Section 9.5
Pursuit-Evasion Problems

Richard B. Borie, University of Alabama
Sven Koenig, University of Southern California
Craig A. Tovey, Georgia Institute of Technology

INTRODUCTION

In *pursuit-evasion* problems, a team of mobile pursuers (or searchers) attempts to capture one or more mobile evaders (fugitives, intruders) within a graph. For example, the pursuers may represent soldiers, policemen, or robots. The evaders might be terrorists, criminals, lost children, or even a poisonous gas. The graph may represent a road map, building floor plan, cave system, pipe network, etc. Many distinct variations of pursuit-evasion problems can be formulated by specifying the rules of movement for the pursuers and for the evaders, the knowledge each opponent has about the other, the rules of capture, the kind of graph, and the objective function. Typical objectives include minimizing the number of pursuers, the distance travelled by pursuers, or the elapsed time until capture. Because finiteness of the latter two objectives is equivalent to optimization of the first objective, the complexity of the latter two problems is bounded below by that of the first. However, optimization of the first objective is usually $\mathcal{NP}$-hard for general graphs, hence the bulk of the graph-theoretic pursuit-evasion literature focuses on that objective.

The following subsections discuss several of the most-studied pursuit-evasion variations. Other surveys on pursuit-evasion include [Bi91], [FoPe96], [Al04], [Ha07], [FoTh08], [ChHoIs11], and [BoYa11]. Now there is also an entire book on this topic [BoNo11]. Throughout this section, except where explicitly specified otherwise, G will denote a connected undirected graph or multigraph, possibly with loops.

9.5.1 Sweeping and Edge Search

FACT

F1: Parsons [Pa78] describes the original pursuit-evasion problem. The following definitions are adapted from [Pa78].

DEFINITIONS

D1: Consider an embedding of G in 3D space such that each vertex resides at a distinct location and no two edges intersect except at a common endpoint. (For every G such an embedding exists.) Let k denote the number of pursuers, and let $P = (P_1, \ldots, P_k)$ where each $P_j : [0, \infty) \to G$ is a continuous function. Then P is a ***sweep strategy*** for G if for every continuous function $E : [0, \infty) \to G$, there exists some pursuer $j \in \{1, \ldots, k\}$ and time t such that $P_j(t) = E(t)$. Here $P_j(t)$ denotes the location within graph G of pursuer j at time t, $E(t)$ denotes the location of an evader at time t, and capture occurs when $P_j(t) = E(t)$.

D2: The ***sweep number*** of G, denoted $\mathrm{sw}(G)$, is the smallest k such that a sweep strategy $P = (P_1, \ldots, P_k)$ exists. The ***sweep problem*** on G is to determine $\mathrm{sw}(G)$, and G is ***k-sweepable*** if $\mathrm{sw}(G) \leq k$.

FACTS

F2: Petrov [Pe82] independently develops another pursuit-evasion model, based on a system of differential equations. See [Pe82] for details.

F3: Golovach [Go89] describes yet another formulation for pursuit-evasion. The following definitions are adapted from [Go89].

DEFINITIONS

D3: An ***edge search operation*** is one of the following: $p(x)$ = place a pursuer at vertex x; $r(x)$ = remove a pursuer from vertex x; and $s(e, x, y)$ = slide a pursuer along edge e from endpoint x to other endpoint y. (We may write $s(e, x, y)$ as $s(x, y)$ if only one edge (x, y) exists, or as $s(e)$ if the sliding direction is forced or inconsequential.)

D4: Initially every edge of G is *contaminated* (might contain an evader). An edge $e = (x, y)$ becomes *clear* if a pursuer slides along e from x to y while either (i) another pursuer resides at x or (ii) every other edge incident to x is clear. If ever any unoccupied vertex x is incident to a contaminated edge, then any clear edges incident to x immediately become recontaminated. (So if a pursuer slides from x to y while neither (i) nor (ii) holds, then edge (x, y) does not become clear.) An ***edge search strategy*** for G is any sequence of edge search operations that ends with every edge of G being simultaneously clear.

D5: The ***edge search number*** of G, denoted $\mathrm{es}(G)$, is the smallest number of pursuers needed to implement any edge search strategy. The ***edge search problem*** on G is to determine $\mathrm{es}(G)$, and G is ***k-edge-searchable*** if $\mathrm{es}(G) \leq k$.

FACT

F4: Golovach [Go89] shows that the formulations of Parsons, Petrov, and Golovach are all equivalent problems. Therefore $\mathrm{sw}(G) = \mathrm{es}(G)$ for every G.

EXAMPLES

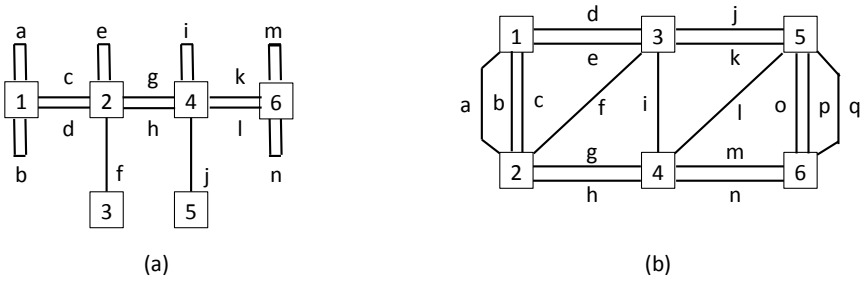

Figure 9.5.1: Typical graphs with edge search numbers 2 and 3.

E1: The graph in Figure 9.5.1(a) has edge search number 2. Here is a strategy that clears this graph using 2 pursuers: p(1), p(1), s(a), s(b), s(c), s(d,1,2), s(e), s(f), s(3,2), s(g), s(h,2,4), s(i), s(j), s(5,4), s(k), s(l,4,6), s(m), s(n).

E2: The graph in Figure 9.5.1(b) has edge search number 3. Here is a strategy that clears this graph using 3 pursuers: p(1), p(2), p(2), s(a,2,1), s(b,1,2), s(c,2,1), s(d), s(e,1,3), s(3,2), s(g), s(h,2,4), s(4,3), s(j), s(k,3,5), s(5,4), s(m), s(n,4,6), s(o,6,5), s(p,5,6), s(q).

FACTS

F5: The edge search number $\text{es}(G) = 1$ if and only if G is a simple path. So $\text{es}(G) = 1$ if and only if G contains neither a cycle nor a vertex of degree 3 or more. Equivalently, $\text{es}(G) = 1$ if and only if G contains neither of these two minimal forbidden minors: a loop with one vertex and one edge, or a star with three edges.

F6: Megiddo et al [MeHaGaJoPa88] shows that $\text{es}(G) \leq 2$ if and only if G contains none of the minimal forbidden minors illustrated in Figure 9.5.2. This paper also provides a structural characterization for the 2-edge-searchable graphs, similar to the graph in Figure 9.5.1(a).

