

Cops and visible robbers: a computational approach

Ath. Kehagias and P. Pralat

9/6/2011

Abstract

We consider the cops and robber game, in which one or more cops move on a graph G with the goal of capturing a robber. The robber may be *adversarial* (he moves so as to avoid capture) or *drunk* (he performs a random walk on G). Our goal is to estimate the *cost of drunkenness* (COD), i.e., the ratio of capture times for the adversarial and drunk robber. We briefly review some theoretically obtained bounds on COD and then concentrate on the CAAR and CADR algorithms for COD computation. We have implemented the algorithms as a (publicly available) *Matlab package* which we briefly present and then use to experimentally evaluate the algorithms.

1 Introduction

The *Cops and Robbers* (CR) game [8, 10] is played on an undirected graph G , where k cops and a robber move in rounds, according to the following rules.

1. In the initial (placement) round the cops choose their initial positions (more than one cop can be in the same node) and then the robber chooses his position.
2. In each of the following rounds, first the cops and then the robber move to nodes neighboring their current positions. Staying in place is also allowed.

It is assumed that the robber is *visible* to the cops, i.e., they always know his position; similarly, the robber always knows the cops' positions.

The cops' goal is to “capture” the robber, i.e., to reach a position where at least one cop occupies the same node as the robber. It is assumed that the cops will always play optimally. In the “classical” version of the CR game, the robber is *adversarial*, i.e., his goal is to avoid capture and he plays optimally with respect to this end. Given a large enough k the cops can win on any graph (for example, if one cop occupies each node). The minimum k required to guarantee that the cops win on G is called the *cop number* of G .

We are also interested in a variant of the CR game in which the robber is “*drunk*”, i.e., he is not interested in avoiding capture and simply performs a random walk on G . The cops are still assumed to play optimally. In [6] we have studied this game and especially the “*cost of drunkenness*”, i.e., the ratio of capture times for the adversarial and drunk robber. Our approach in [6] is mainly theoretical, but we also present algorithms which can be used to compute the cost of drunkenness.

In this technical report we discuss these algorithms in more detail, present their Matlab implementation (the package **CopsRobber**, available at <http://users.auth.gr/~kehagiat/GraphSearch/>) and evaluate them by a series of computer experiments. The report is structured as follows. In Section 2 we present definitions and notation. In Section 3 we show how to compute capture time given a *predetermined* cop schedule (independent of the robber's position). In Section 4 we present the CAAR and CADR algorithms; these *iterative* algorithms compute capture time and optimal cop strategy (against an adversarial and a drunk robber respectively). In Section 5 we present the Matlab implementation of the algorithms. In Section 6 we evaluate CAAR and CADR with computer experiments. In Section 7 we summarize and list possible future research directions.

2 Preliminaries

Let $G = (V, E)$ be a fixed undirected, simple, finite graph without loops. We also assume that G is connected (since the game played on a disconnected graph can be analyzed by investigating each component separately). We will use the following notation and assumptions.

1. $N(v)$ denotes the neighborhood of node $v \in V$ and $N^+(v) = N(v) \cup \{v\}$ (the *closed* neighborhood of v).
2. We define $\hat{V}^2 = V \times V - \{(x, x) : x \in V\}$, i.e., V^2 excluding the diagonal.
3. The minimum number of cops required to capture the adversarial robber is called the *cop number* of G and is denoted by $c(G)$.
4. Given k cops, X_t^i denotes the position of the i -th cop at time t ($i \in \{1, 2, \dots, k\}$, $t \in \{0, 1, 2, \dots\}$); $X_t = (X_t^1, \dots, X_t^k)$ denotes the vector of all cop positions at time t ; $\mathbf{X} = (X_0, X_1, X_2, \dots)$ denotes the positions of all cops during the game (\mathbf{X} may have finite or infinite length).
5. Y_t denotes the position of the robber at time t and $\mathbf{Y} = (Y_0, Y_1, Y_2, \dots)$ the positions of the robber during the game.
6. The moving sequence is as follows: first the cops choose initial positions X_0 , then the robber chooses Y_0 . For the following rounds ($t \in \{1, 2, \dots\}$) first the cops choose X_t and then the robber chooses Y_t .
7. The *capture time* is denoted by T and defined as follows

$$T = \min\{t : \exists i \text{ such that } X_t^i = Y_t\},$$

i.e., it is the first time a cop is located at the same vertex as the robber (note that this can happen either after the cops move or after the evader moves). When $k \geq c(G)$ we will have $T < \infty$ (since $c(G)$ cops can capture the adversarial robber they can also capture the drunk one).

Assume for the moment that both the cops and the robber are adversarial. Given initial cop positions $x \in V^k$ and robber position $y \in V$, we let $\text{ct}_{x,y}(G, k) = T$. The k -*capture time* is defined as follows:

$$\text{ct}(G, k) = \min_{x \in V^k} \max_{y \in V} \text{ct}_{x,y}(G, k).$$

In other words, the players choose their initial positions so as to achieve the best outcome. Finally, when $k = c(G)$ we simply write $\text{ct}(G)$ instead of $\text{ct}(G, c(G))$, and call it the *capture time* instead of $c(G)$ -capture time. Let us stress one more time that the above quantities are defined under the assumption of *optimal play by both players*.

Now let us assume that the cops are adversarial but the robber is *drunk*. More specifically, we assume the robber performs a random walk on G , i.e., given that he is at vertex $v \in V$ at time t , he moves to $u \in N(v)$ at time $t + 1$ with probability equal to $1/|N(v)|$. Note that we do *not* include v in $N(v)$ and that the robber probability distribution does not depend on the current cops position (in particular, it can happen that the robber moves to a vertex occupied by a cop; something the adversarial robber would never do).

Under the above assumptions, the drunk robber game actually is a one-player game. Also, for a given initial configuration and for a given strategy of cops, the capture time T is a random variable. Let

$$\text{dct}_{x,y}(G, k) = \mathbb{E}(T \mid X_0 = x, Y_0 = y, k \text{ cops are used optimally}),$$

in other words, $\text{dct}_{x,y}(G, k)$ is the expected capture time given initial cop and robber configurations x, y and optimal play by the k cops. The drunk robber chooses an initial vertex randomly with uniform probability.

Cops are aware of this and so they choose an initial configuration that minimizes the expected game length. Hence, we define the expected k -capture time as follows:

$$\text{dct}(G, k) = \min_{x \in V^k} \sum_{y \in V} \frac{\text{dct}_{x,y}(G, k)}{|V|}.$$

As before, $\text{dct}(G) = \text{dct}(G, c(G))$.

We define the *cost of drunkenness* as follows

$$F(G) = \frac{\text{ct}(G)}{\text{dct}(G)};$$

we obviously have $F(G) \geq 1$.

In this report we concentrate on the case $k \geq c(G)$. However let us note that even a single cop can catch the drunk robber: in [6] we have shown that $\text{dct}(G, k) < \infty$ for any connected graph G and $k \geq 1$ (even for $k < c(G)$).

Let us also present without proof our main results from [6]. The next theorem shows that, in general, the cost of drunkenness can be any number $c \in (1, \infty)$.

Theorem 2.1 *For every real constant $c \geq 1$, there exists a sequence of graphs (G_m) such that*

$$\lim_{m \rightarrow \infty} F(G_m) = \lim_{m \rightarrow \infty} \frac{\text{ct}(G_m)}{\text{dct}(G_m)} = c.$$

However the following theorems show that, for particular graph families, sharper bounds can be obtained.

Theorem 2.2 *Let P_n denote the path with n nodes. Then*

$$\frac{n}{4} \cdot \left(1 - O\left(\frac{\log n}{n}\right)\right) \leq \text{dct}(P_n) \leq \frac{n}{4};$$

in particular, $\text{dct}(P_n) = (1 + o(1))n/4$. The cost of drunkenness is

$$F(P_n) = \frac{\text{ct}(P_n)}{\text{dct}(P_n)} = 2 + o(1).$$

Theorem 2.3 *Let C_n denote the cycle with n nodes. Then*

$$\frac{n}{8} \cdot \left(1 - O\left(\frac{\log n}{n}\right)\right) \leq \text{dct}(C_n) \leq \frac{n+1}{8};$$

in particular, $\text{dct}(C_n) = (1 + o(1))n/8$. The cost of drunkenness is

$$F(C_n) = \frac{\text{ct}(C_n)}{\text{dct}(C_n)} = 2 + o(1).$$

Theorem 2.4 *Let $T_{d,k}$ denote the d -regular rooted tree of depth k . Then*

$$k - O(\sqrt{k \log k}) \leq \text{dct}(T_{d,k}) \leq k;$$

in particular, $\text{dct}(T_{d,k}) = (1 + o(1))k$. The cost of drunkenness is

$$F(T_{d,k}) = \frac{\text{ct}(T_{d,k})}{\text{dct}(T_{d,k})} = 1 + o(1).$$

Theorem 2.5 *Let Γ_m denote the $m \times m$ grid (it is $\Gamma_m = P_m \square P_m$, the product of the n -node-path with itself). Then $\text{dct}(\Gamma_m) = (1 + o(1))\frac{3}{8}m$. The cost of drunkenness is*

$$F(P_m \square P_m) = \frac{\text{ct}(P_m \square P_m)}{\text{dct}(P_m \square P_m)} = 8/3 + o(1).$$

The above results notwithstanding, sharp bounds on $F(G)$ for an arbitrary G cannot be easily obtained. Hence in the following sections we turn to computational approaches to the problem.

3 Computing the expected capture time for a fixed cop strategy

Suppose that, for a given a graph G , the cops fix a strategy before the game starts, i.e., the cop moves are determined before any robber moves are observed. We will now show how to explicitly compute the probability of capture at time $t \in \{0, 1, 2, \dots\}$ as well as the expected capture time.

