Computing optimal strategy for cop in the game of Cop v.s. Gambler

Shen-Fu Tsai

parity@gmail.com

October 16, 2018

Abstract

We present two efficient algorithms that compute the optimal strategy for cop in the game of Cop v.s. Gambler where the gambler's strategy is not optimal but known to the cop. The first algorithm is analogous to BellmanFord algorithm[3] for single source shortest path problem and runs in $O(|V(G)||E(G)|)$ time. The second is analogous to Dijkstra's algorithm[4] and runs in $O(|E(G)| + |V(G)| \log |V(G)|)$ time. Compared with each other, they are more suitable for sparse and dense graphs, respectively.

1   Introduction

In the probabilistic version of game of graph pursuit[1], a cop plays against a gambler on a graph G. Before the game starts, the cop picks and occupies a vertex from G. In each round of the game, the cop selects and moves to a adjacent vertex or stays at the same vertex, and the gambler chooses to occupy a vertex randomly based on a time-independent distribution, or *gamble*, known to the cop. The gambler is not restricted to only adjacent vertices. Whenever they occupy the same vertex at the same time the cop wins.

It is known that on a connected graph G, the cop can win in less than or equal to $n = |V(G)|$ expected rounds, so the cop *will* win in exactly n expected rounds because the optimal gamble for the gambler is uniform distribution. However, we note that the cop's strategy proposed in [1] that guarantees n expected chase time may not be optimal when a non-uniform gamble is employed. An example is a chain v_0, v_1, v_2, v_3 with $(p_0, p_1, p_2, p_3) = (0.3, 0.7, 0, 0)$ and the cop starting at v_0. By the strategy in [1], the cop should stay at v_0 and expect to win in $10/3$ time. However if she moves to v_1 and stay there, the expected time is $1 + 0.7/0.7 = 2$.

2   Optimal Cop Strategy

We describe two algorithms to compute the optimal strategy for cop starting at *every* vertex $v \in V(G)$ given G and gamble $\{p_v : v \in V(G)\}$, where p_v is the probability that the gambler selects to occupy vertex v. Our algorithms work for both directed and undirected graphs.

We can always represent the optimal strategy for cop when arriving vertex v as to the next vertex $u \in N(v)$ to move to in the next round, where $N(v)$ is the set of adjacent vertices of v plus v itself.

2.1 $O(|V(G)||E(G)|)$ Algorithm

Analogous to Bellman-Ford algorithm[3] for single source shortest path problem, in each of the $O(|V(G)|)$ iterations we update the strategy and chase time for vertex u based on v for each edge $(u, v) \in E(G)$. It runs faster on sparse graphs than the next algorithm in Section 2.2.

---

for all $v \in V(G)$ **do**
    $T_1(v) = 1/p_v$, $\pi(v) = v$
end for
for $i = 2, 3, \ldots$ **do**
    $update = False$
    for all $v \in V(G)$ **do**
        $T_i(v) = T_{i-1}(v)$
        if $T_i(v) > 1 + (1 - p_v) \min_{u \in N(v)} T_{i-1}(u)$ **then**
            $T_i(v) = 1 + (1 - p_v) \min_{u \in N(v)} T_{i-1}(u)$
            $\pi(v) = \arg \min_{u \in N(v)} T_{k-1}(u)$
            $update = True$
        end if
    end for
    if not $update$ **then**
        return
    end if
end for

2.1.1   Analysis

In this section we show the correctness and efficiency of our algorithm. First define $T(v)$ as the optimal expected chase time when the cop enters vertex v.

Lemma 1. *For each $v \in V(G)$,*

$$T(v) = 1 + (1 - p_v) \min_{w \in N(v)} T(w).$$

Moreover if $u \in N(v)$ and $T(v) = 1 + (1 - p_v)T(u)$ then moving to u after entering v is optimal.

The following lemma says that our algorithm will never obtain any $T_i(v)$ that is smaller than $T(v)$.