F7: Megiddo et al [MeHaGaJoPa88] shows that if G is biconnected, then $\text{es}(G) \leq 3$ if and only if G contains none of the minimal forbidden minors illustrated in Figure 9.5.3. This paper also provides a structural characterization for the biconnected 3-edge-searchable graphs, similar to the graph in Figure 9.5.1(b), and also a more general characterization for the non-biconnected 3-edge-searchable graphs.

EXAMPLES

E3: The graph in Figure 9.5.2(a) has edge search number 3: p(1), s(1,4), p(3), s(3,4), r(4), s(4,5), p(2), s(2,6), p(7), s(7,6), s(6,5), s(5,8), s(5,8), s(8,9), s(8,10). However, if edge (8,10) is removed, the resulting graph has edge search number 2: p(1), s(1,4), p(3), s(3,4), s(4,5), s(4,5), s(5,8), s(8,9), r(9), s(5,6), p(2), s(2,6), s(6,7).

E4: The graph in Figure 9.5.2(b) has edge search number 3: p(1), s(1,2), p(2), s(2,3), s(2,4), p(5), s(5,3), s(3,4), s(4,6). However, if edge (4,6) is removed, the resulting graph has edge search number 2: p(1), s(1,2), p(2), s(2,3), s(2,4), s(4,3), s(3,5).

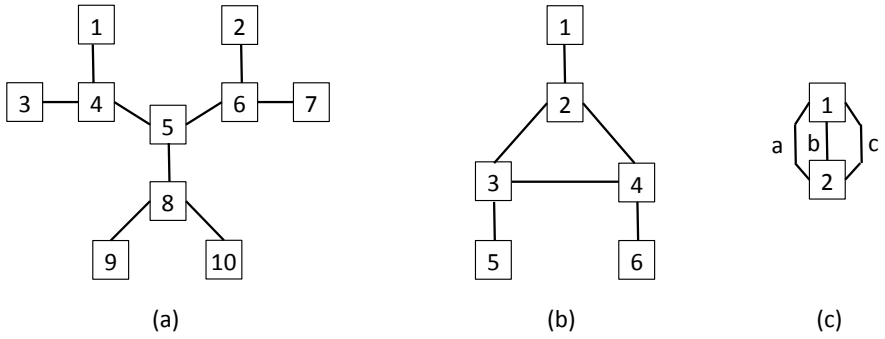

Figure 9.5.2: Forbidden minors for graphs with edge search number ≤ 2.

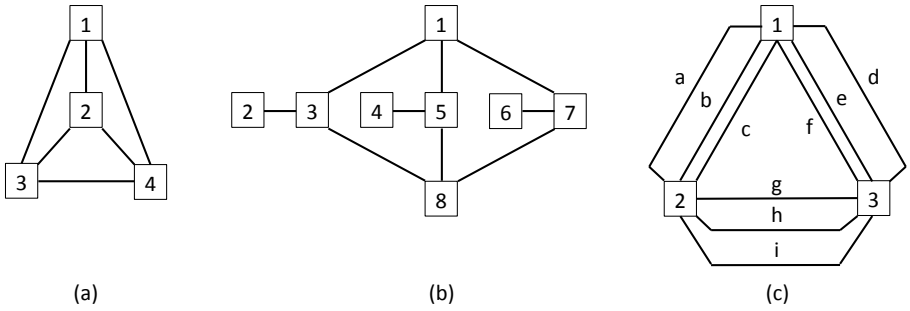

Figure 9.5.3: Forbidden minors for biconnected graphs with edge search number ≤ 3.

E5: The graph in Figure 9.5.2(c) has edge search number 3: p(1), p(1), p(1), s(a), s(b,1,2), s(c). However, if edge c is removed, the resulting graph has edge search number 2: p(1), p(1), s(a), s(b).

E6: The graph K_4 in Figure 9.5.3(a) has edge search number 4: p(1), p(1), p(1), s(1,2), s(1,3), s(1,4), p(3), s(3,2), s(2,4), s(4,3). However, if edge (3,4) is removed, the resulting graph has edge search number 3: p(1), p(1), p(1), s(1,2), s(1,3), s(1,4), s(3,2), s(2,4).

E7: The graph in Figure 9.5.3(b) has edge search number 4: p(1), p(1), p(1), s(1,3), s(1,5), s(1,7), p(2), s(2,3), r(3), s(3,8), p(4), s(4,5), s(5,8), s(8,7), s(7,6). However, if edge (6,7) is removed, the resulting graph has edge search number 3: p(2), s(2,3), p(3), s(3,1), s(3,8), p(1), s(1,7), s(7,8), s(1,5), s(8,5), s(5,4).

E8: The graph in Figure 9.5.3(c) has edge search number 4: p(1), p(1), p(1), p(1), s(a), s(b,1,2), s(c,1,2), s(g), s(h,2,3), s(i,2,3), s(d,3,1), s(e), s(f). However, if edge i is removed, the resulting graph has edge search number 3: p(1), p(1), s(a), p(2), s(b,2,1), s(c,1,2), s(g), s(h,2,3), s(d,3,1), s(e,1,3), s(f).

FACTS

F8: Megiddo et al [MeHaGaJoPa88] proves that the decision version of the edge search problem is $\mathcal{NP}$-complete for arbitrary graphs G.

F9: For every $n \geq 4$, $\text{es}(K_n) = n$, where K_n denotes a complete graph (or clique) with n vertices.

F10: Megiddo et al [MeHaGaJoPa88] presents a linear-time algorithm for computing $\text{es}(G)$ when G is a tree.

F11: There exist polynomial-time algorithms for computing $\text{es}(G)$ when G is a split graph, an interval graph, or a cograph.

REMARK

R1: It currently remains unresolved whether or not polynomial-time algorithms exist for computing $\text{es}(G)$ when G is a permutation graph, an outerplanar graph, a series-parallel graph, or a planar graph.

FACTS

F12: Parsons [Pa78] shows that if G is a tree and $k \geq 2$, then $\text{es}(G) \geq k$ if and only if G has a vertex v with degree $d \geq 3$ such that splitting v into d vertices each having degree 1 yields a forest in which at least three trees have edge search number at least $k - 1$.

F13: Let T_k denote a smallest tree such that $\text{es}(T_k) = k$. Then T_1 has a single edge, T_2 is a star with three edges, and T_3 is the tree shown in Figure 9.5.2(a). In general for $k \geq 2$, T_k may be formed from three copies of T_{k-1} by choosing one leaf from each copy of T_{k-1} and fusing together these three vertices.

F14: Let m_k denote the number of edges in T_k. Then $m_1 = 1$, $m_2 = 3$, and $m_3 = 9$. In general for $k \geq 2$, it follows that $m_k = 3m_{k-1}$, so $m_k = 3^{k-1}$. Hence if T is any tree with m edges, then $\text{es}(T) \leq 1 + \log_3 m$.

F15: LaPaugh [La93] shows that recontamination is not useful for edge search. That is, $\text{es}(G)$ pursuers can always clear G using an edge search strategy in which no clear edge ever becomes recontaminated.

