamma = \Gamma(Q, O, d)$ exists by the usual application of the minimax theorem of Alpern and Gal [5], and we call this number $V(Q)$ the *search value* of Q (or of Q, O, d). In general, both players will require mixed strategies. In the case where Q is a tree studied in Section 4, there are only finitely many undominated pure Hider strategies – namely the leaf nodes. So there the usual minimax theorem would suffice. As long as there is a cycle in the network, all points in the cycle are undominated, and so even the undominated Hider strategies are infinite. In some cases the optimal mixed strategy has infinite support, as in the 'two-arc' network whose solution is given in Theorem 17. In keeping with the original notional convention of S. Gal, we use upper case $S = S(t)$ and H for pure strategies and lower case s and h for mixed strategies. A mixed strategy for the Hider is simply a probability distribution over Q. We consider such a distribution as a probability measure, as that notation is easier, with $h(W)$ denoting the probability the Hider is in the subset W of Q. For mixed Searcher and Hider strategies s and h we define $T(s, h)$ as the expected search time

$$T(s, h) = \int T(S, H) d(s \times h),$$

where $s \times h$ is the product measure.

The game Γ has been widely studied for time-symmetric networks since its formulation by Rufus Isaacs [15]. In this context it has been completely solved for trees (Gal [11]), weakly cyclic networks (Reijnierse and Potter [21]), weakly Eulerian networks (Gal [13]) and networks with an odd number of identical arcs between two nodes (Pavlovic [19]). For general networks there are methods of approximating the solution (Anderson and Aramedia [8]) and analyses of the complexity of the solution (von Stengel and Werchner [22]). More applied work in this area has been carried out by Jotshi and Batta [16]. The topic of search games has been the subject of books by Gal [12], Alpern and Gal [6], and Garnaev [14].

The time-symmetry assumption has recently been dropped in the article [2] of the first author on trees, which gave a recursive method of determining the search value of any tree. Here, we extend that work to general networks by first determining an explicit formula for the search value of a tree and an improved analysis of optimal responses. Our new *search value formula for trees* is $V = (1/2)(\tau + \Delta)$, where τ (called the *tour time*) is the minimum time required to tour the tree and Δ (called the *incline* of Q) is a measure of the asymmetry of the quasimetric d. We define the *height* $\delta(x)$ of a point x in Q as the time difference in getting to and from x with respect to the root O, that is, $\delta(x) = d(O,x) - d(x,O)$. The incline Δ is a weighted average of the heights of the leaf nodes of the tree. The terminology is based on the idea that going up takes more time than going down. The formula for V is easy to obtain independently of the previous work, though we could also have obtained it easily from the recursions in [2]. We choose to adopt the former method so that the arguments of this paper are self contained. We use this formula as one of our tools to extend the analysis of variable speed travel from trees to general networks.

Another idea introduced in this paper is the Arc-Adding Theorem, Theorem 13, which deals with the question of how the search value of a network changes when a new arc is added between points of the original network. By using the Arc-Adding Theorem to relate the search value of a network to the search value of an associated tree and then applying the search value formula for trees, we are able to find or bound the search values of several classes of networks. Our methods also yield a new explicit formula for the value of the Kikuta search game [17] with node searching costs and a simpler derivation of the solution to the foraging (find-and-fetch) search problem of the first author [3].

The main results of this paper concern the determination of the value (or lower bounds on it) for networks other than trees. The main tools for this analysis are the Arc-Adding Lemma and the value formula for trees. We first obtain a complete solution (Theorem 16) for the search game played on a circle where the distance function $d(O, x)$ around the circle is concave in both directions. We then consider the more complicated problem where the circle consists of two arcs from the root to its antipode which have identical travel time functions. It turns out that when the antipode is 'downhill' from the root the solution is quite complicated, requiring both players to use a distributions over a continuum of pure strategies. In particular, the Searcher sometimes goes all the way around the circle but sometimes reverses direction before completing the tour. This is in stark contrast to the way Gal found that Eulerian networks are searched for time-symmetric (not variable speed) networks. We then obtain lower bounds on the value of arbitrary networks by a new technique called *identification-cutting-deleting* involving (i) identifying points on the network, (ii) cutting the resulting loops that are formed and (iii) removing the 'larger' of the two resulting rays. This process gives the general lower bound of Theorem 18, which can be shown (Corollary 19) to generalize a result of Gal [11] for time-symmetric networks. For weakly Eulerian networks, we obtain (Proposition 22) the result that the tree formula $(\tau + \triangle)/2$ is a lower bound if a family of points of the network all lie downhill from the root.

The paper is organized as follows. Section 2 gives our assumptions and notations for travel times. Section 3 presents the general principle of searching regions of higher density first. Section 4 gives our derivation of the search value formula for a tree and applications to the Kikuta game [17]. Section 5 presents the Arc-Adding Theorem, discussed above. Section 6 analyzes the simplest non-tree, a network consisting of a single loop (a circle). This is the only section where variations of travel times within an arc are of importance. Section 7 looks at the circle as consisting of two identical arcs between two points and shows that in certain cases the solution is very complicated, involving backtracking and a continuum of pure strategies. Section 8 uses the Arc-Adding Theorem and the search value formula for trees to obtain lower bounds on the search value of general networks and on weakly Eulerian networks. Section 9 concludes.

2 Assumptions and Notations for Travel Times

The most natural way to define travel times is by having a notion of arc length and to specify two speed functions (one for each direction) along the arcs, piecewise continuously. However it turns out to be easier to work directly from the quasimetric $d(x, y)$ giving the travel time from x to y.

In most cases considered here (in particular, for trees), we will only need to know the travel times from one end of an arc (a^-) to the other (a^+). To this end, we define forward and reverse travel times on a, denoted F_a and R_a, by

$$F_a = d_a\left(a^-, a^+\right),\ R_a = d_a\left(a^+, a^-\right), \text{ and their difference by } D_a = F_a - R_a.$$

(We require the subscript in the form d_a because the shortest time between a^- and a^+ might not be via the arc which connects them.) The orientation of arcs involved in defining F_a and R_a is a matter of choice: for trees, we will always orient arcs away from the root O. For general networks we often choose the orientation so that $F_a \geq R_a$.

3 Searching higher density regions first

This section deals with the problem of searching for an object whose distribution is known. In this case the *search density*, or just *density*, of a region of Q is defined as the probability the Hider is in the region divided by the time taken to search the region. We now give a general analysis of this well known principle of *searching the higher density region* first, taking the particular version established as Proposition 3 of Alpern and Howard [7], and also used in this form in Alpern [2].

We begin by fixing a network Q and a Hider distribution (mixed strategy) h on Q. If $S(t)$ is a Searcher path with cumulative capture distribution $G(t) = \Pr(T(S, H) \leq t) = h(S[0, t])$, then the expected search time, $T(S, h)$ is given by $T(S, h) = \int_0^\infty t\ dG(t)$. Suppose that $a < b < c$, $S(a) = S(b) = S(c)$ and that S searches (probabilistically) disjoint regions A and B of Q in time intervals $[a, b]$ and $[b, c]$, that is, $h(S[a, c]) = h(S[a, b]) + h(S[b, c])$. Here we use measure and function notation so that for example $h(S[a, c])$ represents the probability that the Hider lies in the portion of Q covered by the path S between times a and c. The following theorem considers the question of when the Searcher will do better (reduce

T) by searching in the opposite order. The answer is in terms of what we call the search density. The search density of A is the probability that the Hider is in A (that is, $h(A)$) divided by the time required to search A (in this case $b-a$).

