

# ESP: Pursuit Evasion on Series-Parallel Graphs\*

**Richard Borie**  
Computer Science  
University of Alabama  
borie@cs.ua.edu

**Craig Tovey**  
Industrial and Systems Engineering  
Georgia Institute of Technology  
craig.tovey@isye.gatech.edu

**Kenny Daniel Sven Koenig**  
Computer Science  
University of Southern California  
{kfdaniel,skoenig}@usc.edu

## Abstract

We study pursuit-evasion problems where pursuers have to clear a given graph of fast-moving evaders despite poor visibility, for example, where police search a cave system to ensure that no terrorists are hiding in it. If the vertex connectivity of some part of the graph exceeds the number of pursuers, the evaders can always avoid capture. We therefore focus on graphs whose subgraphs can always be cut at a limited number of vertices, that is, graphs of low treewidth. However, solving pursuit-evasion problems optimally is NP-hard even for the simplest of these graph classes. In this paper, we therefore develop a heuristic approach, called ESP, that solves large pursuit-evasion problems on series-parallel (that is, treewidth-two) graphs quickly and with small costs. It exploits their topology by performing dynamic programming on their decomposition graphs. We apply ESP to different kinds of series-parallel graphs and show that it scales up to larger graphs than a strawman approach based on previous results from the literature.

## Introduction

*Pursuit evasion* is an important problem in artificial intelligence (Gordon, Thrun, and Gerkey 2004), agents (Pellier and Fiorino 2005), robotics (Simov, Slutzki, and LaValle 2000) and theoretical computer science (Parsons 1976). Consider, for example, a scenario where police search a known but twisty cave system to ensure that no terrorists are hiding in it. The police are the pursuers, and the terrorists are the evaders. The cave system can be modeled as a graph with edges that have lengths. The pursuers (which we call robots) and evaders move on this graph. The evaders can hide anywhere on the vertices or edges. They cannot be seen by the robots and can move much faster than them. They get caught only if they collide with a robot on a vertex or edge. The robots move at unit speed. Their travel times

or distances are thus equal to the lengths of their paths. A solution of the pursuit-evasion problem is a movement strategy for a given number of robots with given start vertices on a given graph that enables them to clear the graph, that is, either ensure that no evaders are present or catch them all. An optimal solution minimizes the cost, such as the sum of travel distances or the task-completion time, depending on the desired cost objective. This is a common and very general model of pursuit-evasion problems on graphs (Parsons 1976). For example, a solution remains a solution even if the evaders can be seen by the robots over longer distances or can move only slowly.

If the vertex connectivity of some part of the graph exceeds the number of robots, the evaders can always avoid capture. For instance, suppose that the graph contains a  $K_7$  subgraph. Between any two vertices in that subgraph there are six vertex-disjoint connecting paths, so five robots can not catch an evader. We therefore focus on graphs whose subgraphs can always be cut at a limited number of vertices, that is, graphs of low treewidth. However, solving pursuit-evasion problems optimally is NP-hard even for the simplest of these graph classes (Megiddo et al. 1988; LaPaugh 1993; Borie, Tovey, and Koenig 2009). Yet, large pursuit-evasion problems need to be solved quickly to be of practical help. In this paper, we therefore develop a heuristic approach, called ESP, that solves large pursuit-evasion problems on Series-Parallel graphs quickly and with small costs, by exploiting their topology in the form of their decomposition graphs. We use series-parallel graphs because their topology is realistic for some applications and their pursuit-evasion approaches might be generalizable to even more realistic graph topologies, by generalizing them from the treewidth two of series-parallel graphs to larger treewidths. Indeed, ESP is couched in terms of the decomposition and intended to generalize to graphs of larger treewidths.

In the remainder of the paper, we first define series-parallel graphs, then give a conceptual overview of ESP and finally describe ESP in detail, including how it assigns states to terminal vertices and how it clears subgraphs based on both the states of their terminal vertices and the number of robots that start and end at them. We then apply ESP to different kinds of series-parallel graphs and show that it scales up to larger graphs than does a strawman approach based on previous results from the literature.

\*This material is based upon work supported by, or in part by, the U.S. Army Research Laboratory and the U.S. Army Research Office under contract/grant number W911NF-08-1-0468 and by NSF under contract 0413196. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, companies or the U.S. government.

