Complexity of Searching an Immobile Hider in a Graph

Bernhard von Stengel[1] and Ralph Werchner[2]

Version of June 20, 1996

Discrete Applied Mathematics **78** (1997) 235–249
(page numbers in this manuscript match only approximately)

Abstract. We study the computational complexity of certain search-hide games on a graph. There are two players, called searcher and hider. The hider is immobile and hides in one of the nodes of the graph. The searcher selects a starting node and a search path of length at most k. His objective is to detect the hider, which he does with certainty if he visits the node chosen for hiding. Finding the optimal randomized strategies in this zero-sum game defines a fractional path covering problem and its dual, a fractional packing problem. If the length k of the search path is arbitrary, then the problem is NP-hard. The problem remains NP-hard if the searcher may freely revisit nodes that he has seen before. In that case, the searcher selects a connected subgraph of k nodes rather than a path of k nodes. If k is logarithmic in the number of nodes of the graph, then the problem can be solved in polynomial time. This is shown using a recent technique called color-coding due to Alon, Yuster, and Zwick. The same results hold for edges instead of nodes, that is, if the hider hides in an edge and the searcher searches k edges on a path or on a connected subgraph.

Keywords. Covering and packing, game theory, graph search, NP-completeness.

[1] Theoretische Informatik, ETH Zürich, 8092 Zürich, Switzerland. Email: stengel@inf.ethz.ch
[2] FB Mathematik, Universität Frankfurt, Postfach 11 19 32, 60054 Frankfurt, Germany. Email: werchner@informatik.uni-frankfurt.de

1. Introduction

Communication networks are vulnerable to privacy violations. Surveillance of the network is one way to deter eavesdroppers. This gives rise to various models of pursuit and evasion on graphs and corresponding complexity considerations. One problem that has been examined in depth (see [5, 18] and references) is the search of a graph by a team of searchers traversing the edges of the graph in pursuit of a mobile fugitive. The minimum number of searchers necessary to detect the fugitive with certainty is called the search number of the graph. Computing it is easy for trees but NP-hard for general graphs [18]. Extensions of this approach to models of privacy in distributed environments are studied in [10].

We consider a similar, but more static situation: The hider selects an arbitrary node of the graph for hiding and must stay there. The searcher selects a starting node and traverses the graph along a path, and his search terminates after he has visited k nodes. The search is successful if and only if the searcher visits the node chosen for hiding. Both searcher and hider may randomize. The searcher tries to maximize the probability of detection and the hider tries to minimize it. We also study this game if the hiding location is an edge of the graph. Furthermore, we consider – for both node- and edge-searching – a variant of the game where the searcher may freely revisit a location that he has already searched. In that case the searcher searches a connected subgraph of size k rather than a path of length k.

The optimal searcher path problem has been investigated in the literature as one-sided search, where a prior hiding location distribution is given [24]. This is an important subcase for several reasons. Firstly, the uniform hiding distribution is optimal for the hider if the searcher can search all locations (nodes or edges of the graph) with equal maximal probability. With no restriction on the search length and search on nodes, deciding this question is NP-hard by reduction from the Hamiltonian path problem, as shown in [24] and repeated in Prop. 3.1 below. This argument fails if nodes can be revisited freely or if edges are searched, which is one reason why we study these alternative search models. Secondly, the two-sided search problem will in practice be solved by linear programming with delayed column generation (see Section 2; for heuristics see [6]). Thereby, a profitable column represents a new search path that improves the detection probability against the currently computed hiding distribution, just as for one-sided search. In Section 4, we describe an algorithm for that subproblem, specified for theoretical reasons as the separation problem. This algorithm works by dynamic programming with color-coding, and has polynomial running time if the search length is logarithmic in the size of the graph. In contrast, the standard dynamic programming method is not polynomial in that case.

One-sided search has also been studied for the more general case that the hider is overlooked with a certain probability when his location is inspected [24]. An additional generalization are costs to the searcher for switching from one location to the next [16]. These are even harder problems of Markovian dynamic programming

[25]. We have refrained from such generalizations since a chance of overlooking the hider makes it worthwhile to revisit a location, which complicates the game matrix substantially. Switch costs suggest the adoption of methods used for the Traveling Salesman Problem, which is a wide area of research [15]. Of course, these problems are interesting, and our approach should provide a starting point to tackle them. Our results extend trivially to the case where search can only start from a restricted set of locations, which we therefore do not discuss.

Our two-sided search model is a strong simplification of network surveillance. The bound on inspection resources, here given by the parameter k, is however typical of surveillance models (also studied in the context of inspection games, see [3]). The number of locations that can be searched is usually limited, due to budget constraints or, say, the time that an intruder stays in place and can be detected during a search. We have – somewhat arbitrarily – restricted the searcher's movement by the structure of the graph. Another application of our model may be police patrols in a road network, where k is the typical number of road segments that can be patrolled while a crime to be detected takes place. In arms control, the number k of inspection sites is typically limited by the terms of a disarmament treaty, for example the 1990 Treaty on Conventional Forces in Europe. Here our model applies if the searcher's movement is restricted due to geographic and organizational conditions that can be described by a graph. In these circumstances, the optimal detection probability is of interest as a measure of deterrence from illegal activity.

Books on search theory are [1, 11]. A recent general survey is [4], which would classify our work as two-sided search with immobile target. A continuous search of the edges of a graph is discussed in [12]. A discrete game where both hider and searcher choose a node and try to maximize respectively minimize their distance is studied in [8]. For further references see [4], except for articles related to the search number of a graph, which are cited in [10].

