ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230344

Strategy Repair in Reachability Games

Pierre Gaillard [a;*], **Fabio Patrizi** [b;**] **and Giuseppe Perelli** [b;***]

[a]ENS Paris-Saclay, University Paris-Saclay
[b]Sapienza University of Rome
ORCiD ID: Fabio Patrizi https://orcid.org/0000-0002-9116-251X, Giuseppe Perelli https://orcid.org/0000-0002-8687-6323

Abstract. We introduce *Strategy Repair*, the problem of finding a minimal amount of modifications to turn a strategy for a reachability game from losing into winning. The problem is relevant for a number of settings in Planning and Synthesis, where solutions essentially correspond to winning strategies in a suitably defined reachability game. We show, via reduction from Vertex Cover, that Strategy Repair is NP-complete and devise two algorithms, one optimal and exponential and one polynomial but sub-optimal, which we compare experimentally. The reported experimentation includes some heuristics for strategy modification, which proved crucial in dramatically improving performance.

1 Introduction

A *Reachability Game* (RG) [11] is a finite-state game played by two players, Player 0 (P_0) and Player 1 (P_1). The states of the game are partitioned into P_0's and P_1's; when the game is in a P_0's (resp. P_1's) state s, P_0 (P_1) can move to a new state s' by selecting any of the transitions (s, s') allowed by the game. Transitions do not guarantee strict alternation, i.e., s' can still be P_0's (P_1's), so that P_0 (P_1) can keep moving until (if ever) a P_1's (P_0's) state is reached.

In a RG, some states are designated as *target*, and *solving* the game amounts to finding a *winning strategy* for P_0, i.e., a function which prescribes P_0 the next move to take in each of its states, so as to guarantee that a target state is eventually reached whenever possible, no matter how P_1 moves in its states. This is a very well-studied problem, in Synthesis [8, 3, 2, 7] and in other areas, such as Fully Observable Nondeterministic (FOND) Planning [5], known to be solvable in polynomial time wrt the game size, thus exponential wrt (exponentially) succinct game representations, as typical, e.g., in Planning. Such bounds are in fact tight, the decision version of the problem being P-complete [11, 14, 6, 4].

RGs can serve as semantic models for reasoning about dynamic domains, with the resulting strategy representing the behavior that an agent can execute, in order to achieve a desired state. Typically, however, at execution time, models deviate from the actual trajectory that stems from strategy execution, resulting in a situation where the state does not match that of the model. There may also be situations where the goal changes during strategy execution. In both these examples, the agent is unable to keep executing the computed strategy (which was originally winning) and take appropriate actions to achieve the desired goal. Thus, the problem arises of coming up with a new strategy that guarantees goal achievement.

Observe that the original strategy might have been designed to guarantee not only goal achievement, but also a number of additional properties, such as cost minimization, reward maximization, or forbidden states avoidance, which might yield a significant additional computational effort. Thus, when the unexpected changes are small and yield only a slightly different problem wrt the original one, i.e., only few target states are added or removed and state mismatches occur rarely, it is reasonable to seek for a solution obtained as a slight modification of the original one, under the assumption that the new strategy will retain all (or part of) the properties featured by the initial strategy, without needing the computational overhead required to achieve such properties.

This paper investigates this approach from the general perspective of RGs. We introduce a problem, called *Strategy Repair*, which requires, given a *losing* (i.e., not winning) strategy σ_0, to find a minimal amount of modifications which turn σ_0 into a winning strategy.

We make the following contributions. Firstly, we formally define the problem by introducing a notion of *distance* between two strategies, which intuitively corresponds to the number of states over which the strategies differ. Then, based on such a notion, we devise a solution algorithm and characterize its complexity. Specifically, we prove, by reduction from Vertex Cover, that the decision version of Strategy Repair is NP-complete. We then investigate more efficient, but sub-optimal, alternatives, devising a polynomial greedy algorithm with an effective heuristic, called MustFix, which can be integrated also in the optimal algorithm. Finally, we report on an experimental analysis, which shows that the polynomial algorithm, together with the MustFix heuristic, yields impressive results in terms of running time, scalability and accuracy (measured as distance from the optimal solution). Also the optimal algorithm greatly benefits from the MustFix heuristic, outperforming the running times of the basic version by orders of magnitude. The experiments also measure the impact of repairing a strategy when the goal is subject to small changes wrt computing a strategy from scratch, showing that the former approach produces strategies that are considerably closer to the original one, despite the fact that the latter uses exaclty the same algorithm before and after the goal changes.

