Report of two Works
Analysis of Branch and Bound
The Smoking Robber

ESTEBAN ROMÁN
Universidad Adolfo Ibáñez
INRIA, Sophia Antipolis
October 4, 2014

Contents

Part I
Analysis of Branch and Bound

NO OLVIDAR LOWER BOUNDS

1 Introduction

In this paper we present the analysis made of the results obtained and presented in the paper *Experimental Evaluation of a Branch and Bound Algorithm for computing Pathwidth* by the authors David Coudert, Dorian Mazauric and Nicolas Nisse. The program was developed mainly by David Coudert using Sage, a free open-source mathematics software system licensed under the GPL. Basically, given a graph $G = (V, E)$, the program calculates the pathwidth of G, $pw(G)$, within a time T. Depending on the G, it will be obtaining upper bounds before the final value of $pw(G)$. It also outputs the time used to obtain each one of these upper bounds. En example of this output will be given for a big graph.

We begin with a brief description of the graphs analysed. There are 8 types, mainly *Rome Graphs*, around 87% of all the graphs are this kind. We then analyse the first upper bound obtained, it will be called *First Stop*. The first upper bound value is obtained very fast, in average it takes approximately $0.002[s]$. It is also common to obtain the $pw(G)$ as an upper bound, before it ends checking the rest of the possibilities. The final analysis is about the total time needed to stop and obtain (or not) the pathwidth.

Adding results to this, we also present some lower bounds results. These are obtained in two different ways, the first is an implementation of brambles, an algorithm presented in **Treewidth Lower Bounds with Brambles** by *Bodlaender et. al.* The second way is a personal criteria of Esteban Román.

2 Results

In total we run the program on 12564 different graphs. These (most of them) can be related to eight different types of graphs, the number in parenthesis is the amount of each graph type. These eight types can be grouped as:

1. Test suites of directed and undirected graphs from the GDToolkit.

(http://www.dia.uniroma3.it/∼gdt/gdt4/index.php)

- Rome Graph (11534)

2. Extracted form the TreewidthLIB.

   (http://www.cs.uu.nl/research/projects/treewidthlib/)

   - TWL (212)
   - TWL_tsp (50)

3. Extracted form the VSPLIB.

   (http://www.optsicom.es/vsp/, 2012)

   - HB (69)
   - Tree (45)
   - Grids (50)

4. From SAGE

   - Sage Named (64)
   - Sage Families (463)

We have used max time equal to 200$[s]$, that means that if after this time the algorithm does not obtain the pathwidth for sure, it declares the last value found as an upper bound. Of all these 12564 graphs analysed, the algorithm obtains the $pw(G)$ 11732 times, and declare an upper bound value the other 832 cases.

As an example of the output, so it can be easier to understand, we show the first results on the graph $u2319.tsp$, it has $N = 2319$ nodes and $M = 6869$ edges. This example has not ended searching the $pw(G)$, and was stopped manually. It was not restricted to 200$[s]$.

```
SOL: 90 at time 0.22 and nodes 125
SOL: 89 at time 0.23 and nodes 139
SOL: 88 at time 0.24 and nodes 159
SOL: 87 at time 0.28 and nodes 252
SOL: 86 at time 0.33 and nodes 343
SOL: 85 at time 0.37 and nodes 434
SOL: 84 at time 0.42 and nodes 529
SOL: 83 at time 0.48 and nodes 661
```

```
SOL: 82 at time 0.55 and nodes 793
SOL: 81 at time 0.61 and nodes 929
SOL: 80 at time 0.7 and nodes 1102
SOL: 79 at time 0.78 and nodes 1275
SOL: 78 at time 0.86 and nodes 1450
SOL: 77 at time 0.94 and nodes 1585
SOL: 76 at time 1.01 and nodes 1722
SOL: 75 at time 1.08 and nodes 1857
SOL: 74 at time 1.19 and nodes 2045
SOL: 73 at time 1.3 and nodes 2219
SOL: 72 at time 1.4 and nodes 2386
SOL: 71 at time 1.52 and nodes 2617
SOL: 70 at time 1.64 and nodes 2830
SOL: 69 at time 1.81 and nodes 3144
SOL: 68 at time 1.95 and nodes 3370
SOL: 67 at time 2.14 and nodes 3654
SOL: 66 at time 2.31 and nodes 3921
SOL: 65 at time 13180.34 and nodes 24043895
SOL: 64 at time 19822.74 and nodes 35384943
SOL: 63 at time 20309.35 and nodes 36130610
SOL: 62 at time 20374.09 and nodes 36232475
SOL: 61 at time 179003.68 and nodes 319618255
SOL: 60 at time 309515.02 and nodes 556330573
SOL: 59 at time 440704.21 and nodes 783548003
SOL: 58 at time 558388.34 and nodes 996529598
SOL: 57 at time 660498.87 and nodes 1177339177
SOL: 56 at time 674868.09 and nodes 1201047441
SOL: 55 at time 680349.57 and nodes 1210597948
```

To interpret this *output*, we have three parameters:
`SOL:` P_1 `at time` P_2 `and nodes` P_3

- P_1: Is the actual upper bound.
- P_2: Is the total time required to obtain P_1.
- P_3: Is the number of checked nodes in the *search tree*.

It is not slow to obtain the first results, in fact, it obtains and improves the upper bound 25 times in just $2.31[s]$, but to obtain the next *better* upper bound, it requires $13180.34[s]$ (more than three hours), the last upper bound obtained required 680349.57[s] (189 hours or 7 days and 21 hours). Its necessary to take in to account that this is a big graph for this kind of analysis. In other graphs the last upper bound obtained is indeed the $pw(G)$ but it still have combinations to check.

