European Journal of Operational Research 289 (2021) 338–349

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Interfaces with Other Disciplines

Weighted network search games with multiple hidden objects and multiple search teams☆

Abdolmajid Yolmeh[a], Melike Baykal-Gürsoy[b,*]

[a] Industrial and Systems Engineering Department, Rutgers University, 96 Frelinghuysen Rd, Piscataway, NJ 08854, United States
[b] Industrial and Systems Engineering Department, RUTCOR, CAIT, Rutgers University, 96 Frelinghuysen Rd, Piscataway, NJ 08854, United States

ARTICLE INFO

Article history:
Received 16 September 2019
Accepted 27 June 2020
Available online 3 July 2020

Keywords:
Search games
Multiple search teams
Weighted locations
Zero-sum games
Column and row generation

ABSTRACT

Most search game models assume that the hiding locations are identical and the players' objective is to optimize the search time. However, there are some cases in which the players may differentiate the hiding locations from each other and the objective is to optimize a weighted search time. In addition, the search may involve multiple search teams. To address these, we introduce a new network search game with consideration given to the weights at different locations. A hider can hide multiple objects and there may be multiple search teams. For a special case of the problem, we prove that the game has a closed-form Nash equilibrium. For the general case, we develop an algorithm based on column and row generation. We show that the Searcher's subproblem is NP-hard and propose a branch and price algorithm to solve it. We also present a polynomial time algorithm for the Hider's subproblem. Numerical experiments demonstrate the efficiency of the proposed algorithms, and reveal insights into the properties of this game.

1. Introduction

The search games were first introduced by Isaacs (1965). His "simple search game" is defined on an arbitrary region **R** and is played between a hider and a searcher. The Hider picks a point in **R** and the Searcher selects a unit speed trajectory in **R** to find the Hider. Payoff to the players is the search time, which is the time required for the Searcher's trajectory to meet the Hider for the first time. Gal (1979) provides a more precise formulation of the search game defined on a network Q consisting of a finite set of connected arcs and a predetermined starting point O. The Hider picks a point in Q to hide and the Searcher selects a unit speed path starting from O. Since then, many variations of the network search games have been introduced (Alpern, 2010; 2011; 2017; Alpern & Gal, 2006; Alpern & Lidbetter, 2013a; 2013b; 2015; Baston & Kikuta, 2015; Dagan & Gal, 2008; Gal, 1980; Garnaev, 2012; Zoroa, Fernández-Sáez, & Zoroa, 2013). For example, Dagan and Gal (2008) consider an arbitrary starting point for the Searcher, Alpern (2011) studies a find-and-fetch search model. Other examples include search problems on networks with asymmetric travel times (Alpern, 2010; Alpern & Lidbetter, 2013b), an expanding search paradigm (Alpern & Lidbetter, 2013a), search games on a lattice (Zoroa et al., 2013), search games with searching costs at nodes (Baston & Kikuta, 2015), bi-modal search games to find a small object (Alpern & Lidbetter, 2015), and search games with combinatorial search paths (Alpern, 2017). For a more recent survey of search games see Gal (2013); Hohzaki (2016). For a background in search games see Alpern and Gal (2006), Alpern and Lidbetter (2013b), Gal (1980) and Garnaev (2012).

In this paper, we study a game played between a Hider and a Searcher. The Hider picks one or more locations on a network to hide some objects and the Searcher follows a path to find the hidden objects such that an objective function is optimized. The objective function is usually assumed to be the search time. While the Hider aims at maximizing the search time, the Searcher wants to minimize it. Therefore, this problem can be formulated as a zero-sum game. Majority of the papers in the literature of search games assume that the players do not have preference over different locations on the network and they only care about the search time. However, there are some cases in which the players differentiate the hiding places from each other. In these cases, the players may want to minimize/maximize a weighted search time with node weights representing the rate of damage. For example, in certain attacks (biological or chemical), casualty rate depends on factors such as population density, environment conditions etc.

☆ This material is based upon work supported by the National Science Foundation (Grant No.1901721) and the United States National Institute of Justice (Grant No.2018-R2-CX-0011).

* Corresponding author.
E-mail addresses: abdolmajid.yolmeh@rutgers.edu (A. Yolmeh), gursoy@soe.rutgers.edu (M. Baykal-Gürsoy).

https://doi.org/10.1016/j.ejor.2020.06.046
0377-2217/

Therefore, different locations may have different casualty rates and the overall damage will be proportional to exposure time and casualty rate. Another example is the problem of detecting an eavesdropping agent over communication channels (Garnaev, Baykal-Gürsoy, & Poor, 2016). Different channels may have different transmission capacities and the rate of damage to the network will be proportional to the detection time and the capacity of the channel. The only study that considers node weights in the search games is conducted by Zoroa, Zoroa, and Fernández-Sáez (2009). However, they only consider such games on lattices, not general graphs.

The problem of searching for a hidden object in discrete time and discrete space has been well studied in the literature. Blackwell studied the problem of finding a hidden object in a set of boxes (Matula, 1964). Given a probability distribution over the boxes of containing the hidden object and a search cost for each box, the objective is to minimize the expected cost of finding the object. Game-theoretic variants of this game have also been studied (Bram, 1963; Gittins & Roberts, 1979; Lin & Singham, 2015; Roberts & Gittins, 1978; Ruckle, 1991). Less attention has been given to the search games with multiple hidden objects. Assaf and Zamir (1987) and Sharlin (1987) study a search game with several hidden objects with the objective of finding one of these objects with minimum expected cost. Lidbetter (2013) extends this game to finding all of the hidden objects at minimum expected cost. Alpern, Fokkink, Op, and Lidbetter (2010) introduce caching games in which a Searcher with a limited resource aims to maximize the probability of finding a certain number of hidden objects. Lidbetter and Lin (2017) introduce multi-look search problems in which the Searcher can find at most one hidden object each time a box is opened. In other words, to find all of the hidden objects in a box, the Searcher may have to open it multiple times.

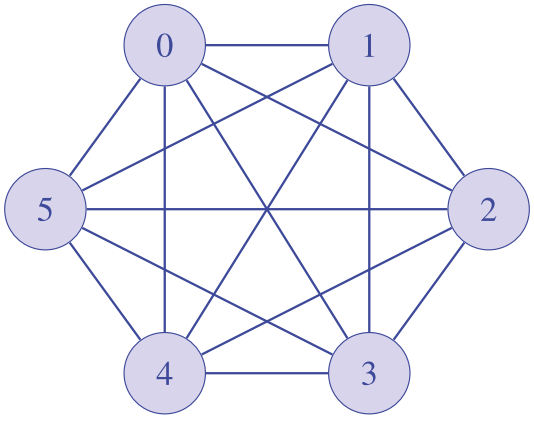
We propose a new discrete search game in which different locations have different weights and the payoff to the players is proportional to the search time and the location weight. In this setting, the location weights represent the rate of damage at that location. The players want to minimize/maximize the overall damage to the network, which is represented as a weighted search time. The game is played between a Searcher who controls a set of homogeneous search teams and a Hider who picks hiding locations to hide the objects. The hiding locations are dispersed on a network and the time it takes to visit a location depends on the previously visited location. We first consider a special case of this game and characterize the Nash equilibrium. Afterwards, we develop a column and row generation procedure to solve the general form of the game. The rest of this paper is organized as follows. Section 2 introduces the weighted discrete search game, presents examples and derives the Nash (saddle-point) equilibrium in closed form for a special case of the original game. Section 3 develops a solution approach based on column and row generation to solve the general game. Section 4 demonstrates numerical experiments to investigate the efficiency of the proposed algorithms and gain insight into the properties of the game. Finally, Section 5 presents the main conclusions of the paper and future research ideas.

2. Problem description

A Searcher and a Hider play a zero-sum weighted search game. The Searcher controls a set of S search teams and the Hider controls a set of H objects to hide. The game is played on a complete graph $G = (N, E)$, where $N = \{0, 1, 2, \ldots, n\}$ is the set of nodes in the graph and $E = \{(i, j) : i, j \in N, i \neq j\}$ is the set of edges. Using the approach in Gal (1979), let node 0 represent the origin, which is a predetermined location where all search teams are initially located. Let $N_h = \{1, 2, \ldots, n\}$ denote the set of n potential hiding locations. These locations are dispersed at the nodes of a network, and each location may differ in its rate of damage, and its difficulty to search. The Hider hides the objects in these potential locations. The Searcher uses a set of homogeneous search teams to find the hidden objects. It takes a search team v_i time units to inspect location $i \in N_h$ and, for each edge $(i, j) \in E$, the time required to travel from location i to location j is denoted by d_{ij}. If a location is inspected, the search team will find the hidden object, if any, i.e., false negative response is not possible. Each location i has a weight denoted by w_i. This weight represents the rate of damage to the network, if the Hider decides to hide an object at location i. In other words, the weight w_i represents the damage incurred in node i per unit of time, if there is an object hidden in this node. This damage rate will be in effect until the object is found by a search team. Therefore, if there is a hidden object in a node, then the overall damage in that node will be equal to the weight of the node multiplied by the time it takes to find the object. Throughout the paper, we assume, without loss of generality, that the locations are sorted in the order of decreasing weights, i.e., $w_1 > w_2 > \cdots > w_n$. We also assume that all parameters of the game are known to both players, i.e., this is a complete information game. Fig. 1 demonstrates an example of the weighted network search game with $n = 5$ nodes, two objects and two search teams. Matrix $\mathbf{d}$ denotes the matrix of travel times d_{ij}. Vectors $\mathbf{v}$ and $\mathbf{w}$ represent the vector of inspection times, v_i, and weights, w_i, respectively.

The objective of the Hider is to maximize the expected total damage to the network, while the Searcher wants to minimize the damage. A pure strategy for the Hider (also called a pure hiding strategy throughout the paper) is to select a hiding location to hide each object. We assume that hiding more than one object in a location does not increase the rate of damage in that location. Hence, it is not beneficial for the Hider to hide multiple objects in a single location. A pure strategy for the Searcher (also called a pure search strategy throughout the paper) is to select a joint schedule for search teams that ensures the search of every node exactly once. In other words, a joint schedule assigns each node to exactly one search team and determines the order in which the search teams visit their assigned nodes. For example, for the search game instance characterized in Fig. 1, an example of joint schedule is to assign schedules $0 \to 1 \to 2 \to 3$ and $0 \to 4 \to 5$ to search teams 1 and 2, respectively. This means that, this joint schedule prescribes search team 1 to start from the origin and visit nodes 1, 2, and 3, respectively. Similarly, search team 2 starts from the origin and visits nodes 4 and 5, respectively.