Lemma 2. *For each $v \in V(G)$ and $i \geq 0$, $T_i(v) \geq T(v)$.*

Proof. We prove the lemma by showing that every $T_i(v)$ is *achievable*. For $i = 0$ it is true, because $T_0(v) = 1/p_v$ is the expected chase time for the cop who stays at vertex v forever. Assume $T_{k-1}(v)$ is achievable,

$$T_k(v) = 1 + (1 - p_v) \min_{u \in N(v)} T_{k-1}(u).$$

By induction, $1 + (1 - p_v) \min_{u \in N(v)} T_{k-1}(u)$ on the right hand side is the upper bound of chase time if the cop moves from vertex v to $\arg\min_{u \in N(v)} T_{k-1}(u)$ when she doesn't capture the gambler in the current round. Therefore $T_k(v)$ is achievable too. $\qquad\square$

Define a *chase path* as a path $\{v_0, v_1, \ldots, v_k\}$ with

$$T(v_i) = 1 + (1 - p_{v_i})T(v_{i+1}), v_{i+1} \in N(v_i)$$

for each $i \in [0, k-1]$ and

$$T(v_k) = 1 + (1 - p_{v_k})T(v_k)$$

Clearly a chase path starting from v and ending at u is an optimal path for cop to start chasing the gambler at v. Ending at u implies that it is optimal for cop to stay at u forever.

Lemma 3. *If $\{v_0, v_1, \ldots, v_k\}$ is a chase path, then so is $\{v_1, \ldots, v_k\}$.*

A chase path starting from v is a *shortest chase path* if these is no shorter chase path that starts from v. We say a vertex v has *shortest chase length k* if the shortest chase path starting from v has length k.

Lemma 4. *If $\{v_0, v_1, \ldots, v_k\}$ is a shortest chase path, then so is $\{v_1, \ldots, v_k\}$.*

Lemma 5. *If v has shortest chase length $k > 1$, then it has a neighbor $u \neq v$ with shortest chase length $k - 1$ such that u follows v in the shortest chase path.*

Proof. Let path $\{v_0 = v, v_1, \ldots, v_{k-1}\}$ be a shortest chase path. By Lemma 4, $\{v_1, \ldots, v_{k-1}\}$ is a shortest chase path of length $k - 1$. By definition $v_1 \in N(v)$, therefore $u = v_1$. $\qquad\square$

Lemma 6. *If v has shortest chase length k, then $T_k(v) = T(v)$ and the computed $\pi(v) \in N(v)$ remains unchanged thereafter satisfying*

$$T(v) = 1 + (1 - p_v)T(\pi(v)).$$

Proof. For $k = 1$ the statement holds because if a vertex has shortest chase length 1, then

$$p_v = \max_{u \in N(v)} p_u$$

as otherwise moving to a neighbor w of v with higher probability is better than staying at v for good.

For a vertex v with shortest chase length $k > 1$, by Lemma 5 it has a neighbor $u \neq v$ with shortest chase length $k - 1$. By induction assumption $T_{k-1}(u) = T(u)$,

$$
\begin{aligned}
T_k(v) &= 1 + (1 - p_v) \min_{w \in N(v)} T_{k-1}(w) \\
&\leq 1 + (1 - p_v)T_{k-1}(u) = 1 + (1 - p_v)T(u) \\
&= T(v)
\end{aligned}
$$

The last equality stems from the fact that u follows v in the shortest chase path. On the other hand, by Lemma 2, $T_k(v) \geq T(v)$, so $T_k(v) = T(v)$. It is obvious from the description of our algorithm that $\pi(v)$ satisfies

$$T(v) = 1 + (1 - p_v)T(\pi(v)).$$

Moreover $\pi(v)$ is only updated in round $j + 1$ when $T_{j+1}(v) < T_j(v)$, so as $T_k(v) = T_{k+1}(v) = \ldots$, $\pi(v)$ does not change beyond round k. $\qquad\square$

Lemma 7. *A shortest chase path is simple.*

Proof. Suppose on the contrary, that a shortest chase path P passes some vertex v twice. Then deleting the first occurrence of v up to but excluding its second occurrence gives another chase path which is shorter than P. $\qquad\square$

Theorem 1. *The algorithm in Section 2.1 correctly computes $\pi(v)$ for all $v \in V(G)$ in $O(|V(G)||E(G)|)$ time.*