2.1 First Stop

As we mentioned before, the first stop is very fast. On average it takes only $2.2393 \cdot 10^{-3}[s]$. In 3388 cases, the first upper bound obtained is in fact the pathwidth of the graph. This fact does not imply that the algorithm stops, it still has to verify that no *better* upper bound exists.

Sorting the time of the first stop we show these three plots. The difference between them is just the amount of graphs plotted so it can be easier to see. The *x-axis* represent the id of the graphs ordered by increasing time required for the first stop, meanwhile the *y-axis* is the time needed. The three plots shown in figure 1 differ in the amount of graphs used; the first one consider them all, the one in the middle consider the *fastest* 99%, and the third, consider the *fastest* 95%.

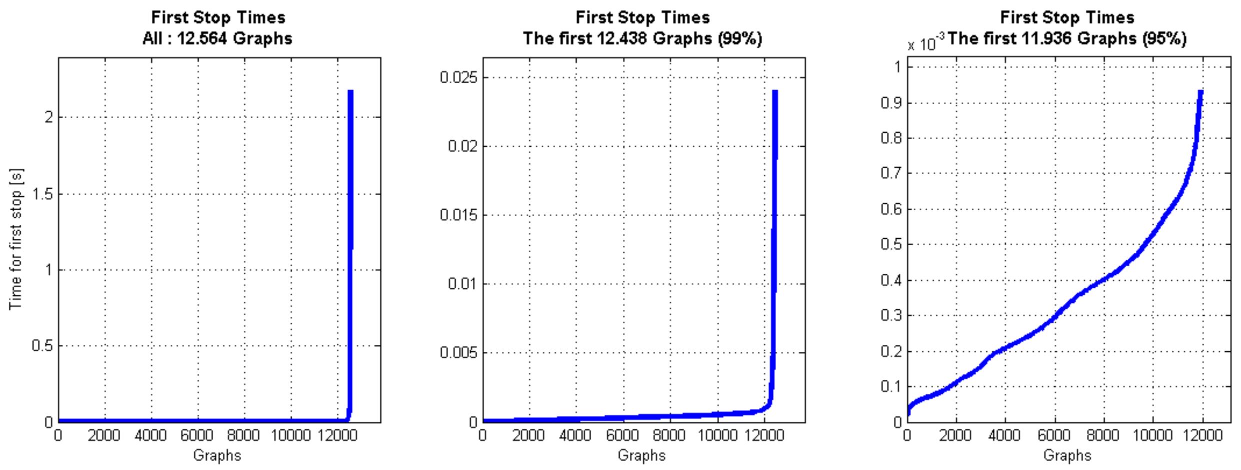

Figure 1: First stop times

As shown in figure 1, we see that 99% of all graphs need less than 0.025[s] and 95% need less than 0.001[s].

If we analyse the 3388 cases where the first stop gives as a result the $pw(G)$ and compare the time needed to stop (pw-stop) and the first stop, we can make a graph of the ratio between these two values. Ideally, we would like to have this ratios near 1, higher values indicate only lost time. The average value of this ratio is 1293, to understand better this average value, its helpful to know also the average time of both times, first stop and pw-stop. The average time for first stop is $2.1 \cdot 10^{-4}[s]$ and for the pw-stop is $6.8 \cdot 10^{-1}[s]$.

Figure 2 shows (in increasing order) all the ratios between pw-stop and first stop. As before, the *x-axis* represent the id of the graphs ordered by increasing ratio, and the *y-axis* is the ratio. We also show different amount of graphs used to facilitate understanding of the behaviour. The highest ratio is near 2×10^5 but more than 90% of these graphs have ratio lower than 10^2 and approximately 30% have ratio no more than 10^1.

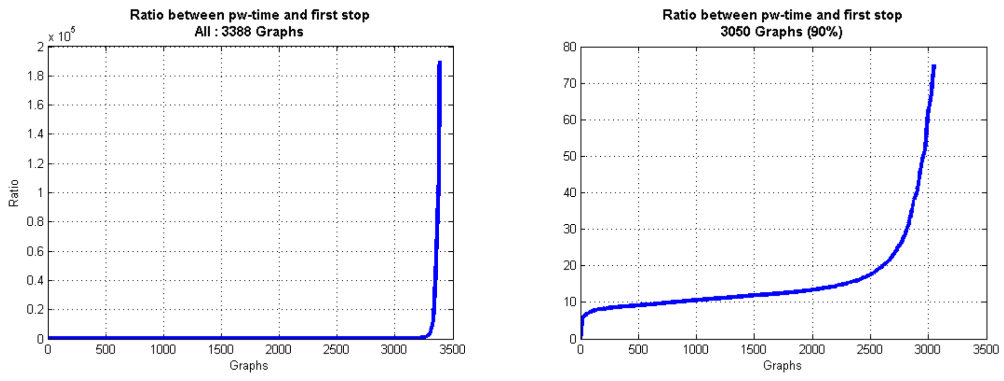

Figure 2: Ratio between stop time and first stop

2.2 Time to stop

Another way of seeing how the upper bounds converge to the pathwidth is studying the ratio of the upper bounds about the pathwidth. The main reason of doing this is to standardize the values obtained. Doing this standardization, it becomes easier to compare different graphs with different pathwidth.For example, lets suppose we have two graphs, G_1 and G_2, with $pw(G)$ 4 and 20 respectively. We use UB_1 as the first upper obtained, UB_2 as the second upper obtained and so on. As usual, the pathwidth is $pw(G)$. As an example, assume that the upper bounds obtained are shown in Table 1.