Strategy Repair is very related to the notion of *plan repair*, originally introduced (as *plan reuse*) in Classical Planning, by [13]. While in that work it is shown that repairing is not more efficient, in general, than replanning, the former exhibits higher effectiveness in practice [10, 17]. Surprisingly, while the problem is actively

[*] Email: pierre.gaillard1@ens-paris-saclay.fr
[**] Email: patrizi@diag.uniroma1.it
[***] Corresponding Author. Email: perelli@di.uniroma1.it

studied in this context , we could not find analogous contributions in the context of Fully Observable Nondeterministic (FOND) Planning, where plans are replaced by *policies*. The results we obtain for RGs are of immediate interest for this setting, since RGs can easily model (FOND) Planning (as well as Classical Planning), with winning strategies corresponding to policies. Moreover, some connections can be made between Strategy Repair and strategy improvement, which consists to generate a strategy and apply iteratively some modifications until a winning strategy is found. This technique can be used to compute a winning strategies in games on graphs [12, 20, 1]. However, notice that no distance minimization is taken into account, as the purpose of strategy improvement is not repairing a strategy, but generating one from scratch.

Our work shares the same motivations as those for the work in Classical Planning, spanning from the idea that repairing is a natural way to save computational efforts by maximizing reuse [13], to more qualitative arguments concerning quality preservation in the new solution (similar solutions are likely to have same properties), impact on human observers (validating a solution similar to one validated before is likely simpler than analyzing a new one), and implementation/execution efforts (modifying a deployed solution is likely to require less effort than implementing a new one). These are somehow summarized in the *stability* maximization principle discussed in [9], although applied to deterministic plans, which essentially states that, for the above reasons, minimal distance solutions are more desirable. In this respect, it is worth observing that, with the recent exception of [16], most of the previous approaches, such as [18, 9, 10, 19], do not offer minimal-distance guarantees.

This work transfers these ideas to the more general setting of RGs, by devising, studying and experimentally evaluating two approaches for strategy repair, one that offers optimality guarantees and one that is suboptimal but extremely efficient and returning near-optimal solutions in practice.

2 Preliminaries

We here introduce the basic notions and definitions related to *reachability games*.

A *2-player arena*, or simply *arena* is a tuple $\mathcal{A} = \langle V, V_0, V_1, \mathrm{E}\rangle$, where V is the set of *nodes*, or *vertices*, with $V = V_0 \cup V_1$ and $V_0 \cap V_1 = \emptyset$, and $\mathrm{E} \subseteq V \times V$ is the set of *edges* of the arena. We say that V_0 is the set of nodes controlled by player 0, (P_0), whereas V_1 is the set of nodes controlled by player 1 (P_1).

Definition 1 (Reachability game) *A* Reachability game *is a pair $\mathcal{G} = \langle \mathcal{A}, \mathcal{T}\rangle$, where $\mathcal{A}$ is an arena, and $\mathcal{T} \subseteq V$ is a subset of nodes of such arena, sometimes called* target *or* winning *nodes.*

A *path* in the arena is a sequence $\pi = v_0 \cdot v_1 \cdot v_2 \ldots \in V^\omega$ such that $(v_i, v_{i+1}) \in \mathrm{E}$ for each $i \in \mathbb{N}$.

As usual, by π_i, we denote the i-th node occurring in the sequence π, whereas by $\pi_{\leq i}$ we denote the prefix of π up to node π_i, also called *partial path*. We say that a path π is *winning* for player 0 if $\pi_i \in \mathcal{T}$ for some $i \in \mathbb{N}$, otherwise it is winning for player 1.

A strategy for player 0 is a function $\sigma_0 : V^* \cdot V_0 \to \mathrm{E}$ mapping partial paths to edges, such that $\sigma_0(v_0 \ldots v_n)$ is an edge outgoing from v_n, for each partial path in $V^* \cdot V_0$. A strategy σ_1 for player 1 is defined accordingly.

A path π is *compatible* with strategy σ_0 if $\sigma_0(\pi_{\leq i}) = (\pi_i, \pi_{i+1})$ for each $\pi_i \in V_0$. Analogously, it is *compatible* with strategy σ_1 if $\sigma_1(\pi_{\leq i}) = (\pi_i, \pi_{i+1})$ for each $\pi_i \in V_1$. By $\mathsf{out}(v, \sigma_0)$ we denote the set of paths starting from v that are compatible with σ_0. Analogously, $\mathsf{out}(v, \sigma_1)$ denotes the set of paths starting from v that are compatible with σ_1. Notice that there exists only one path starting from v that is compatible with both σ_0 and σ_1, which we denote $\mathsf{play}(v, \sigma_0, \sigma_1)$.