Proof. By Lemma 7, all shortest chase paths have lengths less than or equal to n. So by Lemma 6 after round n the computed $\pi(v)$ satisfies

$$T(v) = 1 + (1 - p_v)T(\pi(v))$$

and $T_n(v) = T_{n+1}(v) = T_{n+2}(v) = \ldots$, i.e. the algorithm terminates no later than round $n + 1$. Each round takes $O(|E(G)|)$ time, so the overall time complexity is $O(|V(G)||E(G)|)$. $\qquad\square$

2.2 $O(|E(G)| + |V(G)| \log |V(G)|)$ Algorithm

We present another algorithm computing optimal cop strategy. The time complexity is $O(|E(G)| + |V(G)| \log |V(G)|)$, so it is more suitable for dense graphs. Analogous to Dijkstra's algorithm[4] for single source shortest path, in each of the $|V(G)|$ iterations we only update the strategy and chase time of vertices that links directly to a specific vertex.

> **for all** $v \in V(G)$ **do**
> $t(v) = 1/p_v$, $\pi(v) = v$
> **end for**
> $S = V(G)$
> **while** $|S| > 0$ **do**
> $u = \arg\min_{v \in S} t(v)$
> $S = S - u$
>   **for all** $w \in S$ such that $(w, u) \in E(G)$ **do**
>   **if** $t(w) > 1 + (1 - p_w)t(u)$ **then**
> $t(w) = 1 + (1 - p_w)t(u)$
> $\pi(w) = u$
>   **end if**
>   **end for**
> **end while**

2.2.1 Analysis

In this section we show the correctness and efficiency of this algorithm.

Lemma 8. *If u follows v in a chase path and $T(u) = T(v)$, then $p_u = p_v$ and $T(u) = T(v) = 1/p_u$.*

A chase path always goes from long to short expected chase time.

Lemma 9. *If v follows u in a chase path, then $T(v) \geq T(u)$.*

Proof. Suppose otherwise. Since v follows u, $T(v) = 1 + (1 - p_v)T(u)$, or $T(u) = (T(v) - 1)/(1 - p_v) > T(v)$. So $T(v) > 1/p_v$, a contradiction. $\square$

Lemma 10. *For every $u \in V(G) - S$, $t(u) = T(u)$.*

Proof. It clearly holds when $|S| = n$. Suppose the statement holds for $|S| = k$, and assume $|S| = k - 1 > 0$ at the beginning of some iteration of the while loop. Let $w = \arg\min_{u \in S} t(u)$. It suffices to show that $t(w) = T(w)$.

Assume $t(w) > T(w)$. There exists a chase path $P = \{w, u_0, \ldots, u_m\}$. If P and $V(G) - S$ are disjoint, then by Lemma 9, $T(w) = T(u_0) = \ldots = T(u_m)$. Since u_m is the end of the chase path $T(u_m) = 1/p_{u_m}$, a contradiction as $t(u_m) \leq 1/p_{u_m}$. If $v \in P$ is the vertex closest to w in P such that $v \in V(G) - S$,

5

let u follows v in P and so $u \in S$. By Lemma 9 $T(u) = T(w)$. If $u = w$, then $t(w)$ has already been updated to $1 + (1 - p_w)t(v) = 1 + (1 - p_w)T(v) = T(w)$. If $u \neq w$, then by Lemma 8, $T(w) = 1/p_w \geq t(w)$.

We are then done here, because the computed $t(w)$ is always achievable, i.e. $t(w) \geq T(w)$. $\qquad\square$

Theorem 2. *The algorithm correctly computes $T(v)$ and optimal strategy for all $v \in V(G)$ in $O(|E(G)| + |V(G)| \log |V(G)|)$ time.*

Proof. The correctness follows Lemma 10 immediately. To achieve $O(|E| + |V(G)| \log |V(G)|)$ time complexity, S could be kept as a Fibonacci heap[2] with $O(1)$ amortized element update time and $O(\log |V(G)|)$ element removal time. $\qquad\square$

References

[1] Natasha Komarov and Peter Winkler, Cop vs. Gambler, Discrete Math. 339 (2016), 1677-1681.

[2] https://en.wikipedia.org/wiki/Fibonacci_heap

[3] https://en.wikipedia.org/wiki/Bellman

[4] https://en.wikipedia.org/wiki/Dijkstra