The proposed model is a zero-sum simultaneous-move game with a finite number of strategies. This type of game is also known as a matrix game. In our proposed matrix game model, the Searcher plays as the row player, and the set of all possible pure search strategies constitute the rows of the matrix. The Hider plays as the column player, with the set of all possible pure hiding strategies constituting the columns of the game matrix. We use K to denote the set of all possible pure search strategies and k to index them. Let x_k be the probability of using search strategy k in the Searcher's mixed strategy. Hence $\mathbf{x} = (x_1, x_2, \ldots, x_{|K|})$ represents a mixed strategy of the Searcher, where $|K|$ denotes the cardinality of set K, $x_k \geq 0$, for all $k \in K$ and $\sum_{k=1}^{|K|} x_k = 1$. Similarly, we use Λ to denote the set of all possible pure hiding strategies and index them by l. Let y_l denote the probability of using hiding strategy l. We define binary parameter z_i^l, which is equal to 1 if hiding strategy l involves hiding an object in location i, 0 otherwise. Here, a hiding strategy prescribes which locations will have an object hidden. Thus, if there are two objects hidden at the third and fifth nodes on the five-node network example in Fig. 1, we may have $(z_1^l, z_2^l, \ldots, z_5^l) = (0, 0, 1, 0, 1)$. In case there is only one object hidden, then l may represent a pure hiding strategy of choosing



$$\mathbf{d} = \begin{array}{c} i \setminus j \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 2 & 2 & 2 & 1 \\ 1 & 0 & 1 & 2 & 2 & 2 \\ 2 & 1 & 0 & 1 & 2 & 2 \\ 2 & 2 & 1 & 0 & 1 & 2 \\ 2 & 2 & 2 & 1 & 0 & 1 \\ 1 & 2 & 2 & 2 & 1 & 0 \end{bmatrix} \end{array}, \mathbf{v} = \begin{bmatrix} 3 \\ 4 \\ 3 \\ 1 \\ 2 \end{bmatrix}, \mathbf{w} = \begin{bmatrix} 8 \\ 6 \\ 5 \\ 3 \\ 2 \end{bmatrix}, S = H = 2$$

Fig. 1. A complete instance of the weighted search game.

the lth node to hide the object, i.e., $(z_1^l, z_2^l, \ldots, z_n^l) = (0, \ldots, 0, z_l^l = 1, 0, \ldots, 0)$. Then, y_l corresponds to the probability of choosing the lth node to hide the object. A mixed strategy of the Hider is denoted as $\mathbf{y} = (y_1, y_2, \ldots, y_{|\Lambda|})$, $y_l \geq 0$, $\forall l \in \Lambda$, and $\sum_{l=1}^{|\Lambda|} y_l = 1$. We use parameter t_i^k to denote the time at which joint schedule k completes inspection in location i. For example, for the search game instance in Fig. 1, if joint schedule k is to assign schedules $0 \to 1 \to 2 \to 3$ and $0 \to 4 \to 5$ to search teams 1 and 2, respectively, then search team 1 starts from the origin and visits node 1. Thus $t_1^k = d_{01} + v_1 = 4$ is obtained. After visiting node 1, search team 1 visits node 2. Therefore, $t_2^k = t_1^k + d_{12} + v_2 = 4 + 1 + 4 = 9$. Similarly, we can compute the vector $(t_1^k, t_2^k, \ldots, t_5^k) = (4, 9, 13, 3, 6)$. For $S = 1$, we also use parameter r_i^k to denote the order of visiting location i while using schedule k. If the Searcher and the Hider use mixed strategies $\mathbf{x}$ and $\mathbf{y}$, respectively, then the expected total damage is $v(\mathbf{x}, \mathbf{y}) = \sum_{k \in K} \sum_{l \in \Lambda} \sum_{i \in N_h} w_i t_i^k z_i^l x_k y_l$. The Nash equilibrium (saddle point) of the game is a point $(\mathbf{x}^*, \mathbf{y}^*)$ such that no player can benefit by unilaterally changing his or her strategy. In other words, at a Nash equilibrium point $(\mathbf{x}^*, \mathbf{y}^*)$, the following inequalities hold (Barron, 2013):

$$v(\mathbf{x}^*, \mathbf{y}) \leq v(\mathbf{x}^*, \mathbf{y}^*) \leq v(\mathbf{x}, \mathbf{y}^*).$$

The expected total damage in equilibrium, $v(\mathbf{x}^*, \mathbf{y}^*)$, is called the value of the game. The following theorem describes the saddle-point equilibrium for a special case.

Theorem 1. *If all travel times are the same, i.e. $d_{ij} = d, \forall (i, j) \in E$, the search time is the same at each node, i.e. $v_i = v, \forall i \in N_h$, and there is only one search team and one hidden object, i.e. $S = H = 1$, then the equilibrium is characterized by*

$$y_i = \begin{cases} \frac{\frac{1}{w_i}}{\sum_{j=1}^{m} \frac{1}{w_j}}, & i = 1, 2, \ldots, m, \\ 0, & i = m+1, \ldots, n, \end{cases} \tag{1}$$

$$E[\text{search order of the } i\text{th node}] = \sum_{k \in K} r_i^k x_k = \frac{m(m+1)}{2} \frac{\frac{1}{w_i}}{\sum_{j=1}^{m} \frac{1}{w_j}}, \quad i = 1, 2, \ldots, n, \tag{2}$$

where $m = \arg\max_{i \in \{1,2,\ldots,n\}} \frac{\frac{i(i+1)}{2}}{\sum_{j=1}^{i} \frac{1}{w_j}}$. *Moreover, the value of the game is:* $V^* = (d+v) \frac{\frac{m(m+1)}{2}}{\sum_{j=1}^{m} \frac{1}{w_j}}$.

Note that in this case, one can write t_i^k simply as $t_i^k = r_i^k (d+v)$.

Proof. Based on the definition of Nash Equilibrium, $(\mathbf{x}, \mathbf{y})$ is an equilibrium if and only if for some u:

$$\begin{aligned} &(d+v) w_i \sum_{k \in K} x_k r_i^k \begin{cases} = u, & y_i > 0, \\ \leq u, & y_i = 0, \end{cases} \\ \Rightarrow \quad &w_i \sum_{k \in K} x_k r_i^k \begin{cases} = \frac{u}{d+v} \equiv \bar{u}, & y_i > 0, \\ \leq \bar{u}, & y_i = 0. \end{cases} \end{aligned} \tag{3}$$

In this equation, the term $(d+v) w_i \sum_{k \in K} x_k r_i^k$ is the expected total damage if the Hider decides to hide the object in location i. This equation indicates that, for all nodes i with $y_i > 0$, the expected total damage, $(d+v) w_i \sum_{k \in K} x_k r_i^k$, should be the same. Otherwise, the Hider will be willing to deviate from the current mixed strategy by redistributing the y_i probabilities to increase his payoff. Moreover, for all nodes i with $y_i = 0$, the expected total damage, $(d+v) w_i \sum_{k \in K} x_k r_i^k$, should not be greater than u. Similarly, for the Searcher we have the following condition:

$$\sum_{i=1}^{n} w_i y_i r_i^k \begin{cases} = \bar{u}, & x_k > 0, \\ \geq \bar{u}, & x_k = 0. \end{cases} \tag{4}$$

We define A as the set of active nodes in which the Hider hides the object with a positive probability, i.e., $A = \{i | y_i > 0\}$. Clearly, if $i, j \in A$ with $w_i > w_j$, the expected search orders will satisfy

$$E[\text{search order of the } i\text{th node}] = \sum_{k \in K} x_k r_i^k = \frac{\bar{u}}{w_i} < \frac{\bar{u}}{w_j} = E[\text{search order of the } j\text{th node}]. \tag{5}$$

On the other hand, given the set of active nodes A, if $w_i y_i = c$, a constant for all $i \in A$, then $y_i = c/w_i = \frac{\frac{1}{w_i}}{\sum_{j \in A} \frac{1}{w_j}}$ since $\sum_{i \in A} y_i = 1$. Furthermore, for any $k \in K$, it holds that $\sum_{i \in A} r_i^k \geq \frac{|A|(|A|+1)}{2}$. This is true because the minimum of $\sum_{i \in A} r_i^k$ is achieved only in the case that the nodes in set A are visited before the other nodes. In this case, the order of visiting all nodes in A, follows the natural numbers up to $|A|$, that is $\sum_{i \in A} r_i^k = \sum_{r=1}^{|A|} r = \frac{|A|(|A|+1)}{2}$. Therefore, we have

$$\sum_{i \in A} w_i y_i r_i^k = \frac{1}{\sum_{j \in A} \frac{1}{w_j}} \sum_{i \in A} r_i^k \geq \frac{|A|(|A|+1)}{2} \frac{1}{\sum_{j \in A} \frac{1}{w_j}}. \tag{6}$$

Thus, the Searcher should visit during the first $|A|$ instances, each one of the locations in A, regardless of the exact order, that is $r_i^k \in \{1, \ldots, |A|\}$ for all $i \in A$, and $k \in B = \{l : x_l > 0\}$.

Following the above discussion, given the set of active nodes A, we can show that the following is a solution to Eqs. (3) and (4):

$$y_i = \begin{cases} \frac{\frac{1}{w_i}}{\sum_{j\in A}\frac{1}{w_j}}, & i \in A, \\ 0, & i \notin A. \end{cases} \tag{7}$$

$$E[\text{search order of the } i\text{th node}] = \sum_{k\in K} x_k r_i^k \begin{cases} = \frac{|A|(|A|+1)}{2}\frac{\frac{1}{w_i}}{\sum_{j\in A}\frac{1}{w_j}}, & i \in A, \\ \leq \frac{|A|(|A|+1)}{2}\frac{\frac{1}{w_i}}{\sum_{j\in A}\frac{1}{w_j}}, & i \notin A. \end{cases} \tag{8}$$

Moreover, we can show that, based on this solution, the expected total damage is $ETD(A) = (d+v)\frac{\frac{|A|(|A|+1)}{2}}{\sum_{j\in A}\frac{1}{w_j}}$. Note that, using schedules that visit nodes in set A before other nodes, the quantity $\bar{u}$ can be obtained from (4), which can be used to derive (8) from (3).

