Games on graphs

Innovative ideas to tackle reachability and safety games

LINE OF CONTENTS

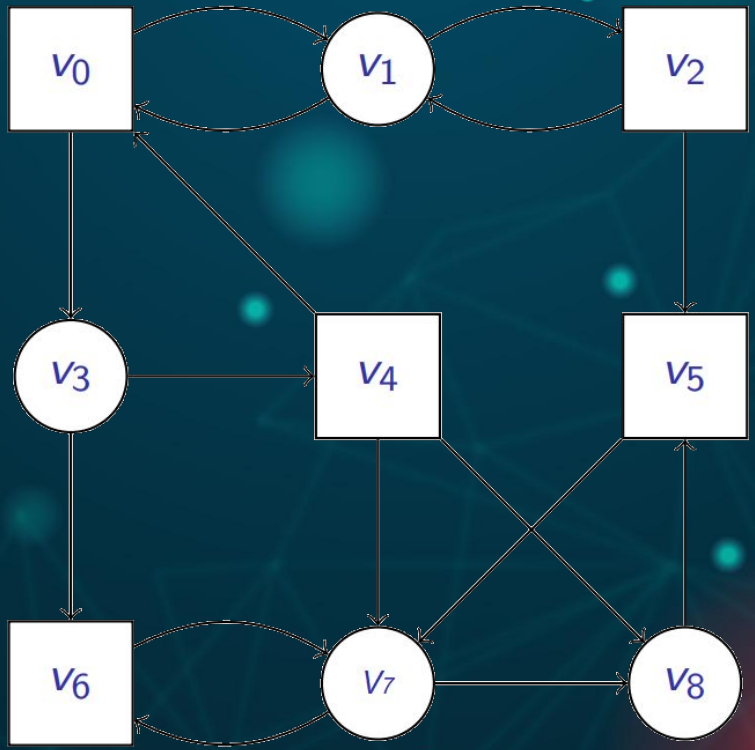
Fundamentals

PROJECT'S SCOPE

- Number of players:
  - ***2 players***
- Interaction:
  - ***Turn-based***
- Information:
  - ***Perfect***
- Nature:
  - ***Deterministic***
- Objective:
  - ***Reachability/Safety***

GAMES ON GRAPHS: FUNDAMENTALS

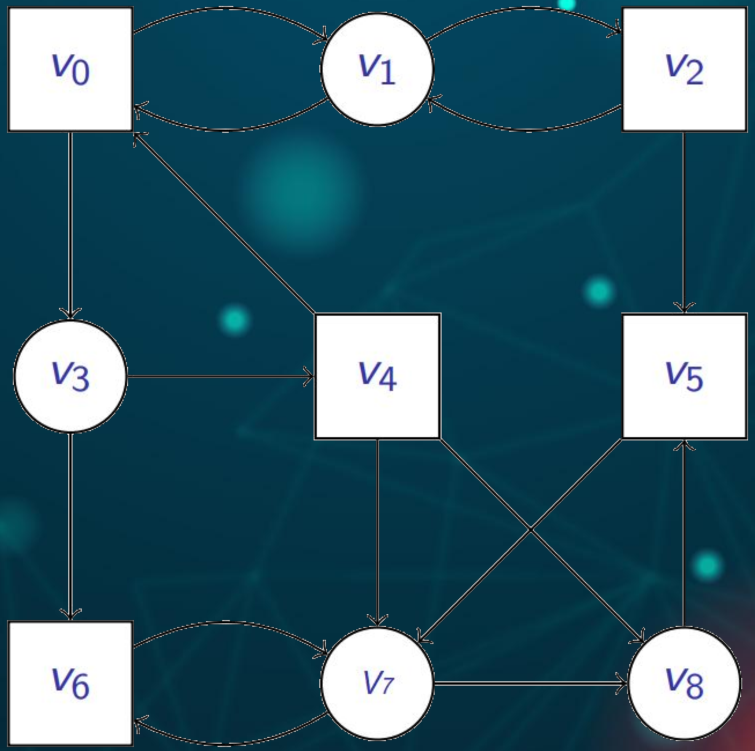
- A ***game*** is played over a (finite) graph $G = (V, E)$, whose vertices are under the control of the two players, i.e., $V = V_0 \cup V_1$
- A ***token*** moves along the vertices and sent to a successor by the controlling player.