In Section 2, we define our problem as a zero-sum game and describe an equivalent linear program (LP). If the length k of the search path is unrestricted, then solving the game – in any of its variants – is NP-hard, as shown in Section 3. For search paths of logarithmic length in the size of the graph, we provide in Section 4 a polynomial-time algorithm, based on a recent color-coding technique [2] applied to the separation problem of the LP in question.

2. LP formulation

The input to our problem is a graph G with node set V and edge set E, and a positive integer k, called the *search length*. This gives rise to the described zero-sum game between two players, called *searcher* and *hider*. Both act simultaneously and will use randomized strategies. Either the nodes or the edges of G serve as *locations* for hiding which are inspected by the searcher. If the game is played on V, where the hider selects a node for hiding, we call it the *node search* game. If the game is played on E, we speak of *edge search.* The searcher selects a starting node and from there a path with up to k nodes in the case of node search, or up to k edges in the case of edge search. If a location that is repeated on the path is counted towards the allowed length k, then we call it a search on *paths.* Alternatively, the searcher may freely revisit a location on a path. This is called a search on *connected subgraphs.* That is, the deterministic strategies of the searcher are, respectively, the paths or connected subgraphs of G of size k, which we often just call *search paths* for brevity. Thus, the four variants of the game are node and edge search on paths and connected subgraphs, respectively.

We can assume that G is connected. Otherwise, it is easy to see that the game is just played on each connected component of G, and that the players choose the component for hiding and searching with a probability that is inversely proportional to the optimal detection probability in the respective component.

The game can be regarded as a matrix game with, say, the hider as row player and the searcher as column player. For node search the rows i are indexed by $1, \ldots, |V|$, for edge search by $1, \ldots, |E|$. The columns j correspond to the possible search paths S_j (paths or connected subgraphs). The column vectors of the game matrix A (with entries a_{ij} zero or one) are the respective incidence vectors of these search paths. Each vector has at most k entries equal to one, the rest is zero. For search on connected subgraphs, every column has exactly k ones since then the searcher can always visit the maximal number of locations, assuming $k \leq |V|$ ($k \leq |E|$).

Let the game matrix A be of dimension $r \times s$ where r is the number of locations ($r = |V|$ for node search, $r = |E|$ for edge search), and s the number of search paths. An optimal strategy of the searcher, the maximizing column player, is given by probabilities $p_1, \ldots, p_s$ so that

$$\sum_{j=1}^{s} a_{ij} p_j \geq d, \qquad i = 1, \ldots, r, \tag{2.1}$$

and so that d is maximal. The inequalities (2.1) say that the search paths are chosen according to a probability distribution so that each location has at least detection probability d. This detection probability is *optimal* if and only if there are probabilities $q_1, \ldots, q_r$ for the hider with the analogous property: The hider hides in location i with probability q_i so that for each search path S_j the detection probability is at most d. In other words, we have $d = d'$ in the constraints

$$\sum_{i=1}^{r} q_i a_{ij} \leq d', \qquad j = 1, \ldots, s. \tag{2.2}$$

Such optimal strategies and the optimal detection probability d define the *solution of the game.*

As an example, node search with $k = 2$ for the graph in Figure 1 has the optimal detection probability $1/3$. The hider hides with probability $1/3$ in one of the nodes 2, 3, 4. The searcher selects, for example, one of the search paths $\{1,2\}$, $\{1,3\}$, $\{4,5\}$ with equal probability to have each node inspected with probability at least $1/3$.

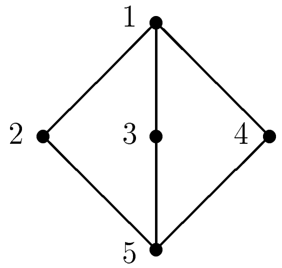

Figure 1

The constraints (2.1), $\sum p_j = 1$, and $p_j \geq 0$ for $j = 1, \ldots, s$ define a linear program (LP) with variables p_j and d, where d is to be maximized. The corresponding dual LP has the variables q_i and d', where d' is to be minimized, subject to the constraints (2.2), $\sum q_i = 1$, and $q_i \geq 0$ for $i = 1, \ldots, r$. The identical optimal value d or d' of either LP defines the optimal detection probability of the game, which is obviously positive.

We simplify these LPs since the variables d and d' can be omitted: Instead of the primal variables p_j, consider instead the variables x_j defined by $x_j = p_j/d$. Then $\sum_{j=1}^{s} x_j = \sum_{j=1}^{s} p_j/d = 1/d$, so maximizing d is equivalent to minimizing $\sum_{j=1}^{s} x_j$. This leads to the LP

$$\text{minimize} \sum_{j=1}^{s} x_j \tag{2.3}$$

$$\begin{aligned} \text{subject to} \qquad \sum_{j=1}^{s} a_{ij} x_j &\geq 1, \qquad i = 1, \ldots, r, \\ x_j &\geq 0, \qquad j = 1, \ldots, s. \end{aligned} \tag{2.4}$$

The dual of this LP is

$$\text{maximize} \sum_{i=1}^{r} y_i \tag{2.5}$$

$$\begin{aligned} \text{subject to} \qquad \sum_{i=1}^{r} y_i a_{ij} &\leq 1, \qquad j = 1, \ldots, s, \\ y_i &\geq 0, \qquad i = 1, \ldots, r. \end{aligned} \tag{2.6}$$

The optimal value w of both LPs is the inverse of the optimal detection probability d of the game. The variables x_j and y_i at optimum define the optimal strategies of the game: normalized, by multiplication with $1/w$, they represent the probabilities p_j and q_i, respectively. The linear programs (2.3), (2.4) and (2.5), (2.6) for a general 0-1-matrix A are known as *fractional covering* and *fractional packing* problems [21, p. 562].