REMARK

R2: LaPaugh's result that recontamination is not useful applies only to edge search but not to sweeping. This is because edge search permits arbitrary removal and placement of a pursuer, which essentially allows pursuers to jump between any vertices of the graph. However, sweeping requires each pursuer to move continuously through the graph, and therefore may require a pursuer to traverse (and unintentionally clear) a contaminated edge.

EXAMPLE

E9: The graph in Figure 9.5.4 illustrates that recontamination is sometimes useful for sweeping. First note that this graph has edge search number 3 as follows: p(1), p(1), p(1), s(a), s(b,1,2), s(c,1,2), r(2), r(2), p(3), p(3), s(f), s(g), s(3,4), s(4,2), s(j), s(e), s(2,5), s(h), s(5,7), r(7), p(6), s(i), r(6), r(6), p(7), p(7), s(l), s(m,7,8), s(n). However, this solution requires jumping to avoid recontamination. That is, rather than removing two pursuers from vertex 2 and placing them on vertex 3, instead let these two pursuers

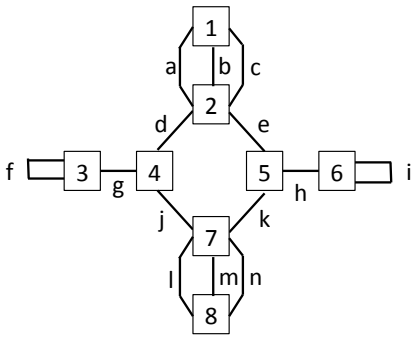

Figure 9.5.4: Recontamination can be useful for sweeping.

slide along edges d and g to reach vertex 3. Then edge d is temporarily cleared, but later it becomes recontaminated when both pursuers depart from vertex 4. Because sweeping only permits moving along edges (no jumping), the graph in Figure 9.5.4 has sweep number 3, but every sweep strategy with 3 pursuers requires recontamination to occur.

DEFINITIONS

D6: A search strategy is ***monotonic*** if recontamination does not occur.

D7: A search strategy is ***internal*** if no pursuer is ever removed from a vertex (so jumping does not occur).

D8: A search strategy is ***connected*** if the set of clear edges always induces a connected subgraph.

FACT

F16: Barriere et al [BaFrSaTh03] provides inequalities that show the relationships between the numbers of pursuers needed to clear a graph when one or more of these constraints (m = monotonic, i = internal, c = connected) are required during edge search. In particular, $\text{es}(G) = \text{m}(G) = \text{i}(G) \leq \text{mi}(G) \leq \text{c}(G) = \text{ic}(G) \leq \text{mc}(G) = \text{mic}(G)$.

9.5.2 Node Search and Mixed Search

FACT

F17: Kirousis et al [KiPa85], [KiPa86] introduce a variation of pursuit-evasion that lacks sliding and that has a novel rule for clearing edges (capturing evaders). The following definitions are adapted from [KiPa86].

DEFINITIONS

D9: A ***node search operation*** is one of the following: $p(x)$ = place a pursuer at vertex x, and $r(x)$ = remove a pursuer from vertex x.

D10: Initially every edge of G is *contaminated*. An edge $e = (x, y)$ becomes *clear* if pursuers simultaneously occupy both endpoint vertices x and y. As previously stated with edge search, if ever an unoccupied vertex x is incident to a contaminated edge, then all clear edges incident to x become recontaminated. A ***node search strategy*** for G is any sequence of node search operations that ends with every edge of G being simultaneously clear.

D11: The ***node search number*** of G, denoted $\text{ns}(G)$, is the smallest number of pursuers needed to implement any node search strategy. The ***node search problem*** on G is to determine $\text{ns}(G)$, and G is ***k-node-searchable*** if $\text{ns}(G) \leq k$.

FACTS

F18: Kirousis et al [KiPa86] shows that recontamination is not useful for node search. That is, $\text{ns}(G)$ pursuers can always clear G using a node search strategy in which no clear edge ever becomes recontaminated.

F19: Bienstock et al [BiSe91] unifies edge search and node search into a more general framework called mixed search. The following definitions are adapted from [BiSe91].

DEFINITIONS

D12: ***Mixed search operations*** are same as edge search operations: $p(x)$ = place a pursuer at vertex x; $r(x)$ = remove a pursuer from vertex x; and $s(e, x, y)$ = slide a pursuer along edge e from endpoint x to other endpoint y.

D13: Initially every edge of G is *contaminated*. As with edge search, edge $e = (x, y)$ becomes *clear* if a pursuer slides along e from x to y while either (i) another pursuer resides at x or (ii) every other edge incident to x is clear. Also, as with node search, edge $e = (x, y)$ becomes *clear* if pursuers simultaneously occupy both endpoint vertices x and y. Recontamination may occur same as with edge search and node search. A ***mixed search strategy*** for G is any sequence of mixed search operations that ends with every edge of G being simultaneously clear.

D14: The ***mixed search number*** of G, denoted $\text{ms}(G)$, is the smallest number of pursuers needed to implement any mixed search strategy. The ***mixed search problem*** on G is to determine $\text{ms}(G)$, and G is ***k-mixed-searchable*** if $\text{ms}(G) \leq k$.

FACTS

F20: Bienstock et al [BiSe91] shows that recontamination is not useful for mixed search. Thus $\text{ms}(G)$ pursuers can always clear G using a mixed search strategy in which no clear edge ever becomes recontaminated.

F21: For any G, construct G^e and G^n by replacing each edge of G with two edges in series or with two edges in parallel, respectively. Bienstock et al [BiSe91] show that $\text{es}(G) = \text{ms}(G^e)$ and $\text{ns}(G) = \text{ms}(G^n)$, so edge search and node search both reduce to mixed search. Therefore the recontamination result for mixed graphs in [BiSe91] implies the previous recontamination results for edge search in [La93] and for node search in [KiPa86].

F22: [KiPa86] and [BiSe91] provide inequalities that show the relationships between the edge search, node search, and mixed search numbers. Combining those inequalities yields that $\max\{es(G), ns(G)\} - 1 \leq ms(G) \leq \min\{es(G), ns(G)\}$, so these three parameter values are always within one of each other.

EXAMPLES

E10: If G is a path with at least one edge then $es(G)=1$, $ns(G)=2$, and $ms(G)=1$.

E11: If G is a loop with one vertex and one edge then $es(G)=2$, $ns(G)=1$, and $ms(G)=1$. Here is a (trivial) node search strategy that requires only 1 pursuer: p(1).

E12: If G is a cycle with two vertices and two edges then $es(G)=ns(G)=ms(G)=2$.

E13: If G is a cycle with at least three edges then $es(G)=2$, $ns(G)=3$, and $ms(G)=2$.

E14: If G is a star with at least three edges then $es(G)=ns(G)=ms(G)=2$.

E15: If G is the graph in Figure 9.5.2(c) then $es(G)=3$, $ns(G)=2$, and $ms(G)=2$. Here is a node search strategy that requires only 2 pursuers: p(1), p(2).

E16: If G is K_4 then $es(G)=ns(G)=ms(G)=4$.