Copyright © 2009, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

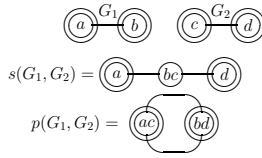


Figure 1: Series and parallel compositions

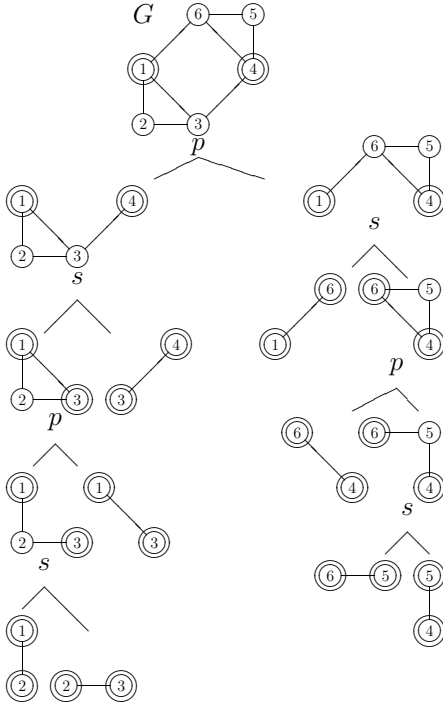


Figure 2: Series-parallel graph and its decomposition tree

### Series-Parallel Graphs

*Series-parallel graphs* (Duffin 1965) are defined recursively by starting with single edges as *base graphs* and successively building larger graphs using series (*s*) and parallel (*p*) compositions. Each composition joins two smaller graphs by fusing at most two designated vertices called *terminal vertices*. The structure of a series-parallel graph can be represented by a *decomposition tree*, whose nodes correspond to subgraphs. A decomposition tree for a series-parallel graph can be constructed in linear time (Valdes, Tarjan, and Lawler 1982). Figure 1 illustrates the series and parallel compositions. The terminal vertices of each graph are doubly circled. Figure 2 shows the decomposition tree of a series-parallel graph that is constructed using several series and parallel operations. As an example of pursuit evasion, consider the following movement strategy for clearing this graph with three robots that all start at vertex 1: Each of the three robots departs vertex 1 along a different incident edge, arriving at vertices 2, 3 and 6, respectively. The robot at vertex 2 proceeds to vertex 3. Next, the two robots at vertex 3 both travel to vertex 4. Finally, one of the robots at vertex 4 proceeds to vertex 6, and the other robot travels from vertex

guard	in	out	status
false	true	false	keeping
false	false	true	lookout
false	true	true	needy
true	any	any	safe
false	false	false	unattached

Figure 3: Five possible statuses for terminal vertices

4 to vertex 5 and then to vertex 6.

### Conceptual Overview of ESP

ESP is a recursive approach for clearing a graph with a given decomposition tree that is given the distribution of the robots at the terminal vertices of a graph before and after clearing the graph. The movement strategy of ESP clears all edges without giving evaders the opportunity to recontaminate edges that have already been cleared. ESP uses divide and conquer to determine such a movement strategy since it is NP-hard to optimize over all possible movement strategies to find one that clears the graph with minimum cost. ESP decomposes the graph into subgraphs and then computes and combines movement strategies on the subgraphs to clear the graph with small cost. This results in ESP optimizing over a subset of possible movement strategies, namely those that are *consistent* with the given decomposition of a graph  $G$  into subgraphs  $G_1$  and  $G_2$ . Informally, by *consistent* we mean that either the robots first clear all of one subgraph and then all of the other subgraph or split into two groups that clear both subgraphs separately but simultaneously. Formally, we mean the following: 1) Each robot begins at a terminal vertex of  $G$ . 2) Each robot ends at a terminal vertex of  $G$ . 3) Once a robot enters the interior of a subgraph  $G_i$  no robot in  $G_i$  may leave  $G_i$  until  $G_i$  has been cleared. 4) No robot may enter the interior of a subgraph after it has been cleared (except for deployment purposes).

### Terminal Status

We exploit the structure of the pursuit-evasion problem by labeling each terminal vertex with a status that represents the state of the vertex. Let  $G$  denote the subgraph of a series-parallel graph corresponding to the node of the decomposition tree that is currently being cleared. Let  $t_1$  and  $t_2$  denote the terminal vertices of  $G$ . Any portion of the entire graph that is not in  $G$  is called *exterior area*. Vertex  $t_i$  might be incident upon a cleared exterior area, an unclesared exterior area, both or neither. We use two boolean variables **in** and **out** to denote whether  $t_i$  is incident upon a cleared or unclesared exterior area, respectively (because we must keep evaders “in” to prevent their escape to cleared exterior areas or “out” to prevent their entry from unclesared exterior areas). We use a third boolean variable **guard** to denote whether a robot has been stationed at  $t_i$ . Different combinations of these three variables yield five statuses, see Figure 3.

- **Keeping:** The term “keeping” derives from the phrase “keep in  $G$ ”. Since  $t_i$  is incident upon a cleared exterior area, we must prevent evaders from leaving  $G$  via  $t_i$ .

Hence, at least one robot must guard  $t_i$  and remain there until all interior areas incident to  $t_i$  are cleared.