We say that a strategy σ_0 is *winning* for player 0 from v, if every path in $\mathsf{out}(v, \sigma_0)$ is *winning*. We say that a node v is winning for player 0 if there exists a strategy σ_0 winning from v. We denote by $\mathrm{Win}_0(\mathcal{G})$ and $\mathrm{Win}_1(\mathcal{G})$ the sets of nodes in $\mathcal{G}$ that are winning for player 0 and 1, respectively. Finally, a strategy is said to be simply *winning* if it is winning from every vertex in $\mathrm{Win}_0(\mathcal{G})$. It is well known that reachability games are *memoryless determined* [15], that is, every node v is either winning for player 0 or winning for player 1 and that there always exists a memoryless winning strategy. Therefore, from now on we restrict our attention to only memoryless strategies, that are defined as $\sigma_0 : V_0 \to \mathrm{E}$ mapping each node belonging to an agent to an outgoing edge.

Such restriction allows us to define a very natural distance between two player 0 strategies σ_0 and σ_0' over the same game, that is

$$\mathsf{dist}(\sigma_0, \sigma_0') = |\{v \in V \mid \sigma_0(v) \neq \sigma_0'(v)\}|$$

Intuitively, we count the number of nodes on which the two strategies map to a different outgoing edge. This can be proved to be an actual distance.

Proposition 1 *For a given arena $\mathcal{A}$, the function* dist *is a distance in the space of memoryless strategies of $\mathcal{A}$.*

Proof We show the properties of a distance separately. That is, for every $\sigma_0, \sigma_0', \sigma_0''$, the following hold:

- $\mathsf{dist}(\sigma_0, \sigma_0) = 0$; (Nonzero)
- $\sigma_0 \neq \sigma_0' \implies \mathsf{dist}(\sigma_0, \sigma_0') \gneq 0$; (Positivity)
- $\mathsf{dist}(\sigma_0, \sigma_0') = \mathsf{dist}(\sigma_0', \sigma_0)$; (Symmetry)
- $\mathsf{dist}(\sigma_0, \sigma_0'') \leqslant \mathsf{dist}(\sigma_0, \sigma_0') + \mathsf{dist}(\sigma_0', \sigma_0'')$ (Triangle Inequality)

The first three are immediate. Regarding the triangle inequality, notice that if $\sigma_0(v) \neq \sigma_0''(v)$, then $\sigma_0(v) \neq \sigma_0'(v)$ or $\sigma_0'(v) \neq \sigma_0''(v)$ (since otherwise $\sigma_0(v) = \sigma_0'(v) = \sigma_0''(v)$). Therefore $\{v \in V \mid \sigma_0(v) \neq \sigma_0''(v)\} \subseteq \{v \in V \mid \sigma_0(v) \neq \sigma_0'(v)\} \cup \{v \in V \mid \sigma_0'(v) \neq \sigma_0''(v)\}$, so $\mathsf{dist}(\sigma_0, \sigma_0'') \leqslant \mathsf{dist}(\sigma_0, \sigma_0') + \mathsf{dist}(\sigma_0', \sigma_0'')$ □

We conclude this section by introducing some useful notation.

For a given game $\mathcal{G}$ and an edge $e = (v_1, v_2) \in \mathrm{E}$, by $\mathcal{G}_e$ we denote the game induced from $\mathcal{G}$ by removing every edge (v_1', v_2') incompatible with e, that is, such that $v_1' = v_1$ and $v_2' \neq v_2$. This can be extended to subsets $\mathrm{E}' \subseteq \mathrm{E}$ of edges, where $\mathcal{G}_{\mathrm{E}'} = (\mathcal{G}_{\mathrm{E}' \setminus \{e\}})_e$ is recursively defined by projecting the edges e of E' one by one.

Notice that a (memoryless) strategy σ_0 can be regarded as a subset of edges, one for each node in V_0, therefore $\mathcal{G}_{\sigma_0}$ denotes the game induced from $\mathcal{G}$ by removing every edge (v, v') incompatible with σ_0, that is, such that $v \in V_0$ and $(v, v') \neq \sigma_0(v)$. Note that every vertex of V_0 has only one successor in $\mathcal{G}_{\sigma_0}$, which means that player 0 has only one strategy available in the game, which is indeed σ_0.