E17: If G is the graph in Figure 9.5.5(a) then $es(G)=5$, $ns(G)=4$, and $ms(G)=4$. Here is a node search strategy that requires only 4 pursuers: p(1), p(3), p(5), p(2), r(2), p(4), r(4), p(6).

E18: If G is the graph in Figure 9.5.5(b) then $es(G)=2$, $ns(G)=3$, and $ms(G)=2$.

E19: If G is the graph in Figure 9.5.5(c) then $es(G)=3$, $ns(G)=3$, and $ms(G)=2$. Here is a mixed search strategy that requires only 2 pursuers: p(4), p(3), s(3,1), r(1), p(5), s(5,2), r(2), p(6), s(6,7).

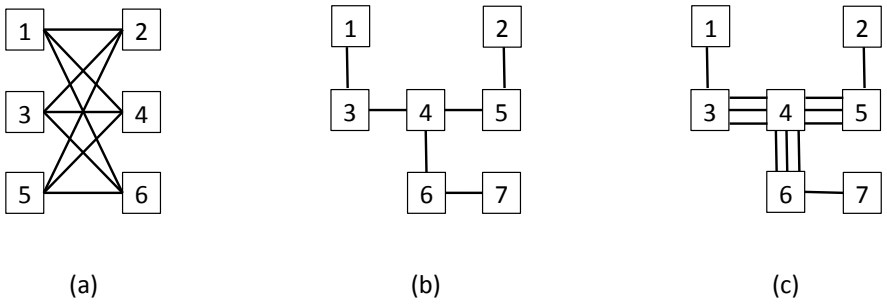

Figure 9.5.5: Examples for edge search, node search, and mixed search.

FACTS

F23: There exist polynomial-time algorithms for computing $ns(G)$ and $ms(G)$ when G is a split graph, an interval graph, a cograph, or a permutation graph.

F24: There exist polynomial-time algorithms for computing ns(G) when G is a tree, an outerplanar graph, or a series-parallel graph.

REMARKS

R3: It currently remains unresolved whether or not polynomial-time algorithms exist for computing ms(G) when G is a tree, an outerplanar graph, or a series-parallel graph.

R4: It currently remains unresolved whether or not polynomial-time algorithms exist for computing ns(G) and ms(G) when G is a planar graph.

FACTS

F25: Kirousis et al [KiPa86] proves that the decision version of the node search problem is $\mathcal{NP}$-complete for arbitrary graphs G.

F26: Bienstock et al [BiSe91] proves that the decision version of the mixed search problem is $\mathcal{NP}$-complete for arbitrary graphs G.

F27: Kirousis et al [KiPa86] shows that for every G, ns(G) is exactly one plus the *vertex separation* of G. Subsequently, Kinnersley [Ki92] shows that the vertex separation of G always equals the *pathwidth* of G, hence ns(G) is exactly one plus the pathwidth of G.

TERMINOLOGY NOTE: See Section 2.4 of this Handbook for a definition of pathwidth.

FACTS

F28: Suppose the evader is *visible*, that is, the evader's position is always known to the pursuers. In this situation Seymour et al [SeTh93] shows that the fewest pursuers needed to implement a node search strategy is exactly one plus the *treewidth* of G.

F29: Suppose instead that the (invisible) evader is *lazy*, that is, the evader can only move immediately before a pursuer is placed on the vertex where it resides (so that if the evader did not move it would be captured). In this situation Dendris et al [DeKiTh97] shows that the fewest pursuers needed to implement a node search strategy is again exactly one plus the *treewidth* of G.

TERMINOLOGY NOTE: See Section 2.4 of this Handbook for a definition of treewidth.

9.5.3 Cops-and-Robbers

The cops-and-robbers problem differs from the previously considered pursuit-evasion problems in several significant ways: both the cops (pursuers) and the robber (evader) must reside only at vertices, the cops and robber take alternating turns; and everybody's location is visible to everyone else.

FACT

F30: Nowakowski et al [NoWi83] and Quilliot [Qu83] each independently originates the cops-and-robbers problem. However, each only considers the special case when there is only one cop.

F31: Aigner et al [AiFr84] extends the cops-and-robbers problem to permit multiple cops. The following definitions are adapted from [AiFr84].

DEFINITIONS

D15: A ***cops-and-robbers game*** on G proceeds as follows. There are two players, C (a team of k cops) and R (a robber). C begins by placing each of the k cops at any vertex of G. (C is permitted to place more than one cop at the same location.) Next, R places the robber at any vertex. The players continue alternating turns. On C's turns, each cop either remains at its present location or moves to an adjacent vertex (so multiple cops may move simultaneously). Similarly, on R's turns, the robber either remains at its present location or moves to an adjacent vertex. Both C and R always know the locations of all participants. A cop *captures* the robber if the cop resides at the same vertex as the robber, and in this case player C wins the game. Player C has a *winning strategy* if no matter what choices R makes, player C can eventually win the game. Otherwise, if the robber can indefinitely avoid capture no matter what choices C makes, then player R wins the game.

D16: The ***cop number*** of G, denoted $c(G)$, is the smallest number of cops k needed for player C to win the cops-and-robbers game on G. The ***cops-and-robbers problem*** on G is to determine $c(G)$, and G is ***k-cop-winnable*** if $c(G) \leq k$.

EXAMPLES

E20: If G is a tree, then $c(G) = 1$. Player C's winning strategy is for the cop to move toward the robber along the shortest path that connects them.

E21: If G is a complete graph (or clique), then $c(G) = 1$.

E22: If G is a cycle with at least four edges, then $c(G) = 2$.

E23: If G is a complete bipartite graph $K_{p,q}$ with $p \geq 2$ and $q \geq 2$, then $c(G) = 2$. Player C initially places one cop on each side of the bipartition.

E24: If G is the graph shown in Figure 9.5.6(a) (the 3-cube), then $c(G) = 2$. If C places cops at vertices {1, 8} then a robber placed at any vertex can be captured immediately. But if only one cop is available, then no matter where it is placed, the robber can always escape to a vertex that is not adjacent to the cop's location.

E25: If G is the graph shown in Figure 9.5.6(b) (Petersen's graph), then $c(G) = 3$. If C places cops at vertices {2, 5, 6} then a robber placed at any vertex can be captured immediately. But if only two cops are available, then no matter where they are placed, the robber can always escape to a vertex that is not adjacent to either cop's location.

E26: If G is a p-by-q grid graph with $p \geq 2$ and $q \geq 2$, then $c(G) = 2$. (Figure 9.5.6(c) illustrates a 4-by-4 grid graph.) Cop 1 moves toward the row of the robber, but if already on the same row, then cop 1 moves toward the column of the robber. Cop 2 moves toward the column of the robber, but if already on the same column, then cop 2 moves toward the row of the robber.