- **Lookout:** Since  $t_i$  is incident upon an uncleared exterior area, we must prevent any potential evaders from entering  $G$  via  $t_i$ . Hence, at least one robot must guard  $t_i$  by the time any interior area incident to  $t_i$  is cleared and remain there until  $G$  is cleared.
- **Needy:**  $t_i$  is incident upon both cleared and uncleared exterior areas but no robot guards it to prevent evaders from leaving or entering  $G$  via  $t_i$  or from entering a cleared exterior area from an uncleared one via  $t_i$ . Hence,  $t_i$  must be converted to “safe” status, described next.
- **Safe:**  $t_i$  is guarded by a robot to prevent evaders from leaving or entering  $G$  via  $t_i$  or from entering a cleared exterior area from an uncleared one via  $t_i$ .
- **Unattached:**  $t_i$  is a terminal vertex of the entire graph and not incident upon any external areas yet. Hence, we do not have to worry about evaders yet.  $t_i$  continues to have “unattached” status until the entire graph subdivides via a parallel decomposition.

## ESP

Figure 4 shows the pseudo code of ESP, and Figure 5 shows the pseudo code of its helper functions.  $ESP(G, r_1, r_2, r'_1, r'_2, s_1, s_2)$  computes a cost sufficient for clearing  $G$  for the given values:  $s_i$  is the status of  $t_i$  and  $r_i$  is the number of robots at  $t_i$  at the time we start clearing  $G$ .  $r'_i$  is the number of robots at  $t_i$  at the time we finish clearing  $G$ . We always require these values to be nonnegative and  $r_1 + r_2 = r'_1 + r'_2$ . We do not count a robot that guards a terminal vertex towards these values since such a robot does not actively participate in the clearing (except in maintaining the status of the terminal vertex). The cost can be the sum of travel distances (“minimize distance”) or the task-completion time (“minimize time”). ESP uses the helper function *select* to select the cost depending on the desired cost objective. A cost of infinity means that the graph cannot be cleared. The cost must admit two operations.  $ESP(G_1, \dots) \oplus ESP(G_2, \dots)$  is the cost of the sequential movement strategies “clear  $G_1$ , then clear  $G_2$ ”, and  $ESP(G_1, \dots) \odot ESP(G_2, \dots)$  is the cost of the simultaneous movement strategies “clear  $G_1$  and  $G_2$  separately but simultaneously.” For example, both  $\oplus$  and  $\odot$  are  $+$  for minimizing distance and  $\oplus$  is  $+$  and  $\odot$  is  $\max$  for minimizing time. ESP is initially called as  $ESP(G, r_1, r_2, r'_1, r'_2, \text{unattached}, \text{unattached})$ , where  $G$  is the entire graph. If one does not care about the distribution of the robots at the terminal vertices before or after clearing the graph, then one can minimize the cost over all such distributions, which is what we do in the experiments.

