

# The CopsRobber Matlab Package: Documentation

Ath. Kehagias and P. Pralat

9/6/2011

## Abstract

The package `CopsRobber` is a small collection of Matlab functions which perform various calculations related to the CR game. In this game one or more cops move along the edges of 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  $G$ ). `CopsRobber` performs computations such as: find an optimal strategy for cops facing an adversarial or drunk robber, simulate a CR game, estimate the *cost of drunkenness* etc.

## 1 Introduction

The `CopsRobber` package can be downloaded from <http://users.auth.gr/~kehagiat/GraphSearch/>). It comes in a single zipped file, which contains the Matlab functions, the documentation (i.e., the document you are currently reading) and a technical report which describes in more detail the algorithms and the theory behind them. Installation is quite simple: unpack the contents of the zip file into a directory and add it to the Matlab path.

`CopsRobber` contains functions to perform various computation related to the *Cops and Robber* (CR) game [7, 9]. This game can be described as follows. On an undirected graph  $G$ ,  $k$  cops and a robber move in turns; 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 robber is *visible* to the cops, i.e., they always know his position; similarly, the robber always knows the cops' positions. The game consists of rounds and each round has two turns.

1. In the initial or 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 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 the version with the *drunk* robber, who performs a random walk on  $G$  and does not attempt to avoid capture.

## 2 Contents of the Package

The main `CopsRobber` functions are the following.

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

Table 1

Several auxiliary functions, not listed here, are also used. Let us note that we use the function `FloydSPR`, an implementation of the Floyd shortest paths algorithm, written by Weihuang Fu and available through the Matlab Central file exchange.

We give a detailed description of the usage of the above function in Section 4. However we first give, in Section 3, a short summary of the theory behind the functions. This section can be skipped at first reading but the user will probably need to read it at some point in order to understand what the `CopsRobber` functions actually compute.

### 3 Theory

The cops and *adversarial* robber game has been introduced independently by Nowakowski [7] and Quilliot [9]. Various versions of the problem have been studied and many papers published. For some pointers to the literature, see our technical report [4] (you can download it from <http://users.auth.gr/~kehagiat/GraphSearch/>) and our paper [5]. The algorithm we call CAAR has been introduced in [3].

The cops and *drunk* robber game has also been studied as a special case of *Markov Decision Processes* (MDP), which has been studied extensively. Good book length treatments of MDP include [1, 8, 10]. Especially for the cops and drunk robber problem see [2], where the algorithm we call CADR has been introduced.

We will only give here a very short summary of the mathematical concepts involved in the problem; for a more detailed treatment see the attached technical report.

#### 3.1 Notation and Definitions

We start with some notation. The game is played on  $G = (V, E)$ , an undirected, connected graph without loops.

1.  $N(v)$  denotes the neighborhood of node  $v \in V$  and  $N^+(v) = N(v) \cup \{v\}$  (the *closed* neighborhood of  $v$ ).
2. The minimum number of cops required to capture the adversarial robber is called the *cop number* of  $G$  and is denoted by  $c(G)$ .
3. Given  $k$  cops,  $X_t^i$  denotes the position of the  $i$ -th cop at time  $t$  ( $i \in \{1, 2, \dots, k\}$ ,  $t \in \{0, 1, 2, \dots\}$ );  $\mathbf{X}_t = (X_t^1, \dots, X_t^k)$  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).
4.  $Y_t$  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.
5. 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  $Y_t$ .
6. 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).

We start with the adversarial robber. 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:

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

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 that the above quantities are defined under the assumption of *optimal play by both players*.

By “drunk robber” we mean one who performs a random walk on  $G$ ; more specifically, when he is at vertex  $v \in V$ , he moves to  $u \in N(v)$  with probability equal to  $1/|N(v)|$ . We do *not* include  $v$  in  $N(v)$ . For a given initial configuration and for a given strategy of cops, the capture time  $T$  is a random variable. Given initial positions  $x, y$  for cops and robber, let

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

and let the expected  $k$ -capture be

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

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

Finally, the *cost of drunkenness* is

$$F(G) = \frac{\text{ct}(G)}{\text{dct}(G)}; \tag{1}$$

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

### 3.2 Computing the expected capture time for a fixed cop strategy

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

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

and  $P$  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, we define  $\bar{V} = V \cup \{n\}$ , where  $n$  is the *capture state*: the corresponding  $(n+1) \times (n+1)$  transition matrix is