GAMES ON GRAPHS: FUNDAMENTALS

- The ***play*** is an infinite sequence of vertices in the graph.
- A winning ***objective*** is a subset Obj $\subseteq V_\omega$ of plays that $Player_0$ wants to occur.
- A sample of an (infinite) play is:

$\Pi = v_0 \cdot v_1 \cdot v_2 \cdot v_1 \cdot v_2 \cdot v_1 \cdot v_0 \cdot \ldots \in V_\omega$



GAMES ON GRAPHS: FUNDAMENTALS

For a subset of vertices $T \subseteq V$:

- ***Reachability***: visit the target set T at least once.
  - $Reach(T) = \{\pi \in V^\omega \mid \exists\ i \in N, \pi_i \in T\}$
- ***Safety***: stay in T forever.
  - $Safe(T) = \{\pi \in V^\omega : \forall\ i \in N, \pi_i \in T\}$

GAMES ON GRAPHS: FUNDAMENTALS

- A ***strategy*** maps finite sequences of vertices into successors and it is of the following form:
  - Player$_0$ strategy $\sigma_0 : V^* \cdot V_0 \rightarrow V$
  - Player$_1$ strategy $\sigma_1 : V^* \cdot V_1 \rightarrow V$
- Strategies "***restricts***" the game only to those play π that are consistent with σ0, that is such that $\pi_{i+1} = \sigma_0 (\pi_0 \cdot \pi_1 \cdot \ldots \cdot \pi_i)$, if $\pi_i \in V_0$.
- For given strategies σ0, σ1, there is only one consistent play starting from v.

GAMES ON GRAPHS: FUNDAMENTALS

- A strategy σ_0 is ***memoryless*** if it is of the form
    - $\sigma_0 : \cancel{V}^* \cdot V \to V$ that is, at every vertex v, the next move does not depend on the past history.
- ***Theorem:***
    - If $v \in Win_0$, then there exists a memoryless strategy σ_0 that is winning from v.

GAMES ON GRAPHS: FUNDAMENTALS

- Formulation of the force function for reachability games:
  - Reach_comp(X) = $\{v \in V_R : E(v) \cap X \neq \emptyset\}$
  - Safety_comp(X) = $\{v \in V_S : E(v) \subseteq X\}$
  - $Force_{Reach}(X)$ = Reach_comp(X) ∪ Safety_comp(X)
- Formulation of the force function for safety games:
  - Reach_comp(X) = $\{v \in V_R : E(v) \subseteq X\}$
  - Safety_comp(X) = $\{v \in V_S : E(v) \cap X \neq \emptyset\}$
  - $Force_{Safety}(X)$ = Reach_comp(X) ∪ Safety_comp(X)

GAMES ON GRAPHS: FUNDAMENTALS

Input:
Graph $G(V, E)$: The graph representing the arena.
Target_reach: The target set for the reachability player.
Begin:

```
Win = Target_reach
while (Win ≠ (Win ∪ force(Win))): do
    Win = Win ∪ force(Win)
end while
return Win
```

- Corresponds to compute the approximate least fixpoint.

Input:
Graph $G(V, E)$: The graph representing the arena.
Target_safe: The target set for the safety player.
Begin:

```
Win = Target_safe
while (Win ≠ (Win ∩ force(Win))): do
    Win = Win ∩ force(Win)
end while
return Win
```

- Corresponds to compute the approximate greatest fixpoint.