If the search length k is constant, then the game can be solved in polynomial time by enumeration, since the LP is of polynomial size. Node search for $k = 2$ has a direct solution based on bipartite matching (described in detail in [7]): If G is represented as a bipartite graph (with edges in $V \times V$), then a minimum edge cover yields an optimal solution to (2.3), (2.4); see [20] and [17, pp. 213–216].

If the search length k is allowed to grow sufficiently fast with the size of the graph, then solving the game is NP-hard, as we will show in the next section. In general, it is even difficult to *verify* the optimality of a pair of strategies $(p_1, \ldots, p_s)$ and $(q_1, \ldots, q_r)$ for searcher and hider. Although the number s of search paths is very large, only r of the probabilities p_j have to be positive, taking a basic solution of the LP (2.1). Thus, an optimal search strategy can by specified in space polynomial in the size of G. With these probabilities p_j, the smallest of the detection probabilities for the nodes (edges) of G gives a lower bound for the optimal detection probability. However, verifying directly a claimed maximal detection probability d' in (2.2) for all s search paths is not possible in polynomial time since s is too large. There is one exception: If the hider hides with equal probability in each location, that is, $q_i = 1/r$, then the detection probability for each search path is obviously at most k/r. If this upper bound for the optimal detection probability can be achieved by a suitable randomized search strategy, then the game is solved. In this best possible case for the searcher, which can be verified in polynomial time, we say that the graph G is *uniformly searchable*.

A graph G is uniformly searchable for node search if it has a Hamiltonian cycle: In that case, the searcher can select any node with equal probability and search k nodes in one direction of the cycle. This is only a sufficient condition (for generalizations see [7]): The graph G in Figure 1 has no Hamiltonian cycle but is uniformly node-searchable for $k = 3$, by selecting one of the search paths $\{1, 2, 3\}$, $\{1, 2, 4\}$, $\{1, 3, 5\}$, $\{2, 4, 5\}$, or $\{3, 4, 5\}$ with equal probability. (We have seen above that G is not uniformly node-searchable for $k = 2$.) Similarly, a sufficient condition that G is uniformly edge-searchable for any k is that G has an Euler cycle (visiting all edges).

In practice, the LP (2.3), (2.4) is very suitable for the *revised simplex* algorithm with implicit column generation, which can be used to compute a search strategy until the detection probability is sufficiently high (compared to its upper bound k/r, for example). The revised simplex algorithm stores only the LP columns A_j of the current basic solution. For the pivoting step, a *profitable column* has to be found, which is generated directly from the graph, using suitable heuristics if necessary [6].

The *separation problem* for the dual LP is equivalent to this problem of finding a profitable column [23, p. 148]. It says: Given a vector $y = (y_1, \ldots, y_r)$, decide if all inequalities (2.6) are valid, and if not, produce an inquality that is violated. If some y_i is negative, then a violated inequality is given directly. If $y \geq 0$, then the separation problem amounts to finding a search path S_j with maximum weight yA_j, interpreting the nonnegative numbers $y_1, \ldots, y_r$ as weights on the nodes (edges) of G. The separation problem is theoretically important since with the ellipsoid algorithm for linear programming one can solve the entire LP using a polynomial number of calls to a 'subroutine' that solves the separation problem [13]:

Theorem 2.1. *With a polynomial-time algorithm for the separation problem, the LP can be solved in polynomial time.*

For node search on connected subgraphs, the separation problem is that of finding a maximum weight k-cardinality subtree of G. If G is a tree, this problem can be solved in polynomial time (see [6, 9] and references), so then the game can be solved in polynomial time. If G is a general graph, then this separation problem is NP-hard [9]. This would entail the NP-hardness of solving the LP if the objective function could be chosen arbitrarily, because in that case the converse of Theorem 2.1 holds as well [13]. However, the objective function is special. In the next section, we will prove directly that solving the search game on connected subgraphs is NP-hard. In Section 4, we will use Theorem 2.1 to show that the search game can be solved in polynomial time for search paths of logarithmic length.

3. NP-hardness for general search length

We have seen that solving the search game for a graph G and search length k is equivalent to solving the primal LP (2.3), (2.4) and its dual LP (2.5), (2.6). In this section, we will prove that solving the primal LP is NP-hard (so that solving the game is also NP-hard). We will do this by showing that it is NP-complete to decide if there is a solution $x_1, \ldots, x_s$ to the constraints (2.4) so that $\sum x_j \leq r/k$. If the objective function in (2.3) can reach this bound, we have also reached optimum, since there is a dual feasible solution, namely $y_i = 1/k$ in (2.6), with dual objective function value r/k. This describes the case where the hider hides in each location with equal probability, and the searcher can achieve the maximum detection probability k/r for each location. In other words, we will show that it is NP-complete to decide if G is uniformly searchable.

Proposition 3.1. *For node search on paths and search length $k = |V|$, a graph G is uniformly searchable if and only if G has a Hamiltonian path.*

Proof. If G has a Hamiltonian path, then this search path S_j visits all nodes, that is, $x_j = 1$ (and all other variables zero) is an optimal solution to the LP (2.3), (2.4) with value 1. If G has no Hamiltonian path, then any search path can visit at most $k - 1$ nodes. Thus, $y_i = 1/(k-1)$ is a feasible solution to the dual LP (2.6) with objective function value $k/(k-1) > 1$, so the primal LP does not have value 1. □

This result has been shown earlier, also for more general detection probabilities [24]. It shows that solving the node search game on paths with general search length k is NP-complete. Even if k is bounded by $|V|^\varepsilon$ for some positive constant ε, then one can also show easily that the Hamiltonian path problem for a graph $G' = (V', E')$ can be polynomially reduced to the question if a graph G is uniformly node-searchable with paths of length k. In that construction, $k = |V'|$ and G consists of about $|V'|^{1/\varepsilon - 1}$ suitably connected copies of G'; for details see [7].