Before proceeding, let us note that the cops would do better if they adjusted their moves depending on the robber moves; this will be treated in the Section 4. However, the approach presented here is less demanding computationally and can be used to provide an upper bound for the optimal expected capture time.

Let $G = (V, E)$ have n nodes: $V = \{0, 1, \dots, n-1\}$. Letting $P_{i,j} = \Pr(Y_t = j | Y_{t-1} = i)$ we have

$$P_{i,j} = \begin{cases} \frac{1}{|N(i)|} & \text{for } j \in N(i) \\ 0 & \text{otherwise.} \end{cases}$$

P is the $n \times n$ transition probability matrix governing the robber's random walk in G in the absence of cops. To account for capture by the cops, define a new state space $\bar{V} = V \cup \{n\}$, i.e., the old state space augmented by the capture state n . The corresponding $(n+1) \times (n+1)$ transition matrix is

$$\bar{P} = \begin{pmatrix} P & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix}.$$

In the absence of cops, the robber performs a standard random walk on G and never enters the capture state; if however he somehow gets captured, he remains so ad infinitum: $\bar{P}_{n,n} = 1$. In other words, the Markov chain governed by \bar{P} contains two noncommunicating equivalence classes: $\{0, 1, \dots, n-1\}$ and $\{n\}$.

Suppose now that a single cop is located in vertex x . We will denote the corresponding transition probability matrix by $\bar{P}(x)$. Obviously, $\bar{P}(x) \neq \bar{P}$. The difference is caused by the possibility of capture. This can occur in two ways.

1. The robber is located at y and, in the first phase of the t -th round, the cop moves into $x = y$. Then the robber is captured w.p.1. So, $\bar{P}_{x,n}(x) = 1$, $\bar{P}_{x,y'}(x) = 0$ for $y' < n$.
2. The robber is located at $y \neq x$ and, in the second phase of the t -th round, he moves from y to $y' = x$. Hence the robber is captured w.p. $P_{y,y'}$. So, for all $y \in V - \{x\}$, $\bar{P}_{y,n}(x) = P_{y,x}$, $\bar{P}_{y,x'}(x) = 0$.

We can summarize the above by writing

$$\bar{P}(x) = \begin{pmatrix} P(x) & \mathbf{p}(x) \\ \mathbf{0} & 1 \end{pmatrix}$$

where $P(x)$ has 0's in the x -th row and column and the corresponding probabilities have been moved into the $\mathbf{p}(x)$ vector. For example, let G be the path with 5 nodes; we list below \bar{P} and $\bar{P}(2)$:

$$\bar{P} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \bar{P}(2) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Especially for the placement round of the game ($t = 0$) we need a different matrix, because the robber does not random-walk, but simply chooses an initial position; if he chooses the one already occupied by the cop, then he is captured. Hence, for this round the appropriate transition matrix is $\hat{P}(x)$, which is the unit matrix with the one of the x -th row moved to the $(n+1)$ -th column.

Let $\pi(t) = [\pi_0(t), \dots, \pi_n(t)]$, where $\pi_i(t) = \mathbb{P}(Y_t = i)$ for $i \in \bar{V}$ and $t \in \{0, 1, \dots, s\}$. Given a strategy $\mathbf{X} = (x_0, x_1, \dots, x_s)$ and $\hat{\pi}(0) = [\frac{1}{n}, \dots, \frac{1}{n}, 0]$, the above formulation yields

$$\pi(0) = \hat{\pi}(0) \hat{P}(x_0) \quad \text{and, for } t \in \{1, 2, \dots\}, \pi(t) = \pi(t-1) \bar{P}(x_t)$$

which also gives $\pi(t) = \hat{\pi}(0) \hat{P}(x_0) \bar{P}(x_1) \dots \bar{P}(x_t)$. To illustrate, let us continue the previous example. Suppose a single cop enters the path and follows the strategy $\mathbf{X} = (0, 1, 2, 3, 4)$ (start on the left end of the path and move to the right). Then we have

$$\begin{aligned} \pi(0) = \hat{\pi}(0) \hat{P}(0) &= \begin{bmatrix} \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \end{bmatrix}, \\ \pi(1) = \pi(0) \bar{P}(1) &= \begin{bmatrix} 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \frac{1}{10} & \frac{3}{10} & \frac{1}{10} & \frac{1}{2} \end{bmatrix}, \\ \pi(2) = \pi(1) \bar{P}(2) &= \begin{bmatrix} 0 & 0 & \frac{1}{10} & \frac{3}{10} & \frac{1}{10} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \frac{1}{10} & \frac{3}{20} & \frac{3}{4} \end{bmatrix}, \\ \pi(3) = \pi(2) \bar{P}(3) &= \begin{bmatrix} 0 & 0 & 0 & \frac{1}{10} & \frac{3}{20} & \frac{3}{4} \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \end{aligned}$$

The elements $\pi_n(t)$ give the probabilities $\Pr(X_t = n)$ at time t , i.e., the probabilities of capture in *at most* t steps. The probabilities of capture *exactly* at time t are given by $\pi_n(t) - \pi_n(t-1)$. The expected capture time (conditional on strategy \mathbf{X} being used) is

$$\mathbb{E}T = \sum_{t=1}^{\infty} t \cdot ((\pi_n(t) - \pi_n(t-1))).$$

In the above example we have

$$\mathbb{E}T = 1 \cdot \left(\frac{1}{2} - \frac{1}{5}\right) + 2 \cdot \left(\frac{3}{4} - \frac{1}{2}\right) + 3 \cdot \left(1 - \frac{3}{4}\right) = \frac{31}{20}.$$

The approach can be generalized to more than one cops, by letting $\mathbf{x} = (x_1, x_2, \dots, x_k)$ be a configuration of cops and defining $\bar{P}(\mathbf{x})$, $P(\mathbf{x})$ analogously to the one cop case. Given that the cops follow the strategy $\mathbf{X} = (X_1, X_2, \dots, X_s)$, the transition probabilities of Y satisfy

$$\Pr(Y_t = j \mid Y_{t-1} = i) = P_{i,j}(X_t)$$

for $t \leq s$. So the robber process is an inhomogeneous Markov chain, with the transitions controlled by the cops' actions. Markov chains of this type are called *Markov Decision Processes* (MDP) or *Controlled Markov Processes*, where the control function is X_t ; it is a (stochastic) control in the sense that it allows us to change the transition probabilities of Y_t . We can use the MDP formulation to compute $\mathbb{E}T$ for any given strategy \mathbf{X} in reasonable time. Computing the *optimal* strategy is not computationally viable; for example, with $|V| = n$ and k cops there may exist up to $(n^k)^t$ strategies of length t (and corresponding $\mathbb{E}T$'s) to evaluate. In the next section we will present a viable approach to compute the optimal strategy.

MDP's were introduced in the book [4]; book-length treatments are [1, 7, 9, 11]; an online tutorial is [5]. MDP's have been applied to a version of the cops-robber problem in [2].

4 Computing the expected capture time for the optimal cop strategy

In this section we will show how to compute $F(G) = \frac{ct(G)}{dct(G)}$ for an arbitrary G . This reduces to computing $ct(G)$ and $dct(G)$, which can be achieved by algorithms which have previously appeared in the literature. To improve the presentation we assign a name to each algorithm and make a few notational modifications; also we point out the similarity between the two algorithms which apparently has not been noticed before.

1. The *CAAR* (Cop Against Adversarial Robber) algorithm computes $ct_{x,y}(G)$ for every initial cop/robber configuration (x, y) . In addition, CAAR computes the optimal cop and robber play for every (x, y) . Capture time $ct(G)$ is easily computed from $ct(G) = \min_x \max_y ct_{x,y}(G)$.
2. Similarly, the *CADR* (Cop Against Drunk Robber) algorithm computes $dct_{x,y}(G)$ and the optimal cop play for every (x, y) ; drunken capture time $dct(G)$ is computed from $dct(G) = \min_x \frac{\sum_y dct_{x,y}(G)}{n}$.

4.1 The CAAR Algorithm

The *CAAR* (Cop Against Adversarial Robber) algorithm computes $ct_{x,y}(G)$ for every initial cop/robber configuration (x, y) . In addition, CAAR computes the optimal cop and robber play for every (x, y) . Capture time $ct(G)$ is easily computed from $ct(G) = \min_x \max_y ct_{x,y}(G)$.

CAAR was introduced by Hahn and MacGillivray in [3]. We present the algorithm for the case of a single cop (the generalization for more than one cops is obvious). Changing notation, we use $C_{x,y}$ to denote the game duration when the cop is located at x , the robber at y and it's the cop's turn to move (in other words $C_{x,y}$ equals $ct_{x,y}(G)$). Similarly $R_{x,y}$ denotes game duration when it's the robber's turn to move. For both $C_{x,y}$ and $R_{x,y}$ we assume optimal play by both cop and robber. CAAR consists of the following recursion (for $i = 1, 2, \dots$):

$$\forall (x, y) \in \widehat{V}^2 : R_{x,y}^{(i)} = \max_{y' \in N^+(y)} C_{x,y'}^{(i-1)}, \quad (1)$$

$$\forall (x, y) \in \widehat{V}^2 : C_{x,y}^{(i)} = 1 + \min_{x' \in N^+(x)} R_{x',y}^{(i)}. \quad (2)$$

C and R are initialized with $C_{x,y}^{(0)} = R_{x,y}^{(0)} = \infty$ for all $x \neq y$. We set $C_{x,x}^{(i)} = R_{x,x}^{(i)} = 0$ for $i = 0, 1, 2, \dots$. In the limit $i \rightarrow \infty$, CAAR computes the solution of the equations

$$\forall (x, y) \in \widehat{V}^2 : R_{x,y} = \max_{y' \in N^+(y)} C_{x,y'}, \quad (3)$$

$$\forall (x, y) \in \widehat{V}^2 : C_{x,y} = 1 + \min_{x' \in N^+(x)} R_{x',y}, \quad (4)$$

$$\forall x \in V : C_{x,x} = R_{x,x} = 0. \quad (5)$$

The interpretation of the equations is the following. Equation (3) says: from configuration (x, y) the robber moves so as to maximize the length of the game; similarly, (4) describes the cop's goal to minimize the game duration (since the cop moves in the first phase of each round, 1 time unit must be added to $\min R_{x',y}$); finally (5) says that the game ends when cop and robber occupy the same vertex.