Theorem 1 (Search Density) *Fix a network Q and a Hider distribution h. Suppose S (with cumulative capture distribution G) searches disjoint regions A and B for the first time during time intervals $[a,b]$ and $[b,c]$, while S' searches in the other order (B during $[a,a+(c-b)]$ and A during $[a+(c-b),c]$).*

$$\text{If } \frac{G(c)-G(b)}{c-b} \geq \frac{G(b)-G(a)}{b-a}, \text{ then } T(S,h) \geq T(S',h).$$

In other words the search with higher search density should be carried out first. If two searches have the same search density they can be carried out consecutively in either order.

The proof of this theorem is straightforward, and can be found in [2].

4 The Search Value of a Tree

In this section we take Q to be a rooted tree, and for simplicity, a *binary* tree (one with at most two arcs *out* of any node, degree at most three). Any tree can be made into a binary tree by adding arbitrarily small additional arcs, so this assumption is not critical. We assume that all arcs are oriented away from O.

An earlier paper [2] gave a recursive method for computing the search value of a tree. Here we present for the first time an explicit formula for the search value, using the notion of the *height* of a point x in Q as the difference between the time to reach x from O and the time to return to O from x. That is, the height $\delta(x)$ of a point x (relative to O which has height 0) in Q is given by

$$\delta(x) = d(O,x) - d(x,O). \tag{1}$$

It is clear that the Hider should only hide at leaf nodes, as all other points of Q are dominated by these. It was already shown in [2] (and by Gal [11] for time-symmetric trees) that the optimal hiding distribution over the leaf

nodes is the *Equal Branch Density* distribution e, which we will formally define later. We define the *incline* of Q, denoted Δ, as the mean height of the leaf nodes, with respect to the distribution (notated as a measure) e, that is

$$\Delta = \sum_{i \in \mathcal{L}} e(i) \cdot \delta(i).$$

For a tree, the tour time τ is simply the sum of all the forward and reverse times of its arcs, $\tau = \sum_{\text{arcs } a} (F_a + R_a)$. The main result of this section is the value formula for trees:

$$V = \frac{1}{2}(\tau + \Delta).$$

4.1 Subtrees and optimal strategies

This subsection shows how our notions of e and Δ can be adapted to subtrees and defines the optimal strategies for the players. By a subtree of a tree Q we mean a subset of Q which is itself a tree.

Definition 2 (subtrees σ_z, EBD distribution e) *If z is a node or arc of a rooted tree Q,O, let σ_z,O_z denote the rooted subtree consisting of all points of Q whose unique path to O intersects with z; the root O_z is the unique closest point to O in σ_z. Define the tour time τ_z to be the time taken to tour σ_z (the sum of all the forward and reverse arcs of σ_z). The EBD distribution (measure) e is the unique one concentrated on the leaf nodes that at every branch node x with out arcs a and b gives equal search density to the two branches σ_a and σ_b. That is,*

$$\frac{e(\sigma_a)}{\tau_a} = \frac{e(\sigma_b)}{\tau_b}, \text{ or simply } \frac{e(\sigma_a)}{e(\sigma_x)} = \frac{\tau_a}{\tau_x} = \frac{\tau_a}{\tau_a + \tau_b}. \tag{2}$$

Figure 1 illustrates the calculation of leaf node measure e and height δ on the given tree, which are indicated above each of the four leaf nodes. We recall the convention of putting the travel times F_a and R_a to the left and right of the arc a. The right side is one fourth the tour time of the tree, so its EBD measure e is $1/4$, while the left side's is $3/4$. Similar ideas give the secondary division of $1/4$ into two weights of $1/8$, and of $3/4$ into two weights of $1/2$ and $1/4$. The leftmost δ is calculated as $(4+7)-(5+2)=4$. The weighted average of the leaf node heights $\delta(i)$ is

$$\Delta = \frac{1}{2}(4) + \frac{1}{4}(0) + \frac{1}{8}(-1) + \frac{1}{8}(+1) = 2, \text{ so } V = \frac{1}{2}(\tau + \Delta) = 17. \tag{3}$$

For the moment, ignore the information about arcs a and b. Readers familiar with the earlier paper [2] will note that there the value (of the full tree) could not be calculated without first analyzing the values of the subtrees, which is not what we do here. This approach is forward looking whereas the earlier was backwards looking (using backwards recursion of the value). A similar difference in approach to the Searcher branching pattern (the βs) will be seen later on.

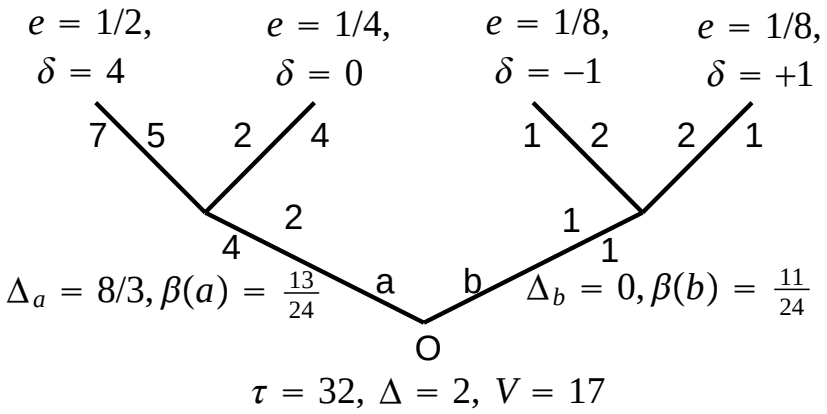

Figure 1. Tree with solution method indicated.

It is useful to extend our notions of $e, \delta, \Delta, \mathcal{L}$ to their values on subtrees.

Definition 3 *For any arc or node z, let $\mathcal{L}_z = \mathcal{L} \cap \sigma_z$ denote the set of leaf nodes of σ_z, let $e_z = e/e(\sigma_z)$ denote the probability measure induced by e on σ_z, and let $\delta_z(x) = d(O_z, x) - d(x, O_z)$ be the relative height of a point x in σ_z. Define the incline Δ_z of σ_z by*

$$\Delta_z = \sum_{i \in \mathcal{L}_z} e_z(i)\delta_z(i).$$

Note that we may rewrite Δ_z in terms of the arcs a in σ_z as

$$\begin{aligned}
\Delta_z &= \sum_{i \in \mathcal{L}_z} e_z(i)\delta_z(i) = \sum_{i \in \mathcal{L}_z} e_z(i) \sum_{\{\text{arcs } a: z \le a < i\}} D_a \\
&= \sum_{\text{arcs } a \in \sigma_z} D_a \sum_{i \in \mathcal{L}_a} e_z(i) \text{ (since } i \in \mathcal{L}_z \text{ and } z \le a < i \iff a \in \sigma_z \text{ and } i \in \mathcal{L}_a) \\
&= \sum_{\text{arcs } a \in \sigma_z} e_z(\sigma_a) D_a \qquad (4)
\end{aligned}$$

We now need a technical result relating to the inclines.

Proposition 4 *If a node x has outward arcs a and b, then*

(i) $\Delta_x = \frac{\tau_a}{\tau_x} \cdot \Delta_a + \frac{\tau_b}{\tau_x} \cdot \Delta_b$, *and*

(ii) $|\Delta_a| + |\Delta_b| \leq \tau_x$.