G_1			
UB_1	6	first stop	0.002
UB_2	5	t_2	0.003
$pw(G_1)$	4	pw-time	0.08

G_2			
UB_1	23	first stop	0.01
UB_2	22	t_2	0.02
UB_3	21	t_3	0.04
$pw(G_2)$	20	pw-time	0.1

Table 1: Example: Original graph data

Then, after the standardization, Table 1 would be looking as Table 2.

G_1			
$\widehat{UB}_1$	1.5	first stop	0.002
$\widehat{UB}_2$	1.25	t_2	0.003
$\widehat{pw(G_1)}$	1	pw-time	0.08

G_2			
$\widehat{UB}_1$	1.15	first stop	0.01
$\widehat{UB}_2$	1.1	t_2	0.02
$\widehat{UB}_3$	1.05	t_3	0.04
$\widehat{pw(G_2)}$	1	pw-time	0.1

Table 2: Example: Standardized graph data

We will show the results of convergence for three cases:

- Graphs such that the $pw(G)$ is obtained.

- Graphs such that the $pw(G)$ is not obtained.
- Graphs such that the first stop upper bound is greater than the $pw(G)$.

Generally spoken, we will obtain the average convergence value for each one of these cases and then more specifically, a similar graph showing convergence for each of the eight classes of graphs.

2.2.1 Graphs such that the $pw(G)$ is obtained.

This group consist of 11732 graphs. In average we can see that these graphs start ($t \sim 10^{-3}$) with an upper bound 30% superior to the $pw(G)$. After approximately $10^{-2}[s]$ the average upper bound is just 5% above the $pw(G)$. After $1[s]$ we have less than a 1% of difference. After 10 seconds, the relative difference is practically none.

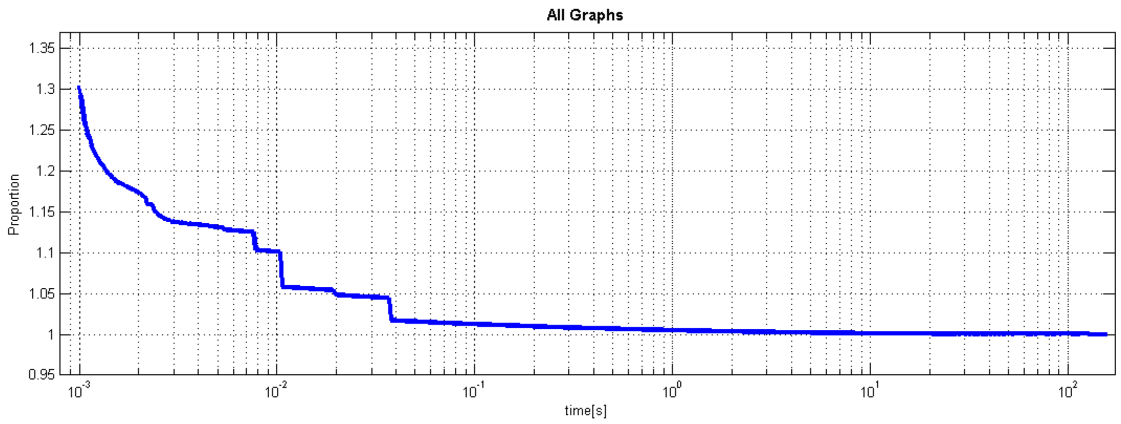

Figure 3: Average upper bound ratio

Figure 4 shows how this behaviour for each of the eight classes. For the graph class Rome, we see the curve *smoother*. In other cases the curve looks more *discrete*, its for the opposite reason, few graphs. The **Tree** graphs are the ones with worst initial upper bound, estimating over 400% the $pw(G)$ but get near the $pw(G)$ very fast.

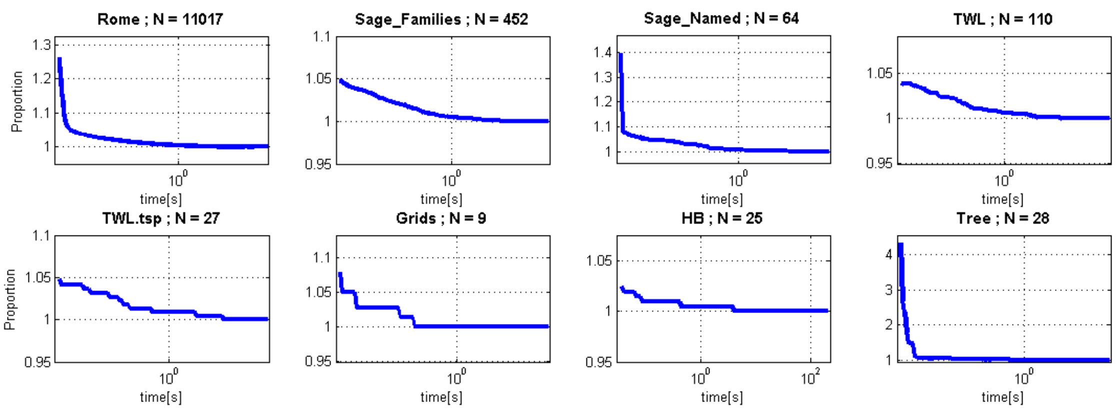

Figure 4: Average upper bound ratio per class

2.2.2 Graphs such that the $pw(G)$ is not obtained.

This group consist of 832 graphs. In average we can see that these graphs start ($t \sim 2 \cdot 10^{-1}$) with an upper bound 50% superior to the lowest upper bound obtained. Comparing with graphs that reach theirs pathwidth, at this time, (figure 3) they were in average only approximately 2% over the $pw(G)$. After 10 seconds, the difference is around the 1%. As this graphs did not reach the $pw(G)$ its clear that the convergence (to the last upper bound) appears at the end time, $200[s]$.