Next step is to characterize the active set A. We show that, for any active set A with $i \in A, j \notin A, w_j > w_i$, the Hider can replace node i with node j to create an active set A' that leads to a higher ETD. This is because $ETD(A) = (d+v)\frac{\frac{|A|(|A|+1)}{2}}{\sum_{j\in A}\frac{1}{w_j}}$ is an increasing function of $w_i, i \in A$. Therefore, replacing w_i with w_j will lead to a higher ETD. Thus, knowing that w_i values are sorted, the active set with the highest ETD is of the form $A = \{1, 2, \ldots, l\}$ and the cut-off index $l = m$ is, by assumption, the cut-off index that leads to the highest ETD. Therefore, the Hider cannot increase the ETD by changing the active set from $A = \{1, 2, \ldots, m\}$ to any other set.

Next, we show that there exists a search strategy characterized by Eq. (2). The space of possible vectors for the expected visiting orders is the convex hull of all permutations of the set $\{1, 2, \ldots, n\}$ given as

$$Conv(R) = \left\{ \bar{r} \in \mathbb{R}^n, \sum_{i=1}^{n} \bar{r}_i = \frac{n(n+1)}{2}, \sum_{i\in Q} \bar{r}_i \geq \binom{|Q|+1}{2}, \quad Q \subseteq \{1, 2, \ldots, n\} \right\}, \tag{9}$$

where $\bar{r}_i$ is the expected visiting order of node i, i.e., $\bar{r}_i = \sum_{k\in K} r_i^k x_k$. This space is called permutahedron and is not full-dimensional (Lancia & Serafini, 2018). To show that there exists a strategy for the Searcher, we need to prove that the expected visiting orders, from Eq. (2), are in the permutahedron. Obviously, $\sum_{i=1}^{n} \bar{r}_i = \frac{n(n+1)}{2}$ is true. We proceed to demonstrate that the other inequalities are also valid. We show that the inequality is valid for $|Q| = l$ with $1 \leq l < n$. For each l, it is enough to prove the inequality for $Q = \{1, 2, \ldots, l\}$ (for other sets of size l, the inequality will follow due to ordered node weights). We need to show that:

$$\sum_{i=1}^{l} \frac{m(m+1)}{2}\frac{\frac{1}{w_i}}{\sum_{j=1}^{m}\frac{1}{w_i}} \geq \binom{l+1}{2} = \frac{(l+1)l}{2},$$

implying

$$\frac{\frac{m(m+1)}{2}}{\sum_{j=1}^{m}\frac{1}{w_l}} \geq \frac{\frac{l(l+1)}{2}}{\sum_{i=1}^{l}\frac{1}{w_i}}.$$

But, this is true based on the assumption that index m is chosen so that this inequality holds. Therefore, there exists a search strategy that satisfies Eq. (2). This completes the proof. □

Remark 1. Using the expected search order values obtained in Eq. (2), we can compute a mixed search strategy for the Searcher in polynomial running time of $O(n^2)$. This can be done using the decomposition algorithm provided in Yasutake, Hatano, Kijima, Takimoto, and Takeda (2011).

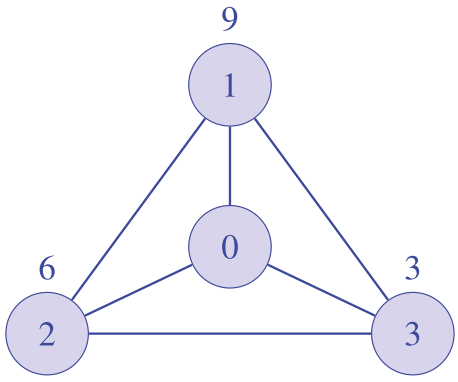


Fig. 2. Example of a weighted network search game with $n = 3$ hiding locations.

Remark 2. Theorem 1 indicates that, when $d_{ij} = d, v_i = v, \ \forall (i, j) \in E, i \in N_h$ and $H = S = 1$, the Nash equilibrium is of threshold type. Meaning that, there exists a cut-off index, m, such that the Hider will only consider the first m locations and ignore the remaining ones. This theorem can be used to obtain an upper bound on the value of the game in general by taking $d = \max_{i,j} d_{ij}$ and $v = \max_i v_i$. Similarly, a lower bound can be computed by taking $d = \min_{i,j} d_{ij}$ and $v = \min_i v_i$.

Remark 3. The Nash equilibrium characterized in Theorem 1 prescribes lower hiding probabilities for locations with higher weights. The reason for this counter-intuitive outcome is that the expected visiting order of the locations with higher weights is smaller in equilibrium. Therefore, even though the rate of damage is higher for locations with higher weights, the expected duration of damage is smaller for these locations. This observation is in line with the results obtained in the area of security games (Baykal-Gürsoy, Duan, Poor, & Garnaev, 2014; Yolmeh & Baykal-Gürsoy, 2017).

Example 1. We consider an example with $n = 3$ nodes to hide on a network shown in Fig. 2. In this figure, node 0 is the origin and the remaining nodes are the potential hiding locations. Node weights are shown on top of each node. We assume that $d_{ij} = 1, v_i = 1, \ \forall (i, j) \in E, i \in N_h$, and $S = H = 1$. Using Theorem 1, we can obtain the critical index $m = 2$. The hiding probabilities obtained are: $y_1 = \frac{\frac{1}{w_1}}{\frac{1}{w_1}+\frac{1}{w_2}} = 0.4, \ y_2 = \frac{\frac{1}{w_2}}{\frac{1}{w_1}+\frac{1}{w_2}} = 0.6, \ y_3 = 0$, and the value of the game is: $V^* = (d+v)\frac{\frac{m(m+1)}{2}}{\sum_{j=1}^{m}\frac{1}{w_j}} = 21.6$. Using Eq. (8), we can compute the expected search order for nodes 1 and 2 as 1.2 and 1.8, respectively. Note that, because node 3 is always the last node that is visited, the expected search order of this node is 3. To transform these expected visiting orders into a mixed strategy for the Searcher, we use the decomposition algorithm described in Yasutake et al. (2011). Using this algorithm, the search schedule $0 \to 1 \to 2 \to 3$ is used with probability 0.8, and the search schedule $0 \to 2 \to 1 \to 3$ is chosen with probability 0.2.

Lemma 1. *Given an upper bound $\overline{V}$ on the value of the game, any search strategy $k \in K$ that completes inspection at any node i at time t_i^k with $t_i^k > \frac{\overline{V}}{w_i}$ does not belong to a Nash equilibrium.*

Proof. We show that the Hider can improve his payoff by hiding an object in location i. The expected damage from hiding an object in location i is: $w_i t_i^k > \overline{V} \geq V^*$. Therefore, using search strategy k leads to an expected damage value that is higher than the expected

damage in equilibrium. Therefore, strategy k does not belong to a Nash equilibrium. □

One way to solve the proposed weighted search game is to generate all of the possible strategies for both players and solve the resulting matrix game. However, for both players, the number of possible strategies grows exponentially as n increases. Therefore, it is not possible to generate all of the possible strategies for problems of larger size. In addition, even though the number of possible strategies is large, only a small number of them are expected to be used in the Nash equilibrium. In the next section, we use this idea to develop an efficient column and row generation algorithm to obtain a Nash equilibrium for this game.

3. Solution approach for general weighted search games

In this section, we develop a solution algorithm based on column and row generation (Muter, Birbil, & Bülbül, 2013; Riedel, Smith, & McCallum, 2012) to find a Nash equilibrium for the weighted search game introduced in the previous section. The proposed solution method can also be described as a modification of the algorithm introduced in Godinho and Dias (2010, 2013). Because this is a zero-sum game, we can write the following linear program (LP) to obtain a Nash equilibrium for this game:

$$\begin{aligned} \text{LPM Minimize} \quad & u \\ \text{subject to} \quad & u \geq \sum_{k\in K} x_k \sum_{i\in N_h} w_i t_i^k z_i^l, \quad \forall l \in \Lambda, \\ & \sum_{k\in K} x_k = 1, \\ & x_k \geq 0, \quad \forall k \in K. \end{aligned}$$

This LP is called the linear programming master problem (LPM). In this formulation, K is the set of all possible joint schedules, indexed by k. x_k is a decision variable representing the probability of using joint schedule $k \in K$ in the Searcher's mixed strategy. In general, the sets K and Λ may be exponentially large; however, the number of strategies used is expected to be much smaller. Our proposed column and row generation algorithm uses this idea to start with small subsets $K' \subset K$ and $\Lambda' \subset \Lambda$ of search and hiding strategies and generates them as needed. The starting subsets K' and Λ' could be any set of feasible strategies. Using the restricted set of strategies K' and Λ' we obtain the following LP:

$$\text{LPM-RS} \quad \text{Minimize} \quad u \tag{10}$$

$$\text{subject to} \quad u \geq \sum_{k\in K'} x_k \sum_{i\in N_h} w_i t_i^k z_i^l, \quad \forall l \in \Lambda', \tag{11}$$

$$\sum_{k\in K'} x_k = 1, \tag{12}$$

$$x_k \geq 0, \quad \forall k \in K'. \tag{13}$$

This problem is called LPM with Restricted Strategies (LPM-RS). The dual of LPM-RS is

$$\text{Dual LPM-RS} \quad \text{Maximize} \quad v \tag{14}$$

$$\text{subject to} \quad v \leq \sum_{l\in\Lambda'} \sum_{i\in N_h} w_i t_i^k z_i^l y_l, \quad \forall k \in K', \tag{15}$$

$$\sum_{l\in\Lambda'} y_l = 1, \tag{16}$$

$$y_l \geq 0, \quad \forall l \in \Lambda'. \tag{17}$$

In this formulation, y_l is the dual variable corresponding to constraint (11) in LPM-RS. This variable represents the probability of using hiding strategy l in the Hider's mixed strategy. Moreover, v is the dual variable corresponding to constraint (12) which represents the minimum expected total damage. Next step is to find new strategies in $K \backslash K'$ and $\Lambda \backslash \Lambda'$ that could improve the current optimal solution for the corresponding players. Given the optimal dual solution y_l of LPM-RS, the reduced cost of joint schedule $k \in K \backslash K'$ is given by $\sum_{l\in\Lambda'} \sum_{i\in N_h} w_i t_i^k z_i^l y_l - v$. Based on the concept of duality in linear programming, optimality of LPM-RS is equivalent to the feasibility of its dual. Therefore, joint schedules that violate the constraint $v \leq \sum_{l\in\Lambda'} \sum_{i\in N_h} w_i t_i^k z_i^l y_l$, can improve the current optimal solution. Consequently, we need to look for a joint schedule k such that: $\sum_{l\in\Lambda'} \sum_{i\in N_h} w_i t_i^k z_i^l y_l - v < 0$. Note that, y_l values are fixed, and the problem is to find a joint schedule k with t_i^k such that: $\sum_{l\in\Lambda'} \sum_{i\in N_h} w_i t_i^k z_i^l y_l - v < 0$. Therefore, we are looking for a new joint schedule k that leads to a smaller expected total damage, $\sum_{l\in\Lambda'} \sum_{i\in N_h} w_i t_i^k z_i^l y_l$, than the current expected total damage, v. To obtain an improving strategy for the Hider, we consider the LPM-RS. Given the optimal solution x_k of LPM-RS, the current expected total damage is u. Therefore, the Hider should look for a new hiding strategy l with z_i^l such that the expected total damage, i.e., $\sum_{k\in K} \sum_{i\in N_h} w_i t_i^k z_i^l x_k$, is greater than the current expected total damage, i.e., u.