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

Suppose now that a single cop is located in vertex  $x$ . The transition probability matrix now becomes  $\bar{P}(x)$ . This must describe the probability of capture, which can happen in two ways:

1. the robber is located at  $y$  and, in the first phase of the  $t$ -th round, the cop moves into  $x = y$ ; hence the robber is captured w.p.1 and  $\bar{P}_{x,n}(x) = 1$ ,  $\bar{P}_{x,y'}(x) = 0$  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 he is captured w.p.  $P_{y,y'}$  and, 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

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

Especially for the placement round of the game ( $t = 0$ ) we need a different matrix,  $\hat{P}(x)$ , which is the unit matrix with the one of the  $x$ -th row moved to the  $(n+1)$ -th column. Letting  $\pi(t) = [\pi_0(t), \dots, \pi_n(t)]$  (where  $\pi_i(t) = \mathbb{P}(Y_t = i)$  for  $i \in \bar{V}$  and  $t \in \{0, 1, \dots, s\}$ ) and given a strategy  $\mathbf{X} = (x_0, x_1, \dots, x_s)$  and  $\hat{\pi}(0) = [\frac{1}{n}, \dots, \frac{1}{n}, 0]$ , we have

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

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 a *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.

### 3.3 Computing the expected capture time for the optimal cop strategy

The cop will do better (in fact optimally) if he uses a *feedback strategy*, i.e., if he adjusts his next move depending on the current position of both himself and the robber. In this section we give the algorithms by which such a strategy can be computed.

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 in [3]. Changing notation, we now 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. Let us also define

$$\widehat{V}^2 = V \times V - \{(x, x) : x \in V\},$$

(i.e.,  $V^2$  excluding the diagonal) and  $N^+(x) = N(x) \cup \{x\}$  (for all  $x \in V$ ). CAAR consists of the following recursion (for  $i = 1, 2, \dots$ ):

$$\forall (x, y) \in \widehat{V}^2 : R_{x,y}^{(i)} = \max_{y' \in N^+(y)} C_{x,y'}^{(i-1)}, \quad (3)$$

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

$C$  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  $i = 0, 1, 2, \dots$ . Then CAAR computes the solution of the equations

$$\forall (x, y) \in \widehat{V}^2 : R_{x,y} = \max_{y' \in N^+(y)} C_{x,y'}, \quad (5)$$

$$\forall (x, y) \in \widehat{V}^2 : C_{x,y} = 1 + \min_{x' \in N^+(x)} R_{x',y}, \quad (6)$$

$$\forall x \in V : C_{x,x} = R_{x,x} = 0. \quad (7)$$

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  $dct(G) = \min_x \frac{\sum_y ct_{x,y}(G)}{n}$ . We now use  $C_{x,y}$  to denote  $dct_{x,y}(G)$ . In other words  $C_{x,y}$  ( $R_{x,y}$ ) is *expected* game duration after the cop's (robber's) move. Recall that  $P_{y,y'}(x)$  is the probability of the robber transiting from  $y$  to  $y'$ , given that the cop is at  $x$ . The analog of (3)-(4) is

$$\forall (x, y) \in \widehat{V}^2 : R_{x,y}^{(i)} = \sum_{y' \in N(y)} P_{y,y'}(x) C_{x,y'}^{(i-1)}, \quad (8)$$

$$\forall (x, y) \in \widehat{V}^2 : C_{x,y}^{(i)} = 1 + \min_{x' \in N^+(x)} R_{x',y}^{(i)} \quad (9)$$

and the analog of (5)-(7) is

$$\forall (x, y) \in \widehat{V}^2 : R(x, y) = \sum_{y' \in N(y)} P_{y,y'}(x) C_{x,y'}, \quad (10)$$

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

$$\forall x \in V : C_{x,x} = R_{x,x} = 0. \quad (12)$$

Actually (8)-(9) can be simplified. Since the drunk robber does not choose his moves, we can eliminate  $R_{x,y}^{(i)}$  from (10)-(11) and obtain the CADR algorithm recursion:

$$\forall (x, y) \in \widehat{V}^2 : C_{x,y}^{(i)} = 1 + \min_{x' \in N^+(x)} \left( \sum_{y' \in N(y)} P_{y,y'}(x') C_{x',y'}^{(i-1)} \right). \quad (13)$$

Eq. (13) is a version of the *value iteration* algorithm, introduced in [2] and further studied in [1, 6, 8, 10].