DEFINITIONS

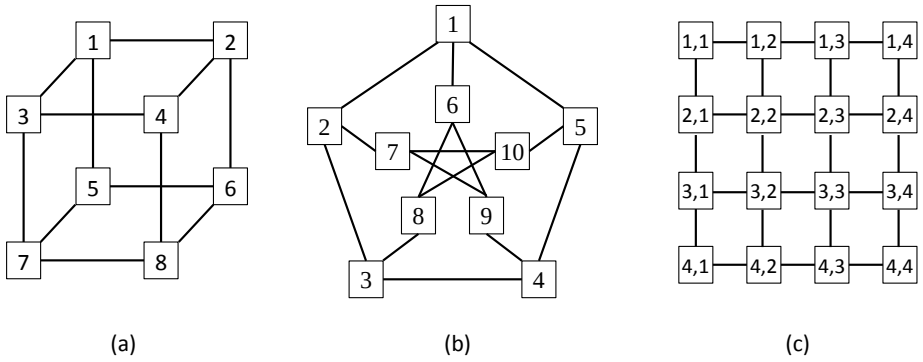

Figure 9.5.6: Examples for cops-and-robbers problem.

D17: Let N[v] denote the *closed neighborhood* of vertex v. Then v is a ***corner vertex*** if there exists some other vertex u with N[v] $\subseteq$ N[u].

D18: G is ***dismantlable*** if there exists a sequence of removing corner vertices that ends when only one vertex remains. Such a sequence is called an ***elimination ordering***.

FACTS

F32: Nowakowski et al [NoWi83] and Quilliot [Qu83] each develops a characterization of the graphs G with c(G) = 1, as follows: c(G) = 1 if and only if G is dismantlable.

F33: If G is a chordal graph, then c(G) = 1. (A chordal graph always has a *simplicial* vertex v such that N[v] is a clique, and any such v is a corner vertex.)

F34: Clarke [Cl02] shows that if G is an outerplanar graph then c(G) $\leq$ 2.

F35: Theis [Th11] shows that if G is a series-parallel graph then c(G) $\leq$ 2.

F36: Aigner et al [AiFr84] shows that if G is a planar graph then c(G) $\leq$ 3.

F37: Schroeder et al [Sc01] shows that if G is a toroidal graph (can be embedded in a torus) then c(G) $\leq$ 4.

F38: Joret et al [JoKaTh10] shows that if G is a treewidth-k graph then c(G) $\leq \lfloor \frac{k}{2} \rfloor + 1$.

F39: Frankl [Fr87] shows that if G is a d-cube then c(G) = $\lceil \frac{d+1}{2} \rceil$.

F40: Aigner et al [AiFr84] shows that if G is a graph with *girth* (length of smallest cycle) $\geq$ 5, then c(G) $\geq$ the minimum degree of any vertex in G.

F41: Fomin et al [FoGoKr08] proves that the cops-and-robbers problem is $\mathcal{NP}$-hard for arbitrary graphs G.

F42: Goldstein et al [GoRe95] shows that if the initial location of each cop is specified as part of the problem instance, then this variation of the cops-and-robbers problem is $EXPTIME$-complete.

F43: Chung et al [ChHoIs11] gives a pseudo-polynomial-time dynamic programming algorithm for solving the cops-and-robbers problem when the number of cops is fixed. The algorithm's running time is $O(n^{2k+2})$, where n is the number of vertices in G and k is the number of cops.

F44: Llewellyn et al [LlToTr89] shows that it is $\mathcal{NP}$-complete to determine whether k cops can capture an infinitely fast robber if the cops are placed sequentially but cannot move once placed.

REMARKS

R5: The complexity status of the decision version of the (standard) cops-and-robbers problem currently remains unresolved. Is it $\mathcal{NP}$-complete? Is it $EXPTIME$-complete?

R6: Another currently unresolved question is known as Meyniel's conjecture: Is c(G) in $O(\sqrt{n})$ for connected graphs G? Chiniforooshan [Ch08] shows that $c(G)$ is in $O(n/\log n)$, which is currently the best known bound. (Here again n denotes the number of vertices of G.)

9.5.4 Additional Variations

The previous subsections have discussed some of the best-known pursuit-evasion problems such as sweeping, edge search, node search, mixed search, and cops-and-robbers. The current subsection discusses some of the many possible additional variations that can be constructed by increasing or restricting the capabilities of the pursuers and/or the evader, and/or by modifying the kind of graph structure through which the pursuers and evaders move.

FACT

F45: Nowakowski [No93], Dyer [Dy04], Barat [Ba06], Alspach et al [AlDyHaYa07], and Yang et al [YaCa07-a] [YaCa07-b] examine the sweeping, edge search, node search, and mixed search problems when G is a directed graph or multidigraph. Different variants occur depending on which participants must obey the specified edge directions.

DEFINITIONS

D19: In ***directed sweeping***, both the pursuers and the evader must obey the specified edge directions.

D20: In ***undirected sweeping***, both the pursuers and the evader may ignore the edge directions (so they can traverse each edge in either direction).

D21: In ***weak sweeping***, the pursuers must obey the edge directions, but the evader may ignore these directions.

D22: In ***strong sweeping***, the evader must obey the edge directions, but the pursuers may ignore these directions.

FACT

F46: Let d(G), u(G), w(G), and s(G) denote the minimum number of pursuers needed to capture an evader using directed, undirected, weak, or strong sweeping, respectively. Dyer [Dy04] shows these inequalities: s(G) $\leq$ min{d(G), u(G)} and max{d(G), u(G)} $\leq$ w(G).

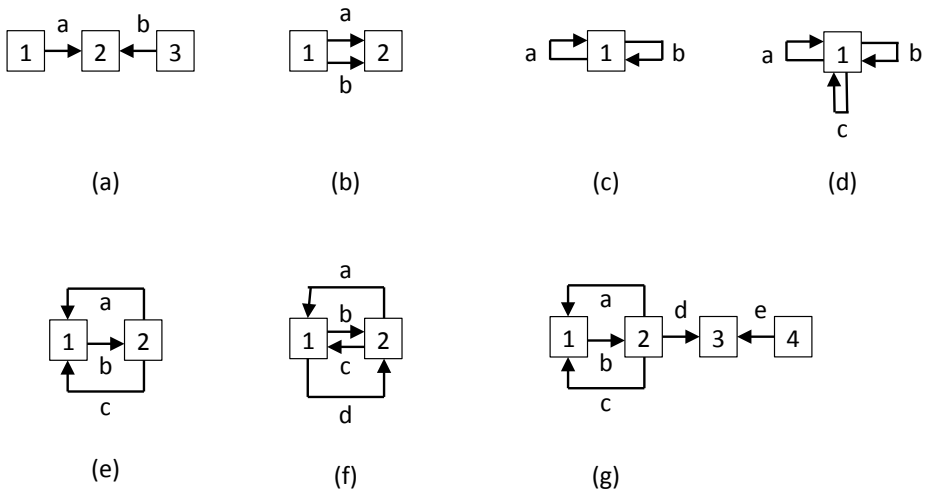

Figure 9.5.7: Directed graph examples.

EXAMPLES

E27: If G is a directed path then d(G)=1, u(G)=1, w(G)=1, and s(G)=1.