Next, we consider edge search on paths. If the search length is $k = |E|$, then the graph G is uniformly searchable if and only if it has an Euler path (proved analogously to Prop. 3.1). However, that question is easy to decide. We therefore need a different argument. For a given graph G', we will reduce, in polynomial time, the question if G' has a Hamiltonian path to the question if a graph G is uniformly edge-searchable. In this construction of G from G', we will append certain paths to nodes u of G'. Such a path P consists of l new nodes and $l-1$ new edges and an additional edge joining an endpoint of P to u. We will call P a *tail* of the new graph G. Its purpose is that in a uniform search, it is either searched entirely or not at all, as stated in the following lemma.

Lemma 3.2. *Suppose that the graph G has a tail P of length l and that G is uniformly searchable for node or edge search on paths or connected subgraphs, for search length $k \geq l$. Then any search path used with positive probability either contains all nodes and edges of P or none.*

Proof. If G is uniformly searchable, then the corresponding solution to the LP (2.3), (2.4) defines an exact fractional cover of all nodes (edges) of G by certain search paths S_j (those with $x_j > 0$, that is, those chosen with positive probability). All these search paths have k nodes (edges). Since $k \geq l$, any such search path S_j starting at a node v of P contains the part of P connecting to the rest of G. Since all nodes (edges) of P are covered equally, v must be the endpoint of P. □

Proposition 3.3. *For edge search on paths and search length $k = |E|/2$, it is NP-complete to decide if a graph $G = (V, E)$ is uniformly searchable.*

Proof. Given a connected graph G' and two nodes u and u' of G', we construct a graph G of polynomial size such that G' has a Hamiltonian path from u to u' if and only if G is uniformly searchable for edge search on paths. This will prove the claim.

Let n and m denote the number of nodes and edges of G'. As indicated in Figure 2, G is obtained from G' as follows: Duplicate every edge of G'. Put a new node on each edge (this makes the graph simple, in case we want to consider the search game only on simple graphs). Let G'' be the resulting graph, which has $4m$ edges, and where every node has even degree. For every original node v of G' except u and u', take a new path with $4m$ nodes and connect *both* endpoints to v. This creates a circle C_v of length $4m+1$ connected only by v to the rest of the graph. Append paths P and P' of length $4m+2$ to u and u', respectively. Let $k = n(4m+3)$. Finally, append a path P'' of length $k-(4m-2(n-1))$ to u. The number $|E|$ of edges of the resulting graph G is therefore $4m+(n-2)(4m+1)+2(4m+2)+k-(4m-2(n-1)) = 2k$.

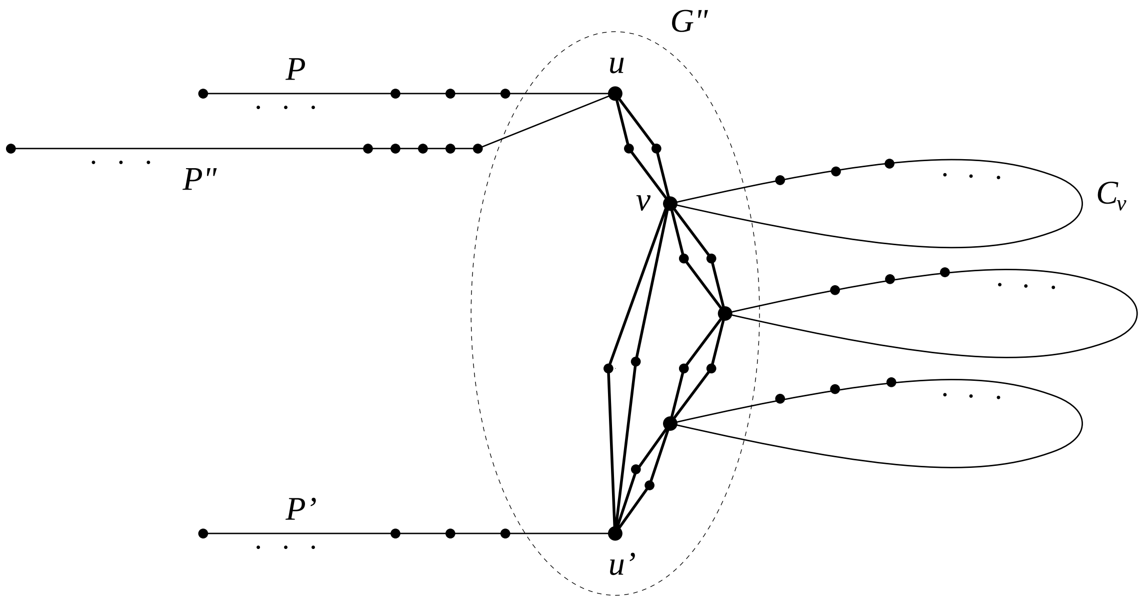

Figure 2

Assume that G' has a Hamiltonian path from u to u', which defines a path in G''. In G, we take that path, which visits all nodes v of G' and has $2(n-1)$ edges, extend it by the two tails P and P', and insert at every node v the path around the circle C_v. This is our first search path, which has k edges. After we remove these edges from G, the remaining graph has k edges, is connected (since G' is connected and we have duplicated the edges of G'), and all nodes have even degree except u and the endpoint of the tail P''. Thus, this graph has an Euler path, which is our second search path. The edges of G are covered exactly once by these two paths, that is, G is uniformly searchable.

Conversely, let G be uniformly searchable and consider an exact fractional cover of the edges of G by paths of length k. One of these paths has edges and nodes in the tail P. By Lemma 3.2, that path starts at the endpoint of P. The path cannot visit P'' since it is too short to reach the endpoint of P''. The path must fit into G without revisiting edges (otherwise, edges would be waisted and we had no exact cover). Therefore, it must visit all circles C_v and thus all nodes v of the original graph G', and end in the tail P'. However, in order to reach the endpoint of P', the path can use only $2(n-1)$ edges of G''. Thus, it defines a Hamiltonian path of G'.