Proof. For (i) we calculate

$$\begin{aligned}
\Delta_x &= \sum_{i \in \mathcal{L}_x} e_x(i)\delta_x(i) = \sum_{i \in \mathcal{L}_a} e_x(i)\delta_x(i) + \sum_{i \in \mathcal{L}_b} e_x(i)\delta_x(i) \\
&= \sum_{i \in \mathcal{L}_a} e_x(\sigma_a)e_a(i)\delta_x(i) + \sum_{i \in \mathcal{L}_b} e_x(\sigma_b)e_b(i)\delta_x(i) \\
&= e_x(\sigma_a)\ \Delta_a + e_x(\sigma_b)\ \Delta_b = \frac{\tau_a}{\tau_x} \cdot \Delta_a + \frac{\tau_b}{\tau_x} \cdot \Delta_b \text{ by (2).}
\end{aligned}$$

For (ii), we calculate

$$\begin{aligned}
|\Delta_a| + |\Delta_b| &\leq \sum_{\text{arcs } c \in \sigma_a} |e_a(\sigma_c)| \cdot |D_c| + \sum_{\text{arcs } c \in \sigma_b} |e_b(\sigma_c)| \cdot |D_c| \ \text{(by (4))} \\
&\leq \sum_{\text{arcs } c \in \sigma_x} |D_c| = \sum_{\text{arcs } c \in \sigma_x} |F_c - R_c| \leq \sum_{\text{arcs } c \in \sigma_x} |F_c| + |R_c| = \tau_x.
\end{aligned}$$

■

Definition 5 (Depth-first (DF) path) *A depth-first (DF) path in a tree is one that, whenever arriving at a node, always takes an unsearched outward arc, if available; otherwise it takes the unique reverse arc.*

Now we give an explicit definition of the strategy β that was defined recursively in [2] and turns out to be optimal for the Searcher.

Definition 6 (Biased depth-first (BDF) strategy β) *At every branch node x with out arcs a and b, define a probability distribution β over these arcs by the formula*

$$\beta\,(a) = \frac{1}{2} + \frac{1}{2\tau_x}(\Delta_a - \Delta_b). \tag{5}$$

(Since Proposition 4 (ii) shows that $|\Delta_a| + |\Delta_b| \leq \tau_x$, this is indeed a probability.) We interpret the strategy β, called the Biased Depth-first (BDF) strategy, as follows:

1. *When arriving at a branch node for the first time, choose an outward arc a with probability $\beta(a)$.*

2. *When arriving at a branch node the second time, choose the unique untraversed outward arc.*

3. *When arriving at a branch node the third time, choose the unique inward arc.*

The strategy β is an example of a *branching strategy*, which is any mixed strategy that follows the three rules listed above (but perhaps with a probability function different from β). Note that branching strategies produce as sample paths only DF paths. For the tree illustrated in Figure 1, we have $\Delta_a = (2/3)(4)+(1/3)(0) = 8/3$, and $\Delta_b = (1/8)(-1)+(1/8)(1) = 0$, so that $\beta(a) = 1/2 + (8/3)/64 = 13/24$. Note in particular that we computed the optimal initial branching (at the root) without working backwards (unlike our recursive approach in [2]).

4.2 A simple formula for the search value of a tree

We now show that the strategy pair (β, e), the Biased Depth-first strategy for the Searcher and the Equal Branch Density strategy for the Hider, form a Nash equilibrium. Since this is a zero sum game this will imply they are optimal strategies. The formula for the value is then obtained by calculating the expected time for the Searcher strategy β to reach any leaf node. A form of this argument for time-symmetric trees is given in [3] - this idea is exactly the same. The following lemma answers a question arising from [2] as to the full set of optimal responses to e, which could not be answered in that paper.

Lemma 7 *Against the EBD Hider distribution e, the optimal responses for the Searcher are precisely the DF searches. Consequently any mixture of DF searches, such as β, is also an optimal response to e.*

Proof. Suppose S is optimal against e and is not DF. We know that S must search the leaf nodes in some order geodesically, moving on the shortest path from one to another. Let x be a node furthest from O for which S searches the reverse arc c (leading to prior node y) when only one of the forward arcs, a but not b, has been searched. Suppose S reaches x at time t_1, then goes

$$\sigma_a, cS'c, \sigma_b,$$

for some path S'. The paths separated by commas are three disjoint searches (with disjoint arc) which start and end at the node x, so they could be carried out in any order. Since S is optimal, the Search Density Theorem (Theorem 1) implies that their search densities must be non-increasing. The definition of e ensures that the search densities of σ_a and σ_b are the same (they are the branches at a common node), so by the previous remark their density must be the same as that of the intervening search $cS'c$. Now consider the behaviour of S from the earlier time that y is reached just before S goes to x. Here there are three arc disjoint paths, starting and ending at y, carried out in the order

$$c\sigma_a c, S', c\sigma_b c.$$

But these are not in non-increasing order of search density, as S' has a higher search density than $cS'c$ and the other two have lower search densities than σ_a and σ_b. Thus S is not optimal.

Next we want to show that all DF searches S have the same expected time $T(S, e)$ against e, so that they are all optimal responses. Consider the graph with the DF searches as nodes and two searches $S_{a,b}$ and $S_{b,a}$ adjacent if they are identical except that at one node x the first searches the two branches σ_a and σ_b consecutively in that order, while the second uses the other order. Note that by the Search Density Theorem and the definition of e, $S_{a,b}$ and $S_{b,a}$ give the same value of T against e. Since this graph is connected, all DF searches have the same value of T against e. In other words, any two DF searches can be transformed into each other by a sequence of searches which differ only in the order they search at a single node, and hence have the same capture time. ■

A similar result is known for time-symmetric networks [3].

Lemma 8 *Using the BDF strategy β, the expected time T for the Searcher to reach each leaf node is $\frac{1}{2}(\tau + \Delta)$. Consequently the leaf nodes are the optimal responses to β. Hence any measure concentrated on leaf nodes, such as the EBD strategy e, is also an optimal response to β.*

Proof. We prove this by induction on the number of arcs. If the tree Q has a single arc a, then clearly $T = F_a$. On the other hand $\tau = F_a + R_a$ and $\Delta = F_a - R_a$, so $\tau + \Delta = 2F_a$. Assume the result is true for all trees with a smaller number of arcs than Q. There are two cases: either O is a branching node, or it is adjacent to only one arc.

Take the first case, that O is a branching node, with outward arcs a and b. Then applying the induction hypothesis to the subtree σ_a, *twice* the expected search time $2 \cdot T(\beta, i)$ to reach a leaf node i in σ_a is given by

$$\beta(a) \cdot (\tau_a + \Delta_a) + \beta(b) \cdot (2\tau_b + (\tau_a + \Delta_a)).$$

By (5), this expression is equal to

$$\begin{aligned}
& \left(\frac{1}{2} + \frac{\Delta_a - \Delta_b}{2\tau}\right) \cdot (\tau_a + \Delta_a) + \left(\frac{1}{2} + \frac{\Delta_b - \Delta_a}{2\tau}\right) \cdot (2\tau_b + (\tau_a + \Delta_a)) \\
= \; & (\tau_a + \Delta_a) + \left(\tau_b + \frac{\tau_b(\Delta_b - \Delta_a)}{\tau}\right) \\
= \; & (\tau_a + \tau_b) + \left(\frac{(\tau_a + \tau_b)\Delta_a + \tau_b(\Delta_b - \Delta_a)}{\tau}\right) \\
= \; & (\tau_a + \tau_b) + \left(\frac{\tau_a\Delta_a + \tau_b\Delta_b}{\tau}\right) = \tau + \Delta, \text{ by Proposition 4 (i).}
\end{aligned}$$

A similar argument applies to leaf nodes in σ_b.

Now consider the second case, that O is adjacent to a single arc a leading to branching node x. Then applying the induction hypothesis to σ_x,

$$\begin{aligned}
T &= F_a + T(\sigma_x) = F_a + \frac{1}{2}(\tau_x + \Delta_x) = F_a + \frac{1}{2}((\tau - F_a - R_a) + (\Delta - (F_a - R_a))) \\
&= \frac{1}{2}(\tau + \Delta), \text{ where } T(\sigma_x) \text{ denotes the expected time from } x.
\end{aligned}$$

■

Theorem 9 *The search value of a tree is half the sum of its tour time and its incline,*

$$V = \frac{1}{2}(\tau + \Delta). \tag{6}$$

The Biased Depth-first strategy β is optimal for the Searcher, and the Equal Branch Density strategy e is optimal for the Hider.