For ease of presentation, function *ESP* includes several nondeterministic choices of values. A deterministic implementation should consider all valid combinations for the nondeterministic choices of values. All nondeterministically chosen numbers must be non-negative integers. Function *ESP* is expressed recursively (top-down), following the structure of the decomposition tree for the given graph.

```

ESP( $G, r_1, r_2, r'_1, r'_2, s_1, s_2$ )
for  $i := 1, 2$  do
  if  $s_i = \text{needy}$  then
     $r_i := r_i - 1$ 
     $r'_i := r'_i - 1$ 
     $s_i := \text{safe}$ 
  if infeasible( $r_1, r_2, r'_1, r'_2, s_1, s_2$ ) then return  $\infty$ 
  if  $G$  is base graph with one edge of length  $L$  then
    if  $r_1 > r'_1$  then return  $\text{select}(r_1 - r'_1, 1) \times L$ 
    else if  $r_1 < r'_1$  then return  $\text{select}(r'_1 - r_1, 1) \times L$ 
    else if  $r_1 = 0$  or  $r_2 = 0$  then return  $2 \times L$ 
    else return  $\text{select}(2, 1) \times L$ 
  else if  $G := p(G_1, G_2)$  then // parallel composition
    choose minimum of
      case 1: // clear  $G_1$  then clear  $G_2$ 
        for  $i := 1, 2$  do
          for  $j := 1, 2$  do
             $s_{ij} := s_i$ 
          for  $i := 1, 2$  do
            if  $s_{i1} = \text{unattached}$  then  $s_{i1} := \text{lookout}$ 
            else if  $s_{i1} = \text{keeping}$  then  $s_{i1} := \text{needy}$ 
            else if  $s_{i2} = \text{unattached}$  then  $s_{i2} := \text{keeping}$ 
            else if  $s_{i2} = \text{lookout}$  then  $s_{i2} := \text{needy}$ 
            choose  $r'_{i1}, r'_{i2}$  such that  $r_1 + r_2 = r'_{i1} + r'_{i2}$ 
            return  $ESP(G_1, r_1, r_2, r'_{i1}, r'_{i2}, s_{i1}, s_{i2}) \oplus ESP(G_2, r'_{i1}, r'_{i2}, r'_{i1}, r'_{i2}, s_{i2}, s_{i2})$ 
          case 2: // clear  $G_2$  then clear  $G_1$ 
            // symmetric to case 1
          case 3: // clear  $G_1$  and  $G_2$  simultaneously
            for  $i := 1, 2$  do
              if  $s_i = \text{safe}$  then  $p_i := 0$ 
              else choose  $p_i := 0$  or  $p_i := 1$ 
              if  $p_i = 1$  then  $s_i := \text{safe}$ 
              else if  $s_i = \text{unattached}$  then choose  $s_i := \text{keeping}$  or  $s_i := \text{lookout}$ 
            choose  $r_{11}, r_{21}, r'_{11}, r'_{21}, r_{12}, r_{22}, r'_{12}, r'_{22}$  such that
               $r_{11} + r_{21} = r'_{11} + r'_{21}, r_{12} + r_{22} = r'_{12} + r'_{22},$ 
               $r_{11} + r_{12} = r_1 - p_1, r_{11} + r'_{12} = r'_1 - p_1,$ 
               $r_{21} + r_{22} = r_2 - p_2$  and  $r'_{21} + r'_{22} = r'_2 - p_2$ 
            return  $ESP(G_1, r_{11}, r_{21}, r'_{11}, r'_{21}, s_1, s_2) \odot ESP(G_2, r_{12}, r_{22}, r'_{12}, r'_{22}, s_1, s_2)$ 
      else if  $G = s(G_1, G_2)$  then // series composition
        choose  $x_1, x_2, x_3$  such that  $r_1 + r_2 = x_1 + x_2 + x_3$ 
        choose minimum of
          case 1: // clear  $G_1$  then clear  $G_2$ 
             $x_3 := x_3 + x_1 - r'_1$ 
            return  $\text{deploy}(G, r_1, r_2, x_1, x_2, x_3) \oplus [ESP(G_1, x_1, x_3, r'_1, x'_3, s_1, \text{lookout})$ 
               $\oplus ESP(G_2, x_2, x_2, 0, r'_2, \text{keeping}, s_2)]$ 
          case 2: // clear  $G_2$  then clear  $G_1$ 
            // symmetric to case 1
          case 3: // clear  $G_1$  and  $G_2$  simultaneously
            choose  $p_3 := 0$  or  $p_3 := 1$ 
            if  $p_3 = 1$  then  $s_3 := \text{safe}$ 
            else choose  $s_3 := \text{keeping}$  or  $s_3 := \text{lookout}$ 
            choose  $x'_1, x'_2$  such that  $x'_1 \leq r'_1$  and  $x'_2 \leq r'_2$ 
            choose  $x_{31}, x_{32}$  such that  $x_3 = p_3 + x_{31} + x_{32}$ 
             $x_{31} := x_{31} + x_1 - x'_1$ 
             $x_{32} := x_{32} + x_2 - x'_2$ 
            return  $\text{deploy}(G, r_1, r_2, x_1, x_2, x_3) \oplus [ESP(G_1, x_1, x_{31}, x'_1, x_{31}, s_1, s_3)$ 
               $\odot ESP(G_2, x_{32}, x_2, x_{32}, x'_2, s_3, s_2)] \oplus \text{cleanup}(G, r'_1 - x'_1, r'_2 - x'_2)$ 

```

Figure 4: ESP

It should be implemented via dynamic programming (either bottom-up or top-down), using a dynamic programming table to store computed values and thereby avoiding repeated calculations of the values. Conventional dynamic programming techniques can then use the information stored in the dynamic programming table to build the movement strategy for clearing the graph. Helper function *dist* should also be implemented via dynamic programming. We now explain the steps of ESP when called as  $ESP(G, r_1, r_2, r'_1, r'_2, s_1, s_2)$ , closely following the pseudo code in Figure 4.