E28: If G is the graph in Figure 9.5.7(a) then d(G)=2, u(G)=1, w(G)=2, and s(G)=1.

E29: If G is the graph in Figure 9.5.7(b) then d(G)=2, u(G)=2, w(G)=2, and s(G)=1.

E30: If G is the graph in Figure 9.5.7(c) then d(G)=2, u(G)=2, w(G)=2, and s(G)=1. For strong sweeping, one pursuer can clear G by starting at vertex 1 and then traversing each edge in a backward direction.

E31: If G is the graph in Figure 9.5.7(d) then d(G)=2, u(G)=2, w(G)=2, and s(G)=2.

E32: If G is the graph in Figure 9.5.7(e) then d(G)=2, u(G)=3, w(G)=3, and s(G)=1. For strong sweeping, one pursuer can clear G by starting at vertex 1 and then traversing edges a, b, c each in a backward direction.

E33: If G is the graph in Figure 9.5.7(f) then d(G)=2, u(G)=3, w(G)=3, and s(G)=2.

E34: If G is the graph in Figure 9.5.7(g) then d(G)=3, u(G)=3, w(G)=4, and s(G)=1.

FACTS

F47: Dyer [Dy04] shows that s(G) = 1 if and only if each strongly connected component of G is either a single vertex or a cycle or a subdivision (homeomorphism) of one of the graphs shown in Figure 9.5.7(c) or (e).

F48: Gottlob [GoLeSc03] examines pursuit-evasion when G in a hypergraph. This variation is known as *robber-and-marshals*. The robber (evader) resides in a vertex, and each marshal (pursuer) resides in a hyperedge. Capture occurs when any marshal occupies a hyperedge that is incident to the robber's vertex.

F49: Barriere et al [BaFlFrSa02], Kolling et al [KoCa08] [KoCa10], Daniel et al [DaBoKoTo10], and Borie et al [BoToKo11] consider pursuit-evasion on graphs in which each vertex and each edge has a specified *width*. These vertex and edge widths represent the number of pursuers needed to guard or clear each vertex or edge. The latter two papers also consider graphs where each edge may have a specified length.

F50: Kolling et al [KoCa08] [KoCa10] define a variation of node search known as the Graph-Clear problem, and also present a polynomial-time algorithm for Graph-Clear on trees.

F51: Fomin et al [FoGo00] [FoHeTe05], Daniel et al [DaBoKoTo10], and Borie et al [BoToKo11] consider variations of pursuit-evasion with different objectives such as minimizing the elapsed time until capture, or the total distance travelled by all the pursuers, or the sum of the times that each pursuer is present in the graph.

F52: Daniel et al [DaBoKoTo10] presents a pseudo-polynomial-time heuristic algorithm called ESP for edge search on series-parallel graphs.

F53: Borie et al [BoToKo11] develops polynomial-time and pseudo-polynomial-time algorithms, and also $\mathcal{NP}$-completeness and strong $\mathcal{NP}$-completeness results, for several variations of sweeping on an assortment of graph classes.

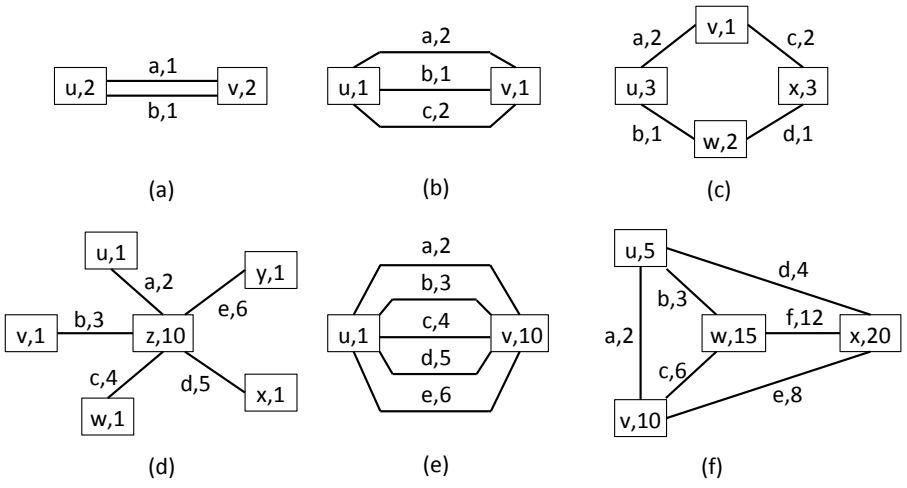

Figure 9.5.8: Examples for sweeping with specified vertex and edge widths.

EXAMPLES

Each vertex and edge in the graphs of Figure 9.5.8 is labeled with both its name and its width. Each edge in these graphs has length 1, and each pursuer travels at speed 1.

E35: The graph in Figure 9.5.8(a) can be cleared with 2 pursuers: Both pursuers start at vertex u and depart u simultaneously. In parallel, one pursuer clears edge a, and one pursuer clears edge b. Finally both pursuers arrive at v simultaneously. The total distance travelled is 2, and the elapsed time is 1.

E36: The graph in Figure 9.5.8(b) can be cleared with 3 pursuers: All pursuers start at vertex u. One pursuer guards u while the other two pursuers depart from u, clear edge a, and arrive at v. One pursuer guards v while the other pursuer departs from v, clears b, and arrives at u. Finally two pursuers depart from u, clear c, and arrive at v. The total distance travelled is 5, and the elapsed time is 3. [Alternatively, if 4 pursuers are available, this graph can be cleared in elapsed time 2. Or, if 5 pursuers are available, the graph can be cleared in elapsed time 1.]

E37: The graph in Figure 9.5.8(c) can be cleared with 3 pursuers: All pursuers start at vertex u and depart u simultaneously; two pursuers clear edge a, while the third pursuer clears edge b. When the first two pursuers arrive at v, one guards v while the other travels through the graph toward w. Two pursuers arrive at w simultaneously and clear w. Next these two pursuers depart w simultaneously; one clears d while the other travels through the graph toward v. When two pursuers reside at v, they both depart v and clear c. Finally all three pursuers arrive at x simultaneously. The total distance travelled is 10, and the elapsed time is 6. [Alternatively, if 4 pursuers are available, this graph can be cleared with total distance 6 and elapsed time 2, as follows: In parallel, two pursuers travel from u to x along edges a and c, while another pursuer travels from u to x along edges b and d. The fourth pursuer remains stationary at w.]

E38: The graph in Figure 9.5.8(d) can be cleared with 10 pursuers: Initially, 2 pursuers reside at vertex u, 3 pursuers reside at vertex v, and 5 pursuers reside at vertex x. In parallel, these 10 pursuers clear edges a, b, and d respectively, and all pursuers arrive at vertex z simultaneously. Next, again in parallel, 4 pursuers clear edge c, while the other 6 pursuers clear edge e. The total distance travelled is 20, and the elapsed time is 2. [Alternatively, if 20 pursuers are available, this graph can be cleared in elapsed time 1.]