Section 3.1 develops a mathematical program and solution algorithms to solve the Searcher's subproblem. Section 3.2 characterizes the Hider's subproblem and gives a polynomial time solution algorithm to solve this subproblem. Section 3.3 presents the overall column and row generation algorithm and provides bounds on the value of the game.

3.1. The Searcher's subproblem

In this section, a flow type formulation is developed for the Searcher's subproblem. Here is a list of parameters and variables used to formulate the Searcher's subproblem:

- x_{ij}: Binary variable, for $(i, j) \in E$, it is equal to 1 if a search team visits location j immediately after visiting location i.
- t_i: Non-negative variable, time at which a search team completes inspection at location i.
- $N^+(i)$: Set of immediate successors of node i in graph G, i.e., $N^+(i) = \{j \in N | (i, j) \in E\}$.
- $N^-(i)$: Set of immediate predecessors of node i in graph G, i.e., $N^-(i) = \{j \in N | (j, i) \in E\}$.
- M: A big number.

Note that the set of edges E may not always correspond to a complete graph. Specifically, when developing a branch and price algorithm in Section 3.1.2, the set of edges E will be modified due to branching. Therefore, the sets of immediate successors and predecessors of a node, $N^+(i)$ and $N^-(i)$, need to be defined with the assumption that the underlying graph may not be complete. Using this notation, the Searcher's subproblem can be formulated as follows:

$$\text{Minimize} \quad \sum_{i\in N_h} w_i \sum_{l\in\Lambda'} z_i^l y_l t_i \tag{18}$$

$$\text{subject to} \quad \sum_{j\in N^+(i)} x_{ij} = 1 \quad \forall i \in N_h, \tag{19}$$

$$\sum_{j\in N^+(i)} x_{ij} - \sum_{j\in N^-(i)} x_{ji} = 0, \quad \forall i \in N_h, \tag{20}$$

$$\sum_{i\in N^+(0)} x_{0i} = S, \tag{21}$$

$$t_i + v_j + d_{ij} - t_j \leq M(1 - x_{ij}), \quad \forall (i, j) \in E, j \neq 0, \tag{22}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E, \tag{23}$$

$$t_i \geq 0, \quad \forall i \in N. \tag{24}$$

In this formulation, the objective function (18) corresponds to the reduced cost. Constraint (19) ensures that each location is visited exactly once. Constraint (20) is the flow conservation constraint for the search teams. Constraint (21) ensures that each search team exits the origin exactly once. Constraint (22) computes the visit times for each location. This constraint also eliminates the infeasible subtours.

The Searcher's subproblem is a generalization of the multiple travelling repairman problem (Luo, Qin, & Lim, 2014) with weighted delays. Because the travelling repairman problem is NP-hard, the Searcher's subproblem is also NP-hard. This indicates that the Searcher's subproblems are hard to solve. Therefore, in the next sections, we develop efficient algorithms to solve the Searcher's subproblems.

Remark 4. Even though the Searcher's subproblem is NP-hard in general, if $S = 1$ and $d_{ij} = d_j$, for all $(i, j) \in E$, then this problem can be solved in polynomial time. Specifically, in this case, the Searcher's subproblem is equivalent to a single machine scheduling problem to minimize the weighted sum of completion times. This problem can be solved by using Smith's rule (Smith, 1956), that is visiting nodes in non-increasing order of $\frac{w_i \sum_{l \in \Lambda'} z_i^l y_l}{d_i + v_i}$.

3.1.1. A simulated annealing search strategy generator for the Searcher's subproblem

In this section, we develop a simulated annealing (SA) algorithm to rapidly obtain a high quality solution to the Searcher's subproblem. SA is a probabilistic search method to approximate the global optimal solution in a large search space (Kirkpatrick, Gelatt, & Vecchi, 1983). Our proposed SA algorithm starts with an initial solution obtained from the strategies that are used with a positive probability in the current LPM-RS. We then randomly generate a neighborhood solution by applying one of the following operations: (1) randomly selecting two hiding locations and swapping their places in the search schedule. (2) randomly selecting a hiding location and randomly assigning it to another search team. If the newly generated solution leads to a better objective function than the current solution, then it replaces the current solution. The new solution may still replace the current solution even if it leads to a worse objective function value. This happens with probability $e^{-\Delta/T}$, where Δ is the amount of deterioration in the objective function if the new solution replaces the current solution and T is a parameter called temperature. The algorithm starts with a relatively high temperature and reduces the temperature as it proceeds. Therefore, in the initial iterations, the algorithm tends to explore more areas in the solution space and in the final phases, it tries to exploit the current area.

SA is an efficient algorithm in providing a fast high quality solution. However, the optimality of a solution cannot be declared when SA fails to result in an improving solution. Therefore, there is a need for an exact method to prove the optimality. To this end, we propose a branch and price algorithm to solve the Searcher's subproblem to optimality.

3.1.2. A branch and price algorithm to solve the Searcher's subproblem

We employ the Dantzig-Wolfe decomposition to reformulate the problem as a set covering model and develop a branch and price (BP) algorithm to solve the Searcher's subproblem. Let Ω denote the set of all admissible search schedules for the search teams. The expected damage of search schedule $s \in \Omega$ is: $ED_s = \sum_{i \in N_h} w_i \sum_{l \in \Lambda'} z_i^l y_l t_{is}$, where t_{is} is the time at which schedule s completes inspection at location i. Let a_{is} denote a binary parameter indicating if search schedule s visits location i. In other words, a_{is} is equal to 1 if search schedule s visits location i, 0 otherwise. For each schedule s, binary variable θ_s is equal to 1 if the schedule s is assigned to a search team, 0 otherwise. Using this notation, the following set covering formulation can be written for the Searcher's subproblem:

$$\text{MP} \quad \text{Minimize} \quad \sum_{s \in \Omega} ED_s \theta_s \tag{25}$$

$$\text{subject to} \quad \sum_{s \in \Omega} a_{is} \theta_s \geq 1, \quad \forall i \in N_h, \tag{26}$$

$$\sum_{s \in \Omega} \theta_s \leq S, \tag{27}$$

$$\theta_s \in \{0, 1\}, \quad \forall s \in \Omega. \tag{28}$$

The objective function (25) is the expected total damage corresponding to the selected search schedules. Constraint (26) indicates that every location needs to be visited at least once. Constraint (27) ensures that the number of selected search schedules is limited by the number of search teams. Constraint (28) is the integrality constraint.

Column generation

This section presents a column generation approach to solve the linear programming relaxation of the model (25)-(28), with the addition of appropriate branching decisions. We call the linear relaxation of the model (25)-(28) the Linear Master Problem (LMP). The optimal solution to LMP is a lower bound to the corresponding node in the branch and bound tree. The Column generation algorithm starts with a small subset of columns $\Omega' \subset \Omega$ and generates new columns as needed. LMP with restricted columns is called restricted linear master problem (RLMP) and the problem of finding a new column is called the pricing subproblem. In other words, given the dual solution of the current RLMP, the goal of the pricing subproblem is to find columns in $\Omega \backslash \Omega'$ with negative reduced cost. If we are unable to find such a column, then the optimal solution of current RLMP is the optimal solution of LMP and we can terminate the procedure. Otherwise, we add new columns to RLMP and repeat the process.

The pricing subproblem

At each branch and bound node, the column generation procedure is performed to get a lower bound of the Searcher's subproblem. The RLMP is

$$\text{Minimize} \quad \sum_{s \in \Omega'} ED_s \theta_s \tag{29}$$

$$\text{subject to} \quad \sum_{s \in \Omega'} a_{is} \theta_s \geq 1, \quad \forall i \in N_h, \tag{30}$$

$$\sum_{s \in \Omega'} \theta_s \leq S, \tag{31}$$

$$0 \leq \theta_s \leq 1, \quad \forall s \in \Omega'. \tag{32}$$

To check if the optimal solution of the current RLMP is optimal for LMP, we solve the pricing subproblem. In other words, we look for a new column with a negative reduced cost. We use $\pi =$

$(\pi_1, \pi_2, \ldots, \pi_n)$ and μ to denote the corresponding dual variables for constraints (30) and (31), respectively. We use $(\hat{\pi}, \hat{\mu})$ to denote the optimal dual solution for the current RLMP. Using this dual solution, the reduced cost of schedule s is $RC_s = ED_s - \sum_i \hat{\pi}_i a_{is} - \hat{\mu}$. Hence, we can formulate the pricing subproblem as follows:

$$\text{Minimize} \quad \sum_{i \in N_h} \left(w_i \sum_{l \in \Lambda'} z_i^l y_l t_i - \hat{\pi}_i \sum_{j \in N^+(i)} x_{ij} \right) - \hat{\mu} \tag{33}$$

$$\text{subject to} \quad \sum_{j \in N^+(0)} x_{0j} = 1, \tag{34}$$

$$\sum_{j \in N^+(i)} x_{ij} = \sum_{j \in N^-(i)} x_{ji}, \quad \forall i \in N_h, \tag{35}$$

$$\sum_{j \in N^+(i)} x_{ij} \leq 1, \quad \forall i \in N_h, \tag{36}$$

$$t_i + v_j + d_{ij} - t_j \leq M(1 - x_{ij}), \quad \forall (i,j) \in E, j \neq 0, \tag{37}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E. \tag{38}$$

The objective is to minimize the reduced cost given in (33). Constraint (34) ensures that the search schedule starts from the origin. Constraint (35) is the flow conservation constraint. Constraint (36) ensures that each location is visited at most once. Constraint (37) indicates that, if location j is visited immediately after location i, then the visit time of location j is at least $t_i + v_j + d_{ij}$. Constraint (38) is the integrality constraint for variables $\{x_{ij}\}$.