The original version of CAAR, as presented in [3], does not actually use a separate copy of $C^{(i)}$ and $R^{(i)}$ for each value of i . Rather, the changes are made “in place”, using the update rule

$$\forall (x, y) \in \widehat{V}^2 : R_{x,y} \leftarrow \max_{y' \in N^+(y)} C_{x,y'}, \quad (6)$$

$$\forall (x, y) \in \widehat{V}^2 : C_{x,y} \leftarrow 1 + \min_{x' \in N^+(x)} R_{x',y}. \quad (7)$$

In other words, the above version (introduced in [3]) uses *Gauss-Seidel* iteration (while our version (1)-(2) uses *Jacobi* iteration). The initializations used in [3] are

$$\forall (x, y) \in \widehat{V}^2 : C_{x,y} \leftarrow \infty, \quad R_{x,y} \leftarrow \infty \quad (8)$$

$$\forall x \in V : C_{x,x} \leftarrow 0, \quad R_{x,x} \leftarrow 0. \quad (9)$$

Hahn and MacGillivray [3] prove that (6)-(9) converges to the solution of (3)-(5) in a finite number of steps (provided $c(G) = 1$).

CAAR provides the optimal cop strategy in *feedback* form $U_{x,y}$, i.e., the optimal cop move when the cop/robber configuration is (x, y) . This is achieved by recording a minimizing x' . The optimal robber strategy

$W_{x,y}$ (for the adversarial robber) is obtained similarly. For every (x, y) configuration we can have more than one optimal move, but they all yield the same (optimal) game duration.

CAAR can be generalized for the case of k cops, replacing x by a k -tuple $\mathbf{x} = (x_1, \dots, x_k)$; however the necessary computations are exponential in k , hence the algorithm is computationally viable only for small k (our implementation **CopsRobber** allows the values $k \in \{1, 2\}$) and “relatively small” $|V|$.

The Jacobi version of the CAAR algorithm is described in pseudocode as follows (for the Gauss-Seidel version, simply remove the i index).

Algorithm 1 The CAAR Algorithm

Input: Node set V , neighborhoods $N(x)$ (for all $x \in V$)
 $i = 0$
 $C_{x,y}^{(i)} = \infty$ for all $(x, y) \in \hat{V}^2$
 $R_{x,y}^{(i)} = \infty$ for all $(x, y) \in \hat{V}^2$
 $C_{x,x}^{(i)} = 0$ for all $x \in V$
 $R_{x,x}^{(i)} = 0$ for all $x \in V$
while $C^{(i)} \neq C^{(i-1)}$ **do**
 $i = i + 1$
for all $(x, y) \in \hat{V}^2$ **do**
 $R_{x,x}^{(i)} = 0$ for all $x \in V$
 $C_{x,x}^{(i)} = 0$ for all $x \in V$
 $R_{x,y}^{(i)} = \max_{y' \in N^+(y)} C_{x,y'}^{(i-1)}$
 $W_{x,y}^{(i)} = \arg \max_{y' \in N^+(y)} C_{x,y'}^{(i-1)}$
 $C_{x,y}^{(i)} = 1 + \min_{x' \in N^+(x)} R_{x',y}^{(i)}$
 $U_{x,y}^{(i)} = \arg \min_{x' \in N^+(x)} R_{x',y}^{(i)}$
end for
end while
 $C = C^{(i)}, R = R^{(i)}, U = U^{(i)}, W = W^{(i)}$
Output: expected capture times C, R , optimal moves U, W

4.2 The CADR Algorithm

The *CADR* (Cop Against Drunk Robber) algorithm computes $dct_{x,y}(G)$ and the the optimal cop play for every (x, y) ; drunken capture time $dct(G)$ is computed from $dct(G) = \min_x \frac{\sum_y ct_{x,y}(G)}{n}$.

Extending the CAAR idea to the *drunk* robber game, let us now use $C_{x,y}$ to denote $dct_{x,y}(G)$. In other words $C_{x,y}$ (resp., $R_{x,y}$) is *expected* game duration after the cop’s (resp., robber’s) move. Recall (Section 3) that $P_{y,y'}(x)$ is the probability of the robber transiting from y to y' , given that the cop is at x ; note that $P(x)$ is a *substochastic* matrix. The analog of (1)-(2) is

$$\forall (x, y) \in \hat{V}^2 : R_{x,y}^{(i)} = \sum_{y' \in N(y)} P_{y,y'}(x) C_{x,y'}^{(i-1)}, \quad (10)$$

$$\forall (x, y) \in \hat{V}^2 : C_{x,y}^{(i)} = 1 + \min_{x' \in N^+(x)} R_{x',y}^{(i)} \quad (11)$$

and the analog of (3)-(5) is

$$\forall (x, y) \in \hat{V}^2 : R(x, y) = \sum_{y' \in N(y)} P_{y,y'}(x) C_{x,y'}, \quad (12)$$

$$\forall (x, y) \in \hat{V}^2 : C_{x,y} = 1 + \min_{x' \in N^+(x)} R_{x',y}. \quad (13)$$

$$\forall x \in V : C_{x,x} = R_{x,x} = 0. \quad (14)$$

We want (10)-(11) to converge to the solution of (12)-(14).

Actually (10)-(11) can be simplified. Since the drunk robber does not choose his moves, we can eliminate $R_{x,y}^{(i)}$ from (12)-(13) and obtain the CADR algorithm recursion:

$$\forall (x, y) \in \widehat{V}^2 : C_{x,y}^{(i)} = 1 + \min_{x' \in N^+(x)} \left(\sum_{y' \in N(y)} P_{y,y'}(x') C_{x',y'}^{(i-1)} \right). \quad (15)$$

This is the Jacobi-iteration version of CADR; a Gauss-Seidel version can also be used.

Before we discuss convergence of the CADR algorithm, we must discuss its connection to MDP's. Consider a general MDP process with state space S , control or *action* space A , transition matrix Q and cost matrix $G(a)$ (i.e., $G_{s,s'}(a)$ is the cost of transition $s \rightarrow s'$ using action a). The state space satisfies $S = S_T \cup S_A$, where S_T are the transient states and S_A the absorbing ones; it is assumed that transitions after absorption have zero cost: $G_{s,s'}(a) = 0$ for $s, s' \in S_A$. Let C_s be the expected total cost of the process starting from state s and continuing until absorption. Then [7] C satisfies the equations

$$\forall s \in S_T : C_s = \min_{a \in A} \left(G_{s,s'}(a) + \sum_{s' \in S_T} Q_{s,s'}(a) C_{s'} \right) \quad (16)$$

and the solutions to (16) can be obtained by the following *value iteration*:

$$\forall s \in S_T : C_s^{(i)} = \min_{a \in A} \left(G_{s,s'}(a) + \sum_{s' \in S_T} Q_{s,s'}(a) C_{s'}^{(i-1)} \right). \quad (17)$$

Once again, (17) is the Jacobi version of value iteration; a Gauss-Seidel version can also be used.

While we have derived (15) from (10)-(11), which we see as an analog of (1)-(2), we will now show that (15) is a version of (17). To this end, let us take $S_T = \widehat{V}^2$ and $A = V$; in other words, states $s = (x, y)$ are cop/robber configurations and actions $a = x'$ are new cop positions. Regarding move costs: (a) before capture every move has unit cost, (b) after capture only moves of the form $(x, x) \rightarrow (x, x)$ are possible and these have zero cost; in short

$$G_{(x,y),(x',y')}(x') = \begin{cases} 1 & \text{iff } x \neq y \\ 0 & \text{else.} \end{cases}$$

Finally,

$$Q_{(x,y),(x',y')}(a) = \begin{cases} P_{y,y'}(x) & \text{iff } a = x' \text{ and } x' \in N^+(x) \text{ and } y' \in N(y) \\ 0 & \text{else.} \end{cases}$$

Using the above, it is easy to reduce (17) to (15).

Returning to convergence issues, value iteration has been studied by several authors, in various degrees of generality [2, 4, 11]. For our purposes the following result [2] is sufficient: if a *proper strategy*¹ exists, the Jacobi version of CADR will converge (to the unique solution of (16)) for any $C^{(0)}$ which satisfies

$$\forall (x, y) \in \widehat{V}^2 : C_{x,y}^{(0)} \geq 0.$$

We have proved in [6] that the cop has a proper strategy for every G . Hence CADR converges (to the unique solution of (12)-(14)) for arbitrary (nonnegative) initialization $C^{(0)}$. Convergence of the Gauss-Seidel version has also been proved.

CADR (similarly to CAAR) provides the optimal cop strategy in *feedback* form $U_{x,y}$, i.e., the optimal cop move when the cop/robber configuration is (x, y) . This is achieved by recording a minimizing x' in (15).

CADR can also be generalized for the case of k cops, replacing x by a k -tuple $\mathbf{x} = (x_1, \dots, x_k)$ (again, our **CopsRobber** implementation allows the values $k \in \{1, 2\}$). Also CADR will work for any transition probability matrix P , not just for random walks. Hence, if desired, we can compute the cost of drunkenness for any

¹I.e., a strategy which yields finite expected cost.

number of cops (not just for $k = c(G)$) and for non-uniform random walks (birth-and-death processes) and other kinds of Markovian robbers.