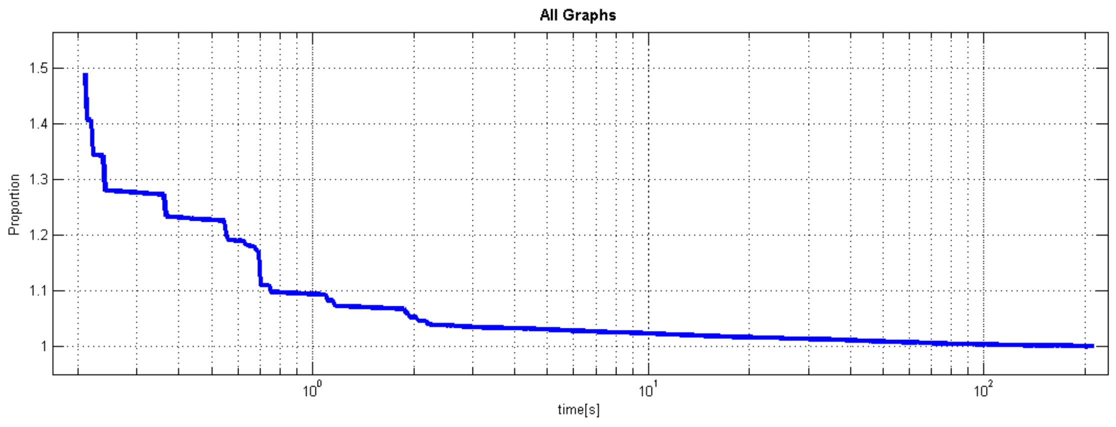

Figure 5: Average upper bound ratio

Plotting by class, we do not see big differences between classes.

The difference between these eight and figure 5 relating the first upper bound is due to the time partition used.

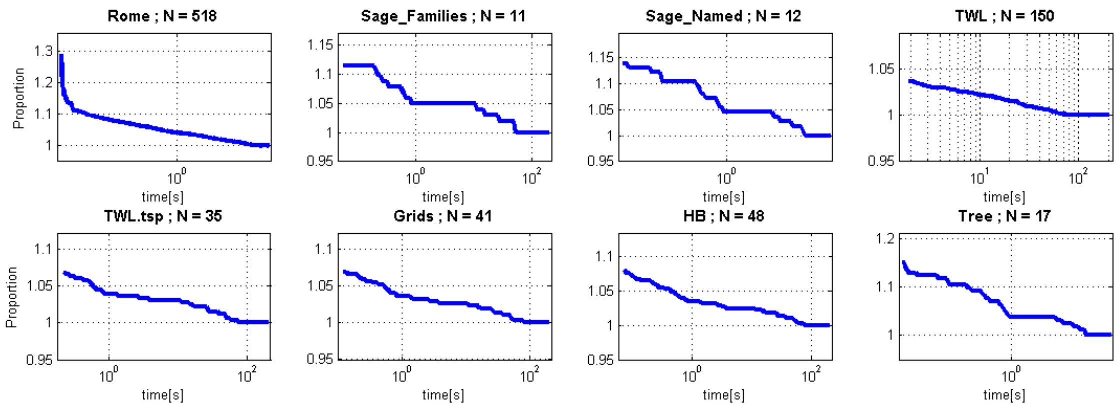

Figure 6: Average upper bound ratio per class

2.2.3 Graphs such that the first stop is greater than the $pw(G)$.

This group consist of 8344 graphs. They all reach the pathwidth. At a similar starting time as the bigger group of all the graphs that reaches the pathwidth these start with a lower upper bound ratio, something that may not make sense at a first look.

Possible reasons because this starting average is less than (2.2.1) may be:

- Time scaling.
- Not necessary these graphs start with a lower proportional upper bound, they may be bigger maybe.

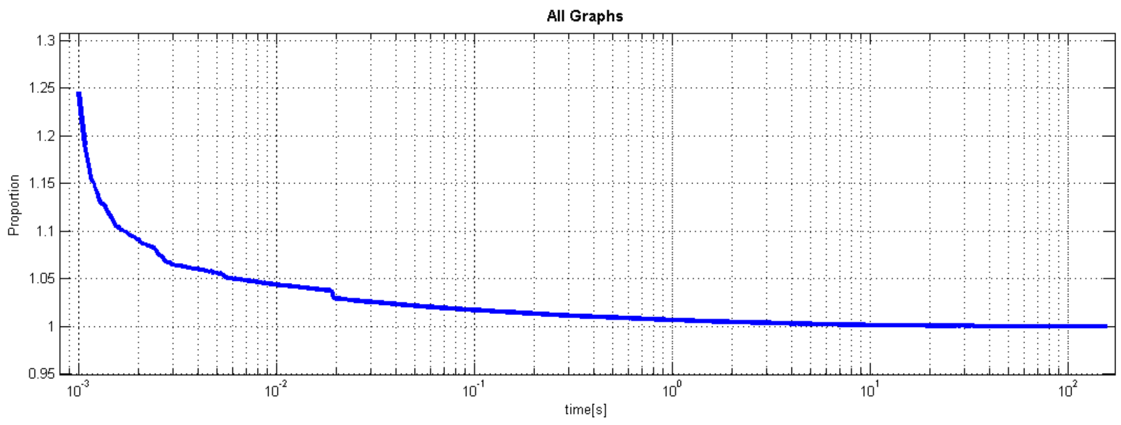

Figure 7: Average upper bound ratio

The missing plot is because there is no *TWL.tsp* graph satisfying the conditions.