E39: The graph in Figure 9.5.8(e) can be cleared with 11 pursuers: All pursuers start at vertex u, and one pursuer remains stationary to guard u. In parallel, 2 pursuers clear edge a, 3 pursuers clear edge b, and 5 pursuers clear edge d. These 10 pursuers arrive at vertex v simultaneously. Next, again in parallel, 4 pursuers clear edge c, while 6 pursuers clear edge e. The total distance travelled is 20, and the elapsed time is 2. [Alternatively, if 20 pursuers are available, this graph can be cleared in elapsed time 1.]

E40: The graph in Figure 9.5.8(f) can be cleared with 25 pursuers: 3 pursuers start at vertex u, 18 pursuers start at vertex x, and 4 pursuers start in the center of edge d. These last 4 pursuers clear edge d; 2 pursuers travel toward u, and 2 pursuers travel toward x. When they arrive, the 5 pursuers at u clear u, and the 20 pursuers at x clear x. Next, 5 pursuers depart from u; 2 pursuers clear edge a, and 3 pursuers clear edge b. Simultaneously, 20 pursuers depart from x; 8 pursuers clear edge e, and 12 pursuers clear edge f. When they arrive, the 10 pursuers at v clear v, and the 15 pursuers at w clear w. Finally, in parallel, 6 pursuers depart v along edge c, and 6 pursuers depart w along c. Eventually these 12 pursuers will meet in the middle of edge c. The total distance travelled is 35, and the elapsed time is 2.

FACTS

F54: Sugihara et al [SuSu89], Dawes [Da92], Neufeld [Ne96], Tanaka [Ta96], Dumitrescu et al [DuKoSuZy08], and Munteanu et al [MuBo10] study sweeping on p-by-q grid graphs such that the evader becomes visible to any pursuer that occupies the same row or column, the pursuers can communicate information such as the evader's position, and the ratio of the speeds of the evader and pursuers is fixed or bounded.

F55: Hahn et al [HaMa06], [Ha07] describe an exponential-time algorithm for solving the cops-and-robbers problem on directed graphs when the number of cops is fixed.

F56: Goldstein et al [GoRe95] shows that the cops-and-robbers problem is $EXPTIME$-complete for directed graphs.

F57: The literature considers many additional variants of pursuit-evasion, some of which we briefly mention here:

- The pursuers and/or evader must begin at specified locations within the graph.
- A pursuer can see and/or capture the evader if within distance at most ϵ.
- Pursuers are non-uniform (different speeds, visibility, and/or capture capabilities).
- Evaders can capture/destroy pursuers.
- Evaders move randomly rather than adversarially, for example, via a Markovian random walk.
- The pursuers' strategy is randomized rather than deterministic.
- The pursuers' strategy yields capture with high probability rather than guaranteed capture.
- The pursuers' strategy performs well in average-case rather than worst-case.
- The algorithm that produces the pursuers' strategy is approximate or heuristic/experimental rather than optimal.
- The size and/or structure of the graph is not known in advance.
- The environment is a geometric space rather than a graph; for example, a circle, convex polygon, non-convex polygon, or any other 2D or 3D region (bounded or unbounded, possibly containing holes or obstacles).

References

[AiFr84] M. Aigner and M. Fromme, A game of cops and robbers, *Discrete Applied Mathematics* **8** (1984), 1–12.

[Al04] B. Alspach, Searching and sweeping graphs: a brief survey, *Matematiche* **59** (2004), 5–37.

[AlDyHaYa07] B. Alspach, D. Dyer, D. Hanson, and B. Yang, Lower bounds on edge searching, in *Lecture Notes in Computer Science* **4614** (2007), Springer, 516–527.

[Ba06] J. Barat, Directed path-width and monotonicity in digraph searching, *Graphs and Combinatorics* **22** (2006), 161–172.

[BaFlFrSa02] L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro, Capture of an intruder by mobile agents, in *Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures* (2002), 200–209.

[BaFrSaTh03] L. Barriere, P. Fraigniaud, N. Santoro, and D. Thilikos, Searching is not jumping, *Graph-Theoretic Concepts in Computer Science* **2880** (2003), 34–45.

[Bi91] D. Bienstock, Graph searching, path-width, tree-width and related problems (a survey), *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **5** (1991), 33–49.

[BiSe91] D. Bienstock and P. Seymour, Monotonicity in graph searching, *Journal of Algorithms* **12** (1991), 239–245.

[Bo98] H. Bodlaender, A partial k-arboretum of graphs with bounded treewidth, *Theoretical Computer Science* **209** (1998), 1–45.

[BoKl96] H. Bodlaender and T. Kloks, Efficient and constructive algorithms for the pathwidth and treewidth of graphs, *Journal on Algorithms* **21** (1996), 358–402.

[BoKlKr95] H. Bodlaender, T. Kloks, and D. Kratsch, Treewidth and pathwidth of permutation graphs, *SIAM Journal on Discrete Mathematics* **8** (1995), 606–616.

[BoMo93] H. Bodlaender and R. Möhring, The pathwidth and treewidth of cographs, *SIAM Journal on Discrete Mathematics* **6** (1993), 181–188.

[BoTh04] H. Bodlaender and D. Thilikos, Computing small search numbers in linear time, in *Proceedings of the International Workshop on Parameterized and Exact Computation* (2004), 37–48.

[BoNo11] A. Bonato and R. Nowakowski, *The Game of Cops and Robbers on Graphs*, American Mathematical Society, 2011.

[BoYa11] A. Bonato and B. Yang, Graph searching and related problems, manuscript (2011), to appear in *Handbook of Combinatorial Optimization*.

[BoToKo11] R. Borie, C. Tovey, and S. Koenig, Algorithms and complexity results for graph-based pursuit-evasion, *Autonomous Robots* **31** (2011), 317–332.

[Ch08] E. Chiniforooshan, A better bound for the cop number of general graphs, *Journal of Graph Theory* **58** (2008), 45–48.

[ChHoIs11] T. Chung, G. Hollinger, and V. Isler, Search and pursuit-evasion in mobile robotics: a survey, *Autonomous Robots* **31** (2011), 299–316.

[Cl02] N. Clarke, Constrained cops and robber, Ph.D. Dissertation, Dalhousie University, 2002.

[DaBoKoTo10] K. Daniel, R. Borie, S Koenig, and C. Tovey, ESP: Pursuit evasion on series-parallel graphs, in *Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems* (2010), 1519–1520.

[Da92] R. Dawes, Some pursuit-evasion problems on grids, *Information Processing Letters* **43** (1992), 241–247.

[DeKiTh97] N. Dendris, L. Kirousis, and D. Thilikos, Fugitive-search games on graphs and related parameters, *Theoretical Computer Science* **172** (1997), 233–254.

[DuKoSuZy08] A. Dumitrescu, H. Kok, I. Suzuki, and P. Zylinski, Vision-based pursuit-evasion in a grid, in *Proceedings of 11th Scandinavian Workshop on Algorithm Theory* (2008), 53–64.