Personal Optimizations

Transpose graph optimization

ON WHICH COMPUTATION ARE WE IMPACTING WITH THIS OPTIMIZATION?

$$Reach\_comp(X) = \{v \in V_R : E(v) \cap X \neq \emptyset\}$$

$$Safety\_comp(X) = \{v \in V_S : E(v) \subseteq X\}$$

$$Force_R(X) = \underline{Reach\_comp(X)} \cup Safety\_comp(X)$$

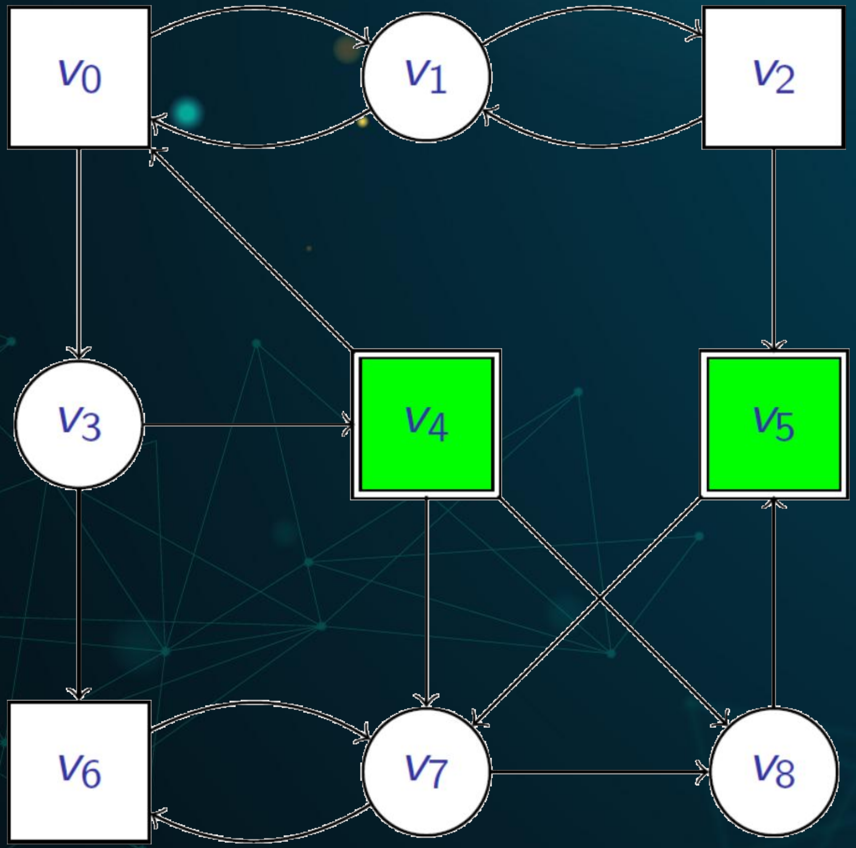
TRANSPOSE GRAPH OPTIMIZATION: COMPUTING REACH_COMP(X)

- Given a graph G and an input target set X, we define the neighborhood of the target set in the transpose graph GT as follows:
  - $N_{GT} := \{ x \in X \mid (x \subseteq \text{Neighbors}\,(GT, x\,)\,) \}$
- With reference to formulation of reachability games, Reach_comp(X) will include all the node $v \subseteq V$ that satisfy the following conditions:
  - a) The node v is controlled by the reachability player (i.e., $v \cap R$).
  - b) The node v belongs to the set N_{GT} (i.e., $v \in N_{GT}$).

⇓

- It is convenient to employ the transpose graph instead of the straight graph to verify whether condition b) is fulfilled.

TRANSPOSE GRAPH OPTIMIZATION: GRAPHICAL INTUITION



- Processed nodes using the "straight" graph:
  - $V_P = \{v_1, v_3, v_7, v_8\}$.
- Processed nodes using the transpose graph:
  - $V_{P'} = \{v_3, v_8\}$.
- Regardless of the data structure employed:
  - Reach_Comp(X) = $\{v_3, v_8\}$.

TRANSPOSE GRAPH OPTIMIZATION: COMPUTING SAFETY_COMP(X)

➢ With reference to formulation of reachability games, Safety_comp(X) will include all the nodes $v \subseteq V$ that satisfy the following conditions:
   ○ a) The node v is controlled by the safety player (i.e., $v \cap S$).
   ○ b) The node v belongs to the set N_{GT} (i.e., $v \in N_{GT}$).
   ○ c) All the outgoing edges of the node $v \subseteq V$ lead to nodes contained in the target set X (i.e., $v \in V \mid E(v) \subseteq X$).

➢ Hybrid approach: we employ the transpose graph to check the fulfillment of condition b), while we use the straight graph to check if condition c) is verified.