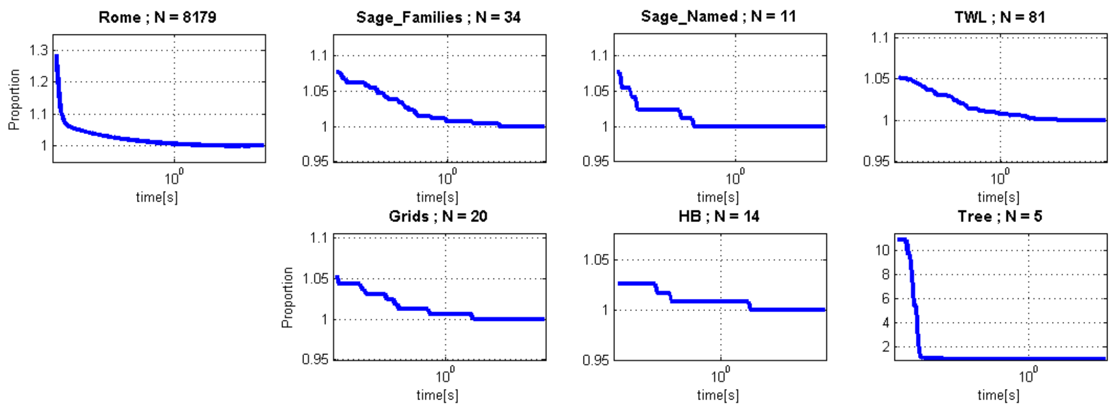

Figure 8: Average upper bound ratio per class

3 Matlab Codes

3.1 working

```
function working(dataN)
    data = dataN;
    data(:,3)=[];
    [a, b]=size(data);
    n=8;    % Different types of graphs
    % C indicates where each type begins
    C=ones(1,n+1);
    for i=2:n
        if not(isempty(find(data(:,1)==i, 1 )))
            C(i)=find(data(:,1)==i, 1 );
        else
                C(i)=C(i-1);
        end
    end
    C(n+1)=a+1;
    % L indicates "amount" of iterations of each Graph
    L=zeros(a,1);
    for i=1:a
        if isempty(find(isnan(data(i,:)), 1 ))
            L(i)=b;
        else
            L(i)=find(isnan(data(i,:)), 1 )-1;
        end
    end
    %  1         2        3   4             5   6        7
              8
    %  Type      Graph    pw         FinalTime        N
               M - [     UB -      time  ]
%      Type=data(:,1);
%      Graph=data(:,2);
%      PW=data(:,3);
%      FT=data(:,4);
    N=data(:,5);
%      M=data(:,6);
    Lmax=max(L);
    UB=data(:,7:2:Lmax);
    UB=[N UB];
    time=data(:,8:2:Lmax);
    clear data
    time=[zeros(a,1)  time];
    LL=(L-6)/2;
    LLmax=max(LL);
    for i=1:a
        if LL(i)<LLmax
            UB(i,LL(i)+2:end)=UB(i,LL(i)+1);
            time(i,LL(i)+2:end)=time(i,LL(i)+1);
        end
        UBR(i,:)=UB(i,:)/min(UB(i,:));
    end
    %%
    minTsort=sort(time(:,2));
    maxTsort=sort(time(:,end));
    MinT= minTsort(round(0.98*a));
    MaxT=max(maxTsort)+10;
    N=max(100,min(a,500));
    LOG=3;
    if LOG==3
        TIME=linspace(log10(MinT),log10(MaxT),N+1);
        TIMEUSE=10.^TIME;
    elseif LOG==2
        TIME=linspace(log2(MinT),log2(MaxT),N+1);
        TIMEUSE=2.^TIME;
    elseif LOG==1
        TIME=linspace(log(MinT),log(MaxT),N+1);
        TIMEUSE=exp(TIME);
    end

        VAL=zeros(a,N+1);
    for i=1:a
        for j=1:N+1
            if not(isempty(find(time(i,:)<=TIMEUSE(j),
                1, 'last' )))
                VAL(i,j)=UBR(i,find(time(i,:)<=TIMEUSE
                    (j), 1, 'last' ) );
            end
        end
    end

    %% Means
    types{1}='Rome';types{2}='Sage\_Families';types
        {3}='Sage\_Named';types{4}='TWL';
    types{5}='TWL.tsp';types{6}='Grids';types{7}='HB';
        types{8}='Tree';
    VALMEAN=mean(VAL);
    figure
    semilogx(TIMEUSE,VALMEAN, 'LineWidth', 2)
    axis([0.8*min(TIMEUSE) 1.1*max(TIMEUSE) 0.95 1.05*
        max(VALMEAN)]);
    grid on;
    title('All Graphs');
    xlabel('time[s]');

    %% By type
    for i =1:n%length(unique(data(:,1)))
        UBDATA{i}=UBR(C(i):C(i+1)-1,:);
        UBDATA{i}(:,max(LL(C(i):C(i+1)-1))+1:end)=[];
        timeDATA{i}=time(C(i):C(i+1)-1,:);
        timeDATA{i}(:,max(LL(C(i):C(i+1)-1))+1:end)
```

```
            =[];
        sizes(i,:)=size(UBDATA{i});
    end
    %%
    figure
    for i=1:n
        clear minTsort maxTsort VAL
        a=C(i+1)-C(i);
        if a>0
            minTsort=sort(time(C(i):C(i+1)-1,2));
    %       maxTsort=sort(time(C(i):C(i+1)-1,end));
            MinT= minTsort(round(0.98*a));
    %       MaxT=max(maxTsort)+10;
            N=max(100,min(a,500));
            LOG=3;
            if LOG==3
                TIME=linspace(log10(MinT),log10(200),N
                    +1);
                TIMEUSE=10.^TIME;
            elseif LOG==2
                TIME=linspace(log2(MinT),log2(200),N
                    +1);
                TIMEUSE=2.^TIME;
            elseif LOG==1
                TIME=linspace(log(MinT),log(200),N+1);
                TIMEUSE=exp(TIME);
            end
            VAL=zeros(a,N+1);
            for k=1:a
                for j=1:N+1
                    if not(isempty(find(time(k,:)<=
                        TIMEUSE(j), 1, 'last')))
                        VAL(k,j)=UBR(k,find(time(k,:)
                            <=TIMEUSE(j), 1, 'last'))
                            ;
                    end
                end
            end
            VALMEAN=mean(VAL);
```

```
                  subplot(2,4,i)
                  semilogx(TIMEUSE,VALMEAN, 'LineWidth', 2)
                  axis([0.8*min(TIMEUSE) 1.1*max(TIMEUSE)
                      0.95 1.05*max(VALMEAN)]);
                  grid on;
                  title({types{i};['N = ',num2str(a)]});
                  xlabel('time[s]');
            end
      end
end
```