A special case of this pricing subproblem has been shown to be NP-hard in Luo et al. (2014). Therefore, this pricing subproblem is also NP-hard. Thus, solving the formulation (33)-(38) directly maybe computationally expensive. To this end, we propose a branch, bound and remember (BBR) algorithm to solve the pricing subproblems. BBR is a branch and bound algorithm that uses memory to avoid revisiting partial solutions that have already been visited. In BBR, before branching on a partial solution, it is looked up in the memory to see if it has already been visited. This idea has been used in different fields of combinatorial optimization (Jouglet, Baptiste, & Carlier, 2004; Morin & Marsten, 1976; Sewell & Jacobson, 2012). The details of the proposed BBR algorithm are as follows.

- Branching: The branch and bound algorithm explores partial solutions through an enumeration tree. Let a partial solution denoted as $\mathcal{P} = (U, l_1, l_2, \ldots, l_p)$, where U is the sets of unvisited locations; p is the number of nodes visited by the partial solution and l_i is the ith visited location for $i = 1, 2, \ldots, p$. Branching on a partial solution means extending it by removing one of the locations from U and adding it to the list of visited locations. This leads to new partial solutions that need to be evaluated by computing upper and lower bounds for their objective function.
- Lower Bound: To obtain a lower bound for a partial solution, we use the Lagrangian relaxation method. Relaxing the constraint that each location needs to be visited at most once results in the following Lagrangian problem:

$$\text{Minimize} \quad \sum_{i \in N_h} \left(w_i \sum_{l \in \Lambda'} z_i^l y_l t_i - \hat{\pi}_i \sum_{j \in N^+(i)} x_{ij} \right) - \hat{\mu} + \sum_{i \in N_h} \lambda_i \left(1 - \sum_{j \in N^+(i)} x_{ij} \right) \tag{39}$$

$$\text{subject to} \quad \sum_{j \in N^+(0)} x_{0j} = 1, \tag{40}$$

$$\sum_{j \in N^+(i)} x_{ij} = \sum_{j \in N^-(i)} x_{ji}, \tag{41}$$

$$t_i + v_j + d_{ij} - t_j \leq M(1 - x_{ij}), \quad \forall (i,j) \in E, j \neq 0, \tag{42}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E, \tag{43}$$

where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$, $\lambda_i \geq 0$ is a vector of Lagrangian multipliers. This problem can be solved in pseudo-polynomial time using a dynamic programming approach. It is well-known that, for any vector of Lagrangian multipliers $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$, $\lambda_i \geq 0$, the optimal solution to the Lagrangian problem (39)–(43) gives a lower bound to the original problem (33)–(38). Moreover, the optimal solution to the Lagrangian problem (39)–(43) is a concave function of λ. We use a subgradient algorithm to estimate the optimal Lagrangian multipliers.
- Use of memory: BBR memorizes already visited partial solutions. Before considering a partial solution $\mathcal{N} = (U, l_1, l_2, \ldots, l_p)$, BBR checks the memory to see if there is a partial solution with the same set of unvisited nodes, U, and the same current location, l_p. If there is such a partial solution $\mathcal{M} = (U' = U, l'_1, l'_2, \ldots, l'_p = l_p)$, such that the objective function computed so far by $\mathcal{N}$ is greater than or equal to the objective function computed so far by $\mathcal{M}$ and the time computed so far by $\mathcal{N}$ is greater than or equal to the time computed so far by $\mathcal{M}$, then $\mathcal{N}$ is dominated by $\mathcal{M}$ and $\mathcal{N}$ can be pruned. BBR uses a hash table to store already visited partial solutions along with their corresponding objective function and time values.
- Search strategy: Preliminary numerical tests revealed that the best first search (BFS) strategy, in which nodes with smaller lower bounds have higher priority of being selected, was unable to rapidly find the optimal solution for some problems. Using BFS, the search algorithm tends to spend a lot of time exploring the nodes in the middle of the enumeration tree rather than choosing the nodes that are deeper in the enumeration tree. Therefore, the algorithm generated very few complete solutions before exploring all of the nodes in the middle of the enumeration tree. To avoid this issue, we use the cyclic best first search (CBFS) strategy (Kao, Sewell, & Jacobson, 2009; Kao, Sewell, Jacobson, & Hall, 2012; Morrison, Sauppe, Zhang, Jacobson, & Sewell, 2017; Sewell, Sauppe, Morrison, Jacobson, & Kao, 2012) in our BB&R algorithm. CBFS systematically chooses the best nodes at all possible depths in the enumeration tree. Specifically, starting by choosing the best nodes at depth 1, CBFS continues to choose the best nodes at deeper levels in the enumeration tree until it reaches the deepest level. At this point, the algorithm goes back to depth 1 and repeats this process until all nodes are explored.

Branching strategy

At each node of the branch and bound tree, we use the column generation algorithm to obtain an optimal solution to the linear relaxation of the model (25)–(28). The resulting value is a lower bound for the corresponding node. If this lower bound is greater than or equal to the current upper bound, then the node is pruned. Otherwise, we must branch further. If the solution obtained at the current node is integral and the solution value is smaller than the current upper bound, then the upper bound is updated.

Branching on θ_s variables is not beneficial. Fixing θ_s variables at 0 leads to stopping BBR from generating a set of specific schedules,

which complicates the solution of the pricing subproblem. Moreover, ruling out a specific schedule is not possible until almost all of the schedule has already been considered by the enumeration tree. Therefore, branching on θ_s variables is not helpful in ruling out candidate solutions early on in the enumeration tree. Thus, it is better to opt for other branching strategies that are more compatible with the pricing subproblem and BBR algorithm. In our proposed branch and price algorithm, we branch on the edges. We choose the edge (i, j) with fractional flow $\hat{x}_{ij}$ farthest to an integer value to branch on. The flow on each edge can be calculated as $\hat{x}_{ij} = \sum_{s\in\Omega'} \theta_s b_{ijs}$, where b_{ijs} is a binary parameter which is equal to 1 if edge (i, j) is used in schedule s, 0 otherwise. After selecting the edge (i, j) to branch on, two child nodes are created: by setting $x_{ij} = 0$ and $x_{ij} = 1$. Fixing $x_{ij} = 0$ implies that the edge (i, j) is forbidden. To enforce this constraint in the pricing subproblem, we delete edge (i, j) from E and, for all schedules s containing edge (i, j), we remove the corresponding variable θ_s from the RLMP. Setting $x_{ij} = 1$ implies that the edge (i, j) must be used. To enforce this constraint in the pricing subproblem, we eliminate all of the edges (i, j') and (i', j) with $i \neq i'$ and $j \neq j'$. Note that branching on the edges only leads to changes in the underlying graph and it does not require modifying the BBR algorithm.

3.2. The Hider's subproblem

In this section, we formulate the Hider's subproblem to obtain an improving hiding strategy. The Hider's subproblem is formulated as follows:

$$\text{Maximize} \quad \sum_{i\in N_h} z_i w_i \sum_{k\in K} t_i^k x_k \tag{44}$$

$$\text{subject to} \quad \sum_{i\in N_h} z_i \leq H, \tag{45}$$

$$z_i \in \{0, 1\}, \quad \forall i \in N_h. \tag{46}$$

In this formulation, z_i is a binary variable which is equal to 1 if the Hider hides an object in location i. Eq. (44), the objective function, is the expected total damage. Eq. (45) corresponds to the constraint on the number of hidden objects. Finally, constraint (46) is the integrality constraint for variable z_i.

The Hider's subproblem is a special case of 0–1 knapsack problem with unit item weights. This problem can be solved in polynomial time by sorting the hiding locations i in a non-increasing order of $w_i \sum_{k\in K} t_i^k$ and choosing the first H hiding locations.

3.3. Overall solution procedure and bounds

Algorithm 1 provides the pseudo-code for the overall solution procedure. The column and row generation algorithm begins by randomly generating a set of initial strategies. Then, using this set of strategies, the LPM-RS is solved to obtain a solution $\bar{\mathbf{x}}$ and a vector of dual values $\bar{\mathbf{y}}$. Dual values $\bar{\mathbf{y}}$ are then used in the Searcher's subproblem to generate a new search strategy. If a new search strategy with a smaller expected total damage is obtained, it is added to K'. Then the Hider's subproblem is solved to generate a new hiding strategy. If a new hiding strategy with a greater expected total damage is obtained, it is added to Λ'. If, during the last two steps, either K' or Λ' has been updated, then the process is repeated; otherwise the procedure terminates. Because the number of possible strategies for both players is finite, the algorithm terminates after a finite number of iterations. Moreover, when the algorithm terminates, no player can improve the expected total damage in their own favor by changing their strategies. Therefore, by definition, the algorithm returns a Nash equilibrium point upon termination.

Algorithm 1: Pseudo-code for the overall solution algorithm.

```
1  Initialize sets K' and Λ'.
2  Solve LPM-RS. Let x̄ = [x_k], ȳ = [y_l] and u be the optimal
   primal solution, dual solution and objective function value,
   respectively.
3  Solve the Searcher's subproblem using ȳ as dual values and
   let t* = [t_i*] denote the optimal solution.
4  if v > Σ_{i∈N_h} t_i* w_i Σ_{l∈Λ'} z_i^l y_l then
5  |  Add the new search strategy t* to K'.
6  end
7  Solve the Hider's subproblem using x̄ as primal values and let
   z* = [z_i*] be the optimal solution.
8  if Σ_{i∈N_h} z_i* w_i Σ_{k∈K'} t_i^k x_k > v then
9  |  Add the new hiding strategy z* to Λ'.
10 end
11 if K' or Λ' has been updated then
12 |  Go to Line 2.
13 else
14 |  Return v as the value of the game.
15 |  Terminate the procedure.
16 end
```

During solution procedure, we have access to lower and upper bounds on the value of the game so that we can terminate the algorithm when a desired solution quality is reached. The following lemma offers solution bounds on the value of the game which can be computed in every iteration of the solution algorithm.