Proof. The two lemmas show that β and e are optimal responses to each other, hence optimal strategies. Evaluating β at any leaf node thus gives the value, as stated. ■

4.3 Uniqueness properties of optimal Searcher strategies[1]

When the search region Q is a tree, we know from [2] that the EBD strategy is the unique optimal Hider distribution, and that the strategy β that we call the Biased Depth-First strategy in this paper is uniquely optimal among the branching strategies for the Searcher. However, there may be other optimal Searcher strategies that are not branching strategies. For instance, Gal's analysis [11] shows that for time-symmetric trees (whose arcs have equal forward and reverse travel times), an equiprobable mixture of *any* Chinese Postman Tour (any depth-first search returning to the root node at the end) and its reverse tour, is optimal. This mixed strategy is known as a Random Chinese Postman Tour (RCPT). Moreover, there may be a large number of Chinese Postman Tours to choose from, each giving rise to a different optimal mixed strategy. Despite the non-uniqueness of optimal strategies, distinct optimal strategies do share some similarities in a 'behavioural sense', as we explain in this section.

First observe that if we fix any binary branch node x of a time-symmetric tree, then a Searcher who is following any RCPT will, upon reaching x for the first time, choose each outward arc with probability 1/2, regardless of the choice of RCPT. This uniqueness property generalizes (with 1/2 modified to the branching probability) to arbitrary *binary* trees, but not otherwise. To describe this generalization, we will use the following definition.

Definition 10 *For a Searcher strategy s on a rooted tree Q, and a branch node x of Q with outward arc a, let $\pi_s(a)$ be the probability that, on reaching x for the first time, s takes the arc a first. We say two Searcher strategies s_1 and s_2 on a tree Q are* ***weakly equivalent*** *if $\pi_{s_1}(a) = \pi_{s_2}(a)$ for every outward arc a of a branch node x.*

For example, for the Biased Depth-First strategy β on a binary tree, if a is the outward arc of some branch node, then $\pi_\beta(a) = \beta(a)$. For non-binary trees the Biased Depth-First strategy is undefined, but as we mentioned at the beginning of Section 4, any tree can be modified to be a binary tree by adding arcs of forward and reverse travel time 0. There are many ways this can be done, giving rise to distinct optimal Searcher strategies which we show may not be weakly equivalent. The following elementary result shows that

[1]We thank an anonymous referee for comments that led to the addition of this section.

uniqueness, up to weak equivalence, holds for binary trees. The positive part is similar to the uniqueness of branching strategies and the negative part depends on the way a non binary tree can be made binary by adding zero length arcs.

Theorem 11 *All optimal Searcher strategies on a rooted tree Q are weakly equivalent if and only if Q is binary. Furthermore $\pi_s(a)$ is constant for optimal Searcher strategies s if and only if a is one of two arcs from its base.*

We will not give a formal proof of this theorem, but instead we explain how it follows from another simple property, in this case shared by all optimal Searcher strategies even for non-binary trees. This property is that the expected time between reaching a node for the first time and taking a given outward arc from this node is constant for all optimal Searcher strategies (as long as the arcs have non-zero travel times). To see why this is the case, suppose for some optimal Searcher strategy, a is an outward arc from a node x to a node y, and let $T_{x\to a}$ be the expected time between reaching x for the first time and taking the arc a. Since any optimal Searcher strategy is depth-first, the value v_x of the game played on the subtree σ_x is equal to the expected time between arriving at x for the first time and reaching any leaf in σ_x, and hence in σ_y. But this is also equal to the expected time between reaching x for the first time and reaching y for the first time (that is $T_{x\to a} + F_a$) plus the value v_y of the game played on σ_y. Hence $v_x = T_{x\to a} + F_a + v_y$, so $T_{x\to a} = v_x - F_a - v_y$ is constant for all optimal Searcher strategies.

When a node has only two outward arcs, this property implies that the probability of taking a given outward arc is constant for all optimal Searcher strategies, so that if a tree is binary, all optimal Searcher strategies must be weakly equivalent. However, suppose a node x has three identical outward branches beginning with arcs a, b and c. Then upon reaching x for the first time, one optimal Searcher strategy might choose equiprobably between each possible order in which to take the outward arcs, and another optimal Searcher strategy might choose equiprobably between a and b, and then always choose c upon reaching x for the second time. In either case, the expected time between reaching x for the first time and choosing a given outward arc is the same. In general, it is not difficult to show that for a non-binary tree, different choices of 'binary modifications' always result in different Biased Depth-first strategies that are not weakly equivalent. Theorem 11 follows easily.

4.4 Application to the Kikuta game with search costs

We now consider the application of the theorem to the search game $K = K(Q, O)$ formulated by Kikuta [17] on a time-symmetric rooted tree Q, O. Kikuta's game is similar to Γ except that each node i is assigned a search cost $c_i \geq 0$, with $\sum c_i = C$. When encountering a node, the Searcher can either search it at cost (time loss) c_i or bypass it (to search it later) without incurring a cost. We have already observed in [2] that $K(Q, O)$ is equivalent to our game Γ on a time-asymmetric tree Q'. We obtain Q' by replacing the search costs with *search arcs* a_i between each node i of Q and a new leaf node i' of Q', with $F_{a_i} = c_i$ and $R_{a_i} = 0$, so that $D_{a_i} = c_i$. We present here for the first time an explicit formula for the value of Kikuta's game, as a corollary of our formula for the search value of a tree. Let τ be the tour time of the original network, not including the search costs of the nodes.

Corollary 12 *The value of Kikuta's game K on a rooted time-symmetric tree Q, O of total length μ and search costs c_i totalling to C is given by*

$$V = \mu + \frac{1}{2}\left(C + \sum_{\text{nodes } i \text{ of } Q} e\left(i'\right) \cdot c_i\right). \tag{7}$$

where e is the EBD distribution on the associated Q'. If the costs at all n nodes of Q are equal to c, then $V = (1/2)(\tau + (n+1)c)$ and the Random Chinese Postman Tour (and searching every node when you come to it) is optimal.

Proof. Since $V = V\left(K\left(Q, O\right)\right) = V\left(Q', O\right)$, and Q' is a (time-asymmetric) tree with no additional search costs, we have that the value V of Kikuta's game is equal to $(1/2)\left(\tau' + \Delta'\right)$, where τ' and Δ' are the tour time and the incline of Q'. Clearly

$$\tau' = \tau + \sum_{\text{nodes } i \text{ of } Q} \left(F_{a_i} + R_{a_i}\right) = \tau + C = 2\mu + C$$

and $\Delta' = \sum_{\text{nodes } i \text{ of } Q} e\left(i'\right) \cdot c_i$, establishing (7).

Now suppose all the $c_i = c$ and $1/2S_1 + 1/2S_2$ is a RCPT, where S_1 is a Chinese Postman Tour and S_2 is its reverse. Then for a fixed leaf node i, in the sum $T(S_1, i) + T(S_2, i)$, all non-leaf arcs contribute precisely their tour time to the sum, and all leaf arcs except the one containing i contribute time

equal to c. The leaf arc containing i contributes time $2c$. Hence, twice the capture time, is given by

$$\begin{aligned} 2T(1/2S_1 + 1/2S_2, i) &= T(S_1, i) + T(S_2, i) \\ &= \tau + (n-1)c + 2c \\ &= \tau + (n+1)c \\ &= 2V(Q', O). \end{aligned}$$

Hence the RCPT is optimal. ■

The case of equal search costs (but not the general case) can also be tackled within the time-symmetric tree theory by adding time-symmetric rays with travel times equal to $c/2$ at each node, observing that while this is not equivalent to the Kikuta problem, it always finds the Hider at time $c/2$ earlier. See [6]. This problem can also be attacked in the more difficult context of an arbitrary Searcher starting point - see Baston and Kikuta [10]. Our methods can be similarly applied to the foraging problem recently introduced by the first author [3].