Both CAAR and CADR can be generalized for the case of  $k$  cops, replacing  $x$  by a  $k$ -tuple  $\mathbf{x} = (x_1, \dots, x_k)$ . Both algorithms provide 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 (13). CAAR similarly provides an optimal robber strategy  $W_{x,y}$ .

The algorithms are described above in their *Jacobi* version, i.e., the  $(i-1)$ -th matrix  $C^{(i-1)}$  is stored and used in the  $i$ -th iteration to compute  $C^{(i)}$ . An alternative approach is the *Gauss-Seidel* iteration, where a single copy of  $C$  is stored and its elements are updated “in place”. Our package implements both the Jacobi and Gauss-Seidel versions of the algorithms.

## 4 Usage

### 4.1 Nomenclature and Conventions

We will give detailed description for the usage of CopsRobber functions below, but it will be useful to remember the following conventions.

The main quantities of interest always are  $C, R$  (the cops’ and robber’s target functions). For the case of a single cop these quantities are denoted by  $\mathbf{C}$  and  $\mathbf{R}$ , respectively. Depending on whether the problem involves an adversarial or drunk robber we use the suffixes **a** or **d**. For example  $\mathbf{C}_a$  means the cop’s cost when facing and adversarial robber. In our implementation  $\mathbf{C}$  and  $\mathbf{R}$  are matrices. For a single cop, they are  $n \times n$  matrices, with  $\mathbf{C}(\mathbf{m}, \mathbf{n})$  denoting the expected duration of the game when the cop is at node  $\mathbf{m}$  and the robber at  $\mathbf{n}$ ; for two cops, they are  $n \times n \times n$  matrices, with  $\mathbf{C}(\mathbf{m1}, \mathbf{m2}, \mathbf{n})$  denoting the expected duration of the game when the first cop is at node  $\mathbf{m1}$ , the second at  $\mathbf{m2}$  and the robber at  $\mathbf{n}$ . We use similar conventions for  $\mathbf{R}$ , wherever applicable. In certain cases (drunk robber) there is no occasion to use  $\mathbf{R}$  and in such cases the *empty matrix*  $\mathbf{R}=[ ]$  may be used in its place (the empty matrix is Matlab construct).

Also of interest are  $U, W$  (the cops’ and robber’s feedback strategies). These are also matrices, of size  $n \times n$  or  $n \times n \times n$ , depending on whether we have one or two cops. These are denoted by  $\mathbf{U}$  and  $\mathbf{W}$  except that, if we have two cops, we need two such matrices (one describes the strategy for cop no.1 and the second for cop no.2) and then they are denoted as  $\mathbf{U1}$  and  $\mathbf{U2}$ .

$P$  will always denote the transition probability matrix for the cop-free random walk on graph  $G$  and in our code will be denoted by  $\mathbf{P}$ .

### 4.2 File Format

CopsRobber functions will need to read two kinds of files.

1. The first kind are *graph files*, used to describe a graph  $G$ . These are ASCII files, with one line per edge, each line containing the numbers of the two respective nodes (nodes are always numbered as  $1, 2, \dots, n$ ); for example see the file `Edge01b.txt`. The user must make sure that a file of this type correctly represents an undirected, connected graph without loops; if this is not the case, our code may produce wrong and unexpected results.
2. The second kind are *strategy files*, used to describe a fixed strategy for the cop(s). These are also ASCII files with number of lines equal to the number of rounds the strategy will be played; in other words the first line describes the move(s) for round 1, ..., the  $T$ -th line the move(s) for round  $T$ . If the strategy is for a one-cop game, the  $t$ -th line contains a single node number: where the cop should move at the

$t$ -th round ; for a two-cop game the line contains two such numbers. The user must make sure that the strategy is *admissible* (i.e., successive cop moves take place only along edges), otherwise the code may again produce wrong and unexpected results.

Usually graph and strategy files are read by Matlab's `load` command and copied into appropriate matrices which are then used as input to the `CopsRobber` functions.