Lemma 2. *Optimal solution to the Searcher's (Hider's) subproblem yields a lower bound (an upper bound) to the expected total damage in equilibrium.*

Proof. Because the Hider's strategy set is restricted, i.e., $\Lambda' \subseteq \Lambda$, solving the Searcher's subproblem leads to a lower bound on the expected total damage in equilibrium. Similarly, because the Searcher's strategy space is restricted, solving the Hider's subproblem leads to an upper bound on the value of the game. □

Remark 5. Note that, in order for the bound in Lemma 2 to be valid, the Searcher's subproblem should be solved to optimality. In general, this bound is not monotone over the iterations, this is called the yo-yo effect (Lübbecke, 2011).

4. Numerical experiments

In this section, we perform computational experiments to investigate the efficiency of the proposed algorithms and to gain insight into the properties of the game. The algorithms are coded in C++ and CPLEX 12.8 solver has been used to solve the LPs and the pricing subproblems. The computational experiments are performed on a computer with 2.6 GH processor and 32 GB of RAM. We used a maximum running time of 2 hours (7200 seconds) for all of the algorithms employed in this section.

Our base set of test instances consists of randomly generated instances. The location of the potential hiding places are randomly generated on a hypothetical square with side of n units. Manhattan distance is used to compute the travel times, d_{ij}, between these locations. Location weights, w_i, are generated randomly from the range $[1, n]$. Similarly, the visit times, v_i, are generated randomly from the range $[1, n]$.

In our first experiment, we compare the performances of different methods for solving the Searcher's subproblem. Specifically, we consider two cases: the flow-type mathematical formulation (MF) of (18) to (24) and the branch and price algorithm (BP) developed

Table 1
Average run times for different number of search teams (in seconds).

n	S=2		S=3		S=4		S=5	
	MF	BP	MF	BP	MF	BP	MF	BP
10	2.02	0.26	1.42	0.34	0.82	0.15	0.52	0.32
15	15.67	2.59	8.01	2.28	3.94	0.69	1.11	0.18
20	7200.00	83.82	838.34	27.42	60.04	40.62	13.83	2.71
25	7200.00	7200.00	7200.00	2696.27	7200.00	4055.85	4537.37	520.81
30	7200.00	7200.00	7200.00	7200.00	7200.00	5480.06	7200.00	1848.59
Mean	4323.54	2897.33	3049.55	1985.26	2892.96	1915.47	2350.57	474.52

Table 2
Average run times for different number of hidden objects (in seconds).

n	H=1		H=2		H=3		H=4	
	MF	BP	MF	BP	MF	BP	MF	BP
10	1.34	0.53	1.08	0.18	1.14	0.18	1.19	0.19
15	4.00	0.71	6.83	0.80	9.37	1.32	8.51	2.91
20	1914.93	38.84	1913.25	21.70	2015.77	51.84	2268.26	42.18
25	5506.43	4214.13	6230.94	4296.92	7200.00	2787.20	7200.00	3174.74
30	7200.00	5192.93	7200.00	5420.28	7200.00	4901.70	7200.00	6214.18
Mean	2925.34	1889.43	3070.42	1947.98	3285.26	1548.45	3335.59	1886.84

Table 3
Average optimality gap for different number of search teams (%).

n	S=2		S=3		S=4		S=5	
	MF	BP	MF	BP	MF	BP	MF	BP
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	0.72	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25	43.50	0.40	35.33	0.00	6.49	0.12	0.50	0.00
30	130.27	8.31	75.61	5.94	45.30	0.10	9.77	0.03
Mean	34.90	1.74	22.19	1.19	10.36	0.04	2.05	0.01

Table 4
Average optimality gap for different number of hidden objects (%).

n	H=1		H=2		H=3		H=4	
	MF	BP	MF	BP	MF	BP	MF	BP
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	0.32	0.00	0.23	0.00	0.08	0.00	0.10	0.00
25	22.78	0.12	18.94	0.23	21.11	0.13	23.01	0.05
30	61.12	4.60	61.36	2.72	53.90	2.41	84.56	4.65
Mean	16.84	0.94	16.11	0.59	15.02	0.51	21.53	0.94

Table 5
Percentage of cases in which an equilibrium is reached for different number of search teams.

n	S=2		S=3		S=4		S=5	
	MF	BP	MF	BP	MF	BP	MF	BP
10	100	100	100	100	100	100	100	100
15	100	100	100	100	100	100	100	100
20	0	100	100	100	100	100	100	100
25	0	0	0	100	0	50	50	100
30	0	0	0	0	0	75	0	75
Mean	40	60	60	80	60	85	70	95

Table 6
Percentage of cases in which an equilibrium is reached for different number of hidden objects.

n	H=1		H=2		H=3		H=4	
	MF	BP	MF	BP	MF	BP	MF	BP
10	100	100	100	100	100	100	100	100
15	100	100	100	100	100	100	100	100
20	75	100	75	100	75	100	75	100
25	25	50	25	50	0	75	0	50
30	0	50	0	25	0	50	0	25
Mean	60	80	60	75	55	85	55	75

in section 3.1.2. Note that the second algorithm (BP) uses SA first to find an improving search strategy. If SA is not able to find an improving search strategy, branch and price algorithm is used. We consider 16 instances for each problem size, with various values of $S \in \{2, 3, 4, 5\}$ and $H \in \{1, 2, 3, 4\}$. Tables 1 and 2 compare the average run times of these algorithms for different values of the number of search teams and the number of hidden objects, respectively. Note that, in Table 1, run times are averaged over all values of $H \in \{1, 2, 3, 4\}$. Similarly, in Table 2, run times are averaged over all values of $S \in \{2, 3, 4, 5\}$. Based on Tables 1 and 2, for all problem sizes, BP performs significantly better than MF. Moreover, for both MF and BP algorithms, the run time increases as n increases and it, generally, decreases as S increases. However, no clear pattern is observed of the effect of increasing H on the run time.

Tables 3 and 4 demonstrate the average optimality gap, in percent, obtained for different number of search teams and hidden objects, respectively. In these experiments, while both algorithms are able to find the optimal solution for problems of smaller size, BP performs significantly better than MF for larger-sized problems. Moreover, for both MF and BP algorithms, the average gap increases as n increases and it decreases as S increases.

Tables 5 and 6 display the percentages of the cases in which an exact equilibrium is reached, for different values of S and H, respectively. Based on these tables, both algorithms are able to solve all problem instances for $n = 10$ and $n = 15$. However, for larger instances, BP performs better than MF. Moreover, as n increases, for both algorithms, the number of cases in which an exact equilibrium is reached decreases in most cases. Based on Table 5, as S increases, for most cases both algorithms are able to solve more instances to optimality. Again, Table 6 does not present any pattern for the effect of H on the performances of the algorithms.

Table 7 exhibits the detailed optimality gap and run time results for $n = 15$ and $n = 30$. Based on this table, for every instance of the problem, BP performs better than MF both in terms of solution quality and run times. Moreover, the differences in performance of the two algorithms for problem instances with $n = 30$ are larger than the performance differences for problem instances with

Table 7
Detailed optimality gaps (in percentages) and run times (in seconds) for $n = 15$ and $n = 30$.

S	H	n=15				n=30			
		GAP		Time		GAP		Time	
		MF	BP	MF	BP	MF	BP	MF	BP
2	1	0.00	0.00	12.30	2.11	123.18	11.72	7200.00	7200.00
2	2	0.00	0.00	16.79	1.43	93.19	7.15	7200.00	7200.00
2	3	0.00	0.00	19.95	2.56	80.55	5.54	7200.00	7200.00
2	4	0.00	0.00	13.63	4.26	224.15	8.83	7200.00	7200.00
3	1	0.00	0.00	2.09	0.30	77.19	6.70	7200.00	7200.00
3	2	0.00	0.00	5.33	0.63	93.23	3.34	7200.00	7200.00
3	3	0.00	0.00	10.85	1.41	81.89	4.11	7200.00	7200.00
3	4	0.00	0.00	13.76	6.77	50.11	9.63	7200.00	7200.00
4	1	0.00	0.00	1.02	0.19	33.01	0.00	7200.00	6335.38
4	2	0.00	0.00	3.73	0.91	50.94	0.39	7200.00	7200.00
4	3	0.00	0.00	5.66	1.17	43.27	0.00	7200.00	5129.00
4	4	0.00	0.00	5.34	0.50	53.98	0.00	7200.00	3255.84
5	1	0.00	0.00	0.61	0.24	11.11	0.00	7200.00	36.23
5	2	0.00	0.00	1.48	0.22	8.07	0.00	7200.00	80.43
5	3	0.00	0.00	1.01	0.15	9.91	0.00	7200.00	77.60
5	4	0.00	0.00	1.33	0.12	10.00	0.14	7200.00	7200.00
Mean		0.00	0.00	7.18	1.44	65.24	3.60	7200.00	5432.16

Table 8
Average number of search strategies generated (SSG), search strategies used (SSU), and percentage of generated search strategies used in equilibrium.

n	S=2			S=3			S=4			S=5		
	SSG	SSU	%	SSG	SSU	%	SSG	SSU	%	SSG	SSU	%
10	23.50	5.25	22.34	18.50	5.00	27.03	13.25	4.25	32.08	14.00	3.50	25.00
15	43.50	7.75	17.82	36.00	7.50	20.83	33.75	5.50	16.30	18.75	4.00	21.33
20	109.75	12.25	11.16	97.25	10.75	11.05	68.50	9.75	14.23	53.50	8.50	15.89
Mean	58.92	8.42	17.11	50.58	7.75	19.64	38.50	6.50	20.87	28.75	5.33	20.74

Table 9
Average number of search strategies generated (SSG), search strategies used (SSU), and percentage of generated search strategies used in equilibrium.

n	H=1			H=2			H=3			H=4		
	SSG	SSU	%	SSG	SSU	%	SSG	SSU	%	SSG	SSU	%
10	22.25	5.50	24.72	17.25	4.00	23.19	16.25	4.25	26.15	13.50	4.25	31.48
15	36.25	7.00	19.31	36.00	6.00	16.67	31.75	5.50	17.32	28.00	6.25	22.32
20	86.50	11.50	13.29	73.25	8.75	11.95	87.00	11.25	12.93	82.25	9.75	11.85
Mean	48.33	8.00	19.11	42.17	6.25	17.27	45.00	7.00	18.80	41.25	6.75	21.89

$n = 15$. For the cases in which BP algorithm cannot reach the optimum, the optimality gap is much smaller than the gap obtained via MF algorithm. In the remaining experiments in this section, we will only consider BP algorithm as it is the more efficient algorithm.

