ESP: Pursuit Evasion on Series-Parallel Graphs[*]

(Extended Abstract)

K. Daniel[1] R. Borie[2] S. Koenig[1] C. Tovey[3]

[1]Univeristy of Southern California
Computer Science
{kfdaniel,skoenig}@usc.edu

[2]University of Alabama
Computer Science
borie@cs.ua.edu

[3]Georgia Tech
ISYE
craig.tovey@isye.gatech.edu

ABSTRACT

We develop a heuristic approach, called ESP, that solves large pursuit-evasion problems on series-parallel (that is, treewidth-2) graphs quickly and with small costs. It exploits their topology by performing dynamic programming on their decomposition graphs. We show that ESP scales up to much larger graphs than a strawman approach based on previous results from the literature.

Categories and Subject Descriptors

I.2.11 [**Distributed AI**]: Multi-Agent Systems

General Terms

Algorithms, Experimentation, Theory

Keywords

Pursuit Evasion, Robotics, Series-Parallel Graphs, Treewidth

1. INTRODUCTION

We consider scenarios where robots search a known environment (such as a cave system) to ensure that no evaders are hiding in it. The environment can be modeled as a graph with edges that have lengths and widths. The evaders can hide anywhere on the vertices or edges. The evaders can move arbitrarily fast, and they cannot be seen by the robots unless caught. They get caught only if they collide with one or more robots on a vertex or a number of robots at the same point on an edge that is no smaller than the width of the 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 of evaders. An optimal solution minimizes some cost objective, such as the sum of travel distances or their task-completion time. This pursuit-evasion model is called edge searching [5]. Many variants of edge searching with edge widths one have been studied in the literature [3] but the typical results are the same: polynomial algorithms for determining the minimal number of robots required to clear a given tree, NP-hardness for determining the minimal number of robots required to clear a general graph, and no algorithms for minimizing distance or time.

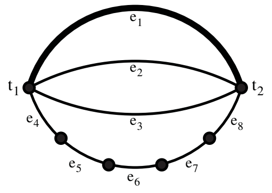

Figure 1: Series-parallel graph

If the vertex connectivity of some part of a graph exceeds the number of robots, the evaders can always avoid capture. We therefore focus on graphs whose subgraphs can always be cut at a limited number of vertices, namely series-parallel (that is, treewidth-2) graphs [2]. Series-parallel graphs are defined recursively by starting with single edges as base graphs and successively building larger graphs using series and parallel 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 and which can be constructed in linear time [6]. Solving our pursuit-evasion problems on series-parallel graphs is NP-hard [1]. We have therefore developed a heuristic approach, called ESP, that solves large pursuit-Evasion problems on Series-Parallel graphs with n edges and r robots in time $O(nr^6)$ and with small distance or time by performing dynamic programming on their decomposition trees. ESP has the advantage over other pursuit-evasion algorithms that it allows for edges of different widths and for different cost objectives.

2. CONCEPTUAL OVERVIEW OF ESP

ESP is a recursive approach for clearing a graph with a given decomposition tree and given start and end locations of the robots at the terminal vertices. The movement strategy of ESP clears all edges without giving evaders the opportunity to recontaminate edges that have already been cleared. 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 begins to clear 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.

[*]This material is based upon work supported by, or in part by, NSF under contract/grant number 0413196, ARL/ARO under contract/grant number W911NF-08-1-0468 and ONR in form of a MURI under contract/grant number N00014-09-1-1031.

Our design philosophy results in a natural but non-trivial heuristic approach to pursuit evasion, which we illustrate by example. Consider the series-parallel graph G in Figure 1 and make the following assumptions: Four robots start at terminal vertex t_1. Two robots must end at terminal vertex t_1 and the other two robots at terminal vertex t_2 after clearing the graph. The graph is the parallel composition of subgraphs G_1 and G_2. G_1 is the parallel composition of three edges, namely e_1 of width two and e_2 and e_3 of width one. G_2 is the series composition of five edges that form a path, namely e_4 to e_8, all of width one. In this example our goal is to find a movement strategy with a small sum of travel distances. An example of an inconsistent movement strategy for clearing G is this: At time 0, send three robots from t_1 to t_2 along e_1. At time 1, send one robot from t_2 to t_1 along e_8 to e_4. At time 6, send one robot from t_1 to t_2 along e_2 and one robot from t_2 to t_1 along e_3. This movement strategy is inconsistent with the given decomposition of G into G_1 and G_2 since robots first clear part of G_1, then a robot that helped to clear G_1 clears G_2 and finally robots clear the rest of G_1. The consistent movement strategies fall into the following three categories.