Current set optimization

ON WHICH COMPUTATION ARE WE IMPACTING WITH THIS OPTIMIZATION?

$$Reach\_comp(X) = \{v \in V_R : E(v) \cap X \neq \emptyset\}$$

$$Safety\_comp(X) = \{v \in V_S : E(v) \subseteq X\}$$

$$Force_R(X) = \underline{Reach\_comp(X)} \cup Safety\_comp(X)$$

CURRENT SET OPTIMIZATION

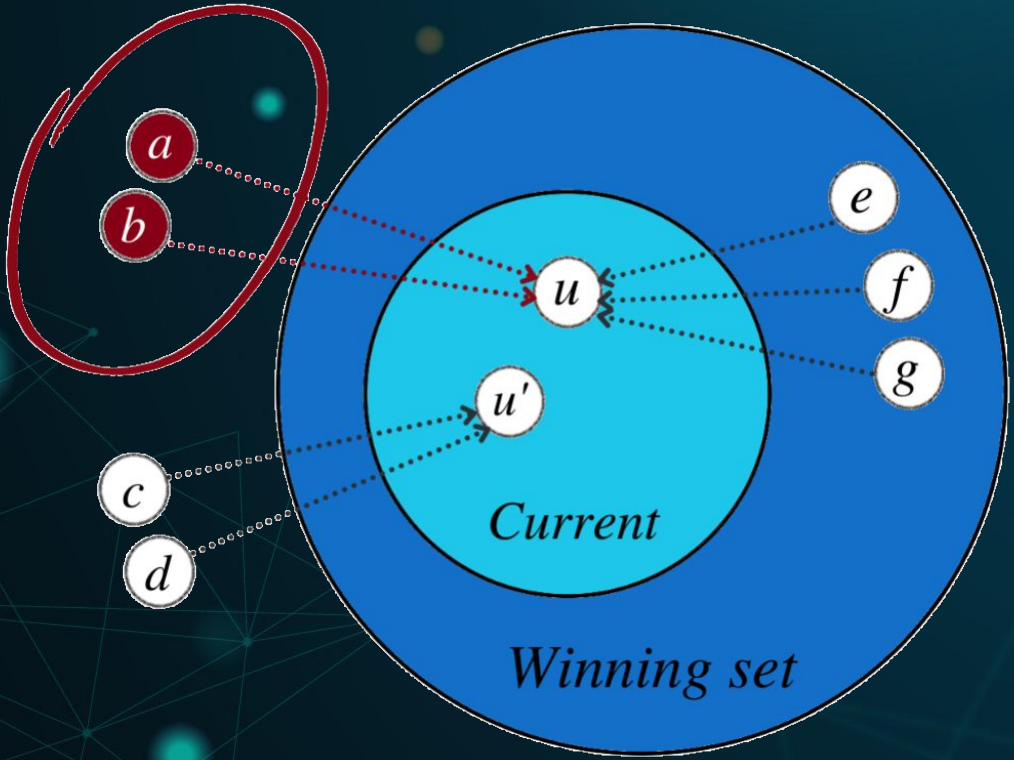
- With reference to the formulation of reachability games:
  - **Current set** := *set of nodes added in the last iteration to the reachability player's winning set.*
  - It corresponds to the $Force_{Reach}(X)$ computed in the algorithm's previous iteration.
  - The Current set is generally a subset of the winning set W.

- We combine this optimization with the transpose graph optimization:
  - We verify the fulfillment of the conditions a) and b) exclusively for the nodes reachable from the Current set, instead of verifying whether these conditions hold for the set of nodes reachable from the entire winning set.

CURRENT SET OPTIMIZATION: PROOF OF CORRECTNESS

- Using this optimization, the nodes candidate to enter the target set T are only the nodes that have not been considered yet.
- The nodes added in the previous iteration fall within the following cases:
  - *The node is not reachable from the target set with a single edge*
    - → The node has not been added to the target set in the previous iteration
  - *The node is not reachable from the target set through two edges*
    - → The node must be necessarily added to the force set (and thus to the target set T) in the previous iteration.