Because our solution approach involves iteratively generating new strategies, it is important to investigate the size of the generated strategy sets for both players. It is also interesting to see how many of the generated strategies are used with a positive probability in the final solution. For this experiment, we only consider the problem sizes for which an equilibrium has been reached for all values of S and H, i.e., $n \in \{10, 15, 20\}$. Tables 8 and 9 present the average number of search strategies generated (SSG), as well as the average number of search strategies used (SSU) in the final solution, for different values of S and H, respectively. The columns labeled "%" display SSU as a percentage of SSG. Based on these tables, the number of generated search strategies increases as n increases. However, only a small percentage of these strategies are used in the final solution. Moreover, this percentage decreases as n increases. Based on Table 8, as S increases, in most cases both SSG and SSU decrease. No clear pattern is observed regarding the effect of H on the number of generated or used search strategies.

Tables 10 and 11 present the average number of hiding strategies generated (HSG), as well as the average number of hiding strategies used (HSU) in the final solution, for different values of S and H, respectively. The columns labeled "%" display HSU as a pecentage of HSG. Based on these tables, the number of generated hiding strategies is smaller than the number of generated search strategies shown in Tables 8 and 9. However, the number of used hiding strategies is comparable to the number of used search strategies shown in Tables 8 and 9. Therefore, as seen in Tables 10 and 11, the percentage of generated hiding strategies used in the final solution is larger than the percentage of generated search strategies used in the final solution. The tables also show that the number of generated hiding strategies increases as n increases. Based on Table 10, as S increases, the number of used hiding strategies decreases. However, no clear pattern can be observed on the effect of S on the number of generated hiding strategies. Moreover, no clear pattern is observed regarding the effect of H on the number of generated or used hiding strategies.

Fig. 3 illustrates the convergence of the solution for different problem sizes. The number of iterations needed for convergence increases as the problem size increases. Moreover, the bounds are not monotone over the iterations and the yo-yo effect is visible due to the non-monotonicity of the bound in Lemma 2. Another interesting observation is that, the upper bound values stabilize much earlier than the termination of the algorithm. This means that after the upper bound values stabilize, we can terminate the solution algorithm without undermining the solution quality too much.

Table 10
Average number of hiding strategies generated (HSG), hiding strategies used (HSU), and percentage of generated strategies used in equilibrium.

n	S=2			S=3			S=4			S=5		
	HSG	HSU	%	HSG	HSU	%	HSG	HSU	%	HSG	HSU	%
10	11.25	5.25	46.67	9.75	4.00	41.03	7.50	2.50	33.33	7.50	1.75	23.33
15	16.50	7.25	43.94	17.50	6.75	38.57	14.50	4.50	31.03	10.75	2.25	20.93
20	35.25	12.00	34.04	39.25	9.75	24.84	30.50	6.50	21.31	27.50	5.25	19.09
Mean	21.00	8.17	41.55	22.17	6.83	34.81	17.50	4.50	28.56	15.25	3.08	21.12

Table 11
Average number of hiding strategies generated (HSG), hiding strategies used (HSU), and percentage of generated strategies used in equilibrium.

n	H=1			H=2			H=3			H=4		
	HSG	HSU	%	HSG	HSU	%	HSG	HSU	%	HSG	HSU	%
10	6.75	3.00	44.44	12.00	3.00	25.00	9.00	3.50	38.89	8.25	4.00	48.48
15	9.50	5.00	52.63	15.75	5.25	33.33	17.00	5.25	30.88	17.00	5.25	30.88
20	13.50	7.25	53.70	34.75	6.75	19.42	36.75	11.25	30.61	47.50	8.25	17.37
Mean	9.92	5.08	50.26	20.83	5.00	25.92	20.92	6.67	33.46	24.25	5.83	32.25

Fig. 3. Convergence of the proposed column and row generation algorithm.

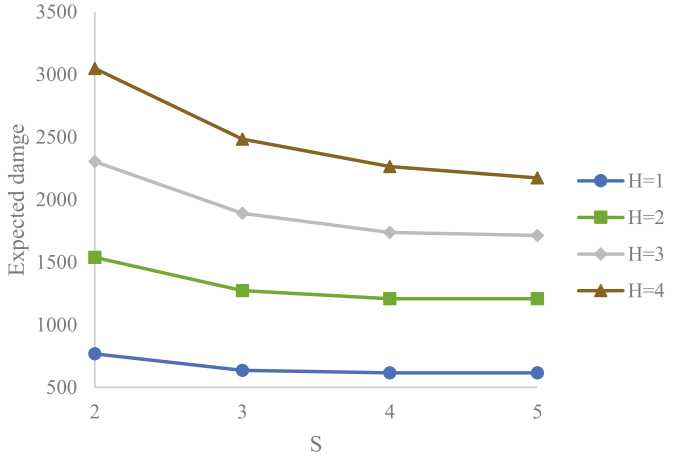


Fig. 4. The effect of number of search teams on the expected total damage in equilibrium.

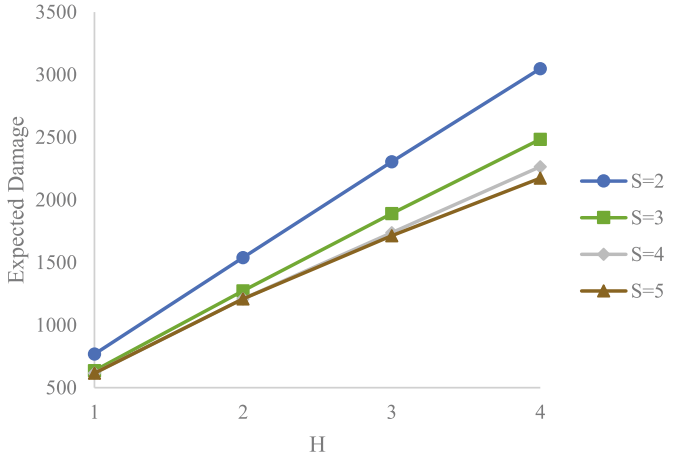


Fig. 5. The effect of number of hidden objects on the expected total damage in equilibrium.

Next, we study the effect of the number of search teams and objects hidden on the expected total damage in equilibrium. An experiment is designed with $S \in \{2, 3, 4, 5\}$ and $H \in \{1, 2, 3, 4\}$. Fig. 4 demonstrates that, as the number of search teams increases, the expected damage decreases. Moreover, a diminishing returns effect is visible in the reduction in expected total damage for each unit increment in S. Fig. 5 shows that the expected total damage in-

creases almost linearly with the number of hidden objects. Moreover, the rate of increase is greater for smaller number of search teams.

5. Conclusions and future research

In this paper, we propose a weighted search game model in which players want to minimize/maximize a weighted search time. We characterize the solution for a special case of the problem. In order to solve the general form of this problem, we develop an efficient algorithm based on column and row generation. To solve the arising subproblems for the Searcher, we propose a branch and price approach. We also present bounds for the expected total damage in equilibrium which can be used in each iteration of the column and row generation algorithm to estimate the quality of the current solution so far and terminate the algorithm if the solution quality is satisfactory. We then perform numerical experiments to investigate the performance of the proposed solution algorithms. Our computational results demonstrate the efficiency of the use of branch and price algorithm to solve the Searcher's subproblems. They also display a diminishing returns effect when increasing the number of search teams.

This paper addresses a gap in the literature by considering a search game model with different node weights. It also develops efficient algorithms to solve this search game. However, there are some limitations that need to be addressed in future research. In our proposed model, the Hider can only hide objects on the vertices of the graph. This limitation can be partly addressed by further discretizing the edges and placing extra nodes. However, explicitly modelling hiding on the edges may yield more realistic results. Extending the model to accommodate uncertainty of parameters, such as travel times and location weights, is another avenue for future research in this area.

References

Alpern, S. (2010). Search games on trees with asymmetric travel times. *SIAM Journal on Control and Optimization, 48*(8), 5547–5563.

Alpern, S. (2011). Find-and-fetch search on a tree. *Operations Research, 59*(5), 1258–1268.

Alpern, S. (2017). Hide-and-seek games on a network, using combinatorial search paths. *Operations Research, 65*(5), 1207–1214.

Alpern, S., Fokkink, R., Op Den Kelder, J., & Lidbetter, T. (2010). Disperse or unite? a mathematical model of coordinated attack. In *Proceedings of the international conference on decision and game theory for security* (pp. 220–233). Springer.

Alpern, S., & Gal, S. (2006). *The theory of search games and rendezvous, volume 55*. Springer Science & Business Media.

Alpern, S., & Lidbetter, T. (2013a). Mining coal or finding terrorists: The expanding search paradigm. *Operations Research, 61*(2), 265–279.

Alpern, S., & Lidbetter, T. (2013b). Searching a variable speed network. *Mathematics of Operations Research, 39*(3), 697–711.

Alpern, S., & Lidbetter, T. (2015). Optimal trade-off between speed and acuity when searching for a small object. *Operations Research, 63*(1), 122–133.

Assaf, D., & Zamir, S. (1987). Continuous and discrete search for one of many objects. *Operations Research Letters, 6*(5), 205–209.

Barron, E. (2013). *Game theory: An introduction*. John Wiley & Sons.

Baston, V., & Kikuta, K. (2015). Search games on a network with travelling and search costs. *International Journal of Game Theory, 44*(2), 347–365.

Baykal-Gürsoy, M., Duan, Z., Poor, H. V., & Garnaev, A. (2014). Infrastructure security games. *European Journal of Operational Research, 239*(2), 469–478.

Bram, J. (1963). *A 2-player n-region search game*. Center for Naval Analyses Alexandria Va Operations Evaluation Group.

Dagan, A., & Gal, S. (2008). Network search games, with arbitrary searcher starting point. *Networks, 52*(3), 156–161.

Gal, S. (1979). Search games with mobile and immobile hider. *SIAM Journal on Control and Optimization, 17*(1), 99–122.

Gal, S. (1980). Search games, volume 149 of Mathematics in Science and Engineering.