3.2 Study

```
function [X, T] = Study(data)
    %    1          2          3     4     5               6     7               8          9
    %    Type       Graph      OPT         pw              FinalTime             N          M - [    UB -    time ]
    DATA = data;
    [a, b]    = size(DATA);
    UB=DATA(:,8:2:b);
    Time=DATA(:,9:2:b);
    Sol_1     = DATA(:,8);      % First solution
    Time_1    = DATA(:,9);      % First solution Time
    OptVal    = DATA(:,4);      % Best Value obtained (pw or UB)
    T2Stop    = DATA(:,5);      % Total time to stop
    for i=1:a                   % Time needed to compute OptVal
        T2gPW(i,1)   = DATA(i,7+2*find(DATA(i,8:2:end)==OptVal(i),1));
    end
    %% Some Basic Analysis
    %Percent_Succes_First_Solution=X(8);
    X(1)=sum(T2Stop);
    X(2)=round(X(1)/60);
    X(3)=round(X(1)/3600);
    X(4)=sum(T2Stop-T2gPW);
    X(5)=round(X(4)/60);
    X(6)=round(X(4)/3600);
    X(7)=100*X(4)/X(1);
    Total_Time_Used_Seconds=X(1);
    Total_Time_Used_Minutes=X(2);
    Total_Time_Used_Hours=X(3);
    Total_Useless_Time_Seconds=X(4);
    Total_Useless_Time_Minutes=X(5);
    Total_Useless_Time_Hours=X(6);
    Percent_Useless_Time=X(7);
    %% Other
    T=zeros(size(UB,2),2);
```

```
        for i = 1:size(UB,2)
            T(i,1) = sum(UB(:,i)==OptVal);
            T(i,2) = 100*T(i,1)/a;
%            YesPos  = find(Sol_1==OptVal);
%            X(8)=100*length(YesPos)/a;
        end
end
```

Part II

The Smoking Robber

4 Introduction

This new problem is similar to the typical $Cop - Robber$ problem, but with the difference that here the cops does not catch the robber. They just try to be close enough to the robber, closer than a certain distance. We can assume for simplicity that the robber is a smoker and wants to smoke, but he cannot smoke if there is another cop near him. So the cops try to be near this smoker as much as they can, so the robber do not smoke. They all live on a ring of length N, and the robber is faster than the cops.

First we will analyze the case where there are 3 cops, one robber and the robber is twice as fast as the cops; $v_r = 2v_c$. Then we will generalize this problem to M cops and a velocity ratio given by $v_r = \alpha v_c$.

5 Definitions

We will define some concepts, so it is easier later to understand the explanations.

Control Distance (v): Distance from the cop where the robber can not smoke.

Control Zone: Place near the cops where the robber cannot smoke. It is defined by the control distance.

Freedom: Place outside the *Control Zone*.

Clock Ordering: A clock ordering (x, y, z) is a clockwise ordering that means that y is in between x and z. A clock ordering (x, y, z, w) means that we have (y, z, w) and (x, y, z).

Go By: We say that a robber r goes by a cop c if there are times ($t_1 < t_2 < t_3$ such that the clock ordering are:

- (r, c) at $t = t_1$ (or (c, r)).
- r and c share position at $t = t_2$.
- (c, r) at $t = t_3$ (or (r, c)).

(we assume that the robber does not change move direction)

Pass By: We say that a robber r pass by a cop c if he manages to *go by* the cop and find *Freedom* immediately after.

Guard: We say that c_2 guards c_1 if given a clock ordering (r, c_1, c_2), the robber r cannot pass by c_1 because of c_2.

Guard Distance Given a clock ordering (r, c_1, c_2), the guard distance is the maximum distance between c_1 and c_2 such that c_2 guards c_1. This distance will be a function of:

- the proportional velocity of the robber,
- the distance $d(r, c_1)$.