5 Adding arcs to a network

In this section we discuss new general ideas that may be applied to the study of search games. They answer the question of how the search value $V(Q, O)$ changes when an additional arc is added between points of Q. By repeated use of these ideas, we can compare any network with an associated tree network, to which we can apply the formula for the search value found in Section 4. The idea of comparing general networks with trees goes back to Gal.

Suppose we add an arc to a network. How does the search value $V(Q, O)$ change (the root remains the same)? The following results give simple but useful answers to this question.

Theorem 13 (Arc-Adding) *Let Q' be obtained from the rooted network Q, O by adding an arc α from x to y, where x and y are points of Q. Then setting $V = V(Q, O)$ and $V' = V(Q', O)$, we have*

(i) $V' \leq V + F_\alpha + R_\alpha$. In particular, $V' \leq V$ if we simply identify x and y.

(ii) If $F_\alpha \geq d_Q(x, y)$ and $R_\alpha \geq d_Q(y, x)$, then $V' \geq V$.

Proof.

(i) Let s be an optimal mixed Searcher strategy on Q, so that the expected search time $T(s,x) \leq V$. Let s' be the strategy on Q' based on s, which modifies each pure strategy (search path) S chosen by s to a path S' on Q so that after the first time $S(t) = x$, S' goes to y and back along e before continuing with the original path S. Then if $z \in \alpha$, $T(s',z) \leq T(s,x) + F_\alpha \leq V + F_\alpha$ and if $z \in Q$, $T(s',z) \leq T(s,z) + F_\alpha + R_\alpha \leq V + F_\alpha + R_\alpha$, giving (i).

(ii) Every optimal mixed Hider strategy h on Q guarantees an expected capture time at least V on Q'. To see this, note that every strategy S' in Q' is no better against h than the strategy S obtained by replacing any use of α by a shortest path in Q between x and y, in the appropriate direction. ■

The second part of the Arc-Adding Theorem can be used to convert loops (arcs a with $a^- = a^+$) to leaf arcs, which leads in some cases (as in Figure 3, below) to trees, for which we will determine a formula for the search value. We first need the notion of the *in-radius* $\rho = \rho(a)$ and *in-midpoint* $m = m(a)$ of an arc a with initial and final ends a^- and a^+. These are uniquely defined (via the Intermediate Value Theorem) by the equation

$$d\left(a^-, m\right) = d\left(a^+, m\right) = \rho.$$

In other words the in-midpoint is the unique point which is equidistant *from* the two ends of its arc and this common distance is the in-radius.

Definition 14 (de-looping $DL(Q)$) *Let a be any loop of a network Q, directed so that $F_a \geq R_a$. Let a^1 denote the arc from a^- to $m = m(a)$ and let a^2 denote the arc from a^+ to m, where both a^- and a^+ are identified with some common node of Q. Let $DL(Q)$, the de-looping of Q, be the network obtained from Q by removing all the (open) arcs a^2 from Q, for every loop a. For every loop a of Q, $DL(Q)$ has a leaf ray a^1, with travel times $F_{a^1} = \rho(a)$ and $R_{a^1} = R_a - \rho(a)$. If a has constant velocities then*

$$F_{a^1} = \rho(a) = \frac{F_a R_a}{F_a + R_a} \text{ and } R_{a^1} = R_a - \rho(a) = \frac{R_a^2}{F_a + R_a}. \tag{8}$$

To illustrate the de-looping algorithm, consider the network drawn on the left of Figure 3. We use the convention that loop travel times are indicated inside and outside, and for other arcs forward time is on left, reverse time is on right. The right hand circle, with forward and reverse travel times 12 and 6 has its in-midpoint one third the way around the circle in the forward

direction. The formulae (8) give the travel times for its associated leaf arc on the tree (4 and 2). (The left circle is turned into a leaf arc of half its length, as in the time-symmetric constructions of Gal [13].)

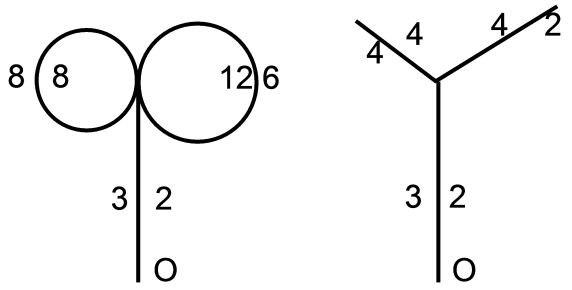

Figure 3. A network and its delooped tree.

Corollary 15 *For any rooted network Q', O, we have*

$$V(Q', O) \geq V(DL(Q'), O). \tag{9}$$

Proof. Observe that we can obtain the original network Q' from the delooped network $Q = \mathrm{DL}(Q')$ by adding (putting back) the (deleted) arc $\alpha = a^2$. The result will follow from the second part of the Arc-Adding Theorem upon showing that α (directed towards $m(a)$) is larger (slower) than a^1 in both directions. But for each loop a, we have, taking x to be endpoint $a^- = a^+$ of the loop and y to be its in-midpoint $m(a)$, that

$$\begin{aligned} F_\alpha &= F_{a^1} = \rho(a) = d_Q(x, y), \text{ and} \\ R_\alpha &= F_a - \rho(a) \geq R_a - \rho(a) = R_{a^1} = d_Q(y, x). \end{aligned}$$

■

Note that there is an alternative way we could "deloop" the network, where we replace each loop a with a single arc of forward length R_a and backward length 0. This arc has the same total travel time as the arc a^1 used in Definition 14, but a larger incline. The alteration represents an advantage to the Hider, since it effectively forces the Searcher to go around the loop a in a particular direction. Hence this alternative delooping process would provide an upper bound for the value of the game.[2]

[2]We thank an anonynous referee for making this observation.

6 Circle with Concave Travel Times

The simplest non-tree network is the circle, represented as a single loop arc a, with its initial and terminal ends a^- and a^+ identified with the start node O. For simplicity, we call the forward direction clockwise. We restrict the nature of the travel times on the arc a by parameterizing a (directed from end a^- to a^+) by $0 \leq x \leq 1$ so that $d(a^+, x)$ is a decreasing linear function of x and $f(x) = d(a^-, x)$ is an increasing continuous function. In this case we say that the arc a has *constant velocities* if f is linear and has *concave travel times* if f is concave. Note that the fact that an arc has concave travel times is independent of the orientation choice for the arc.

In the time-symmetric case there is a simple solution to the game on a circle: the value is half the travel time along the loop, two optimal strategies for the Hider are hiding uniformly along the arc or hiding at the midpoint m, the optimal Searcher strategy is to tour the loop equiprobably in either direction. The general solution for arbitrary travel times is unknown. Even for simple travel times (as shown in the next section when the circle is viewed as two identical arcs, with uniform velocities, from O to m) the solution can be very complicated, requiring backtracking paths and mixtures over a continuum of pure strategies.

In this section we consider a version in which the solution for the time-symmetric circle has a natural generalization to hiding at the in-midpoint and searching *with unequal probabilities* in the clockwise or anticlockwise directions. The travel time assumption needed for this simplification is concave travel times.