### 4.3 Syntax of the Functions

**CRstrat:** This function computes the expected and maximum capture time when the cop(s) use a fixed strategy. It uses (2).

```
% function [p, Texp, Tmax]=CRstrat(P,X)
% compute expected capture time given fixed strategy X
%
% P:    transition probability matrix for RW on graph G
% X:    strategy as T-by-K matrix (T length of game, K number of cops)
% p:    prob. of robber in node n at time t (T-by-N)
% Texp: expected capture time
% Tmax: maximum capture time
```

**CRcaar:** This function computes optimal strategies for one or two cops and an adversarial robber; the optimal game duration is also computed. It is an implementation of the recursion (3)-(4) for one cop, or its generalization for two cops.

```
% function [U1,U2,W,C,R]=CRcaar(P,T,thrs1,inf1,copnum,itermethod)
% optimal costs and strategies for the Cops + Adversarial Robber game
% it can handle 1 or 2 cops and use Jacobi or Gauss-Seidel value iteration
%
% P:    N-by-N transition prob. matrix of G (used only for node connectivity)
% T:    max number of iterations
% thrs1: threshold to break iteration
% inf1: initial value for C
% copnum: number of cops (allowed: 1 or 2)
% itermethod: iteration method (1: Jacobi, 2: Gauss-Seidel)
%
% ONE COP CASE:
% U1:    N-by-N opt. cop move matrix; U1(mc,mr) is optimal move starting from (mc,mr)
% U2:    empty matrix
% W:    N-by-N opt. robber move matrix; W(mc,mr) is optimal move starting from (mc,mr)
% C:    N-by-N opt. capture time matrix; C(mc,mr) is OCT starting from (mc,mr)
%        and cop's turn
% R:    N-by-N opt. capture time matrix; R(mc,mr) is OCT starting from (mc,mr)
%        and robber's turn
%
% TWO COPS CASE:
% U1:    N-by-N-by-N opt. cop1 move matrix; U1(mc1,mc2,mr) is optimal move starting
%        from (mc1,mc2,mr)
% U2:    N-by-N-by-N opt. cop2 move matrix; U2(mc1,mc2,mr) is optimal move starting
%        from (mc1,mc2,mr)
% W:    N-by-N-by-N opt. robber move matrix; W(mc1,mc2,mr) is optimal move starting
%        from position (mc1,mc2,mr)
% C:    N-by-N-by-N opt. capture time matrix; C(mc1,mc2,mr) is OCT starting from
```

```

%           (mc1,mc2,mr) and cops' turn
% R:       N-by-N-by-N opt. capture time matrix; R(mc1,mc2,mr) is OCT starting from
%           (mc1,mc2,mr) and robber's turn

```

**CRcadr:** This function computes optimal strategies for one or two cops and a drunk robber; the expected game duration is also computed. It is an implementation of the recursion (13) for one cop, or its generalization for two cops.

```

% function [U1,U2,C]=CRcadr(P,T,thrs1,inf1,copnum,itermethod)
% optimal costs and strategies for the Cops + Drunk Robber game
% it can handle 1 or 2 cops and use Jacobi or Gauss-Seidel value iteration
%
% P:       N-by-N transition prob. matrix of G (used only for node connectivity)
% T:       max number of iterations
% thrsh1:  threshold to break iteration
% inf1:    initial value for C
% copnum:  number of cops (allowed: 1 or 2)
% itermethod: iteration method (1: Jacobi, 2: Gauss-Seidel)
%
% ONE COP CASE:
% U1:      N-by-N optimal cop move matrix; U1(mc,mr) is optimal move starting
%           from position (mc,mr)
% U2:      empty matrix
% C:       N-by-N optimal capture time matrix; C(mc,mr) is OCT starting from position
%           (mc,mr) and cop's turn
%
% TWO COPS CASE:
% U1:      N-by-N-by-N optimal cop1 move matrix; U1(mc1,mc2,mr) is optimal move
%           starting from position (mc1,mc2,mr)
% U2:      N-by-N-by-N optimal cop2 move matrix; U2(mc1,mc2,mr) is optimal move
%           starting from position (mc1,mc2,mr)
% C:       N-by-N-by-N optimal capture time matrix; C(mc1,mc2,mr) is OCT starting
%           from position (mc1,mc2,mr) and cops' turn

```

**CRsim:** This function simulates a game between one or two cops and a drunk or adversarial robber. The players follows feedback strategies (provided as matrices, in the format computed by the CRcaar and CRcadr functions).

```

% function [X1,X2,Y,Z,CT]=CRsim(x10,x20,y0,P,U1,U2,W,Tmax,output)
% simulate (one OR two) cops vs. (adversarial OR drunk) robber
%
% x10:     initial position of cop1
% x20:     initial position of cop2 (use x20<0 for one cop play)
% y0:      initial position of robber
% P:       N-by-N transition prob. matrix of G (used only for node connectivity)
% U1:      N-by-N optimal move matrix for cop1; U1(mc,mr) is optimal move starting from
%           position (mc,mr)
% U2:      N-by-N optimal move matrix for cop2; (if one cop game, use anything)
% W:       N-by-N optimal robber move matrix; (for drunk robber use W=[])
% Tmax:    max duration of game
% output:  1 (positions of cops+robber displayed) or 0 (positions of cops+robber