CURRENT SET OPTIMIZATION: GRAPHICAL INTUITION



- Suppose the node considered in the current iteration is node “u”:
  - The only nodes we desire to process are nodes “*a*” and “*b*”.
  - Considering nodes “*e*”, “*f*,” “*g*” is detrimental!
  - Nodes “*c*” and “*d*” will be processed when node “ u’ ” is considered.

“Processed” list optimization

ON WHICH COMPUTATION ARE WE IMPACTING WITH THIS OPTIMIZATION?

$$Reach\_comp(X) = \{v \in V_R : E(v) \cap X \neq \emptyset\}$$

$$Safety\_comp(X) = \{v \in V_S : E(v) \subseteq X\}$$

$$Force_R(X) = Reach\_comp(X) \cup \underline{Safety\_comp(X)}$$

PROCESSED LIST OPTIMIZATION: COMPUTING SAFETY_COMP(X)

- As before, we refer to formulation of reachability games.
- Thus, we define:
  - L := The losing set for the reachability player (i.e., the current set of safety nodes)
  - V_S := The set nodes controlled by the safety player
  - SCL := The set of nodes controlled by the safety player that are solely connected with nodes belonging to L.

⇩

The ***goal*** is to ***compute SCL***, which coincides with Safety_comp(X) in the previous formulation.

PROCESSED LIST OPTIMIZATION: THE PROBLEM

- To verify the condition *c*):
  - We iterate through the set V_S and for each node u ∈ L
    - We examine all the nodes controlled by the safety player solely connected with “u” ∈ L or with another node “ u’ ” ∈ L.
- Possible scenario → A node v is encountered, firstly, as a neighbor of the node “u” and, secondly, of the node “ u’ ”.
  - ***Problem***: *some nodes may be processed multiple times*!

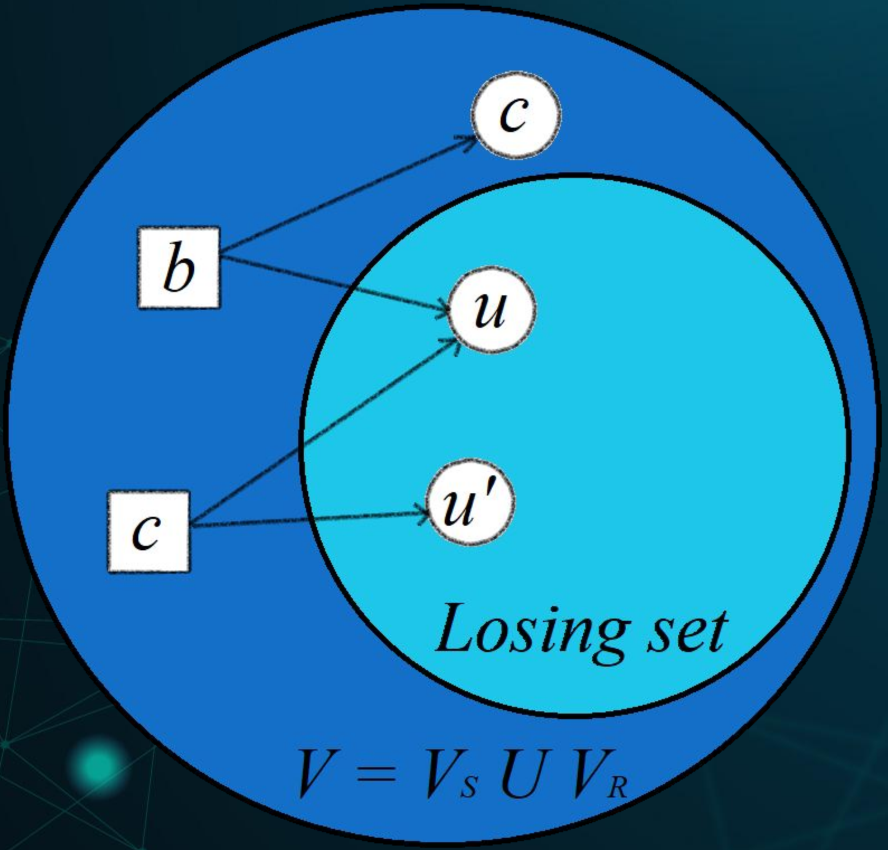
PROCESSED LIST OPTIMIZATION: THE SOLUTION

- The result of the operations carried out ***is independent*** of the fact that the node “v” is a neighbor either of the node “u” or the node “ u’ ”!
- Hence:
  - We insert each considered node into Processed, a list that contains the processed nodes.
  - In doing so, once a node has been already examined as the neighbor of a node “u”, it will not be considered again, if it is encountered as the neighbor of another node “ u’ ”.