Theorem 16 *Let C be the network consisting of a single loop a at the start node O, with concave travel times. Let ρ and m denote the in-radius and in-midpoint of a. Then*

1. $V \equiv V(C, O) = \rho$
2. *Hiding at m is optimal.*
3. *The mixed Searcher strategy $\bar{s}_p$ of going around loop a clockwise (forwards) with probability $p = 1/(1+\lambda)$ and anticlockwise with probability $1-p$, is optimal for any $\lambda \in \left[\frac{f'_+(m)}{R_a}, \frac{f'_-(m)}{R_a}\right]$, where f'_+ and f'_- are respectively the right and left derivatives of f.*

4. *In particular, if loop a has constant velocities, then $m = R_a/(F_a + R_a)$, $V = \rho = F_a m$ and the unique optimal value of p is m.*

Proof. The de-looping $DL(C)$ of the loop a is a single ray with forward travel time ρ, so the value of $DL(C)$ is ρ, and by Corollary 15 we have $V \geq \rho$. (This is clear in any case as hiding at m ensures the Searcher cannot reach you in time less than the in-radius ρ.)

Since the forward time function $f(x) = d_a(a^-, x)$ is concave, it has left and right derivatives $f'_-(x_0)$ and $f'_+(x_0)$ at any x_0 in the interior of a such satisfying

$$f'_-(x_0) = \inf_{x<x_0} \frac{f(x_o) - f(x)}{x_o - x} \geq f'_+(x_0) = \sup_{x>x_0} \frac{f(x) - f(x_0)}{x - x_0}. \tag{10}$$

Suppose that the Searcher adopts the strategy $\bar{s}_p$, $p = 1/(1+\lambda)$, for some $\lambda \in \left[\frac{f_+(m)}{R_a}, \frac{f_-(m)}{R_a}\right]$. If the Hider is anticlockwise of m, that is, at some point $H = x \leq m$, then

$$\begin{aligned}
& T(\bar{s}_p, x) - \rho \\
= \;& p\, f(x) + (1-p)\; g(x) - \rho = p\, f(x) + (1-p)\; R_a(1-x) - f(m) \\
= \;& \frac{f(x)}{1+\lambda} + \frac{\lambda R_a(1-x)}{1+\lambda} - \frac{f(m)(1+\lambda)}{1+\lambda} = \frac{f(x) + \lambda R_a(1-x) - f(m) - \lambda R_a(1-m)}{1+\lambda} \\
= \;& \frac{f(x) - f(m) + \lambda R_a(m-x)}{1+\lambda} \leq \frac{f(x) - f(m) + f'_-(m)(m-x)}{1+\lambda} \\
\leq \;& \frac{m-x}{1+\lambda}\left(f'_-(x) - \frac{f(m) - f(x)}{m-x}\right) \leq 0.
\end{aligned}$$

By an analogous argument, if the Hider hides at some $x \geq m$ the Searcher will find him in expected time $\leq \rho$. Hence $V \leq \rho$, so that $V = \rho$. Points 2 to 4 follow easily. ■

7 Solution of Two-Arc Networks

In the previous section we showed that the circle network has a simple solution if it has concave travel times. In this section we show that the solution can become quite complicated if concavity is lost, even for a very simple class of circle networks $U(b)$ consisting of two identical constant velocity arcs from

O to m (so labeled because it is the in-midpoint of $U(b)$ if we view it as a single loop). That is, the two arcs have identical forward and reverse travel times F and R. Without loss of generality we can take the forward travel time F to be 1, and for notational simplicity denote $R = b$. Of course if $b \leq 1$ then the network can be viewed as a single arc (loop) with concave travel times, so in this case $V(U(b)) = \rho = 1$. Optimally, the Hider goes to m and the Searcher adopts strategy $\bar{s}_p$ with $p = 1/2$ (in fact any $p \in \left[\frac{b}{1+b}, \frac{1}{1+b}\right]$). As we shall see, the case $b > 1$ (where the network goes 'downhill' from the start O) has a rather complicated solution, reminiscent of the solution of the search game on three (time-symmetric) arcs given in [19]. We view each arc from O to m as having unit length, parameterized by x going from 0 to 1, with forward velocity 1 and reverse velocity $1/b$.

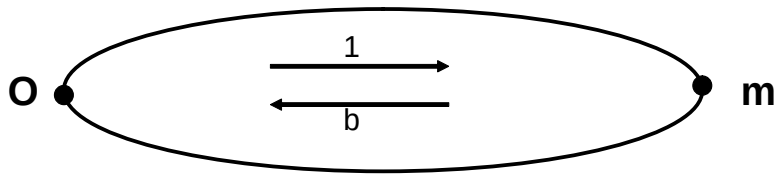

Figure 4. The 'two-arc' network $U(b)$.

Theorem 17 *Consider the network $U(b)$ consisting of two identical arcs from O to m with forward travel time 1 and reverse time $b \geq 1$. Then*

1. *The search value is $V = V(U(b), O) = 1 + \frac{1}{2}(b-1)\ln 2$.*

2. *An optimal strategy for the Hider is to pick x according to the density function $4e^{-2x}$ on the interval $[0, \ln 2/2]$. Then he hides equiprobably on the two points at forward distance x from O.*

3. *An optimal strategy for the Searcher begins by choosing a number y from $[0, \ln 2/2]$ according to the density function $2e^{2y}$. With probability $p = (b+3)/(2b+2)$ he tours the circle equiprobably in either direction. With probability $1-p$ he goes around in an equiprobable direction until he is at forward distance y from O; then reverses direction and goes around the circle until he has reached all points.*

Proof. Suppose the Hider follows the strategy described in the statement of the proof. Then the expected discovery time if the Searcher goes all the

way around the circle is

$$\frac{1}{2}\int_0^{\frac{1}{2}\ln 2} 2x4e^{-2x}dx + \frac{1}{2}\int_0^{\frac{1}{2}\ln 2}(1+b(1-2x))4e^{-2x}dx = 1+\frac{1}{2}(b-1)\ln 2.$$

If the Searcher backtracks at some point $y \leq \frac{1}{2}\ln 2$ then the expected discovery time is

$$\begin{aligned} & \frac{1}{2}\left(\int_0^{\frac{1}{2}\ln 2}(2(1+b)y+2x).4e^{-2x}dx\right) + \\ & \frac{1}{2}\left(\int_0^{y} 2x\ 4e^{-2x}dx + \int_y^{\frac{1}{2}\ln 2}(2(1+b)y+1+b(1-2x))4e^{-2x}dx\right) \\ = \ & 1+\frac{1}{2}(b-1)\ln 2. \end{aligned}$$

If the Searcher backtracks a some point $y > \frac{1}{2}\ln 2$, the expected search time will be greater than if he backtracks at $y = \frac{1}{2}\ln 2$.

