REVISTA DE LA FACULTAD DE INGENIERIA
www.ingenieria.uda.cl
22 (2008) 41-45

Dynamic control nodes in graphs

Ricardo Contreras[1] and Julio Godoy[1]
1. DIICC, Universidad de Atacama, Copiapó, Chile
E-mail: rcontreras@diicc.uda.cl, jgodoy@diicc.uda.cl

Abstract

Coloring problem, in graph theory, is known to be NP-Hard for the chromatic number and NP-Complete for the corresponding decision problem. However, for a clique, it is also known that the chromatic number equals the size of that clique. We use this principle and propose a control algorithm, based on a laminar structure, that performs a control over certain graphs families in O(|V| + |E|) time. For an illustrating and corroborating purpose we based our study on a classical chase problem.

Keywords: Ptolemaic graphs, control nodes, tree-covering, laminar structures, chordal graphs.

1. Introduction

In this article we present an efficient time algorithm for traffic controlling using as few resources as possible. The idea is to cover and control a certain area in a graph, and once the covering has been done, keep on controlling the covered area while moving towards other areas using as few resources as possible for this controlling. We based our study in the popular classic board game Scotland Yard. We changed the original problem (in which Mr. X was forced to use specific transportation systems and show his presence after a certain number of turns) letting the fugitive run as long as he isn't caught (i.e. same position, in the same time, as a detective). We also gave him the possibility to move as far as he wants at each turn, and being invisible. This reformulation will be called "Catch Problem". We'll start with the formal definition of the Catch Problem followed by a graphs analysis, establishing for which graphs families it is possible to obtain, in linear time, a new problem representation structure (i.e. a tree) for controlling. We'll present our Catch Problem control strategy based on a reduced number of dynamic control nodes, and finally we will analyze if an optimization is possible, for the number of dynamic control nodes needed.

the catch problem

Let $G = (V, E)$ a connected graph representing a map; $A \subseteq V / |A| \geq 1$, a subset with a certain number of agents (control resources); and $X \subseteq V / |X| = 1$, a subset for an unique element (i.e. Mr. X).

There are only two possible states: on the run, Mr. X is still a fugitive, and catched, Mr. X has been caught. Using the above subsets we could represent this events as $A \cap X = \{\emptyset\}$ and $A \cap X \neq \{\emptyset\}$ respectively.

We will consider the movements as changes in the subsets's elements. The changes will be done alternately in a cycle, and only one element is allowed to change at each cycle.

Since we consider changes in the subsets, for each movement there will be two events: before and after. For the subset A, this could be represented as an initial event ($A - \{u \in A\}$), where we keep the node we eliminate (u). Latter, for the second event ($w \in \{V - A\} / w \neq u$), we select a new node that hasn't been used. However, for the subset X there's a difference; its state could remain without changes (i.e. Mr. X remained in the same position) or could eventually change but, and here we impose a critical rule, it changes if and only if there is a path (a connected subgraph) $P \subseteq G$ so that: $\forall u \in P , u \cap A = \{\emptyset\}$ and $\exists u/u \in X \wedge u \in P$. Despite the action taken, the subset X will always be defined by a) the graph's topology and b) the presence of nodes belonging to the subset A. See figure 1 for an example.

The cardinality of each path P is bounded by A, therefore $P \cap A = \{\emptyset\}$ and the cardinality of P (i.e. the longest path) could be at most $| V | - | A |$. According to this, the cardinality of A should be $| V | -1$ so that in the next state of A, $A \cap X \neq \{\emptyset\}$ (i.e. Mr. X gets caught). However, in terms of the size of subset A, it doesn't represent an efficient way to solve the problem. In order to solve this problem we focused on an efficient use of the agents subset A, specifically its cardinality and its settings in G, for the delimitation of P paths and therefore, X.