6 Basic Case

As we said, the basic case consists of one robber and three cops. The robbers is twice as fast as the cops. We have two ways of analyzing this case. The first one, our *Basic Case*, and easiest, is when the robber chooses a place first, and later so do the cops. Given the initial position of the robber, the three cops can choose the places given by the left image in Figure 9. The marks on the ring, equals the *Control Distance.* The ratio of the ring and the *control distance* needed so the cops can avoid that the robber never smokes is 14 : 1. We see that, if the robber tries to *pass by* cop C_1 and runs clockwise, the cop, C_3, standing in a opposite initial place to the robber, C_3, can stop the robber to smoke.

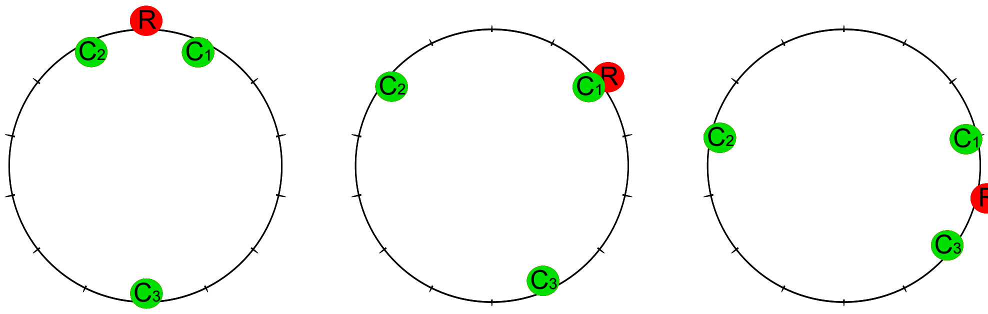

Figure 9: Basic Case

This is an example for C_3 *guards* C_1. In this case the *Guard Distance* is equal 6 times the *control distance.* To calculate this *Guard Distance*, we can use as help Figure 10, where the robber is red and the two cops are green. It shows two different times, the first, the robber is at a distance R to his nearest cop, and the distance, we want to find, between the two cops is D. The second shows the exact moment when the robber is in the limit of both *control zones*.

The *control zone* is painted in yellow, and its size equals two v, where v is *control distance*. So, assuming that both the robber and the left cop move to the right, and the right cop moves to the left, and that the robber is α times faster than the cops, then to find D (*Guard Distance*) as a function of α and R, we will have:

$$\begin{aligned} R + \frac{D}{2} &= \alpha \cdot \left(\frac{D}{2} - v\right) \\ R + \alpha \cdot v &= \frac{D}{2}(\alpha - 1) \\ 2\left(\frac{R + \alpha \cdot v}{\alpha - 1}\right) &= D \end{aligned}$$

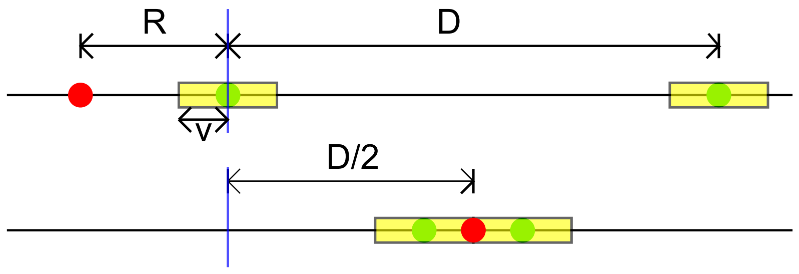

Figure 10: Basic Case

In our basic case, $\alpha = 2$, so we finally get

$$D = 2R + 4v \tag{1}$$

If we see Figure 9, the *Basic Case*, we have that $R = v$, so the *guard distance* should be $6v$, which it is. So C_3 guards both C_1 and C_2.

The second way to analyze this problem is when the cops choose initial place first. To simplify a little bit the numbers, we will assume the length of the ring $N = 42$, $v = 3$ ($N/v = 14$) and $\alpha = 2$. We assume that the initial position for the cops are equidistant from each other. and that the robber choose an initial position at a distance d to the left from the center of two of the cops. Its illustrated in figure 11. Without loss of generality , we assume the robber moves to the right (clockwise). It is clear this is not anymore the basic case. The basic case is all ready solved. Studying the basic case, with $N = 42$, $v = 3$, the positions of the four players can be given by:

$$\begin{aligned} R(t) &= 2t \\ C_1(t) &= 3 + t \\ C_2(t) &= -3 - t \\ C_3(t) &= 21 - t \end{aligned}$$

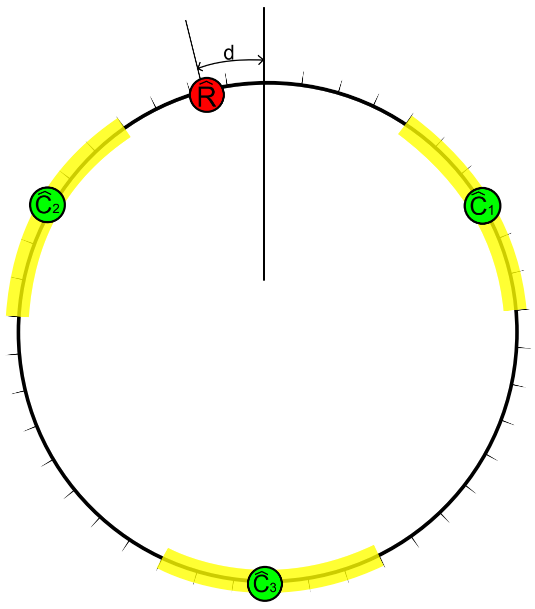

Figure 11: Basic Case

To solve the case when the three cops choose initial place first, what we will do is obtain the same positions, for some $\tau = \max(\tau_2, \tau_3)$. We will be *synchronizing* the the positions, one by one, at the times $\tau_1 \leq \tau_2, \tau_3$ such that after each τ_i one of the cops will be moving with the robber in the same way he does in the basic case. As this cops, will be *synchronizing*, they're movement equation will change in time. At time zero (we use τ as time for this case) these four equation for the new case are:

$$\begin{aligned} \widehat{R}(\tau) &= -d + 2\tau \\ \widehat{C_1}(\tau) &= 7 - \tau \end{aligned}$$

$$\begin{aligned}\widehat{C_2}(\tau) &= -7+\tau \\ \widehat{C_3}(\tau) &= 21-\tau\end{aligned}$$