## Preparatory Steps

If any terminal vertex  $t_i$  has “needy” status, then a robot needs to guard it since it is incident upon both cleared and uncleared exterior areas. Therefore, ESP changes the status of  $t_i$  to “safe” and decrements the number  $r_i$  and  $r'_i$  of robots at  $t_i$  before and after, respectively, clearing the graph, which can result in a negative number of robots, indicating that the graph cannot be cleared. ESP returns a cost of infinity if

```

infeasible( $r_1, r_2, r'_1, r'_2, s_1, s_2$ )
  for  $i := 1, 2$  do
    if  $r_i < 0$  or ( $r_i = 0$  and  $s_i = \text{keeping}$ ) or  $r'_i < 0$  or ( $r'_i = 0$  and  $s_i = \text{lookout}$ ) then
      return true
    return  $r_1 + r_2 = 0$ 
select( $a, b$ )
  if the objective is minimizing distance then return  $a$ 
  else if the objective is minimizing time then return  $b$ 
 $\oplus(a, b)$  // infix operator
  return  $a + b$ 
 $\odot(a, b)$  // infix operator
  return  $\text{select}(a + b, \max(a, b))$ 
dist( $G$ )
  if  $G$  is base graph with one edge of length  $L$  then return  $L$ 
  else if  $G = p(G_1, G_2)$  then return  $\min(\text{dist}(G_1), \text{dist}(G_2))$ 
  else if  $G = s(G_1, G_2)$  then return  $\text{dist}(G_1) + \text{dist}(G_2)$ 
deploy( $G, r_1, r_2, x_1, x_2, x_3$ )
  if  $x_1 > r_1$  then
    return  $\text{select}((x_1 - r_1) \times \text{dist}(G) + x_3 \times \text{dist}(G_2), \max(\text{dist}(G), \text{dist}(G_2)))$ 
  else if  $x_2 > r_2$  then
    return  $\text{select}((x_2 - r_2) \times \text{dist}(G) + x_3 \times \text{dist}(G_1), \max(\text{dist}(G), \text{dist}(G_1)))$ 
  else
    return  $\text{select}((r_1 - x_1) \times \text{dist}(G_1) + (r_2 - x_2) \times \text{dist}(G_2), \max(\text{dist}(G_1), \text{dist}(G_2)))$ 
cleanup( $G, z_1, z_2$ )
  return  $\text{select}(z_1 \times \text{dist}(G_1) + z_2 \times \text{dist}(G_2), \max(\text{dist}(G_1), \text{dist}(G_2)))$ 

```

Figure 5: Helper functions for ESP

the combination of parameter values indicates immediately that the graph cannot be cleared. Helper function *infeasible* returns true iff any of the following conditions is violated: the number of robots at each terminal vertex must be non-negative; a terminal vertex with “keeping” status must start with at least one robot; a terminal vertex with “lookout” status must finish with at least one robot; and the number of robots must be positive. We now separately consider what ESP does if the graph is a base graph and if it is formed by a parallel or series composition.

### Base Graph

Suppose  $G$  consists of one edge  $(t_1, t_2)$  of length  $L$ .

- If  $r_1 > r'_1$ , ESP sends  $r_1 - r'_1$  robots from  $t_1$  to  $t_2$ .
- If  $r_1 < r'_1$ , ESP sends  $r'_1 - r_1$  robots from  $t_2$  to  $t_1$ .
- If  $r_1 = r'_1$  (and hence also  $r_2 = r'_2$ ):
  - If  $r_2 = 0$ , ESP sends one robot from  $t_1$  to  $t_2$  and return.
  - If  $r_1 = 0$ , ESP sends one robot from  $t_2$  to  $t_1$  and return.
  - If  $r_1 > 0$  and  $r_2 > 0$ , ESP sends one robot from  $t_1$  towards  $t_2$  and one robot from  $t_2$  towards  $t_1$ . When the two robots meet, each robot returns to where it started.

### Parallel Composition

Suppose  $G$  is the parallel composition of subgraphs  $G_1$  and  $G_2$ . ESP then returns the smallest cost of three subcases, namely “clear  $G_1$ , then clear  $G_2$ ,” “clear  $G_2$ , then clear  $G_1$ ” and “clear  $G_1$  and  $G_2$  separately but simultaneously.”

**Clear  $G_1$ , then clear  $G_2$**  The terminal vertices of  $G$  pass on their status to the corresponding terminal vertices of both subgraphs. ESP then changes each terminal vertex with “unattached” status to “lookout” status and each one with “keeping” status to “needy” status when clearing  $G_1$  (resulting in status  $s_{i1}$  of  $t_i$ ) since it is incident upon an uncleared exterior area, namely  $G_2$ . Similarly, ESP changes each terminal vertex of  $G_2$  with “unattached” status to “keeping” status and each one with “lookout” status to “needy” status when clearing  $G_2$  (resulting in status  $s_{i2}$  of  $t_i$ ) since it

is then incident upon a cleared exterior area, namely  $G_1$ . ESP then nondeterministically chooses the number  $r''_i$  of robots at each terminal vertex  $t_i$  of  $G$  after clearing  $G_1$  but before clearing  $G_2$  such that  $r_1 + r_2 = r''_1 + r''_2$ , resulting in two subproblems  $ESP(G_1, r_1, r_2, r''_1, r''_2, s_{11}, s_{21})$  and  $ESP(G_2, r''_1, r''_2, r'_1, r'_2, s_{12}, s_{22})$ , that can be solved recursively. ESP combines the costs of clearing both subgraphs with the  $\oplus$  operator, which is used for combining the costs of sequential movement strategies.

**Clear  $G_2$ , then clear  $G_1$**  This subcase is symmetric to the previous subcase, exchanging the roles of  $G_1$  and  $G_2$ .