Suitable graphs familias

For the catch problem we could generalize and state that there are two possible extreme configurations (depending on the graph topology): a chain and a clique (K_n). For the first case, a simple agent situated at an extreme could perform a control of G moving towards the other extreme (despite the position of Mr. X, he will eventually be found), see figure 2.a. For the second case, however, the amount of agents needed is in direct relation with the size of the clique (i.e. $k - 1$), see figure 2.b. A third case would consider all graphs G that are neither a chain nor a clique (e.g. a cycle C_n with $n \geq 4$).

Our proposal takes into consideration only the first two cases and their combination. Therefore, as an obvious starting point, we will focus on the chordal graphs [2]. Now, since the clique still represents our worst scenario in resources allocation, we need a structure that allows us to control each of the cliques (one at a time) and, once this control over a clique has been done, to "free" the resources used for controlling (i.e. the dynamic control nodes)

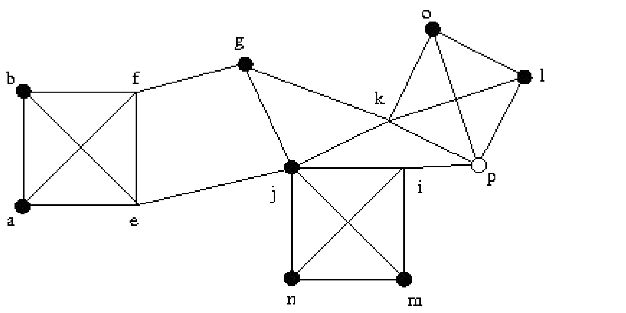

Figure 1: Graph G = (V, E) A = { a, b, n, m, j, g, o, l }, X = { p }, two possible P paths {i, p, k}, {f, e} however, only the first alternative is a valid one

The structure selected for this control is a Tree-covering structure; it allows us to perform a control over G from one starting point, like the DFS [4]. It is important to precise that the Tree-covering structure needs to have a particular configuration which is that all v ∈ Kn are consecutive in the tree (despite their order, all nodes belonging to a clique should be represented as a chain in the tree), we will call this structure T. It is also important to precise that not all chordal graphs have a T structure. See figure 2.c for an example.
We need a structure T that allows us to represent the two extreme configurations, considering at the same time all those subsets of chordal graphs that have a T representation (e.g. a chain of cliques), in a linear time. We know that ptolemaic graphs [7] could represent some of the chordal graphs that have a T structure and also the two extreme cases (see figure 3).
We know that, for ptolemaic graphs, their recognition is linear [5]; even more, we know that it is possible to obtain a T structure also in linear time, based on the laminar principle [7]. Therefore, because of the complexity in obtaining the T structure, our problem will be focused, from this point forward, on the ptolemaic family and their equivalent classes ([1], [3], [6]).
Dynamic control nodes for ptolemaic graphs
For an optimal use of resources (i.e. agents) we will use the structure defined by T; for nodes covering, the following operations will be used: Pred(T ,v), that refers to the father of v in T and Suc(T ,v) that refers to the immediate predecessor of v in T [7]. The idea is to use these operations to control each clique at a time (this is the principle of the dynamic control nodes).

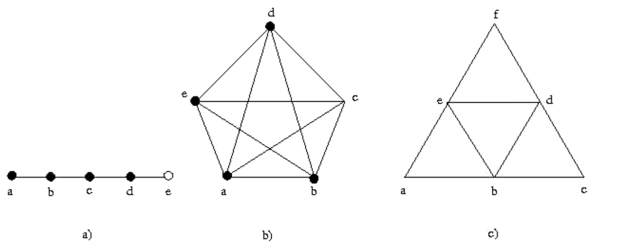

Figure 2: a) a chain where controlling the node e allows us to perform a control moving towards a. b) a clique K5 where four nodes are needed to find Mr. X in the following move. c) a chordal graph where no T representation is possible so that all cliques are a chain