⇩

In doing so, the computation of *SCL* (*i.e.*, Safety_comp(X)) has been optimized!

"PROCESSED" LIST OPTIMIZATION: GRAPHICAL INTUITION



- Suppose the node considered in the current iteration is node "u":
    - The node "*b*" is processed. It is not safe since it is not solely connected to nodes in the losing set. We add "*b*" to the "processed" list.
    - The node "*c*" is processed. This node is safe, hence it is added to the set SCL. Then we add this node to the "processed" list.
- Then we consider the node " u' ":
    - The node "*c*" has already been processed as a neighbor of "u", hence <u>*it is not processed again*</u> as a neighbor of " u' ".

Multiple-perspective algorithm

MULTIPLE PERSPECTIVE ALGORITHM: INTUITION

- The multiple perspective algorithm combines the two logic underpinning:
  - The pure forward
  - The pure backward
- At each iteration, the algorithm determines whether it is convenient to tackle the problem from either the reachability or safety player's point of view.

MULTIPLE PERSPECTIVE ALGORITHM: INNER FUNCTIONING

- Start → The algorithm starts to solve the game as a safety game

- Execution →
  If the winning set's size is less than or equal than a threshold
  $t = floor(|V|/2)$
  - It is convenient to perform the forward step

  Otherwise:
  - it is convenient to execute the backward step.

- End →The algorithm returns the winning set for the safety player.

THE EMPLOYED HEURISTIC AT A GLANCE

It proceeds "backward".

The algorithm solve the game as a safety game

The algorithm solve the game as a reachability game

Threshold = floor(|V|/2)

It proceeds "forward".

MULTIPLE PERSPECTIVE ALGORITHM: CAVEAT

- ➢ The actual complexity of each step depends *on the degree of the nodes* involved in the computation!

- ➢ Hence, the criterion employed to determine whether it is convenient to switch point of view is not an accurate indicator, but *it is just used as a heuristic!*

MULTIPLE PERSPECTIVE ALGORITHM: PROOF OF CORRECTNESS

- Note that *taking a step instead of another does not undermine the correctness of the algorithm*!
- To provide an immediate intuition:
    - Changing the point of view at each iteration can be viewed as generating at each iteration a new game, in which the target set given in input is the winning set returned by the step executed at the previous iteration.
- In this respect, each iteration *is independent of the previous one*.
- Hence switching between the two points of view as the game progresses *does not affect* the algorithm's correctness.

Experimental phase: Results and discussion

$Experiment\_label$	$Total\_nodes$	$Total\_edges$	$Target\_nodes$	$Safety\_nodes$	$Reachability\_nodes$
1	996	2432	23	26	970
2	573	1734	119	30	543
3	789	2882	218	361	428
4	514	914	55	112	402
5	529	2517	2	196	333
6	174	420	49	49	125
7	502	2167	77	26	476
8	793	2768	357	614	179
9	313	489	24	9	304
10	203	445	196	11	192

$Experiment\_label$	FW_time	BW_time	MP_time	$Time\_saving\_wrt\_FW$	$Time\_saving\_wrt\_BW$
1	0.0012s	0.0096s	0.0003s	76.19%	97.01%
2	0.0004s	0.0070s	0.0004s	-4.65 %	94.29%
3	0.0026s	0.0389s	0.0023s	13.79%	94.17%
4	0.0005s	0.0127s	0.0005s	3.10%	96.11%
5	0.0002s	0.0106s	0.0001s	14.94%	98.75%
6	0.0005s	0.0020s	0.0005s	1.72%	74.65%
7	0.0004s	0.0056s	0.0003s	8.29%	94.07%
8	0.0033s	0.0429s	0.0033s	1.49%	92.32%
9	0.0001s	0.0026s	0.0001s	11.52%	95.06%
10	0.0006s	0.0011s	0.0005s	15.38%	50.55%

TABLE III

THIRD EXPERIMENTS BATTERY. NUMBER OF NODES RANGING BETWEEN [100, 1000], TARGET_SAFE_RATIO RANGING BETWEEN [0.1,1.0].