The Jacobi version of the CADR algorithm is described in pseudocode as follows (for the Gauss-Seidel version, simply remove the i index).

Algorithm 2 The CADR Algorithm

Input: Node set V , neighborhoods $N(x)$ (for all $x \in V$), transition matrix P , initialization level M , termination criterion ε

$i = 0$

$C_{x,y}^{(i)} = M$ for all $(x, y) \in \hat{V}^2$

$C_{x,x}^{(i)} = 0$ for all $x \in V$

while $|C^{(i)} - C^{(i-1)}| < \varepsilon$ **do**

$i = i + 1$

for all $(x, y) \in \hat{V}^2$ **do**

$C_{x,x}^{(i)} = 0$ for all $x \in V$

$C_{x,y}^{(i)} = 1 + \min_{x' \in N^+(x)} \left(\sum_{y' \in N(y)} P_{y,y'} C_{x',y'}^{(i-1)} \right)$

$U_{x,y}^{(i)} = \arg \min_{x' \in N^+(x)} \left(\sum_{y' \in N(y)} P_{y,y'} C_{x',y'}^{(i-1)} \right)$

end for

end while

$C = C^{(i)}, U = U^{(i)}$

Output: expected capture times C , optimal moves U

4.3 Computing the Cost of Drunkennes

$F(G)$ can be easily computed from $ct(G)$ and $dct(G)$. Recall the definition of $F(G)$:

$$F(G) = \frac{ct(G)}{dct(G)} = \frac{\min_{x \in V} \max_{y \in V} ct_{x,y}(G)}{\min_{x \in V} \left(\frac{1}{|V|} \sum_{y \in V} dct_{x,y}(G) \right)}. \quad (18)$$

Since $ct_{x,y}(G) = C_{x,y}$ as computed by CAAR and $dct_{xy}(G) = C_{x,y}$ as computed by CADR, (18) can be easily computed after completion of these algorithms.

The optimal initial positions for the cops and robber can also be easily computed once $ct_{x,y}(G)$ and $dct_{xy}(G)$ are available. Consider the two terms of the fraction in (18).

1. The denominator concerns the drunk CR game, which (as already mentioned) is a one-player game. Since the cost to the cop for starting in position x is $\frac{1}{|V|} \sum_{y \in V} dct_{x,y}(G)$, his optimal starting position will be $x^* = \arg \min_{x \in V} \frac{1}{|V|} \sum_{y \in V} dct_{x,y}(G)$ (there may be more than one such x^*). Hence

$$dct(G) = \frac{1}{|V|} \sum_{y \in V} dct_{x^*,y}(G) = \min_{x \in V} \left(\frac{1}{|V|} \sum_{y \in V} dct_{x,y}(G) \right)$$

2. The numerator concerns the adversarial CR game. Since the cop plays first and he can expect the optimal response by the robber, he will choose $x^* = \arg \min_{x \in V} \max_{y \in V} ct_{x,y}(G)$. The robber will observe x^* (since he plays after the cop) and will play $y^* = \arg \max_y ct_{x^*,y}(G)$ ². In short,

$$ct(G) = ct_{x^*,y^*}(G) = \min_{x \in V} \max_{y \in V} ct_{x,y}(G).$$

²Note that the situation would be different if cop and robber chose their initial positions simultaneously; then we would have a zero-sum two-player game in normal form and it might be advisable to use *mixed* strategies.

5 The CopsRobber Package

In the current section we give a brief presentation of our Matlab package **CopsRobber**. The package is publicly available and can be downloaded from <http://users.auth.gr/~kehagiat/GraphSearch/>; installation consists in unpacking the contents of the file **CopsRobbers.zip** into a directory and adding it to the Matlab path. The main CopsRobber functions are the following.

CRstrat.m	Compute expected capture time for a fixed strategy and adversarial robber
CRcaar.m	Compute optimal capture times and optimal moves for cops and adv. robber
CRcadr.m	Compute expected capture times and optimal moves for cops and drunk robber
CRsim.m	Simulate cops and (adv. or drunk) robber playing feedback strategies
CRcod.m	Compute the cost of drunkenness
CRcheq.m	Checks that a pair of C (and R) satisfy the optimality equations

Table 1

Details on use of the functions are given in the Matlab files. Let us present the usage of the main commands by examples along with usage comments (the user can test all the examples listed below by typing **CRdemo.m** in the Matlab command lie).

We start Matlab and move to the directory which contains the code. We first type

```
>> P=grf2P01('Edge01b.txt');
```

to load a graph from the file **Edge01b.txt** (it is a text file with each line containing two node numbers, i.e., an edge of the graph). The graph G with which we will work is P_{20} , a path with 20 nodes.

On our first example we compute **Texp** and **Tmax**, the expected and maximum capture time for a drunk robber and a fixed cop strategy X . We read the strategy from file **strat01b.txt** and invoke the function **CRstrat**. The **disp** command prints **Texp** and **Tmax**.

```
>> X=load('strat01b.txt')
>> [p, Texp, Tmax]=CRstrat(P,X);
>> disp([Texp Tmax])
8.9665    18.0000
```

Next we will compute optimal strategies with the CAAR algorithm. First we determine certain algorithm parameters.

```
>> T=200;
>> epsilon=0.01;
>> initial=0;
>> copnum=1;
>> intermethod=1;
```

T is the maximum number of iterations for which the algorithm will run; **epsilon** is the termination criterion ε ; **initial** is the initial value for $C^{(0)}$, $R^{(0)}$; **copnum** is the number of cops; **intermethod=1** means we will use the Jacobi iteration. Now we are ready to invoke the CAAR algorithm as follows.

```
>> [U1,U2,W,Ca,Ra]=CRcaar(P,T,epsilon,initial,copnum,intermethod);
```

The algorithm runs and produces the following output (time step and max difference between old and updated C and R values).

```
1      1.000000
2      1.000000
...
19     1.000000
20     0.000000
```

We now have $U1, W, Ca, Ra$, the U, W, C, R matrices of optimal moves for cop and adversarial robber (CRcaar also outputs $U2$, the optimal moves for the second cop, which in the one-cop problem is an empty matrix). We will now check that the obtained C and R satisfy the optimality equations (3)-(5). To this end we run

```
>> [EC,ER]=CRcheq(Ca,Ra,P);
>> disp([ sum(sum(EC)) sum(sum(ER))])
0 0
```

The first command invokes CRcheq.m, which returns two $N \times N$ matrices. $EC(x,y)$ equals zero if $C_{x,y} = 1 + \min_{x'} R_{x',y}$ (and $C_{x,x} = 0$) and one otherwise (similarly for ER). Hence summing all elements of the matrices (by the second command) we see that no equations are violated; in other words we have obtained the unique solution of (3)-(5).

We can run CAAR on the same graph using two cops (and then check the solution) as follows.

```
>> copnum=2;
>> [U1,U2,W,C,R]=CRcaar(P,T,thrs1,inf1,copnum,itermethod);
>> [EC,ER]=CRcheq(C,R, P);
>> disp([ sum(sum(sum(EC))) sum(sum(sum(ER)))])
```

Note the triple sums in the last command; they are needed because in our implementation C and R (and also EC and ER) are $N \times N \times N$ matrices.

We can also run the CADR algorithm (for one cop) and check the solutions as follows.

```
>> copnum=1;
>> [U1,U2,Cd]=CRcadr(P,T,thrs1,inf1,copnum,itermethod);
>> [EC,ER]=CRcheq(Cd,[], P);
>> disp([ sum(sum(EC)) sum(sum(ER))])
```

Note the R matrix used as input to CRcheq. It is the second, empty ($[]$) matrix. Since the robber is drunk, CRcheq does not produce any R output. Let us use the obtained cost matrices C_a, R_a, C_d to compute the cost of drunkenness F .

```
>> [F,CT,DCT,mopta,moptd]=CRcod(Ca,Ra,Cd);
>> disp([CT DCT F])
10.0000 4.4588 2.2428
```

The optimal capture time (for adversarial robber) is $CT = 10$, the expected capture time (for adversarial robber) is $DCT = 4.4588$ and the cost of drunkenness is $F = \frac{CT}{DCT} = 2.2428$.

Finally, we can simulate a game between one cop and a drunk robber. First we give values to the simulation parameters.

```
>> x10=1;
>> x20=-1;
>> y0=5;
>> Tmax=100;
>> output=1;
```

The initial position of the first cop is $x10=1$. The initial position of the second cop is taken as $x20=-1$ to indicate there is actually a single cop. The initial position of the robber is $y0=5$. We will use a maximum of $Tmax=100$ steps in the simulation and $output=1$ means the program will actually print out the cop and robber positions for every time step. Now we can run the CRSim command, which produces the following output.

```
>> [X1,X2,Y,Z,CT]=CRsim(x10,x20,y0,P,U1,U2,[],Tmax,output);
```

```
0      1      15      14
1      2      16      14
```

2	3	15	12
3	4	16	12
....			
14	15	17	2
15	16	18	2
16	17	17	0

The first column is time, the second is the cop's position and the third the robber's position; the final column shows the cop-robber distance, when this becomes zero the robber is captured.

6 Experiments

6.1 Experiment Group 1: Paths

In this section we perform experiments on paths. We will use the notation P_n to denote the path of n nodes.