Category 1: Clear G_1, then clear G_2. After clearing G_1, each robot must be at t_1 or t_2. Let r_1'' denote the number of robots at t_1 after clearing G_1. There must be at least one robot each at t_1 and t_2 since otherwise an evader could sneak into G_1 from G_2. Therefore, $r_1'' = 1, 2, 3$. The sum of travel distances is minimized as follows. **Case $r_1'' = 1$:** At time 0, send two robots from t_1 to t_2 along e_1 and one robot from t_1 to t_2 along e_2; at time 1, send one robot from t_2 to t_1 along e_3; and at time 2, send one robot from t_1 to t_2 along e_3. G_1 is cleared at time 3. We do not consider the graph cleared at time 2.5 since the robots are not yet where they are supposed to be after clearing the graph. At time 3, send one robot from t_2 to t_1 along e_8 to e_4. The resulting task-completion time for clearing the graph is 8 and the sum of travel distances is 10. **Case $r_1'' = 2$:** At time 0, send two robots from t_1 to t_2 along e_1 and one robot from t_1 to t_2 along e_2; and at time 1, send one robot from t_2 to t_1 along e_3. G_1 is cleared at time 2. At time 2, send one robot from t_1 to t_2 along e_4 to e_8 and one robot from t_2 to t_1 along e_8 to e_4. The resulting task-completion time for clearing the graph is 7 and the sum of travel distances is 14. **Case $r_1'' = 3$** : At time 0, send two robots from t_1 to t_2 along e_1 and one robot from t_1 to t_2 along e_2; and at time 1, send two robots from t_2 to t_1 along e_3. G_1 is cleared at time 2. At time 2, send one robot from t_1 to t_2 along e_4 to e_8. The resulting task-completion time for clearing the graph is 7 and the sum of travel distances is 10.

Category 2: Clear G_2, then clear G_1. Similar to the previous category.

Category 3: Clear G_1 and G_2 separately but simultaneously. In this case we divide the robots into two teams and optionally robots to guard one or both of the terminal vertices. Team 1 clears G_1, and team 2 clears G_2. If a terminal vertex is not guarded, then the teams must agree that no evader should escape G (to an exterior area) via the terminal vertex or that no evader should find refuge in G (from an exterior area) via the terminal vertex. We call this agreement the state of the terminal vertex. For example, a graph of two parallel edges of width one could be cleared by two teams of one robot each by sending both robots from t_1 to t_2 along each edge because both teams ensure no escape at t_1 and no refuge at t_2 - the two robots are correctly coordinated. On the other hand, the graph cannot be cleared by two teams of one robot each by sending one robot from t_1 to t_2 along one edge and the other robot from t_2 to t_1 along the other edge, because the first robot ensures no escape at t_1 and no refuge at t_2 while the second robot ensures no escape at t_2 and no refuge at t_1 - the robots are not correctly coordinated.

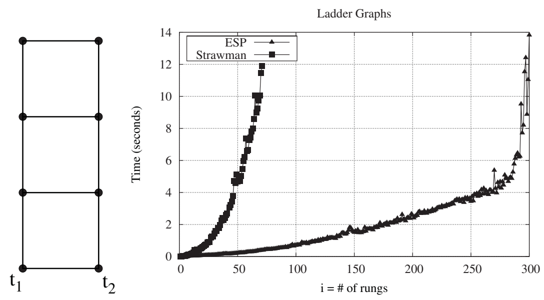

Figure 2: Runtime on ladder graphs

For our example, one robot must initially be stationed at t_1. There must be at least two robots in team 1 (due to the edge of width two) and one robot in team 2. The sum of travel distances is minimized as follows. At time 0, send the robot from team 2 from t_1 to t_2 along e_4 to e_8 and both robots from team 1 from t_1 to t_2 along e_1; at time 1, send a robot from team 1 from t_2 to t_1 along along e_2; at time 2, send a robot from team 1 from t_1 to t_2 along e_3; and at time 3, send a robot from team 1 from t_2 to t_1 along e_3 (or e_1 or e_2). The resulting task-completion time for clearing the graph is 5 and the sum of travel distances is 10.

3. EXPERIMENTAL RESULTS

We created a strawman approach to evaluate ESP against, which is based on the idea that "recontamination does not help to clear a graph" [4]. The strawman approach determines the minimal number of robots required to clear arbitrary graphs but does not determine movement strategies, neither for minimizing distance nor time. We compare ESP and the strawman approach on ladder graphs with edges of lengths and widths one with all robots starting at t_1. At most 3 robots are required to clear ladder graphs of any size. Both ESP and the strawman approach correctly minimized the number of robots required to clear ladder graphs. We use L_i to denote the ladder graph with i rungs. For ladder graphs L_i for $i > 2$ with 3 robots, ESP cleared L_i with distance $d_{ESP} \leq 7i - 11$, which is about a factor of $7/3$ worse than the lower bound $d_{OPT} \geq 3i - 2$ given by the number of edges. ESP cleared L_i in time $t_{ESP} = 4i - 6$, which is about a factor of 2 worse than the lower bound $t_{OPT} \geq 2i$ given by twice the distance from the start to the farthest vertex (robots must end at t_1 in this case). The runtimes of ESP and the strawman approach are compared in Figure 2. The strawman approach could solve graphs only up to L_{68} within 10 seconds whereas ESP could solve graphs up to L_{295} (where it hit memory limitations) since its runtime increased much less. For the first 5 ladder graphs we were able to compute the optimal solutions in time and distance by brute force search and determined that ESP performed optimally.

4. REFERENCES

[1] R. Borie, C. Tovey, and S. Koenig. Algorithms and complexity results for pursuit-evasion problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 59–66, 2009.

[2] R. Duffin. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, 10:303–318, 1965.

[3] R. Fomin and D. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008.

[4] A. LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40(2):224–245, April 1993.

[5] T. Parsons. Pursuit-evasion in a graph. In *Theory and Applications of Graphs*, pages 426–441. Springer-Verlag, 1976.

[6] J. Valdes, R. Tarjan, and E. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, May 1982.