□

At first glance, search on connected subgraphs looks like an easier problem since it is not related to Hamiltonian paths. For node search and $k = |V|$, a connected graph G is always uniformly searchable using any spanning tree as search path. The argument of Prop. 3.1 therefore no longer applies. However, the problem is still NP-hard. We will reduce the NP-complete problem of finding an exact three-cover to the question if a graph G is uniformly searchable. The construction of G is similar to the reduction of the exact three-cover problem to the Steiner tree problem [14], with additional tails appended to the graph so that we can apply Lemma 3.2. The same proof works for both node and edge search.

Proposition 3.4. *For node (edge) search on connected subgraphs and search length $k = |V|/2$ ($k = |E|/2$), it is NP-complete to decide if a graph $G = (V, E)$ is uniformly searchable.*

Proof. We show a reduction from the exact three-cover problem. An instance of this problem is given by a set $U = \{1, \ldots, n\}$, where n is a multiple of three, and a collection $C = \{c_1, \ldots, c_m\}$ of three-element subsets c_i of U. The problem is to decide whether there are $n/3$ of these sets that cover U. Given U and C, we construct the graph G as follows (see Figure 3). G consists of a graph G' with node set $\{u, u', v_1, \ldots, v_m, w_1, \ldots, w_n\}$, and certain paths appended to some nodes of G'. The edges in G' join v_i to the nodes u, u', and w_j for the three elements j of c_i, for $i = 1, \ldots, m$. Thus G' has $5m$ edges. To obtain G from G', we append new paths $P, P', P_1, \ldots, P_n$ with one of their endpoints to the nodes $u, u', w_1, \ldots, w_n$, respectively, all of which have length $6m$ except P'; the length of the tail P' depends on whether we consider node or edge search. Let us consider node search first, where $k = (n + 1)\,6m + {}^4\!/_3 n + 1$. In that case, P' has length $k - m + n/3 - 1$, and G has $2k$ nodes.

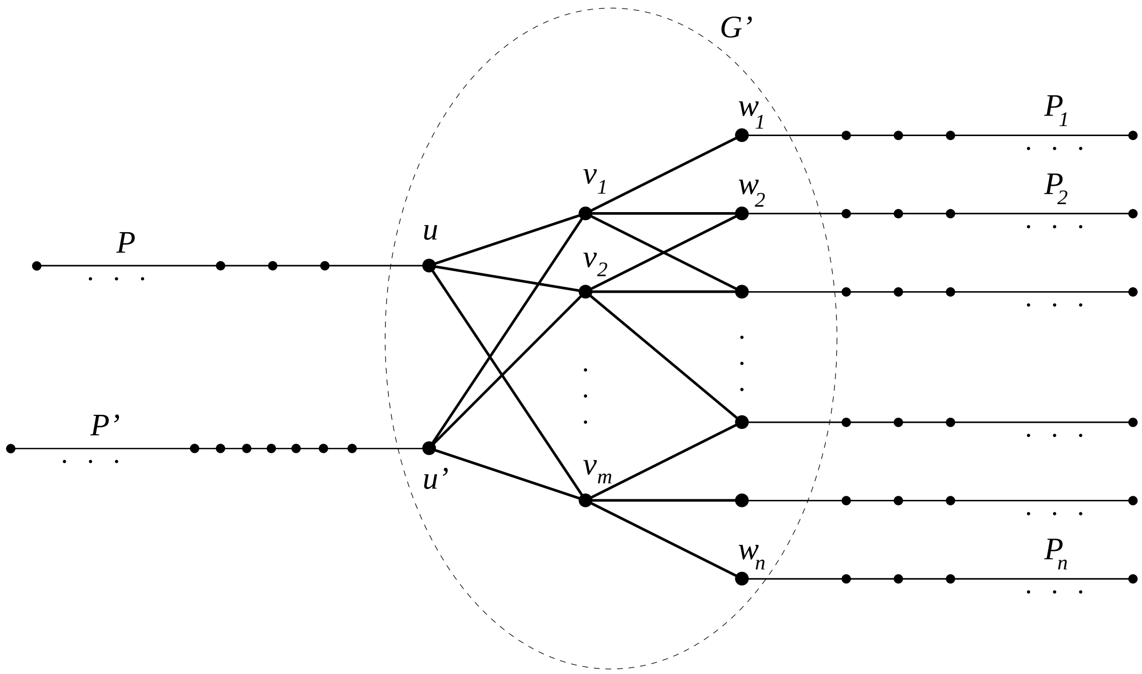

Figure 3

Suppose D is an exact cover of U, that is, $D \subseteq C$, $|D| = n/3$, and $\bigcup D = U$. Then the nodes of G can be partitioned into two trees T and T': The tree T is the unique spanning tree on the nodes $u, w_1, \ldots, w_n$, the nodes v_i with $c_i \in D$, and the paths $P, P_1, \ldots, P_n$. The tree T' contains u', the nodes v_i with $c_i \notin C$, and the tail P'. Both T and T' consist of k edges.

Conversely, assume that G is uniformly searchable, that is, there is an exact fractional cover of the nodes of G with connected subgraphs of size k. Let T be one of these subgraphs (with positive weight) that includes a node of P. By Lemma 3.2, T includes the endpoint of P. Since T is too small to reach the endpoint of P', it is disjoint from P'. Since T has more than $(n+1)\,6m$ nodes it contains all of $P_1, \ldots, P_n$, and thus the nodes u and $w_1, \ldots, w_n$. So T contains at most $n/3$ of the nodes v_i. The corresponding sets c_i define a cover of U since every node w_j for $j \in U$ belongs to T.