Let us first consider the case of one-cop vs. an adversarial and then a drunk robber. The optimal cop strategy against the adversarial robber actually is quite obvious: in case n is odd, the cop should select the middle node $\frac{n+1}{2}$ and, after the robber selects his initial node, the cop should go in the robber's direction; in case of even n , the cop has two optimal initial node choices: $\frac{n}{2} - 1$, $\frac{n}{2}$. For $n \geq 5$ there are many optimal robber strategies; they must all satisfy the following conditions: (a) initial robber position must have a distance at least 2 from the cop and (b) the robber must try to keep a distance of at least 2 for as long as he can. For one-cop and drunk robber, the optimal cop strategy remains the same; rather than following a particular strategy, the robber simply random-walks on P_n . We perform the corresponding computations for various n values, and present the results in the following table.

n	$ct(P_n)$	$dct(P_n)$	$F(P_n)$	Execution time in sec (CAAR)	Execution time in sec (CADR)
5	2.00	0.8000	2.5000	0.0168	0.0136
10	5.00	2.0000	2.5000	0.0183	0.0206
20	10.00	4.4588	2.2428	0.1496	0.1679
30	15.00	6.9522	2.1576	0.4265	0.6139
40	20.00	9.4517	2.1160	0.9936	1.5587
50	25.00	11.9526	2.0916	1.9511	3.4767
60	30.00	14.4540	2.0755	3.4224	6.5090

Table 2: Cost of drunkenness on path P_n with one cop.

We see that, as n (the number of nodes) increases, F tends to 2, the theoretically computed value (see Theorem 2.2). Computation time is very small even for large values of n . Also note that CADR requires longer time to complete than CAAR.

We perform similar computations for the case of two cops. From CAAR it turns out (quite reasonably) that the optimal cop strategy is to place the cops at approximately $\frac{n}{4}$ and $\frac{3n}{4}$; the robber's optimal placement is at (approximately) $\frac{n}{2}$. The same cop placements remain optimal for the drunk robber. We present the results, for various n values, in Table 3. The results are similar to the one cop case, but note that execution time increases quite rapidly with n . For the case $n = 60$, CADR requires 844 sec, a little over 14 min. While this is still manageable, it is rather on the long side.

n	$ct(P_n)$	$dct(P_n)$	$F(P_n)$	Execution time (CAAR)	Execution time (CADR)
5	1.00	0.6000	1.6667	0.0271	0.0198
10	2.00	0.9000	2.2222	0.2523	0.2851
20	5.00	2.0500	2.4390	4.4772	6.3269
30	7.00	3.2708	2.1401	26.3938	38.6483
40	10.00	4.5044	2.2201	85.5995	135.9235
50	12.00	5.7459	2.0885	212.3132	388.6472
60	15.00	6.9928	2.1451	404.4863	844.8199

Table 3: Cost of drunkenness on path P_n with two cops.

6.2 Experiment Group 2: Cycles

In this section we perform experiments on the n nodes cycle C_n . Since in the cycle a single cop cannot capture the adversarial robber, we will only consider the case of two cops hunting first an adversarial and then a drunk robber. For both the adversarial and drunk robber and even n , an optimal cop strategy is to place the two cops in any two diametrically opposite nodes; the optimal strategies for the case of odd n are similar. The adversarial robber's optimal strategies can be described as follows: (a) initial robber position must have a distance at least 2 from the nearest cop and (b) the robber must try to keep a distance of at least 2 from the nearest cop for as long as he can. The results for various n values appear in Table 4. We see that, as n (the number of nodes) increases, F tends to 2, the theoretically computed value (see Theorem 2.3). In general, the results of Table 4 are very similar to those of Table 3.

n	$ct(P_n)$	$dct(P_n)$	$F(P_n)$	Execution time in sec (CAAR)	Execution time in sec (CADR)
5	1.00	0.6000	1.6667	0.0224	0.0230
10	2.00	1.0000	2.0000	0.1667	0.2186
20	5.00	2.1000	2.3810	2.6292	3.9532
30	7.00	3.3333	2.1000	13.3594	21.8254
40	10.00	4.5500	2.1978	42.2385	74.3258
50	12.00	5.8000	2.0690	103.9459	206.5505
60	15.00	7.0333	2.1327	215.4420	458.9260

Table 4: Cost of drunkenness on cycle C_n with two cops.

6.3 Experiment Group 3: Trees

We examine regular trees $T_{d,k}$ (d is node degree and k tree depth). In Table 5 we list results for one cop and various d and k values; total number of nodes is listed in the third column.

d	k	n	$ct(T_{d,k})$	$dct(T_{d,k})$	$F(T_{d,k})$	Execution time in sec (CAAR)	Execution time in sec (CADR)
2	2	7	2.0000	0.8571	2.3333	0.0185	0.0122
2	3	15	3.0000	1.6444	1.8243	0.0279	0.0165
2	4	31	4.0000	2.4014	1.6657	0.1388	0.0916
2	5	63	5.0000	3.3161	1.5078	0.7142	0.6402
3	2	13	2.0000	0.9231	2.1667	0.0144	0.0074
3	3	40	3.0000	1.8188	1.6495	0.1803	0.1218
3	4	121	4.0000	2.6513	1.5087	2.2611	2.9700
4	2	21	2.0000	0.9524	2.1000	0.0355	0.0195
4	3	85	3.0000	1.8918	1.5858	0.8421	0.8509
5	2	31	2.0000	0.9677	2.0667	0.0783	0.0450

Table 5: Cost of drunkenness on $T_{d,k}$ with one cop.

In Table 6 we perform the same computations for two cops.

d	k	n	$\text{ct}(T_{d,k})$	$\text{dct}(T_{d,k})$	$F(T_{d,k})$	Execution time in sec (CAAR)	Execution time in sec (CADR)
2	2	7	1.0000	0.7143	1.4000	0.0593	0.0302
2	3	15	2.0000	0.8667	2.3077	0.6608	0.4188
2	4	31	3.0000	1.6237	1.8477	8.1467	6.0328
2	5	63	4.0000	2.3792	1.6812	94.8508	94.8802
3	2	13	2.0000	0.8462	2.3636	0.3011	0.1720
3	3	40	3.0000	1.5078	1.9896	13.9444	10.5496
3	4	121	4.0000	2.3688	1.6886	557.7370	864.7016
4	2	21	2.0000	0.9048	2.2105	1.3541	0.8400
4	3	85	3.0000	1.6541	1.8137	143.1214	153.4834
5	2	31	2.0000	0.9355	2.1379	4.5125	2.9715

Table 6: Cost of drunkenness on $T_{d,k}$ with two cops.

Comparing Tables 5 - 6 with Tables 2 - 4 we see that trees can be searched faster than paths and cycles of the same size (i.e., the same number of nodes). We also note an interesting effect regarding $\text{ct}(T_{d,k})$. For $d = 2$, when going from one to two cops, $\text{ct}(T_{2,k})$ decreases by one; e.g., $\text{ct}(T_{2,2})$ is 2 for one cop and 1 for two cops. However, for $d > 2$, we see no corresponding decrease; e.g., $\text{ct}(T_{3,2})$ is 2 for both one and two cops. This may at first seem paradoxical, but actually is easily explained. In the binary tree two cops can divide the graph $T_{2,k}$ into two subtrees $T_{2,k-1}$ and each cop searches a subtree separately; this effectively reduces the depth of the tree by one. However this reduction is not possible when $d > 2$; in this case the optimal cop placement is (for both cops) at the root of the tree and only one cop is actively pursuing the robber. These considerations do not apply to the drunk robber. Hence, for binary trees, the cost of drunkenness is actually smaller for two cops than for one.

We also perform a few experiments involving non-regular trees, which we denote with $T_{d,k}^p$. Here d, k are as previously and p has the following meaning: every node can spawn up to a maximum of d nodes, but each node will actually be generated w.p. p . We generate 40 such trees for each set of (d, k, p) values and apply to each such tree CAAR and CADR; in Table 7 we list the *average* (over the 40 instances) of n , $\text{ct}(T_{d,k}^p)$, $\text{dct}(T_{d,k}^p)$ and $F(T_{d,k}^p)$.

d	k	p	aver. n	No. of Cops	$\text{ct}(T_{d,k}^p)$	$\text{dct}(T_{d,k}^p)$	$F(T_{d,k}^p)$	Execution time in sec (CAAR)	Execution time in sec (CADR)
3	6	0.50	33.05	1	4.45	2.3536	1.9345	0.3203	0.3285
4	6	0.35	23.50	1	4.10	2.0608	2.0297	0.1534	0.1233
3	7	0.50	51.92	1	5.15	2.8431	1.9282	1.0679	1.5013
3	6	0.50	33.05	2	3.60	1.6229	2.3397	43.2821	50.7222
4	6	0.35	23.50	2	3.30	1.3738	2.7420	11.7323	10.6124

Table 7: Cost of drunkenness on random tree $T_{d,k}^p$.

6.4 Experiment Group 4: Grids

In this section we perform experiments on square grids. We denote the $m \times m$ grid by Γ_m . Note that $\Gamma_m = P_m \square P_m$, i.e., it is the product graph of P_m by itself. We only list experiments with two cops (since a single cop cannot capture the adversarial robber on Γ_m). In [6] we show that, against the adversarial robbers, the optimal initial cop placements are at the nodes with coordinates (approximately) either $(\frac{m}{2}, \frac{m}{4})$ and $(\frac{m}{2}, \frac{3m}{4})$ or $(\frac{m}{4}, \frac{m}{2})$ and $(\frac{3m}{4}, \frac{m}{2})$.

m	n	$ct(\Gamma_m)$	$dct(\Gamma_m)$	$F(\Gamma_m)$	Execution time in sec (CAAR)	Execution time in sec (CADR)
3	9	2	0.8519	2.3478	0.1306	0.2198
4	16	3	1.1667	2.5714	1.1086	2.5146
5	25	4	1.5541	2.5739	5.9837	16.0363
6	36	5	1.9636	2.5463	23.3114	71.0747
7	49	6	2.3607	2.5416	72.5183	269.0830

Table 8: Cost of drunkenness on Γ_m with two cops.