**Clear  $G_1$  and  $G_2$  separately but simultaneously** ESP first nondeterministically chooses for each terminal vertex  $t_i$  whether to change it to “safe” status (in which case it changes the value of  $p_i$  from zero to one), except if it already has “safe” status (in which case guarding it again would be wasteful) or “needy” status (in which case guarding it is required). If it afterwards still has “unattached” status, ESP nondeterministically chooses to assign it either “keeping” or “lookout” status. The terminal vertices of  $G$  pass on their updated status to the corresponding terminal vertices of both subgraphs. ESP then nondeterministically chooses the number of robots  $r_{ij}$  and  $r'_{ij}$  at  $t_i$  before and after, respectively, clearing  $G_j$  among those that clear  $G_j$  such that  $r_{1j} + r_{2j} = r'_{1j} + r'_{2j}$ ,  $r_{i1} + r_{i2} = r_i - p_i$  and  $r'_{i1} + r'_{i2} = r'_i - p_i$ , resulting in two subproblems  $ESP(G_1, r_{11}, r_{21}, r'_{11}, r'_{21}, s_1, s_2)$  and  $ESP(G_2, r_{12}, r_{22}, r'_{12}, r'_{22}, s_1, s_2)$ , that can be solved recursively. ESP combines the costs of clearing both subgraphs with the  $\odot$  operator, which is used for combining the costs of simultaneous movement strategies.

### Series Composition

Suppose  $G$  is the series composition of subgraphs  $G_1$  and  $G_2$ . Different from a parallel composition, there is now a middle vertex  $t_3$  whose number of robots is zero before clearing  $G$ . Let  $s_3$  denote the status of  $t_3$ . ESP can move robots to  $t_3$  to aid in clearing  $G$ . It nondeterministically deploys the robots from the terminal vertices  $t_1$  and  $t_2$  to  $t_1$ ,  $t_2$  and  $t_3$  so that these vertices receive  $x_1$ ,  $x_2$  and  $x_3$  robots, respectively. Thus,  $r_1 + r_2 = x_1 + x_2 + x_3$ . The helper function  $deploy(G, r_1, r_2, x_1, x_2, x_3)$  computes the cost of this deployment step. ESP then returns the smallest cost of three subcases, namely “clear  $G_1$ , then clear  $G_2$ ,” “clear  $G_2$ , then clear  $G_1$ ” and “clear  $G_1$  and  $G_2$  separately but simultaneously.”

**Clear  $G_1$ , then clear  $G_2$**  The terminal vertices of  $G$  pass on their status to the corresponding terminal vertices of both subgraphs. ESP assigns  $t_3$  “lookout” status when clearing  $G_1$  since it is incident upon an uncleared exterior area, namely  $G_2$ . Similarly, ESP assigns it “keeping” status when clearing  $G_2$  since it is then incident upon a cleared exterior area. The number of robots  $x'_3$  at  $t_3$  after clearing  $G_1$  but before clearing  $G_2$  is determined by the previous choices to be  $x'_3 = x_3 + x_1 - r'_1$ , resulting in two subproblems  $ESP(G_1, x_1, x_3, r'_1, x'_3, s_1, \text{lookout})$  and  $ESP(G_2, x'_3, x_2, 0, r'_2, \text{keeping}, s_2)$ , that can be solved recursively. ESP combines the costs of the deployment step and

the costs of clearing both subgraphs with the  $\oplus$  operator, which is used for combining the costs of sequential movement strategies.

**Clear  $G_2$ , then clear  $G_1$**  This subcase is symmetric to the previous subcase, exchanging the roles of  $G_1$  and  $G_2$ .

**Clear  $G_1$  and  $G_2$  separately but simultaneously** The terminal vertices of  $G$  pass on their status to the corresponding terminal vertices of both subgraphs. ESP nondeterministically chooses for  $t_3$  whether to assign it “safe” status (in which case it changes the value of  $p_3$  from zero to one). If it afterwards does not have “safe” status, ESP nondeterministically assigns it either “keeping” or “lookout” status. ESP then nondeterministically chooses the number of robots  $x'_i$  at  $t_i$  after clearing  $G_i$ , restricted to  $x'_i \leq r'_i$  to simplify the cleanup step described below, and nondeterministically partitions the robots at  $t_3$  into  $x_{31}$  robots that help to clear  $G_1$  and  $x_{32}$  robots that help to clear  $G_2$ . Thus,  $x_3 = p_3 + x_{31} + x_{32}$ . The number of robots  $x'_{3i}$  at  $t_3$  after clearing  $G_i$  among those that clear  $G_i$  is determined by the previous choices to be  $x'_{3i} = x_{3i} + x_i - x'_i$ , resulting in two subproblems  $ESP(G_1, x_1, x_{31}, x'_1, x'_{31}, s_1, s_3)$  and  $ESP(G_2, x_{32}, x_2, x'_{32}, x'_2, s_3, s_2)$ , that can be solved recursively. Finally, ESP deploys  $r'_i - x'_i$  robots from  $t_3$  to  $t_i$  to make the number of robots at  $t_i$  equal to  $r'_i$ . The helper function  $cleanup(G, r'_1 - x'_1, r'_2 - x'_2)$  computes the cost of this cleanup step. ESP combines the costs of clearing both subgraphs with the  $\odot$  operator, which is used for combining the costs of simultaneous movement strategies. It then combines the resulting costs with the costs of the deployment step and the cleanup step with the  $\oplus$  operator, which is used for combining the costs of sequential movement strategies.