In the case of edge search, let $k = (n+1)\,6m + {}^4\!/\!_3 n$, and let P' have length $k - 5m + {}^4\!/\!_3 n$. Then G has $2k$ edges. In the same way as before, an exact cover D defines a tree T in G, where T and its complement T' in G partition the edges of G into two connected subgraphs of size k. Conversely, if G is uniformly searchable, then an exact fractional cover of the edges of G has one connected subgraph T that includes all edges of $P, P_1, \ldots, P_n$. Thus, T contains ${}^4\!/\!_3 n$ edges of G'. Since T is connected, it contains at most $1 + {}^4\!/\!_3 n$ nodes of G', including all nodes $u, w_1, \ldots, w_n$. It thus contains at most $n/3$ of the nodes v_i. As before, the corresponding sets c_i cover U. This completes the proof that the reduction is correct. $\square$

If the game is played on the nodes of the graph, then connected subgraphs with k nodes are equivalent to trees with k nodes. One could also consider edge search on trees (instead of connected subgraphs), although this is not very natural. The preceding proof shows that solving this game is also NP-hard.

4. Polynomial-time algorithms for logarithmic search length

Solving the search game is NP-hard if the search length k is arbitrary. For constant k, the problem can be solved in polynomial time by enumeration. In this section, we show that if k is proportional to the logarithm of the size of the graph G, then the search game can still be solved in polynomial time. By Theorem 2.1, it suffices to show that the separation problem can be solved in polynomial time, which says: Given nonnegative weights on the nodes (edges) of G, find a search path of maximum weight. Note that the weight of a location that is revisited by the searcher is counted only once.

We solve this separation problem by modifying an algorithm for *finding* a simple path of length k in G. Alon, Yuster, and Zwick [2] recently presented a technique called *color-coding* that solves this problem in polynomial time if $k = O(\log |V|)$, answering a long open question. The standard dynamic programming approach requires to store intermediate results for all paths with up to k nodes. If $k = |V|$, then there are in total about $2^{|V|}$ such paths, so that the total running time is

$O(|V|^2 2^{|V|})$ [15, p. 40]. If k is much smaller than $|V|$, however, then there are $O(|V|^k)$ such paths. With color-coding, in contrast, each node is given a random color and only sets of colors need to be stored intermediately, in total about 2^k if k colors are used. One has to try different colorings, but if these are taken from a suitable family of hash functions, not too many are needed to succeed. We adapt the color-coding algorithm to finding a (not necessarily simple) path of maximum weight. Edge search and search on connected subgraphs can be solved similarly.

Proposition 4.1. *Consider node (edge) search on paths or connected subgraphs for graphs $G = (V, E)$, search length k, and nonnegative weights on the nodes (edges) of G. Then a search path of maximum weight can be found in time $2^{O(k)}|E| \log |V|$. For $k = O(\log |V|)$, this is polynomial in the size of G.*

Proof. We show first the algorithm for node search on paths. We proceed in rounds, where in each round, every node v is assigned a *color* $c(v)$ in $\{1, \ldots, k\}$. We will make sure that every set of k nodes is *colorful* in at least one round, that is, all its nodes have different colors. Within each round, we are looking for a colorful path of maximum weight. This is done by dynamic programming, as follows: For $i = 0, \ldots, k-1$ and all nodes v, we consider paths that can be walked in i steps and end in v (for $i = 0$, such a path consists of the single node v). For each set C of colors, $C \subseteq \{1, \ldots, k\}$, let $M(v, C, i)$ denote the maximum weight of these paths that have C as the set of colors of their nodes, where for each color, only the weight of the first node with that color is counted. (Let $M(v, C, i) = -\infty$ if there is no such path, for example if $c(v) \notin C$ or if $|C| > i + 1$.) The initialization $M(v, C, 0)$ is trivial. If $y(v)$ is the weight of v, then $M(v, C, i + 1)$ (for $c(v) \in C$) is computed as the maximum of the numbers $y(v) + M(u, C - \{c(v)\}, i)$ and $M(u, C, i)$ for all edges (v, u) of G. In a round, computing the numbers $M(v, C, i)$ takes a constant amount of work for every edge, every C, and every i. So the running time for each round is $O(|E| \cdot 2^k \cdot k)$.

The maximum weight of a search path is the maximal $M(v, C, i)$ ever computed in all rounds. The search path itself is easily found, for example by keeping track of the computation. For the correctness of the algorithm, we have to make sure that every set of k nodes is colored with k different colors in at least one round. There is such a *k-perfect* family of colorings consisting of only $2^{O(k)} \log |V|$ colorings (see [2]; a k-perfect family of $2^{O(k)} \log^2 |V|$ colorings is somewhat simpler to construct). Each of those colorings can be generated in $O(|V|)$ time. Thus the total running time for finding a path of maximum weight is $2^{O(k)}|E| \log |V|$.

For edge search on paths, the edges (u, v) of G have weights $y(u, v)$. We use the same approach, coloring each edge with one of k possible colors $c(u, v)$. The number $M(v, C, i)$ represents the maximum weight on a path that ends in the node v, has the set C of colors of its edges, and is walked in i steps; again, we count only the weight of the first edge of a color. Inductively for $i = 1, \ldots, k - 1$, we compute $M(v, C, i + 1)$ as the maximum of the numbers $y(v, u) + M(u, C - \{c(v, u)\}, i)$ and

$M(u, C, i)$ for all edges (v, u) with $c(v, u) \in C$. Each round has again running time $O(|E| \cdot 2^k \cdot k)$. Performing these rounds for a k-perfect family of colorings of edges, the overall running time is again $2^{O(k)}|E| \log |V|$ since $\log |E|$ is proportional to $\log |V|$.