6.5 Experiment Group 5: Lollipops

In this section we perform experiments on “*lollipop*” graphs. The (m, c) lollipop is denoted by $L_{m,c}$ and is a graph that is obtained from the complete graph $K_{[cm]}$ connected to the path P_m (that is, one end of the path belongs to the clique). For all values (m, c) , it is easy to see that one cop can catch the robber (either adversarial or drunk) on $L_{m,c}$. In [6] we have shown that, by appropriate selection of c , $F(L_{m,c})$ can take any value in $(1, \infty)$.

In Table 9 we present several combinations of m and c ; as usual, n denotes the total number of nodes. Similarly, in Table 10 we present the same results for two cops.

m	c	n	$ct(L_{m,c})$	$dct(L_{m,c})$	$F(L_{m,c})$	Execution time in sec (CAAR)	Execution time in sec (CADR)
5	1/2	8	3.00	1.4688	2.0426	0.0222	0.0194
5	1	10	3.00	1.5750	1.9048	0.0153	0.0213
5	2	15	3.00	1.4167	2.1176	0.0403	0.1126
10	1/2	15	6.00	2.9733	2.0179	0.0486	0.0724
10	1	20	6.00	2.8953	2.0723	0.1094	0.2951
10	2	30	6.00	2.2757	2.6366	0.4062	2.5187
20	1/2	30	11.00	5.9841	1.8382	0.4072	1.0404
20	1	40	11.00	5.4333	2.0246	1.1356	6.3614
20	2	60	11.00	3.9623	2.7762	5.0502	72.7441
30	1/2	45	16.00	8.9352	1.7907	1.5439	5.7239
30	1	60	16.00	7.9485	2.0130	4.8809	44.3143
30	2	90	16.00	5.6370	2.8384	23.2740	564.0467

Table 9: Cost of drunkenness with one cop on the lollipop $L_{m,c}$.

m	c	n	$ct(L_{m,c})$	$dct(L_{m,c})$	$F(L_{m,c})$	Execution time in sec (CAAR)	Execution time in sec (CADR)
5	1/2	8	2.00	0.7500	2.6667	0.1279	0.1137
5	1	10	2.00	0.8000	2.5000	0.2511	0.4601
5	2	15	2.00	0.8667	2.3077	2.0906	10.11405
10	1/2	15	3.00	1.2333	2.4324	1.3910	2.6694
10	1	20	3.00	1.1750	2.5532	6.8654	31.7424
10	2	30	3.00	1.1167	2.6866	87.6725	1026.7434
20	1/2	30	5.00	2.3188	2.1563	35.7257	136.6107
20	1	40	5.00	1.9891	2.5137	267.5297	2791.5937
20	2	60	not completed	not completed	not completed	not completed	not completed
30	1/2	45	not completed	not completed	not completed	not completed	not completed
30	1	60	not completed	not completed	not completed	not completed	not completed
30	2	90	not completed	not completed	not completed	not completed	not completed

Table 10: Cost of drunkenness with two cops on the lollipop $L_{m,c}$.

We see that in two cops experiments execution time increases rather rapidly and in fact several experiments did not run to completion (the actual behavior observed was that the Matlab program ran out of memory).

These negative results are interesting and useful to get an idea of the limits of the CAAR and CADR algorithms or, more precisely, of our implementations thereof. We hope a C implementation and more careful coding can extend the size of the problems which can be dealt with; but of course the “curse of dimensionality” will sooner or later catch up with us.

6.6 Experiment Group 6: Barbells

In this section we perform experiments on “barbell” graphs. The (m, c) barbell is denoted by $B_{m,c}$ and is a graph that is obtained from two complete graphs $K_{\lfloor cm \rfloor}$ connected by a path P_m (that is, one end of the path belongs to the first clique and the other end belongs to the second one). For all values (m, c) , it is easy to see that one cop can catch the robber (either adversarial or drunk) on $B_{m,c}$. In [6] we have shown that, by appropriate selection of c , $F(B_{m,c})$ can take any value in $(1, 2)$.

We present our results in Tables 11 (one cop) and 12 (two cops). Once again, there are cases for which the experiments were not completed (this is also true of two cops on $B_{20,c}$ but we do not include these negative results in Table 12).

m	c	n	$ct(B_{m,c})$	$dct(B_{m,c})$	$F(B_{m,c})$	Execution time in sec (CAAR)	Execution time in sec (CADR)
5	1/2	11	4	2.1970	1.8207	0.0554	0.0357
5	1	15	4	2.7383	1.4607	0.0400	0.0778
5	2	25	4	3.2786	1.2200	0.1510	0.6131
10	1/2	20	7	4.1978	1.6675	0.1011	0.1911
10	1	30	7	5.0317	1.3912	0.3154	1.1854
10	2	50	7	5.6410	1.2409	1.4801	13.4713
20	1/2	40	12	8.1385	1.4745	0.8796	2.9956
20	1	60	12	9.2946	1.2911	3.3353	27.3446
20	2	100	12	10.1874	1.1779	17.6969	351.9576
30	1/2	60	17	11.9492	1.4227	3.5567	19.2307
30	1	90	17	13.4889	1.2603	14.6170	190.7509
30	2	150	17	14.7000	1.1565	81.2394	2630.6598

Table 11: Cost of drunkenness with two cops on the barbell $B_{m,c}$.

m	c	n	$ct(B_{m,c})$	$dct(B_{m,c})$	$F(B_{m,c})$	Execution time in sec (CAAR)	Execution time in sec (CADR)
5	1/2	11	2	1.0000	2.0000	0.3613	0.5489
5	1	15	2	1.0000	2.0000	1.2929	3.6679
5	2	25	2	1.0000	2.0000	16.0763	116.0589
10	1/2	20	3	1.6500	1.8182	4.8205	11.9200
10	1	30	3	1.4667	2.0455	37.5107	245.9268
10	2	50	not completed	not completed	not completed	not completed	not completed

Table 12: Cost of drunkenness with two cops on the barbell $B_{m,c}$.

6.7 Experiment Group 7: Randomly generated graphs

In this section we apply CAAR and CADR to two families of random graphs.

In the first family, a graph is generated in the following manner. First a path of m nodes is generated. Then we take every pair of non-adjacent nodes x, y and add an edge xy w.p. p . Note that the resulting graph will always be connected. We generate twenty such graphs and apply CAAR and CADR to each one of them, always using two cops. It is possible (though unlikely) that, on one of the generated graphs, the adversarial robber cannot be captured by two cops; if this is the case we drop the particular graph and generate another one, with the goal that the total number of searched graphs is twenty (actually there were very few such cases). We list the obtained results in Table 13. The first three columns list the graph parameters (especially, n is the total number of nodes), the remaining columns give the usual information.

m	p	n	$ct(B_{m,c})$	$dct(B_{m,c})$	$F(B_{m,c})$	Execution time in sec (CAAR)	Execution time in sec (CADR)
10	0.05	10	2.1000	0.8667	2.4279	0.2357	0.3528
10	0.10	10	2.0500	0.8371	2.4502	0.1877	0.3897
10	0.15	10	1.8500	0.8183	2.2598	0.1733	0.4467
10	0.20	10	1.9000	0.8108	2.3426	0.1803	0.5681
10	0.25	10	1.6500	0.8056	2.0459	0.1794	0.7090
20	0.05	20	3.8500	1.3113	2.9257	2.6174	6.4532
20	0.10	20	3.3500	1.1636	2.8774	2.6635	9.4859
20	0.15	20	2.7000	1.0890	2.4676	3.0130	15.0438
20	0.20	20	2.5000	1.0389	2.3962	3.0761	19.3343

Table 13: Cost of drunkenness on random graphs of the first family.

In the second family, a graph is generated in the following manner. First a regular d -ary tree of depth k is generated. Then, we take every pair of non-adjacent nodes x, y and add an edge xy w.p. p . Again, the resulting graph will always be connected. We generate twenty such graphs and apply CAAR and CADR to each one of them, always using two cops; once again we reject graphs on which the robber cannot be captured and keep the total number of used graphs equal to twenty. We list the obtained results in Table 14.

d	k	p	n	$ct(B_{m,c})$	$dct(B_{m,c})$	$F(B_{m,c})$	Execution time (CAAR)	Execution time (CADR)
2	3	0.05	15	2.3500	0.9204	2.5407	0.8114	1.4018
2	3	0.10	15	2.3500	0.9450	2.4748	0.8268	2.1052
2	3	0.15	15	2.0500	0.9275	2.2095	0.8311	2.9978
2	3	0.20	15	2.0500	0.9459	2.1688	0.9049	4.2888
2	3	0.25	15	2.0000	0.9146	2.1887	0.9492	4.9867
3	3	0.05	40	6.9000	1.4670	4.6604	40.5833	106.3985
3	3	0.10	40	8.8824	1.4146	6.2497	65.3016	240.5962
3	3	0.15	40	8.6746	1.3227	6.5583	85.5467	389.56
3	3	0.20	40	8.5294	1.2601	6.7535	109.7863	627.8949

Table 14: Cost of drunkenness on random graphs of the second family.

6.8 Experiment Group 8: Execution Time

In the final group of experiments we want to explore the dependence of *execution time* on several factors.

First, we want to explore dependence on the size of the graph. We quantify “size” by n , the number of nodes. This is a very rough description of size, because it is expected (and will be observed in the following graphs) that CADR and CAAR can take very different times to complete on graphs with the same number of nodes but different topology. So, for example, the number of edges or the number of cycles of a graph will probably influence execution time decisively. At any rate, in this preliminary exploration we simply collect the data of the previous sections and plot them in Figures 1 and 2. In all figures, the horizontal axis corresponds to n , number of nodes, and the vertical axis to the logarithm of execution time (in sec) for CAAR (diamond) and CADR (square). For the same n value we can have several different execution times, corresponding to graphs of the same number of nodes but different topology. For the reader’s convenience we summarize the quantities plotted in each figure in the following table.