## Runtime of ESP

ESP, like many algorithms on series-parallel graphs (Borie, Parker, and Tovey 1992), runs in time linear in the size of the graph. In particular, the implementation in Figure 4 runs in time  $O(nr^8)$  when graph  $G$  has  $n$  edges and  $r = r_1 + r_2$  is the number of robots. Its most costly part is to clear the subgraphs of a series composition separately but simultaneously, so we justify the runtime bound for this case. There are  $O(n)$  nodes in the decomposition tree. For each node, there are  $O(r)$  possible values for  $r_1, r_2$  and  $r'_1$  each, which then force the value of  $r'_2$ . There are a constant number of values for  $s_1$  and  $s_2$  each. There are  $O(r)$  choices for  $x_1$  and  $x_2$  each, which force the value of  $x_3$ . There are two choices for  $p_3$ . There are  $O(r)$  choices for  $x_{31}$ , which then forces the value of  $x_{32}$ . Finally, there are  $O(r)$  choices for  $x'_1$  and  $x'_2$  each. Thus, there are  $O(nr^8)$  combinations. Helper function  $dist$  runs in time  $O(n)$  if it is implemented via dynamic programming because each traversal does  $O(1)$  work per node of the decomposition tree. However, by more careful analysis, ESP runs in time only  $O(nr^6)$ . The key observation is that the subexpressions can be stored in a dynamic programming table to avoid recomputations. For each  $G$ , there are  $O(r^3)$  distinct calls to ESP on each subgraph  $G_1$  and  $G_2$ , which yields  $O(r^6)$  calls to operator  $\odot$ , of which only the  $O(r^3)$  minimum values for each combination of  $x_1, x'_1,$

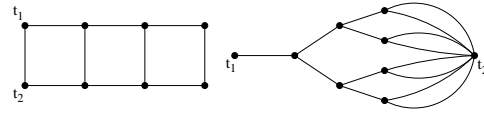


Figure 6: Ladder graph  $L_4$  (l) and BTL graph  $B_4$  (r)

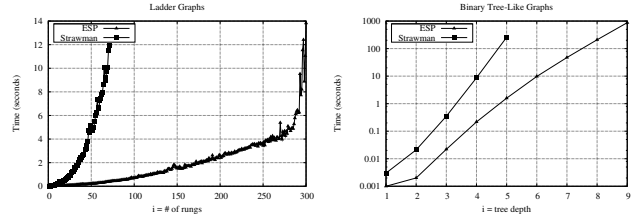


Figure 7: Runtime on ladder graphs (l) and BTL graphs (r)

$x_2$  and  $x'_2$  must be remembered. These results can then be combined similarly with the results from helper functions  $deploy$  and  $cleanup$ , such that only  $O(r^6)$  calculations are performed at each of the  $O(n)$  nodes of the decomposition tree.

## Experimental Results

We do not know of any publicly available implementation of pursuit-evasion approaches on graphs. We therefore created a strawman approach to compare against ESP. The strawman approach determines the minimum number of robots required to clear arbitrary graphs but does not determine movement strategies, neither for minimizing distance nor time. It is based on the idea that “recontamination does not help to clear a graph” (LaPaugh 1993), meaning that, if there is a movement strategy that clears a graph, then there must also be a movement strategy that clears a graph such that no cleared vertex becomes recontaminated. We then improved the runtime of the strawman approach by noticing that, if a robot is at a vertex that has only one uncleared adjacent vertex, it might as well move to that vertex right away (“forced move”) and thereby reduce the branching factor of the search.

The strawman approach assumes that the evaders hide only on the vertices (node searching), while ESP assumes that the evaders hide on the vertices or edges (edge searching). However, it is simple to reduce edge searching on graphs to node searching (Bienstock and Seymour 1991). The reduction takes a series-parallel graph  $G$  as input and then constructs the dual graph  $G'$  by creating one vertex in  $G'$  for each edge in  $G$ . Two vertices in  $G'$  are adjacent iff the corresponding edges share a common vertex in  $G$ . The dual graph  $G'$  is then used as input to the strawman approach.

### Ladder Graphs

The first class of graphs we used in our experiments are ladder graphs with edges of length one, see Figure 6 (l). These graphs resemble a physical ladder in that there are two sides connected by rungs. We use  $L_i$  to denote the ladder graph with  $i$  rungs. We can construct them as series-parallel graphs

with the following recurrence, where  $e$  refers to a single edge:

$$\begin{aligned} L_1 &= e \\ L_{i+1} &= p(e, s(e, s(L_i, e))) \end{aligned}$$

The number of vertices and edges of the graphs  $L_i$  are linear in the number of iterations  $i$  since each iteration increases the number of edges by 3 and the number of vertices by 2. At most 3 robots are required to clear any graph.