For search on connected subgraphs, a search path of maximum weight contains always k locations. Thus, in each round with given coloring, we can omit the parameter i above since $i = |C|$. We define $M(v, C)$ to be the maximum weight of a subgraph that contains the node v and has exactly one node (edge) of each color in C. For node search, $M(v, C)$ is computed as the maximum of $M(v, D)+M(u, C-D)$ for all edges (v, u) and proper subsets D of C. For edge search, that maximum is taken over the numbers $M(v, D)+y(v, u)+M(u, C-D-\{c(v, u)\})$ where $c(v, u) \in C-D$. Since we consider all subsets C of $\{1, \ldots, k\}$ and their subsets D, these numbers $M(v, C)$ are computed with $O(3^k) = 2^{O(k)}$ steps per edge. Thus, the running time for one round is $2^{O(k)} \cdot |E|$.

For all four versions of the search game, the total running time of the algorithm is thus $2^{O(k)}|E| \log |V|$. This is polynomial in the size of G if $k = O(\log |V|)$. □

It is natural to ask if the game can be solved in polynomial time for graphs $G = (V, E)$ and a search length k that is asymptotically larger than $\log |V|$. Suppose this holds for node search on paths and, say, for all $k = O(\log^\alpha |V|)$ for some $\alpha > 1$. We claim that this would give us an algorithm to solve an instance of 3SAT with n clauses in time $2^{O(n^{1/\alpha})}$, which would be a rather unexpected result. Namely, the satisfiability of such an instance of 3SAT can be reduced to the question if a graph G' with $k = O(n)$ nodes has a Hamiltonian path [22, p. 193]. We append a very long path to the starting node of the Hamiltonian path whose length is a multiple of k to turn G' into a graph $G = (V, E)$ so that $k = \log^\alpha |V|$ and thus $|V| = 2^{k^{1/\alpha}}$. Then Lemma 3.2 implies that G' has a Hamiltonian path if and only if G is uniformly searchable. Since we can decide this in running time that is polynomial in $|V|$, this would give us the mentioned algorithm for solving the 3SAT instance.

In practice, an algorithm for the separation problem will not be used with the ellipsoid method but rather for finding a profitable column when using the revised simplex algorithm. It is typical to generate such a column by dynamic programming. In the case of node search on paths, for example, a path of maximum weight is computed for successively longer paths. In order to decide if the weight of a new node v can be added to the current path of maximum weight, it is necessary to know if v belongs to that path or not. In the proof of Prop. 4.1, this information is represented by sets of colors, instead of storing directly the possible sets of nodes on the current paths. However, this requires storing $O(|V|^k)$ such sets instead of only $O(2^k)$ color sets. Prior to the color-coding technique of [2], Monien [19] proposed an algorithm for finding a simple path of length k in time $O(k! \, |E|)$. He showed that for the dynamic programming step, much fewer sets of nodes have to be stored than with the naive approach. This algorithm can also be modified to solve the separation problem for node and edge search on paths. Even though its running time is asymptotically worse than when using color-coding, it may be superior for small values of k, and simpler to implement. We conclude our study with an outline of this method.

Consider node search on paths, search length k, and nonnegative weights on the nodes of G. For each node v and $i = 1, \ldots, k$, let $F(v,i)$ denote the family of node sets S so that there is a path ending in v that is walked in $i-1$ steps and visits exactly the nodes in S (so $|S| \leq i$). The families $F(v,i)$ can be constructed inductively for successively larger values of i, starting with $i = 1$. Analogously to [19], we only store a certain *representative* subfamily $F'(v,i)$ of $F(v,i)$ that has the following property: For any set A of $k-i$ or fewer nodes and a set S in $F(v,i)$ *disjoint* to A of maximum weight, there exists some S' in $F'(v,i)$ that is also disjoint to A and has the same maximum weight. The reason for this condition is that some S in $F(v,i)$ is eventually extended by a set A of nodes, where A is disjoint to S and $|A| \leq k-i$, to obtain a search path of maximum weight. For that purpose, S can be replaced by a set S' from $F'(v,i)$.

The sets of the representative family $F'(v,i)$ are organized in a rooted tree $T(v,i)$ with node and edge labels, defined as follows: Every node of $T(v,i)$ is either labeled with a set S from $F(v,i)$ or with a special symbol λ (it is also useful to store the weight of S). Every edge of $T(v,i)$ is labeled with a node of G. If A is the set of edge labels on the path from the root of $T(v,i)$ to some node t of $T(v,i)$, then t is labeled with a set from $F(v,i)$ disjoint to A of maximum weight. If there is no set in $F(v,i)$ disjoint to A (for example if $v \in A$), then t is labeled with λ. If a node t of $T(v,i)$ has depth less than $k-i$ (with the root at depth 0) and is labeled with a set $S \in F(v,i)$, then t has $|S|$ direct successors connected to t by edges labeled with the elements from S. All other nodes of $T(v,i)$ are terminal nodes.

Note that the tree $T(v,i)$ has depth at most $k-i$ and consists of $O((i-1)^{k-i})$ nodes. The labels of these nodes form a representative family $F'(v,i)$: If $A \subseteq V$ and $|A| \leq k-i$, then a set $S' \in F(v,i)$ disjoint to A of maximum weight (if such a set exists) is found as follows: Start with the root of $T(v,i)$ as the current node t in $T(v,i)$. If the label of t is λ, then there is no set $S' \in F(v,i)$ disjoint to A. If the label S of t is disjoint to A, the result is $S' = S$. Otherwise, descend in $T(v,i)$ along an edge labeled with some element of $S \cap A$ and repeat the step with the respective successor of t as the new current node. This procedure terminates with a node t of $T(v,i)$ where the set of edge labels on the path from the root of $T(v,i)$ to t is a subset A' of A. The label S' of t has maximum weight among the sets in $F(v,i)$ disjoint to A', so it certainly has maximum weight among the sets disjoint to A.