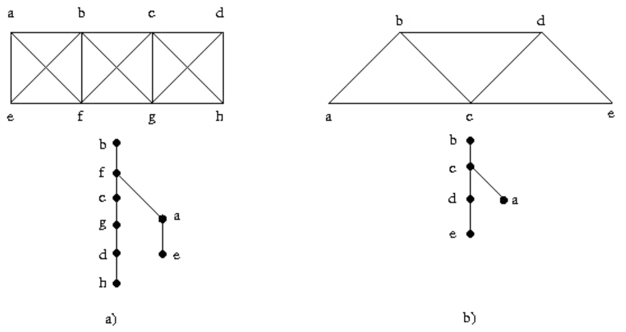

Figure 3: a) a ptolemaic graph that has a T structure such that every maximal clique is a chain. b) a chordal graph that has a T structure such that every maximal clique is a chain but that is not ptolemaic

A problem will arise when a node belongs to more than one clique, because we will have to control the clique and, once the control has been done, keep on controlling the intersection(s). Since we defined that the subset X could change its state to a node u if and only if there is a path P ($\forall$ u ∈ P, u ∩ A = {∅} and $\exists$ u/u ∈ X $\wedge$ u ∈ P); once that a P has been completely checked, we could free the control nodes in the intersection, therefore reducing the size of subset A. See figure 4.a for an example.

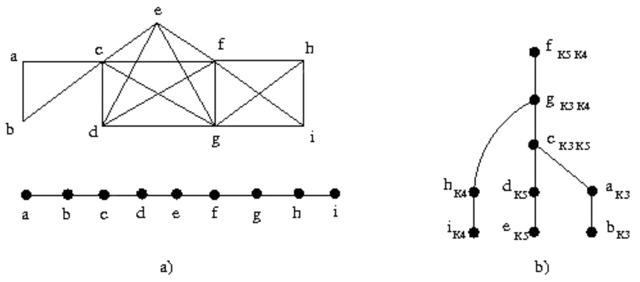

Figure 4: a) A ptolemaic graph G and its T structure where c, f and g are the cliques intersections. b) Another T representation of the original graph G with clique labels for each node

Finally the intersection of maximal cliques (found while obtaining T) could be used to label each node according to its clique using the algorithm described in [7] (it is important to note that a node always belongs to at least one clique). See figure 4.b for an example.

2. Algorithm

For the following algorithm we will use a structure called label(u) (obtained in linear time from [7]) that includes, for each u ∈ A, the maximal cliques to which u belongs (e.g. a node u could have a label referring to a clique K1 and another label to K7). Note that a node belonging to X (i.e. Mr. X) could be changed only if there is a path P such that P ∩ A = { ∅ } (i.e. Agents not in the way).

```
1. Data: Laminar tree with labels T
2. Result n ⊆ X
3. found ← 0, A ← {∅}, j←0;
4. arrayleafs ← leafs from V /* Obtaining the leafs and labels*/
5. repeat
6.   flag ←1;
7.     /*Process a leaf and considers cliques chain */
8.     if arrayleafs[j] Not Empty then u ← arrayleafs[j];
9.   else obtain new arrayleafs[j] /*new leafs are needed (apply DFS)*/
10.  end_if
11.    ki ←label(u); /*Assign Ki the label corresponding to u */
12.  while ki ⊆ label(u) and found = 0 and flag = 1 do
13.   A ← A ∪ {u};/*Assign agents to nodes associated with ki*/
14.     if A ∩ X ≠ {∅} then found ←1 /* Mr. X has been found */
15.     else label(u) ←label(u) – ki;
16.     end_if
17.     if Pred(T,u) ≠ {∅} then u ← Pred(T,u)
18.     else flag ← 0;
19.     end if
20.  end_while
21.  while Suc(T,u) ≠ {∅} and found = 0 do
22.     aux ← u;
23.     label(u) ← label(u) –{ki};
24.     if ki ∈ label(Suc(T,u)) then u ← Suc(T,u) /* This ensures only 25. one route */
26.     end_if
27.     if label(aux) = {∅} then Suc(T,Pred(T,aux)) ← Suc(T,aux);
28.        Pred(T, Suc(T,aux)) ← Pred (T,aux);
29.        /*Eliminates all nodes belonging to only one clique */
30.        A ← A – aux;
31.     end_if
32.  end_while
33.  A ← A-u;
34.   Suc(T, Pred (T,u)) ←{∅}; /*Eliminates the last node*/
35.  j ← j+1;
36.until found = 1;
```