Suppose the Searcher follows the strategy described in the statement of the proof. Then, if the Hider is at a distance $x > \frac{1}{2}\ln 2$ from O, the expected search time is

$$\begin{aligned} & \frac{b+3}{2b+2}\left(\frac{1}{2}2x+\frac{1}{2}(1+b(1-2x))\right) + \\ & \frac{b-1}{2b+2}\left(\frac{1}{2}\int_0^{\frac{1}{2}\ln 2}(2(1+b)y+1+b(1-2x))2e^{2y}dy + \frac{1}{2}\int_0^{\frac{1}{2}\ln 2}(2(1+b)y+2x)2e^{2y}dy\right) \\ = \ & 1+(b-1)\ln 2-(b-1)x \\ \leq \ & 1+(b-1)\ln 2-(b-1)\frac{1}{2}\ln 2 \\ = \ & 1+\frac{1}{2}(b-1)\ln 2 \end{aligned}$$

If the Hider is at a distance $x \leq \frac{1}{2}\ln 2$ then the expected search time is

$$\frac{b+3}{2b+2}\left(\frac{1}{2}2x+\frac{1}{2}(1+b(1-2x))\right) + \left(\frac{b-1}{2b+2}\right)\cdot$$
$$\left(\frac{1}{2}\left(\int_0^{x}(2(1+b)y+1+b(1-2x)2e^{2y}dy + \int_x^{\frac{1}{2}\ln 2} 2x2e^{2y}dy + \frac{1}{2}\int_0^{\frac{1}{2}\ln 2}(2(1+b)y+2x)2e^{2y}dy\right)\right)$$
$$= 1+\frac{1}{2}(b-1)\ln 2$$

Hence the value is $1 + \frac{1}{2}(b-1)\ln 2$. ■

The method for discovering these strategies is as follows. Suppose $b = 2$. For the Hider distribution, we would like to find a density function $h(x)$ where $x \in [0, z]$ such that the expected search time is the same whether the Searcher backtracks after travelling some distance $y \leq z$, or goes all the way around the circle (which is effectively backtracking after travelling distance 0). That is,

$$\frac{1}{2}6y + \int_0^z 2xh(x)dx + \int_0^y 2xh(x)dx + \int_y^z (6y + 3 - 4x)h(x)dx = C.$$

where C is a constant, independent of y. Simplifying this and differentiating with respect to y we obtain

$$4 - h(y) - 2\int_0^y h(x)dx = 0.$$

Putting $H(y) = \int_0^y h(x)dx$ gives the differential equation

$$\frac{dH}{dy} = 4 - 2H.$$

Solving this gives $h(x) = 4e^{-2x}$, and using $\int_0^z h(x)dy = 1$ we obtain $z = \frac{1}{2}\ln 2$.

A similar method can be used to find the Searcher strategy.

8 General Lower Bounds on the Search Value

In this section we obtain lower bounds for the search value of a network by an algorithm of *identification-cutting-deleting*. We first (identification) identify either all (Section 8.1) or some (Section 8.2) of the nodes. The identification of two nodes may be regarded as the addition of an arc between these nodes with 0 travel time in both directions. Then we cut any loops at their in-midpoint (cutting). Finally, for each former loop, we remove (deleting) the larger of the two resulting rays to the in-midpoint. Actually, we have seen the last two (cutting and deleting) parts of the algorithm - this is simply the de-looping algorithm. The result of the algorithm is a tree network with a lower search value than the original one, by the Arc-Adding Theorem and its corollary. Since we have a formula for the search value of a tree, this

becomes a lower bound for the original network. By choosing which nodes to identify, this gives us different lower bounds. This is shown for the 'circle with two spikes' network of Figure 5 (with time-symmetric semicircles. Here Q^* identifies all nodes and Q^{**} identifies all nodes on the Eulerian subnetwork.

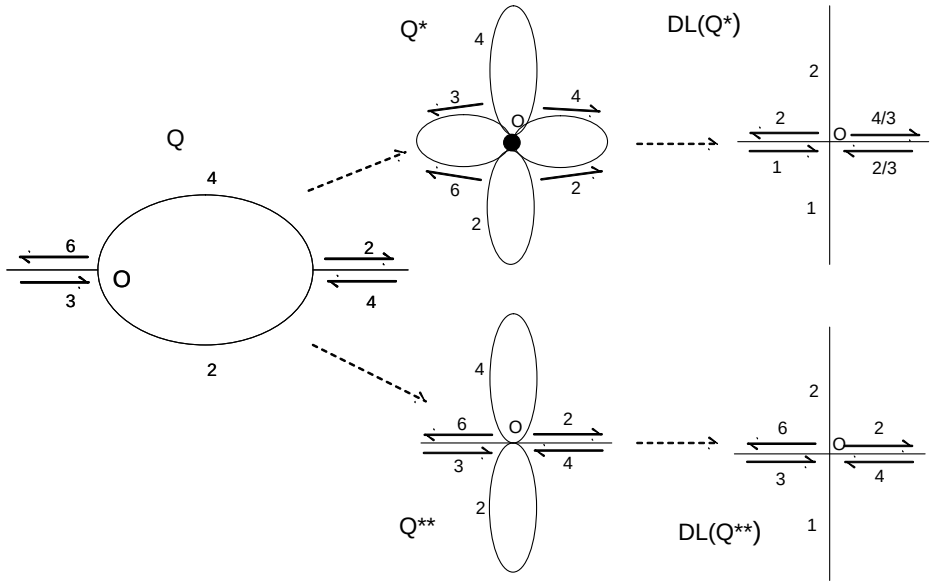

Figure 5. Network Q, with identifications followed by de-looping.

8.1 Lower bounds for arbitrary networks

Here we apply the identification-cutting-deleting algorithm to obtain a general lower bound for the search value.

Theorem 18 *Let Q, O be a rooted network with every arc a oriented so that $F_a \geq R_a$. Let $\omega = \sum_{a \in \mathcal{A}} R_a$ denote the time required to traverse all the arcs of Q in the quicker direction. Then, with ρ_a denoting the in-radius of a,*

$$V(Q) \geq \frac{1}{2}\left(\omega + \sum_{a \in \mathcal{A}} \frac{R_a}{\omega}\left(2\ \rho_a - R_a\right)\right). \tag{11}$$

Proof. Let Q^* denote the network obtained from Q by identifying all of its nodes. Thus Q^* is an $n-$leaf clover around a central node O (n is the number

of arcs of Q). Let STAR(Q) =DL(Q^*) be the star obtained by de-looping Q^*. So by the first part (identification remark) of the Arc-Adding Theorem (Theorem 13) and Corollary 15 on de-looping, we have

$$V(Q,O) \geq V(Q^*,O) \geq V(\text{STAR}(Q),O).$$

It remains only to show that the right hand side of (11) is the search value of the star network STAR(Q), or that its tour time τ and incline Δ satisfy

$$\tau = \sum_{a\in\mathcal{A}} R_a \equiv \omega, \text{ and} \tag{12}$$

$$\Delta = \sum_{a\in\mathcal{A}} \frac{R_a}{\omega}(2\rho(a) - R_a). \tag{13}$$

It follows from (8) that for each arc a of Q, the corresponding arc a^1 with leaf node $m(a)$ of STAR(Q) has tour time

$$\begin{aligned} \tau_{a^1} &= F_{a^1} + R_{a^1} = \rho(a) + (R_a - \rho(a)) = R_a, \text{ establishing (12), and} \\ \delta\left(a^1\right) &= F_{a^1} - R_{a^1} = \rho(a) - (R_a - \rho(a)) = 2\rho(a) - R_a. \end{aligned} \tag{14}$$

Since the EBD probability of the leaf node of a^1 is τ_{a^1}/τ, the remaining formula (13) now follows from (14). ■

If Q is a time-symmetric network, then R_a is simply the length of the arc a, so that ω is simply the total length of Q, a number called μ by Gal [11]. In this case the radius $\rho(a)$ of a is simply half its length R_a, so we have another proof of Gal's result.

Corollary 19 *For any time-symmetric rooted network Q, O the search value V satisfies*

$$V \geq \mu/2, \text{ where } \mu \text{ is the total length of } Q.$$

For time-symmetric networks, this is easily established by having the Hider adopt the uniform distribution. Our proof is distinct from the Gal's - it involves no integration. Of course we can use the same idea here, to assert that $V(Q) \geq \omega/2$, using the uniform Hider strategy.