The distance between $\widehat{C_1}$ and $\widehat{R}$ is $dist(\widehat{C_1},\widehat{R}) = 7+d-3\tau$, and if $\widehat{C_1}$ change its direction when this distance equals 3, which is the *control distance*, then we find τ_1:

$$\tau_1 = \frac{4}{3}+\frac{d}{3} \tag{2}$$

After $\tau \geq \tau_1$ we can say that $\widehat{C_1}$ is synchronized and fix it position equation to:

$$\widehat{C_1}(\tau) = \begin{cases} 7-\tau & ; \quad \tau \leq \tau_1 \\ \frac{17}{3}-\frac{d}{3}+\tau & ; \quad \tau > \tau_1 \end{cases} \tag{3}$$

To find $\tau_2 \geq \tau_1$ and fix $\widehat{C_2}$, we'll need two equations and two variables. We will match distances:

$$\begin{aligned} C_1(t) - R(t) &= \widehat{C_1}(\tau) - \widehat{R}(\tau) \\ R(t) - C_2(t) &= \widehat{R}(\tau) - \widehat{C_2}(\tau) \end{aligned}$$

This two equations become:

$$\begin{aligned} 3 - t &= \frac{17}{3} + \frac{2d}{3} - \tau \\ 3 + 3t &= 7 - d + \tau \end{aligned}$$

The solution for τ gives us

$$\tau_2 = 6 + \frac{d}{2} \tag{4}$$

and we fix $\widehat{C_2}$:

$$\widehat{C_2}(\tau) = \begin{cases} -7 + \tau & ; \quad \tau \leq \tau_2 \\ -1 + \frac{d}{2} - \tau & ; \quad \tau > \tau_2 \end{cases} \tag{5}$$

.

So, for $\tau \geq \tau_2$, we will have $\widehat{C_1}$ and $\widehat{C_2}$ synchronized. To find τ_3, we want that the distance between $\widehat{C_2}$ and $\widehat{C_3}$ is 6. $\widehat{C_3}(\tau) - \widehat{C_1}(\tau) = 6$:

$$21 - \tau - \left(\frac{17}{3} - \frac{d}{3} + \tau\right) = 6$$

we obtain:

$$\tau_3 = \frac{23}{3} + \frac{d}{6} \tag{6}$$

and:

$$\widehat{C_3}(\tau) = \begin{cases} 21 - \tau & ; \quad \tau \leq \tau_3 \\ \frac{40}{3} - \frac{d}{6} + \tau & ; \quad \tau > \tau_3 \end{cases} \tag{7}$$

.

Now the three cops are synchronized with the robber.

7 Generalization

7.1 Speed

Using equation the reasoning to obtain equation (1) for the guard distance $D = 2R + 4v$, and using the initial configuration for the basic case, and using $R = v$ we have the value:

$$D = 2v\left(\frac{p+1}{p-1}\right)$$

And given that the total length of the ring is $L = 2D + 2v$, we obtain, the length as a function of the proportional speed as:

$$L = 2v\left(\frac{3p+1}{p-1}\right)$$

and the ratio between this length and the control distance as:

$$ratio(L/cd) = 2\left(\frac{3p+1}{p-1}\right)$$

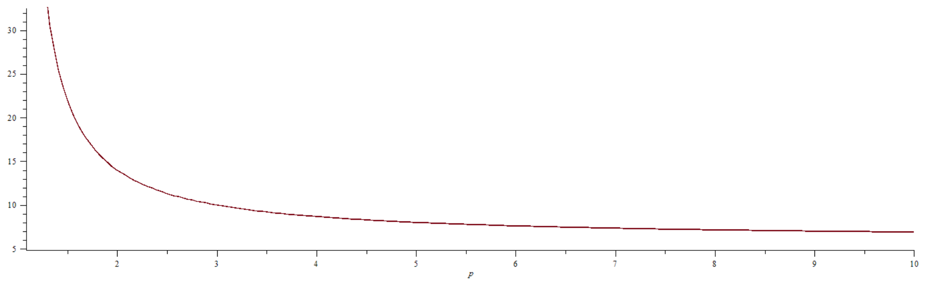

Figure 12: Ratio between this length and the control distance

Clearly, this ratio converges to 6 when the robber is much faster.

7.2 Amount of Cops

Given N cops, to solve this problem, we can make it the same as if there where only 3 cops.the nearest two cops, one per each side, represent C_1 and C_2 from before. The $N-2$ rest of cops, that are not *neighbor* with the robber, act as one. We fix the distance between them, so they move all together, doing this, we can see the problem equal as one where we join together all of these $N-2$ cops.

In this case we have:

$$L = (N-1)D + 2v = 2v\left(\frac{N(p+1)-2}{p-1}\right)$$

and the ratio:

$$ratio(L/cd) = 2\left(\frac{N(p+1)-2}{p-1}\right)$$

We see that for big values of α, this ratio converges to $2N$, which is logic.