Gal, S. (2013). Search games: A review. In *Search Theory* (pp. 3–15). Springer.

Garnaev, A. (2012). *Search games and other applications of game theory*: 485. Springer Science & Business Media.

Garnaev, A., Baykal-Gürsoy, M., & Poor, H. V. (2016). A game theoretic analysis of secret and reliable communication with active and passive adversarial modes. *IEEE Transactions on Wireless Communications, 15*(3), 2155–2163.

Gittins, J., & Roberts, D. (1979). The search for an intelligent evader concealed in one of an arbitrary number of regions. *Naval Research Logistics Quarterly, 26*(4), 651–666.

Godinho, P., & Dias, J. (2010). A two-player competitive discrete location model with simultaneous decisions. *European Journal of Operational Research, 207*(3), 1419–1432.

Godinho, P., & Dias, J. (2013). Two-player simultaneous location game: Preferential rights and overbidding. *European Journal of Operational Research, 229*(3), 663–672.

Hohzaki, R. (2016). Search games: Literature and survey. *Journal of the Operations Research Society of Japan, 59*(1), 1–34.

Isaacs, R. (1965). *Differential games*. New York: Wiley.

Jouglet, A., Baptiste, P., & Carlier, J. (2004). Branch-and-bound algorithms for total weighted tardiness. In *Handbook of scheduling: algorithms, models, and performance analysis Edited by J. Y-T. Leung. Chapman&Hall/CRC*.

Kao, G. K., Sewell, E. C., & Jacobson, S. H. (2009). A branch, bound, and remember algorithm for the $1|r_i|\Sigma t_i$ scheduling problem. *Journal of Scheduling, 12*(2), 163.

Kao, G. K., Sewell, E. C., Jacobson, S. H., & Hall, S. N. (2012). New dominance rules and exploration strategies for the $1|r_i|\Sigma U_i$ scheduling problem. *Computational Optimization and Applications, 51*(3), 1253–1274.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.

Lancia, G., & Serafini, P. (2018). *Compact extended linear programming models*. Springer.

Lidbetter, T. (2013). Search games with multiple hidden objects. *SIAM Journal on Control and Optimization, 51*(4), 3056–3074.

Lidbetter, T., & Lin, K. (2017). Searching for multiple objects in multiple locations. ArXiv preprint arXiv:1710.05332.

Lin, K. Y., & Singham, D. I. (2015). *Robust search policies against an intelligent evader*. Naval Postgraduate School Monterey United States. Technical report

Lübbecke, M. E. (2011). Column generation. In *Wiley encyclopedia of Operations Research and Management Science. John Wiley and Sons Ltd*.

Luo, Z., Qin, H., & Lim, A. (2014). Branch-and-price-and-cut for the multiple traveling repairman problem with distance constraints. *European Journal of Operational Research, 234*(1), 49–60.

Matula, D. (1964). A periodic optimal search. *The American Mathematical Monthly, 71*(1), 15–21.

Morin, T. L., & Marsten, R. E. (1976). Branch-and-bound strategies for dynamic programming. *Operations Research, 24*(4), 611–627.

Morrison, D. R., Sauppe, J. J., Zhang, W., Jacobson, S. H., & Sewell, E. C. (2017). Cyclic best first search: Using contours to guide branch-and-bound algorithms. *Naval Research Logistics (NRL), 64*(1), 64–82.

Muter, I., Birbil, S. I., & Bülbül, K. (2013). Simultaneous column-and-row generation for large-scale linear programs with column-dependent-rows. *Mathematical Programming, 142*(1–2), 47–82.

Riedel, S., Smith, D., & McCallum, A. (2012). Parse, price and cut: Delayed column and row generation for graph based parsers. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning, pages 732–743*. Association for Computational Linguistics.

Roberts, D., & Gittins, J. (1978). The search for an intelligent evader: Strategies for searcher and evader in the two-region problem. *Naval Research Logistics Quarterly, 25*(1), 95–106.

Ruckle, W. H. (1991). A discrete search game. In *Stochastic games and related topics* (pp. 29–43). Springer.

Sewell, E. C., & Jacobson, S. H. (2012). A branch, bound, and remember algorithm for the simple assembly line balancing problem. *INFORMS Journal on Computing, 24*(3), 433–442.

Sewell, E. C., Sauppe, J. J., Morrison, D. R., Jacobson, S. H., & Kao, G. K. (2012). A bb&r algorithm for minimizing total tardiness on a single machine with sequence dependent setup times. *Journal of Global Optimization, 54*(4), 791–812.

Sharlin, A. (1987). Optimal search for one of many objects hidden in two boxes. *European Journal of Operational Research, 32*(2), 251–259.

Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly, 3*(1–2), 59–66.

Yasutake, S., Hatano, K., Kijima, S., Takimoto, E., & Takeda, M. (2011). Online linear optimization over permutations. In *Proceedings of the international symposium on algorithms and computation* (pp. 534–543). Springer.

Yolmeh, A., & Baykal-Gürsoy, M. (2017). A robust approach to infrastructure security games. *Computers & Industrial Engineering, 110*, 515–526.

Zoroa, N., Fernández-Sáez, M.-J., & Zoroa, P. (2013). Tools to manage search games on lattices. In *Search theory* (pp. 29–58). Springer.

Zoroa, N., Zoroa, P., & Fernández-Sáez, M. J. (2009). Weighted search games. *European Journal of Operational Research, 195*(2), 394–411.

Update

European Journal of Operational Research
Volume 292, Issue 3, 1 August 2021, Page 1209

DOI: https://doi.org/10.1016/j.ejor.2020.12.057

European Journal of Operational Research 292 (2021) 1209

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Erratum

Corrigendum to "Weighted Network Search Games with Multiple Hidden Objects and Multiple Search Teams" [European Journal of Operational Research, Vol. 289 (1) 2021, 338-349]☆

Abdolmajid Yolmeh[a], Melike Baykal-Gürsoy[b,*]

[a] Industrial and Systems Engineering Department, Rutgers University, 96 Frelinghuysen Rd, Piscataway, NJ 08854, USA
[b] Industrial and Systems Engineering Department, RUTCOR, CAIT, Rutgers University, 96 Frelinghuysen Rd, Piscataway, NJ 08854, USA

In the statement of Theorem 1, Eq. (2) and the full sentence should read:

$$E[\text{search order of the } i\text{th node}] = \sum_{k\in K} r_i^k x_k = \begin{cases} \dfrac{1}{w_i}\dfrac{\frac{m(m+1)}{2}}{\sum_{j=1}^{m}\frac{1}{w_j}}, & i = 1,2,\ldots,m, \\ \dfrac{1}{w_i}\dfrac{[\frac{l_1(l_1+1)}{2} - \frac{m(m+1)}{2}]}{\sum_{j=m+1}^{l_1}\frac{1}{w_j}} \le \dfrac{1}{w_i}\dfrac{\frac{m(m+1)}{2}}{\sum_{j=1}^{m}\frac{1}{w_j}}, & i = m+1,\ldots,l_1, \\ \dfrac{1}{w_i}\dfrac{[\frac{l_k(l_k+1)}{2} - \frac{l_{k-1}(l_{k-1}+1)}{2}]}{\sum_{j=l_{k-1}+1}^{l_k}\frac{1}{w_j}} \le \dfrac{1}{w_i}\dfrac{\frac{m(m+1)}{2}}{\sum_{j=1}^{m}\frac{1}{w_j}}, & i = l_{k-1}+1,\ldots,l_k,\ \ k\ge 2, \end{cases}$$

where $m = \arg_{i\in\{1,2,\ldots,n\}} \max \frac{\frac{i(i+1)}{2}}{\sum_{j=1}^{i}\frac{1}{w_j}}$, $l_1 = \arg_{m+1\le l\le n} \max \frac{(\frac{l(l+1)}{2} - \frac{m(m+1)}{2}}{\sum_{j=m+1}^{l}\frac{1}{w_j}}$ and for $k \ge 2$ until $l_k = n$, $l_k = \arg_{l_{k-1}+1\le l\le n} \max \frac{(\frac{l(l+1)}{2} - \frac{l_{k-1}(l_{k-1}+1)}{2}}{\sum_{j=l_{k-1}+1}^{l}\frac{1}{w_j}}$.

Proof. The proof of Theorem 1 should include "Note that for $l \le m$," instead of "Thus, we need to show that:"

Moreover, the proof should include the following additional statements after the sentence ending with "...so that this inequality holds.": For $m+1 \le l \le l_1$

$$\sum_{i=1}^{l}\bar{r}_i = \frac{m(m+1)}{2} + \frac{[\frac{l_1(l_1+1)}{2} - \frac{m(m+1)}{2}]}{\sum_{j=m+1}^{l_1}\frac{1}{w_j}}\sum_{i=m+1}^{l}\frac{1}{w_i} \ge \frac{m(m+1)}{2} + \frac{[\frac{l(l+1)}{2} - \frac{m(m+1)}{2}]}{\sum_{j=m+1}^{l}\frac{1}{w_j}}\sum_{i=m+1}^{l}\frac{1}{w_i} = \frac{l(l+1)}{2},$$

and similarly for all $l_{k-1} < l \le l_k$ until $l_k = n$, for $l_k = n$ for all $l_{k-1} < l < n$,

$$\sum_{i=1}^{l}\bar{r}_i = \frac{l_{k-1}(l_{k-1}+1)}{2} + \frac{[\frac{l_k(l_k+1)}{2} - \frac{l_{k-1}(l_{k-1}+1)}{2}]}{\sum_{j=l_{k-1}+1}^{l_k}\frac{1}{w_j}}\sum_{i=l_{k-1}+1}^{l}\frac{1}{w_i} \ge \frac{l(l+1)}{2},$$

both inequalities follow from the definition of l_i's. Therefore, the search strategy defined by (2) is feasible. □

Acknowledgement

The authors are grateful to Thomas Lidbetter for bringing this discrepancy to their attention and his helpful comments.

☆ This material is based upon work supported by the National Science Foundation (Grant No. 1901721) and the United States National Institute of Justice (Grant No.2018-R2-CX-0011)

DOI of original article: 10.1016/j.ejor.2020.06.046

* Corresponding author.
E-mail addresses: abdolmajid.yolmeh@rutgers.edu (A. Yolmeh), gursoy@soe.rutgers.edu (M. Baykal-Gürsoy).

https://doi.org/10.1016/j.ejor.2020.12.057