Let us consider these estimate for the two-arc network $U(b)$ for $b = 2$, where Theorem 17 gives the search value as $1+\ln(2)/2 = 1.3466$. The lower bound $\omega/2$ gives $(1+1)/2 = 1$, which is the same as if we apply Theorem 18 to $U(2)$ considered as a single loop. If we consider $U(1/2)$ as having two arcs and two nodes, then Theorem 18 (with $R_a = 1$, the shorter way, and $\rho_a = 2/3$) gives the lower bound as $(1/2)(2 + 2(1/2)(2/3 - 1/3)) = 7/6 = 1.1667$.

8.2 Lower bounds for weakly Eulerian networks

In this section we show how to improve on the lower bound of Theorem 18 for weakly Eulerian networks. These are defined as follows (recall that a network is Eulerian if it contains a closed path which visits each point of the network exactly once).

Definition 20 *A network Q is called weakly Eulerian if it contains a set of disjoint Eulerian networks $E_1, \ldots, E_k$ such that shrinking each of them to a single point transforms Q into a tree. Formally, "shrinking" E_i to a point means replacing it by a point incident to all arcs in $Q - E_i$ that are incident to E_i.*

Let $\mathcal{A}_1$ denote the set of arcs of Q lying on the Eulerian networks and let $\mathcal{M}$ denote the set of in-midpoints of these arcs. Let $\mathcal{L}$ denote (as earlier for trees) the set of leaf nodes of Q.

We may also define $\Delta = \Delta(Q)$ for an Eulerian network in the following way.

Definition 21 *Let Q^{**} be the network obtained from Q by identifying into a single node j all the nodes on the same Eulerian component E_j of Q. Next define the tree $\hat{Q} = DL(Q^{**})$ to be the de-looping of Q^{**}. Then we define $\Delta = \Delta(Q)$ by*

$$\Delta = \Delta(\hat{Q})$$

Proposition 22 *Let Q be a weakly Eulerian network such that*

1. *$F_a = R_a$ for all a in $\mathcal{A}_1$, and*
2. *$d(O,i) \geq d(i,O)$ for all $i \in \mathcal{M} \cup \mathcal{L}$.*

Then $V(Q,O) \geq \frac{1}{2}(\tau + \Delta) \geq \frac{\tau}{2}$.

Proof. The first part of the Arc-Adding Theorem (Theorem 13) says that $V(Q) \geq V(Q^{**})$ and Corollary 15 on de-looping guarantees that $V(Q^{**}) \geq V\left(\hat{Q}\right)$. Hence $V(Q) \geq V\left(\hat{Q}\right)$. We evaluate $V\left(\hat{Q}\right)$ using the formula (6) for the search value of a tree. Note that for any $i \in \mathcal{M} \cup \mathcal{L}$, we have that

$$d_{\hat{Q}}(O,i) - d_{\hat{Q}}(i,O) = d_Q(O,i) - d_Q(i,O) \geq 0, \tag{15}$$

by assumption 2. Note that the original network Q and the tree $\hat{Q}$ have the same tour time

$$\tau = \sum_{a \in \mathcal{A} - \mathcal{A}_1} (F_a + R_a) + \sum_{a \in \mathcal{A}_1} F_a.$$

Observe that the leaf nodes of the tree $\hat{Q}$ are the points i of $\mathcal{M} \cup \mathcal{L}$, which by assumption have non-negative height $\delta(i)$. Hence the incline Δ of $\hat{Q}$ is a weighted average of non-negative numbers, so by (15), $\Delta \geq 0$. The tree formula for the search value (6) gives

$$V(Q) \geq V\left(\hat{Q}\right) = \frac{1}{2}(\tau + \Delta) \geq \frac{\tau}{2}. \tag{16}$$

■

If we specialize this result to time-symmetric networks, we obtain a well known result of Gal (2000), which generalized a similar result of Reijnierse and Potter [21] for a more restricted class of networks (weakly cyclic networks).

Corollary 23 *If Q is a time-symmetric Weakly Eulerian network, then $V(Q) = \tau/2$ (or in Gal's notation, $\bar{\mu}/2$).*

Proof. The assumption is stronger than that of the Proposition, so we have $V(Q) \geq \tau/2$. However for all time-symmetric networks, the random Chinese Postman Tour for the Searcher ensures that $V(Q) \leq \tau/2$. ■

9 Conclusion

This paper has analyzed the problem of searching a variable speed network for a Hider who is hidden according to an unknown distribution. A worst case scenario was considered, that is, a game against Nature in the form of a Hider who does not want to be found. We solved the game explicitly for a tree, improving over the recursive approach given in an earlier paper. We then used our explicit solution for trees to analyze more general networks.

We note that even for the time-symmetric case, the general solution is known only for a fairly small family of networks; for the time-asymmetric case studied here even less is known. We believe the associated cooperative problem of rendezvous (e.g. [1],[4]) could also be attacked with some success on time-asymmetric networks using ideas presented here.

References

[1] Alpern, S. Rendezvous search: a personal perspective. *Operations Research* 50 (2002):772-795.

[2] S. Alpern. Search games on trees with asymmetric travel times. SIAM J. Control Optim. 48 (2010), no. 8, 5547-5563.

[3] S. Alpern. A search game model of optimal foraging on a network. Operational Research Working Paper Series, LSEOR 10.124 (2010).

[4] S. Alpern and A. Beck. Asymmetric rendezvous on the line is a double linear search problem. *Mathematics of Operations Research* 24 (1999), 604-618.

[5] S. Alpern and S. Gal, A Mixed Strategy Minimax Theorem without Compactness, *SIAM J. Control and Optim.* 26 (1988), 1357-1361.

[6] S. Alpern and S. Gal. The Theory of Search Games and Rendezvous. Kluwer International Series in Operations Research and Management Sciences, 319 pp, Kluwer, Boston, 2003.

[7] S. Alpern and J. V. Howard. Alternating search at two locations. *Dynamics and Control* 10 (2000), no. 4, 319-339.

[8] E. J. Anderson and M. Aramendia. The search game on a network with immobile hider. *Networks* 20 (1990), 817-844.

[9] V. J. Baston and F. A. Bostock. An evasion game on a finite tree. *SIAM J. Control Optim.* 28 (1990), no. 3, 671-677.

[10] V. J. Baston and K. Kikuta. Search games on networks with travelling and search costs and with arbitrary searcher starting points. Preprint (2010).

[11] S. Gal, Search games with mobile and immobile hider. *SIAM J. Control Optim.* 17 (1979), 99-122.

[12] S. Gal, Search Games. Academic Press, 216 pp, 1980.

[13] S. Gal, On the optimality of a simple strategy for searching graphs. *Int. J. Game Theory* 29 (2000), 533-542.

[14] A. Garnaev. Search Games and Other Applications of Game Theory (Lecture Notes in Economics and Mathematical Systems Vol. 485), Springer, 2000.

[15] R. Isaacs. Differential Games, Wiley, New York, 1965.

[16] A. Jotshi and R. Batta. Search for an immobile entity on a network. *European Journal of Operational Research* 191 (2008), no. 2, 347-359

[17] K. Kikuta, A search game with travelling cost on a tree, *J. Oper. Res. Soc. Japan*, 38 (1995), no. 1, 70-88.

[18] K. Kikuta and W. Ruckle, Initial point search on weighted trees. *Naval Res. Logistics* 41 (1994), no. 6, 821-831.

[19] L. Pavlovic. A search game on the union of graphs with immobile hider. *Naval Res. Logistics* 42 (1995), no. 8, 1177-1199.

[20] W. L. Pearn and M. L. Li. Algorithms for the windy postman problem. *Computers and Operations Research* 21 (1994), no.6, p.641-651.

[21] J. H. Reijnierse and J. A. M. Potter. Search games with immobile hider. *Int. J. Game Theory* 21 (1993), 385-394.

[22] B. von Stengel and R. Werchner . Complexity of searching an immobile hider in a graph. *Discrete Appl. Math.* 78 (1997), 235-249.