```

```

%           not displayed)
%
% ONE COP CASE:
% X1:       Tmax-by-1 cop position matrix
% X2:       empty matrix
% Y:       Tmax-by-1 robber position matrix
% Z:       Tmax-by-1 cop-to-robber distance matrix
% CT:       capture time
%
% TWO COPS CASE:
% X1:       Tmax-by-1 cop1 position matrix
% X2:       Tmax-by-1 cop2 position matrix
% Y:       Tmax-by-1 robber position matrix
% Z:       Tmax-by-1 min cop-to-robber distance matrix
% CT:       capture time

```

**CRcod:** This function computes the cost of drunkenness (as given by (1)).

```

% function [F,CT,DCT,mopta,moptd]=CRcod(Ca,Ra,Cd)
% computes the cost of drunkenness
%
% Ca:       matrix of cop-optimal capture times, either N-by-N (1 cop) or
%           N-by-N-by-N (2 cops) / ADVERSARIAL robber
% Ra:       matrix of rob-optimal capture times, either N-by-N (1 cop) or
%           N-by-N-by-N (2 cops)/ ADVERSARIAL robber
% Cd:       matrix of cop-optimal expected capture times, either N-by-N (1 cop) or
%           N-by-N-by-N (2 cops) / DRUNK robber
% F:       the cost of drunkenness, ratio of adversarial to drunk optimal capture times
% mopta:    optimal initial positions, adversarial robber; 1-by-2 (1 cop)
%           or 1-by-3 (2 cops)
% moptd:    optimal initial positions, drunk robber; 1-by-1 (1 cop) or 1-by-2
%           (2 cops)

```

**CRcheq:** This function checks that the  $C$  and  $R$  expected game durations (as computed by `CRcaar` or `CRcadr`) satisfy the optimality equations (5)-(7) or (10)-(12).

```

% function [EC,ER]=CRcheq(C,R,P)
% check if C, R are solutions of the (adversarial or drunk) optimality equations
%
% C:       cop cost matrix
% R:       robber cost matrix (use empty matrix for drunk optim. equations)
% E:       satisfaction matrix
%           EC(mc,mr)=0 iff (mc,mr) cop eq. is satisfied
%           ER(mc,mr)=0 iff (mc,mr) rob eq. is satisfied

```

## 5 Examples of Using CopsRobber

Details on use of the functions are given in the Matlab files. The user can also type `CRdemo` in the Matlab command line and examples of the main `CopsRobber` functions will be run with commentary. In this section let us present the usage of the main commands by examples along with usage comments.

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  $T_{exp}$  and  $T_{max}$ , 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  $T_{exp}$  and  $T_{max}$ .

```
>> 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;
>> itermethod=1;
```

$T$  is the maximum number of iterations for which the algorithm will run; `epsilon` is the termination criterion  $\varepsilon$ ; `initial` is the initial value for  $C^{(0)}$ ,  $R^{(0)}$ ; `copnum` is the number of cops; `itermethod=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,itermethod);
```

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 (5)-(7). 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 (5)-(7).

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  $R$  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.

## 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, *Discrete Mathematics*, **306**, 2492–2497, 2006.

- [4] Ath. Kehagias and P. Pralat, “Cops and visible robbers: a computational approach”, Technical Report, available at <http://users.auth.gr/~kehagiat/GraphSearch/TRCODvis.pdf>.
- [5] Ath. Kehagias and P. Pralat, “Some remarks on cops and drunk robbers”, Submitted for publication.
- [6] R. Pallu de la Barriere. *Optimal Control Theory*. Dover, 1980.
- [7] R. Nowakowski and P. Winkler, Vertex to vertex pursuit in a graph, *Discrete Mathematics* **43** (1983) 230–239.
- [8] M. L. Puterman. *Markov Decision Processes*, Wiley, 1994.
- [9] A. Quilliot, Jeux et pointes fixes sur les graphes, Ph.D. Dissertation, Université de Paris VI, 1978.
- [10] D. J. White, *Markov decision processes*, Wiley, 1993.