The trees $T(v,i)$ can be computed successively for $i = 1, \ldots, k$. Let $y(v)$ denote the weight of v and let $y(S)$ denote the weight of a set $S \subseteq V$. For $i < k$ and $v \in V$, the tree $T(v, i+1)$ is constructed from the trees $T(u,i)$ for the neighbors u of v in G. Let S_u be the label of the root of $T(u,i)$, and, if $v \in S_u$, let S'_u be the label of its successor along the edge labeled with v. To determine the label of the root of $T(v,i+1)$, one has to compute the maximum of $y(v) + y(S_u)$ if $v \notin S_u$ and of $y(v) + y(S'_u)$ and $y(S_u)$ if $v \in S_u$, for all neighbors u of v. The construction of

$T(v, i+1)$ continues in that manner, descending along the same edge labels in the trees $T(u, i)$ as in $T(v, i+1)$. Finally, the search path of maximum weight is one of the labels of the one-node trees $T(v, k)$. The total running time of this algorithm is $O(k! \cdot |E|)$ which follows from the bound $\sum_{i=1}^{k}(i-1)^{k-i} = O((k-1)!)$ and the time $O(k)$ required for testing disjointness.

A straightforward modification of this algorithm solves the separation problem for edge search on paths in the same running time. It is an open question if the approach of [19] can be applied to the separation problem for node or edge search on connected subgraphs.

Acknowledgments

The node search game was posed as a problem jointly with Buyang Cao [7]. We thank Jeff Edmonds, Ulrich Faigle, András Frank, Fred Glover, and the referees for helpful comments. This research was done at the International Computer Science Institute in Berkeley.

References

[1] R. Ahlswede and I. Wegener, *Search problems* (Wiley, New York, 1987).

[2] N. Alon, R. Yuster, and U. Zwick, Color-coding: a new method for finding simple paths, cycles and other small subgraphs within large graphs, *Proc. 26th ACM Symp. Theory of Computing* (1994) 326–335.

[3] R. Avenhaus, B. von Stengel, and S. Zamir, Inspection games, in: R. J. Aumann and S. Hart, eds., *Handbook of Game Theory, Vol. 3* (North-Holland, Amsterdam, 1996, to appear).

[4] S. Benkoski, M. G. Monticino and J. R. Weisinger, A survey of the search theory literature, *Naval Research Logistics* 38 (1991) 469–494.

[5] D. Bienstock and P. Seymour, Monotonicity in graph searching, *Journal of Algorithms* 12 (1991) 230–245.

[6] B. Cao, Search-hide games on trees, *European J. Operational Research* 80 (1993) 175–183.

[7] B. Cao and B. von Stengel, *Search-hide games on graphs*, Technical Report S-9303, University of the Federal Armed Forces Munich, Germany (1993).

[8] F. R. K. Chung, J. E. Cohen and R. L. Graham, Pursuit-evasion games on graphs, *J. Graph Theory* 12 (1988) 159–167.

[9] M. Fischetti, H. W. Hamacher, K. Jørnsten, and F. Maffioli, Weighted k-cardinality trees: complexity and polyhedral structure, *Networks* 24 (1994) 11–21.

[10] M. Franklin, Z. Galil, and M. Yung, Eavesdropping games: A graph-theoretic approach to privacy in distributed systems, *Proc. 34th IEEE Symp. Foundations of Computer Science* (1993) 670–679.

[11] S. Gal, *Search Games* (Academic Press, New York, 1980).

[12] S. Gal, Continuous search games. In: D. V. Chudnovsky and G. V. Chudnovsky, eds., *Search Theory – Some Recent Developments* (Dekker, New York, 1989) 33–53.

[13] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization* (Springer, Berlin, 1988).

[14] R. M. Karp, Reducibility among combinatorial problems, in: R. E. Miller and J. W. Thatcher, eds., *Complexity of Computer Computations* (Plenum, New York 1972) 85–103.

[15] E. L. Lawler et al., eds., *The Traveling Salesman Problem* (Wiley, Chichester, 1985).

[16] U. Lössner and I. Wegener, Discrete sequential search with positive switch cost, *Mathematics of Operations Research* 7 (1982) 426–440.

[17] L. Lovász and M. D. Plummer, *Matching Theory*, Annals of Discrete Mathematics, Vol. 29 (North-Holland, Amsterdam, 1986).

[18] N. Megiddo et al., The complexity of searching a graph. *J. ACM* 35 (1988) 18–44.

[19] B. Monien, How to find longs paths efficiently, in: G. Ausiello and M. Lucertini, eds., *Analysis and Design of Algorithms for Combinatorial Problems*, Annals of Discrete Mathematics, Vol. 25 (North-Holland, Amsterdam, 1985) 239–254.

[20] G. L. Nemhauser and L. R. Trotter, Jr., Properties of vertex packing and independence system polyhedra, *Mathematical Programming* 6 (1974) 48–61.

[21] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization* (Wiley, New York 1988).

[22] C. H. Papadimitriou, *Computational Complexity* (Addison-Wesley, Reading, 1994).

[23] A. Schrijver, *Theory of Linear and Integer Programming* (Wiley, Chichester, 1986).

[24] K. E. Trummel and J. R. Weisinger, The complexity of the optimal searcher path problem, *Operations Research* 34 (1986) 324–327.

[25] I. Wegener, Optimal search with positive switch cost is NP-hard, *Information Processing Letters* 21 (1985) 49–52.