Figure	No. of Cops	Iteration
1-left	1	Jacobi
1-right	1	Gauss-Seidel
2-left	2	Jacobi
2-right	2	Gauss-Seidel

Table 15. Summary of figures 1-4

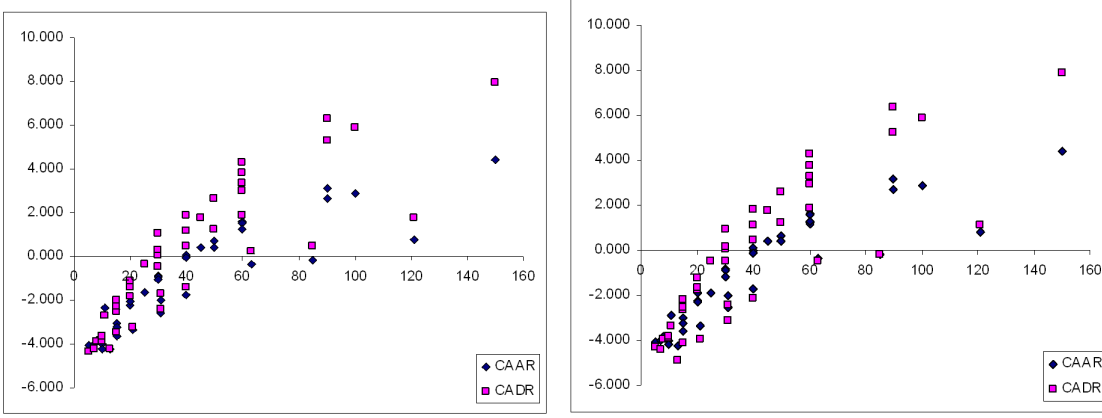


Figure 1: For one cop, we plot the natural logarithm of execution time vs. n , number of nodes of the graph. Left panel: Jacobi iteration; right panel: Gauss-Seidel iteration. In both panels, diamonds denote execution times for CAAR and squares for CADR.

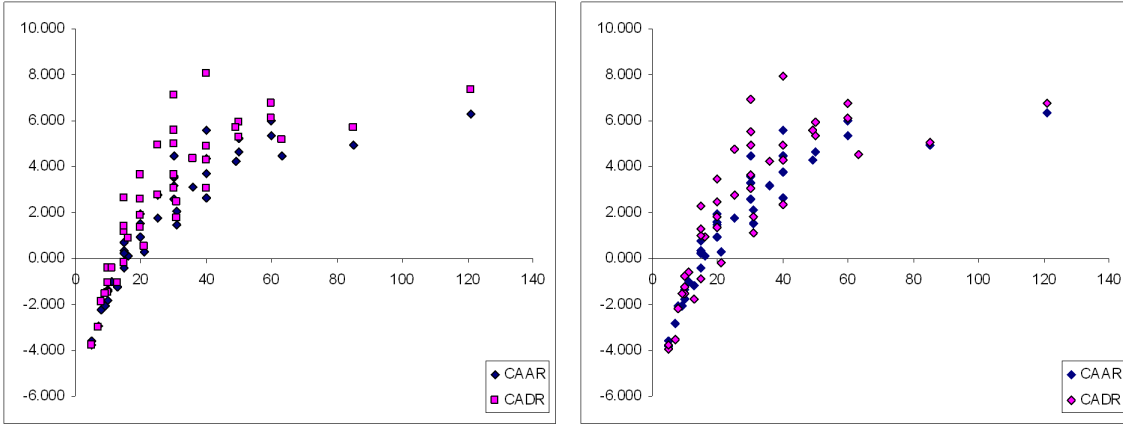


Figure 2: For two cops, we plot the natural logarithm of execution time vs. n , number of nodes of the graph. Left panel: Jacobi iteration; right panel: Gauss-Seidel iteration. In both panels, diamonds denote execution times for CAAR and squares for CADR.

We see in Figures 1-2 the rapid increase of execution time as a function of n . We also see that graphs with the same n can have very different execution times. For example, in Figure 1 we see that for $n \simeq 60$ execution time for a lollipop can be more than 100 times that for a tree. This stands to reason, since in certain cases a lollipop has a much higher number of edges (and cycles) than a tree with the same number of nodes.

We also see in Figures 1 and 2 that CADR generally requires considerably higher execution time than CAAR and, especially, moving from one to two cops will increase the execution time considerably. (In addition to execution time, another effect of increasing the number of cops is the increased memory requirement; this effect does not show in the above figures but we have verified its existence by monitoring memory usage during execution of the algorithms.)

On the other hand, figures 1 and 2 show that there is no appreciable difference in execution time between the Jacobi and Gauss-Seidel versions of the algorithms.

The final factor that we want to explore is the influence of *initial conditions* on execution time. To this end we run both CAAR and CADR on a number of graphs starting with different initializations $C_{x,y}$ and $R_{x,y}$ (for $x \neq y$). The results are summarized in Figures 3-7. We have used the initializations $C_{x,y} \in \{0, 1, 10, 100, 1000\}$. The graphs we have used are paths, cycles and trees; in each family we have used an increasing number of nodes, resulting in increasing execution times. The way we present the results is as follows. Let $T_e(\text{init})$ be the execution time of an algorithm for a particular graph and with initialization $C_{x,y} = \text{init}$. In Figures 3-7

the horizontal axis is execution time for the initialization $C_{x,y} = 0$, in other words $T_e(0)$. Four curves are included in each graph, one curve corresponding to each value $init \in \{1, 10, 100, 1000\}$. Each such curve depicts the ratio $f(T_e(0)) = \frac{T_e(init)}{T_e(0)}$. In other words, a particular point in such a curve shows the factor by which execution time changed (for a particular graph, requiring execution time $T_e(0) = \tau$) when the initialization changed from $C_{x,y} = 0$ to $C_{x,y} = init$. In all cases (except for the case of CADR on trees, i.e., the right panels of Figures 4 and 7) we see values close to 1; in other words: initialization values do not seem to play a critical role in execution time.

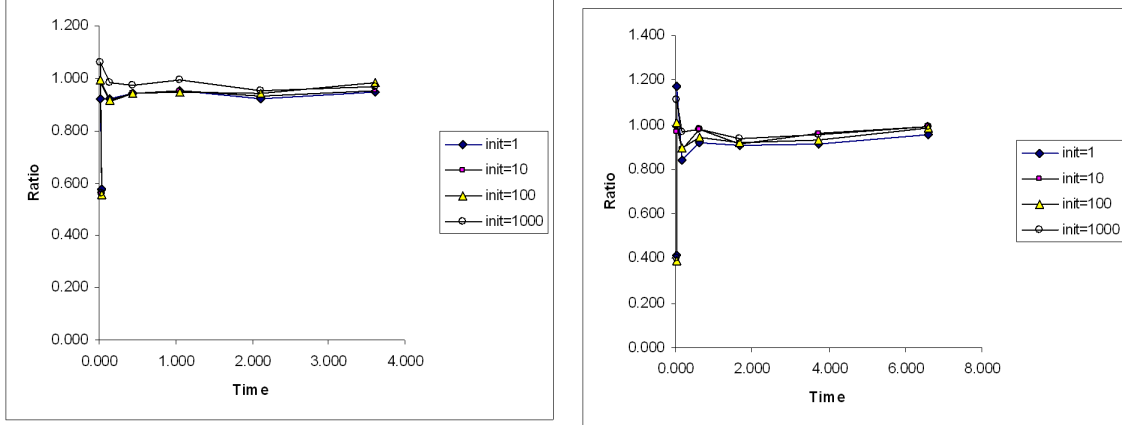


Figure 3: One cop on paths: ratio of execution times for various initializations of C and R . Let $T_e(init)$ be the execution time of CADR (resp. CAAR) when C (resp. C and R) is initialized at value $init$. The horizontal axis is $T_e(0)$. The vertical axis is the ratio $T_e(init)/T_e(0)$ for various $init$ values. Left panel: CAAR; right panel: CADR.

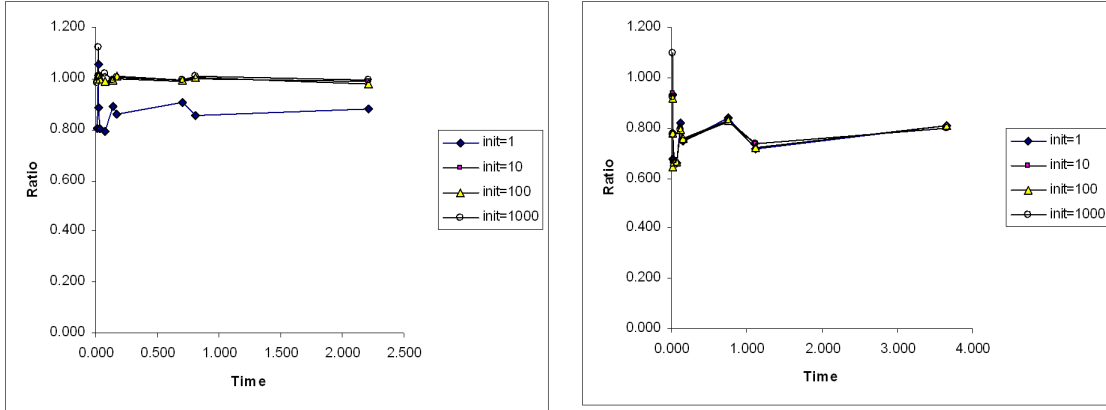


Figure 4: One cop on trees: ratio of execution times for various initializations of C and R . Let $T_e(init)$ be the execution time of CADR (resp. CAAR) when C (resp. C and R) is initialized at value $init$. The horizontal axis is $T_e(0)$. The vertical axis is the ratio $T_e(init)/T_e(0)$ for various $init$ values. Left panel: CAAR; right panel: CADR.