Algorithm 1. Dynamic control nodes algorithm

For demonstrative purpose let's consider the graph in figure 4.a and the corresponding T structure (figure 4.b with the clique labels for each node). We will assume Mr. X is at node c. We will start from the leaf node i, obtain the label (only one label at this node) and perform a control node by node assigning resources (i.e. Agents) to the predecessors of i while the label, in each subsequent predecessor, equals the one obtained at the leaf. Once nodes g and f, the only ones with a double labeling, are controlled, Mr. X will not be able to reach h or i but will be able to change its position to d, e, a, b or remain in c (A = {i, h, g, f } reduces to A = {f, g}). Considering the leaf node e we will repeat the process and end up with a single node in control, c. Because of their labeling, e, d, c, g and f will be controlled and since c is the only one with a diﬀerent label, it will remain (A = {e, d, c, g, f } obtained at this stage, is reduced to A = {c}). Assuming that Mr. X is still on the run (i.e. a or b), he will be catched in the last cycle.

2.1 Complexity

Since the structures used to represent the original graph as T are obtained in linear time [7], we'll focus on the algorithm's complexity. The main problem is on line 4, that is, the leafs identification. For this, we create an array (arrayleafs) that contains all the leafs present in T so that on each cycle, a diﬀerent leaf is used. This process will be done using the DFS algorithm, so that each time a node has a Suc(T ,u) = {0} it will be included in the array arrayleafs. The complexity of this step is $O(|V| + |E|)$. Finally, the complexity of the algorithm is $O(n + k')$, where n represents the nodes of G and k' the sum of the cardinalities of all nodes belonging to more than one clique.

3. Conclusion

The dynamic control nodes algorithm works for ptolemaic graphs and their equivalent classes; however we have shown that it is possible to have a T structure and therefore to apply the algorithm for certain chordal graphs that do not necessarily belong to these graphs classes.
Because of the bases considered for the study, it is not possible to apply the algorithm neither to all chordal graphs or non-chordal graphs, however this does not implies that it can't be done. As further work, we plan to treat some diﬀerent classes.
Despite the topology of the T structure, our algorithm will always use the minimal amount of control nodes due to the fact that we study the structure from a bottom-up approach (i.e. contrary to the DFS approach), ensuring at each cycle at least a partial clique node control. Therefore an optimization for the amount of dynamic control nodes is not necessary. Finally, because of the correspondence with ptolemaic graphs, this algorithm is suitable for some acyclic hypergraphs, in particular the $\gamma -$ acyclic ones.

4. References

[1] Chartrand, G. and Kay, D.C. 1965. A characterization of certain Ptolemaic graphs. In Canada Journal of Mathematics, Vol. 17, No. 2, pp 342 – 346.

[2] Golumbic, M.C. 2004. Algorithmic graph theory and perfect graphs, North-Holland Publishing Co., Amsterdam, The Netherlands.

[3] Howorka, Edward. 1981. A characterization of Ptolemaic graphs. In Journal of Graph Theory. Vol 5, pp 323-331.

[4] Knut, Donald. 1997. The Art of Computer Programming. Addison-Wesley, Boston, USA.

[5] Maffray, F. and Hammer, P.L. 1990. Completely separable graphs. In Discrete Applied Mathematics. Vol. 27, No. 1-2, pp 85-99.

[6] Mulder, H.M. and Bandelt, H.J.. 1986. Distance-hereditary graphs. In Journal of Combinatorial Theory Series B, Vol. 41, No. 2, pp 182-208.

[7] Uehara, Ryuyhei and Uno, Yushi. Laminar structure of ptolemaic grapas and its application. In Lecture Notes in Computer Science. Vol 3827, pp 186-195.