We used ESP with iterative deepening to determine the number of robots required to clear ladder graphs. Both ESP and the strawman approach correctly minimized this number. ESP cleared  $L_i$  with the minimum number of robots with distance  $4i - 2$ , which is very close to the optimal value of  $4i - 4$ . It cleared  $L_i$  with the minimum number of robots in time  $4i - 2$ , which is about a factor of 2 worse than the optimal value of  $2i - 1$ . The runtimes of ESP and the strawman approach are compared in Figure 7 (l). Ladder graphs are well suited for the strawman approach since almost all moves are forced. Yet, the strawman approach could solve graphs only up to  $L_{68}$  within 10 seconds whereas ESP could solve graphs up to  $L_{295}$  since its runtime increased much less. Around  $L_{275}$ , ESP hit memory limitations. ESP would have been able to solve larger graphs with additional memory that would have allowed for a larger dynamic programming table.

### BTL Graphs

The second class of graphs we used in our experiments are what we call BTL (binary tree-like) graphs with edges of length one, see Figure 6 (r). These graphs are complete binary trees with one edge added from the root to the first terminal vertex and two parallel edges added from each leaf vertex to the second terminal vertex. We use  $B_i$  to denote the BTL graph constructed from a complete binary tree of depth  $i - 2$ . We can construct them as series-parallel graphs with the following recurrence:

$$\begin{aligned} B_1 &= e \\ B_{i+1} &= s(e, p(B_i, B_i)) \end{aligned}$$

The number of vertices and edges of the graphs  $B_i$  are exponential in the number of iterations  $i$  since, if the number of vertices in graph  $B_i$  is  $V_i$ , then  $V_{i+1} = 2 \times V_i - 1$  and, if the number of edges in graph  $B_i$  is  $E_i$ , then  $E_{i+1} = 2 \times E_i + 1$ . The number of robots required to clear  $B_i$  is  $i$ .

Both ESP and the strawman approach correctly minimized the number of robots required to clear BTL graphs. ESP cleared  $B_i$  with the minimum number of robots with distance  $2^i + 2i + 3$ , which is quite close to the lower bound  $2^i - 1$  given by the number of edges. It cleared  $B_i$  with the minimum number of robots in time  $2^i + 2i + 3$ , which is the same as the distance because every subgraph is cleared sequentially. The runtimes of ESP and the strawman approach are compared in Figure 7 (r). Due to the exponential growth of the graphs, the strawman approach could solve graphs

only up to  $B_4$  within 10 seconds but ESP could solve graphs up to  $B_6$ . Note that the runtime scale in Figure 7 (r) is logarithmic, implying that the runtime of ESP is approximately  $O(n)$  but the runtime of the strawman approach is approximately  $O(n^{4/3})$ , where  $n$  is the number of vertices of the graphs.

### Conclusion

We presented ESP, a heuristic approach to pursuit and evasion on series parallel graphs, and demonstrated that it scales up to larger graphs than a strawman approach based on previous results from the literature. ESP has the advantage that it allows for edges of different lengths and for different cost objectives, such as minimizing the sum of travel distances or the task-completion time. It is future work to investigate improved deployment steps and to extend ESP to more general graph classes.

### References

- Bienstock, D., and Seymour, P. 1991. Monotonicity in graph searching. *Journal of Algorithms* 12(2):239–245.
- Borie, R.; Parker, R.; and Tovey, C. 1992. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica* 7(1–6):555–581.
- Borie, R.; Tovey, C.; and Koenig, S. 2009. Algorithms and complexity results for pursuit-evasion problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*.
- Duffin, R. 1965. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications* 10:303–318.
- Gordon, G.; Thrun, S.; and Gerkey, B. 2004. Visibility-based pursuit-evasion with limited field of view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 20–27.
- LaPaugh, A. 1993. Recontamination does not help to search a graph. *Journal of the ACM* 40(2):224–245.
- Megiddo, N.; Hakimi, S.; Garey, M.; Johnson, D.; and Papadimitriou, C. 1988. The complexity of searching a graph. *Journal of the ACM* 35(1):18–44.
- Parsons, T. 1976. Pursuit-evasion in a graph. In *Theory and Applications of Graphs*. Springer Verlag. 426–441.
- Pellier, D., and Fiorino, H. 2005. Coordinated exploration of unknown labyrinthine environments applied to the pursuit evasion problem. *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems* 895–902.
- Simov, B.; Slutzki, G.; and LaValle, S. 2000. Pursuit-evasion using beam detection. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1657–1662.
- Valdes, J.; Tarjan, R.; and Lawler, E. 1982. The recognition of series parallel digraphs. *SIAM Journal on Computing* 11(2):298–313.