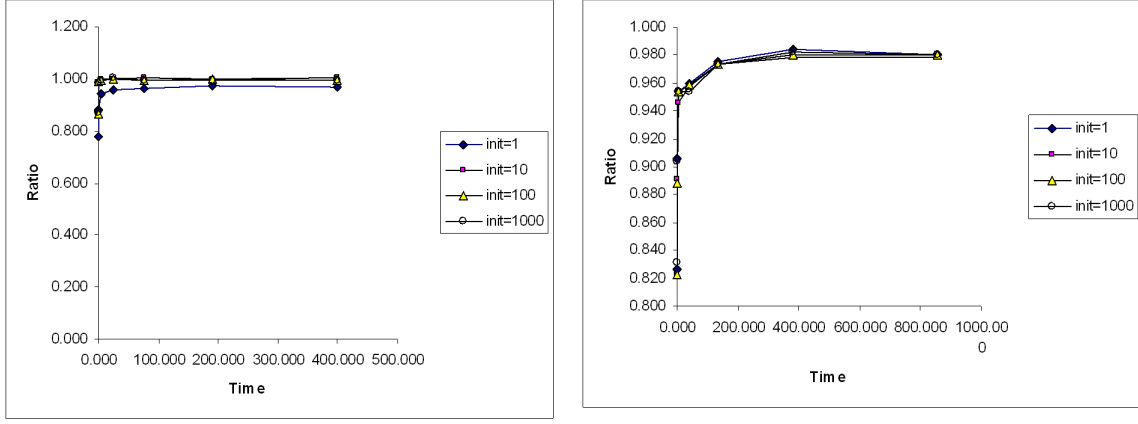


Figure 5: Two cops on paths: ratio of execution times for various initializations of C and R . Let $T_e(init)$ be the execution time of CADR (resp. CAAR) when C (resp. C and R) is initialized at value $init$. The horizontal axis is $T_e(0)$. The vertical axis is the ratio $T_e(init)/T_e(0)$ for various $init$ values. Left panel: CAAR; right panel: CADR.

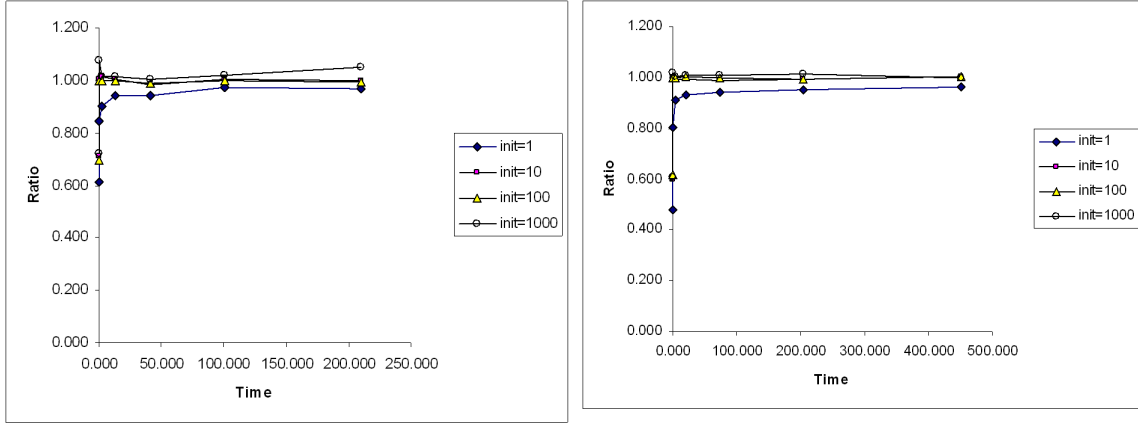


Figure 6: Two cops on cycles: ratio of execution times for various initializations of C and R . Let $T_e(init)$ be the execution time of CADR (resp. CAAR) when C (resp. C and R) is initialized at value $init$. The horizontal axis is $T_e(0)$. The vertical axis is the ratio $T_e(init)/T_e(0)$ for various $init$ values. Left panel: CAAR; right panel: CADR.

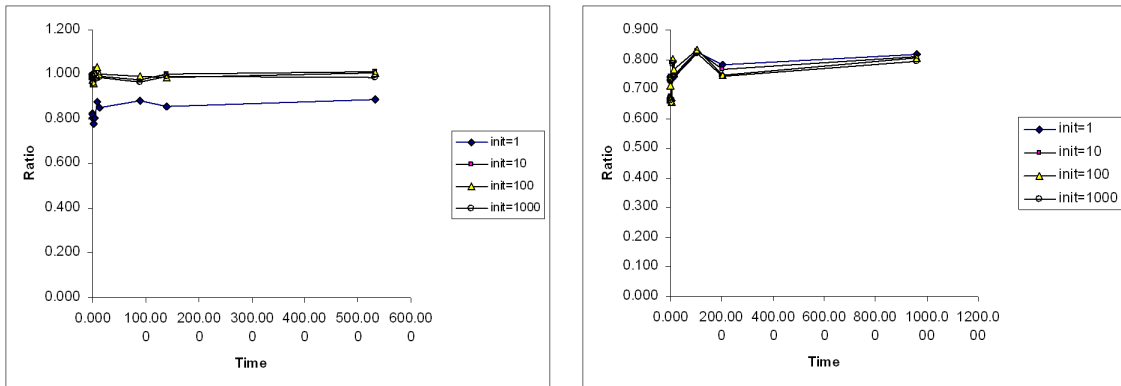


Figure 7: Two cops on trees: ratio of execution times for various initializations of C and R . Let $T_e(init)$ be the execution time of CADR (resp. CAAR) when C (resp. C and R) is initialized at value $init$. The horizontal axis is $T_e(0)$. The vertical axis is the ratio $T_e(init)/T_e(0)$ for various $init$ values. Left panel: CAAR; right panel: CADR.

7 Conclusion

We have studied the cops and robber game from the computational point of view. Our main goal is to compute the cost of drunkenness; to this end, however, we have used algorithms which have more general applicability, in computing optimal strategies for both the cops (playing against a drunk or adversarial robber) as well as for the adversarial robber. These algorithms offer an alternative, computational approach to the theoretical determination of the cost of the drunkenness (which can be achieved in relatively few, special cases).

While both the CAAR and CADR algorithms have been previously available in the literature, apparently their similarity has not been previously noticed. While implementations of the value iteration algorithm are available in the public domain, we believe CADR (our implementation of value iteration specifically for the cops and drunk robber problem) is the first one publicly available; we believe the same is true of CAAR (our implementation of Hahn and MacGillivray’s algorithm for the cops and adversarial robber problem).

Our implementation can be significantly improved. A version which can handle an arbitrary number of cops is desirable; such an implementation must be written in faster language than Matlab (C, C++ and Java are prime candidates) and careful coding optimization should be applied to turn a final application capable of handling truly large graphs and more than two cops. Ultimately however, the exact algorithms presented in this report will be defeated by the curse of dimensionality. To handle large numbers of cops (and robbers) *approximate* algorithms must be developed. These algorithms will be *suboptimal* and will likely make use of *heuristics*; hence a careful theoretical analysis will be necessary to establish performance guarantees³. These tasks we intend to undertake in the future.

An additional case of interest is when the cops chase an *invisible* robber. This problem is very important for applications but also computationally much harder than the one we have examined in the current report. In a future publication we hope to present algorithms which will handle this case.

References

- [1] J. Bertsekas and J. Tsitsiklis. *Parallel and Distributed Computation*. Addison-Wesley, 1989.
- [2] J.H. Eaton and L.A. Zadeh, Optimal pursuit strategies in discrete-state probabilistic systems, *Trans. ASME Ser. D, J. Basic Eng*, vol. 84, pp. 23–29, 1962.
- [3] G. Hahn and G. MacGillivray, A note on k -cop, l -robber games on graphs, *Disc. Math.*, **306**, 2492–2497, 2006.
- [4] R.A. Howard, *Dynamic programming and Markov process*, MIT Press, 1960.
- [5] L. Kallenberg, Markov Decision Processes, <http://www.math.leidenuniv.nl/~kallenberg/Survey%20MDP.pdf>
- [6] Ath. Kehagias and P. Pralat, “Some remarks on cops and drunk robbers”, Submitted for publication.
- [7] R. Pallu de la Barriere. *Optimal Control Theory*. Dover, 1980. (1998), 253–268.
- [8] R. Nowakowski and P. Winkler, Vertex to vertex pursuit in a graph, *Disc. Math.* **43** (1983) 230–239.
- [9] M. L. Puterman. *Markov Decision Processes*, Wiley, 1994.
- [10] A. Quilliot, Jeux et pointes fixes sur les graphes, Ph.D. Dissertation, Université de Paris VI, 1978.
- [11] D. J. White, *Markov decision processes*, Wiley, 1993.

³Let us give just one example of such an algorithm, for the case of drunk robber. We use the following simple heuristic: the cop always randomly chooses a vertex among those which are closer to the robber. If the max degree of G is d , then w.p. at least $\frac{1}{d}$ the cop-robber distance decreases and w.p. no greater than $\frac{d-1}{d}$ it remains the same. Hence the expected cop-robber distance decreases monotonically and, if $c(G) = 1$, we can expect it to go to zero (this has been mentioned by P. Winkler) . However, this heuristic will not necessarily work for k cops when $k = c(G) > 1$. Furthermore, even for $c(G) = 1$, this heuristic does not provide a method for optimal initialization of cop position. But perhaps a theoretical analysis would provide refinements of the heuristic which would answer these questions. .