[Dy04] D. Dyer, Sweeping graphs and digraphs, Ph.D. Dissertation, Department of Mathematics, Simon Fraser University, 2004.

[ElSuTu94] J. Ellis, I. Sudborough, and J. Turner, The vertex separation and search number of a graph, *Information and Computation* **113** (1994), 50–74.

[ElWa08] J. Ellis and R. Warren, Lower bounds on the pathwidth of some grid-like graphs, *Discrete Applied Mathematics* **156** (2008), 545–555.

[FoGo00] F. Fomin and P. Golovach, Graph searching and interval completion, *SIAM Journal on Discrete Mathematics* **13** (2000), 454–464.

[FoGoKr08] F. Fomin, P. Golovach and J. Kratochvil, On tractability of cops and robbers game, in *Proceedings of 5th IFIP International Conference on Theoretical Computer Science* (2008), 171–185.

[FoPe96] F. Fomin and N. Petrov, Pursuit-evasion and search problems on graphs, in *Proceedings of 27th Southeastern International Conference on Combinatorics, Graph Theory and Computing*, in *Congressus Numerantium* **122** (1996), 47-58.

[FoHeTe05] F. Fomin, P. Heggernes, and J. Telle, Graph searching, elimination trees, and a generalization of bandwidth, *Algorithmica* **41** (2005), 73–87.

[FoTh08] F. Fomin and D. Thilikos, An annotated bibliography on guaranteed graph searching, *Theoretical Computer Science* **399** (2008), 236–245.

[Fr87] P. Frankl, On a pursuit game on Cayley graphs, *Combinatorica* **7** (1987), 289–295.

[GoRe95] A. Goldstein and E. Reingold, The complexity of pursuit on a graph, *Theoretical Computer Science* **143** (1995), 93–112.

[Go89] P. Golovach, A topological invariant in pursuit problems, *Differentsial'nye Uravneniya (Differential Equations)* **25** (1989), 923–929.

[GoPeFo00] P. Golovach, N. Petrov, and F. Fomin, Search in graphs, in *Proceedings of the Steklov Institute of Mathematics* (2000), S90–S103.

[GoLeSc03] G. Gottlob, N. Leone, and F. Scarcello, Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width, *Journal of Computer and System Sciences* **66** (2003), 775-808.

[Gu93] J. Gustedt, On the path width of chordal graphs, *Discrete Applied Mathematics* **45** (1993), 233–248.

[Ha07] G. Hahn, Cops, robbers and graphs, *Tatra Mountains Mathematical Publications* **36** (2007), 163–176.

[HaMa06] G. Hahn and G. MacGillivray, A note on k-cop, l-robber games on graphs, *Discrete Mathematics* **306** (2006), 2492–2497.

[Ha87] Y. Hamidoune, On a pursuit game on Cayley digraphs, *European Journal of Combinatorics* **8** (1987), 285–289.

[JoKaTh10] G. Joret, M. Kaminski, and D. Theis, The cops and robbers game on graphs with forbidden (induced) subgraphs, *Contributions to Discrete Mathematics* **5** (2010), 40–51.

[Ki92] N. Kinnersley, The vertex separation number of a graph equals its path-width, *Information Processing Letters* **42** (1992), 345–350.

[KiPa85] L. Kirousis and C.H. Papadimitriou, Interval graphs and searching, *Discrete Mathematics* **55** (1985), 181–184.

[KiPa86] M. Kirousis and C. Papadimitriou., Searching and pebbling, *Theoretical Computer Science* **47** (1986), 205–218.

[KoCa08] A. Kolling and S. Carpin, Multi-robot surveillance: an improved algorithm for the GRAPH-CLEAR problem, in *Proceedings of the 2008 IEEE International Conference on Robotics and Automation* (2008), 2360–2365.

[KoCa10] A. Kolling and S. Carpin, Pursuit-evasion on trees by robot teams, *IEEE Transactions on Robotics* **26** (2010), 32–47.

[La93] A. LaPaugh, Recontamination does not help to search a graph, *Journal of the ACM* **40** (1993), 224–245.

[LlToTr89] D. Llewellyn, C. Tovey, and M. Trick, Local optimization on graphs, *Discrete Applied Mathematics* **3** (1989), 157–178.

[MeHaGaJoPa88] N. Megiddo, S. Hakimi, M. Garey, D. Johnson, and C. Papadimitriou, The complexity of searching a graph, *Journal of the ACM* **35** (1988), 18–44.

[MuBo10] B. Munteanu and R. Borie, Variations of the vision-based pursuit-evasion problem on a grid, in *Proceedings of the International Conference on Foundations of Computer Science* (2010), 139–144.

[Ne96] S. Neufeld, A pursuit-evasion problem on a grid, *Information Processing Letters* **58** (1996), 5–9.

[No93] R. Nowakowski, Search and sweep numbers of finite directed acyclic graphs, *Discrete Applied Mathematics* **41** (1993), 1–11.

[NoWi83] R. Nowakowski and P. Winkler, Vertex-to-vertex pursuit in a graph, *Discrete Mathematics* **43** (1983), 235–239.

[Pa78] T. Parsons, Pursuit-evasion in a graph, in *Theory and Applications of Graphs, Lecture Notes in Mathematics* **642** (1978), Springer, Berlin, 426–441.

[Pe82] N. Petrov, A problem of pursuit in the absence of information on the pursued, *Differentsial'nye Uravneniya (Differential Equations)* **18** (1982), 1345–1352.

[Qu83] A. Quilliot, Problemes de jeux, de point fixe, de connectivite et de representation sur des graphes, des ensembles ordonnes et des hypergraphes, These d'Etat, Universite de Paris VI, 1983.

[Sc01] B. Schroeder, The copnumber of a graph is bounded by $\lfloor \frac{3}{2} \text{ genus}(G) \rfloor + 3$, in *Categorical Perspectives* (2001), 243–263.

[SeTh93] P. Seymour and R. Thomas, Graph searching and a min-max theorem for tree-width, *Journal of Combinatorial Theory Series B* **58** (1993), 22–33.

[SuSu89] K. Sugihara and I. Suzuki, Optimal algorithms for a pursuit-evasion problem in grids, *SIAM Journal on Discrete Mathematics* **2** (1989), 126–143.

[Ta96] K. Tanaka, An improved strategy for a pursuit-evasion problem on grids, manuscript (1996).

[Th11] D. Theis, The cops and robber game on series-parallel graphs, manuscript (2011), to appear in *Graphs and Combinatorics*.

[Th00] D. Thilikos, Algorithms and obstructions for linear-width and related search parameters, *Discrete Applied Mathematics* **105** (2000), 239–271.

[YaCa07-a] B. Yang and Y. Cao, Directed searching digraphs: monotonicity and complexity, in *Lecture Notes in Computer Science* **4484** (2007), Springer, 136–147.

[YaCa07-b] B. Yang and Y. Cao, Digraph strong searching: monotonicity and complexity, in *Lecture Notes in Computer Science* **4508** (2007), Springer, 37–46.