$Experiment\_label$	$Total\_nodes$	$Total\_edges$	$Target\_nodes$	$Safety\_nodes$	$Reachability\_nodes$
1	5341	9515	1226	4587	754
2	6368	8771	1037	4883	1485
3	5887	28 147	1887	5452	435
4	5990	8531	148	1111	4879
5	6275	25 183	325	1038	5237
6	5667	20 984	2613	2933	2734
7	5961	23 184	1461	5679	282
8	5543	7214	554	3868	1675
9	5318	16 082	2270	1648	3670
10	5581	20 494	1236	61	5520

$Experiment\_label$	FW_time	BW_time	MP_time	$Time\_saving\_wrt\_FW$	$Time\_saving\_wrt\_BW$
1	0.0494s	2.8728s	0.0409s	17.06%	98.57%
2	0.0689s	4.3557s	0.0596s	13.40%	98.63%
3	0.0479s	3.7482s	0.0438s	8.68%	98.83%
4	0.0101s	1.1958s	0.0113s	−12.04 %	99.06%
5	0.0196s	1.6087s	0.0177s	9.72%	98.90%
6	0.1544s	1.8811s	0.1561s	−1.08 %	91.70%
7	0.0275s	3.9568s	0.0231s	16.22%	99.42%
8	0.0368s	3.0642s	0.0316s	14.10%	98.97%
9	0.1412s	1.3840s	0.1292s	8.48%	90.66%
10	12.5055s	0.7117s	0.0048s	99.96%	99.33%

TABLE IV

FOURTH EXPERIMENTS BATTERY. NUMBER OF NODES RANGING BETWEEN [5000, 6500], TARGET_SAFE_RATIO RANGING BETWEEN [0.1,1.0].

DISCUSSION OF THE RESULTS

- The multiple-perspective algorithm performs better than its naive counterparts on the average case!
- The cases in which the multiple-perspective algorithm is slower than the naive ones show that the difference in terms of the required time is not particularly relevant.
- There are some cases in which the algorithm profoundly outperforms both the naive counterparts (e.g., first table experiment 1, second table experiment 10).

DISCUSSION OF THE RESULTS

- From all the experiments conducted, we can state that the multiple-perspective algorithm *profoundly outperforms* the backward algorithm in every conducted experiment.
- This is because most of the optimizations proper of the multiple-perspective algorithm have been designed to improve the algorithm’s capabilities to solve reachability games.

DISCUSSION OF THE RESULTS

- When the pure forward algorithm performs better than the multiple-perspective algorithm?
  - The multiple-perspective algorithm has two additional costs when solving games:
    - The cost related to the the generation of the transpose graph
    - The cost of switching point of view (i.e., computing the complement of the winning set).
- The multiple-perspective algorithm *is slower when the time required to compute the additional data structures exceeds the time saved through the carried-out optimizations*!

DISCUSSION OF THE RESULTS

- However, we remark another crucial factor:
  - The game starts as a safety game, hence it is given as input the target set for the safety player.
  - The *backward algorithm is disadvantaged* because it has to compute the complement of the target set and then solve the game as a reachability game.
  - If the starting size of the winning set is greater than half of the total number of nodes, *the multiple perspective algorithm experiences the same disadvantage*!
  - (It performs better than the backward algorithm thanks to the optimizations)

CONCLUSIONS - THE PROJECT'S STEPS

Implementation of the canonical versions of the algorithms to solve reachability and safety games

Objective 1 ✔

Design and implementation of several algorithmic optimizations

Objective 2 ✔

Design and development of a novel algorithm that combines the logic underpinning safety and reachability games

Objective 3 ✔

Testing of the proposed algorithm through randomized experiments

Objective 4 ✔

Questions?

Thank you