3 The Strategy Repair Problem

We now introduce the *strategy repair* problem for reachability games. First, for a given reachability game $\mathcal{G}$ and a player 0 strategy σ_0, define $\mathrm{Win}(\mathcal{G}, \sigma_0)$ to be the set of nodes from which σ_0 is winning. It is not hard to show that $\mathrm{Win}(\mathcal{G}, \sigma_0) = \mathrm{Win}(\mathcal{G}_{\sigma_0})$, that is, the nodes that are winning for player 0 when it is using strategy σ_0 can be obtained by considering the game $\mathcal{G}_{\sigma_0}$ where the choices incompatible with σ_0 have already been ruled out. Observe that it always holds that $\mathrm{Win}(\mathcal{G}, \sigma_0) \subseteq \mathrm{Win}(\mathcal{G})$, with $\mathrm{Win}(\mathcal{G}, \sigma_0) = \mathrm{Win}(\mathcal{G})$ if, and only if, σ_0 is winning for player 0.

We define the strategy repair problem as follows.

Definition 2 (Strategy repair problem) *For a given reachability game $\mathcal{G}$ and a strategy σ_0, find a winning strategy σ'_0 such that $\mathsf{dist}(\sigma_0, \sigma'_0) \leq \mathsf{dist}(\sigma_0, \sigma''_0)$ for each winning strategy σ''_0.*

The problem introduced requires to minimize the number of modifications that are required to turn a strategy σ_0 into a strategy σ'_0 winning for a given reachability game $\mathcal{G}$. The corresponding decision problem, instead, consists in fixing a given threshold $k \in \mathbb{N}$ and checking whether some winning strategy σ'_0 exists with $\mathsf{dist}(\sigma_0, \sigma'_0) \leq k$. We now prove that the strategy repair problem for reachability games is NP-complete. To do so, we show a reduction from the NP-complete problem *vertex cover* [4], defined below.

Definition 3 (Vertex cover) *For a given undirected graph $G = \langle S, A \rangle$ with A a set of pairs of S elements, and a natural number $k \in \mathbb{N}$, find a subset $S' \subseteq S$, with $|S'| \leq k$, such that, for each $\{v, v'\} \in A$, $v \in S'$ or $v' \in S'$.*

Theorem 1 *The strategy repair problem for reachability games is* NP*-complete.*

Proof First, observe that checking whether a strategy σ'_0 is winning can be done in polynomial time, as it amounts to checking whether $\mathrm{Win}(\mathcal{G}, \sigma'_0) = \mathrm{Win}(\mathcal{G})$ and so to solve two reachability games of size linear in $|\mathcal{G}|$. Also, checking that $\mathsf{dist}(\sigma_0, \sigma'_0) \leq k$ can be done in time linear with respect to the number of nodes in V_0. This then proves membership in NP.

To prove that the problem is hard for NP, consider an instance $\langle S, A \rangle, k$ of the vertex cover problem, and construct the following strategy repair problem $\mathcal{G} = \langle \mathcal{A}, \mathcal{T} \rangle, \sigma_0, k$ where $\mathcal{A} = \langle V, V_0, V_1, \mathrm{E} \rangle$ is defined as

- $V = S \times \{0, 1\} \cup \{t\}$ where t is a fresh new vertex;
- $V_0 = S \times \{0\} \cup \{t\}$;
- $V_1 = S \times \{1\}$;
- $\mathcal{T} = \{t\}$
- $\mathrm{E} = \{((v, 0), (v, 1)) \mid v \in S\} \cup \{((v, 0), t) \mid v \in S\} \cup \{(v, 1), (v', 0) \mid \{v, v'\} \in A\}$

And strategy σ_0 is such that $\sigma_0(v, 0) = (v, 1)$, for each $v \in S$.

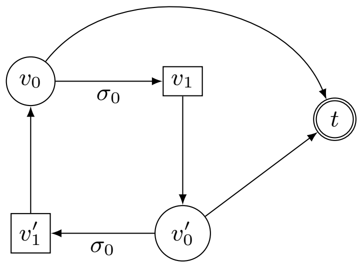

Figure 1: The structure of vertex cover reduction.

Intuitively, for each node v of the original graph, we construct two nodes in the arena $(v, 0), (v, 1)$ under the control of player 0 and 1 respectively, and an extra target node t, which is the only winning node of the game. In each node under his control, player 0 can either move towards the 1-type version of the same vertex or to the target node t. Player 1, instead, can move towards the 0-type version of any node that is connected to the current vertex in the original undirected graph.

The strategy σ_0, instead selects the 1-type version of every vertex in the original graph.

Essentially, for each edge $\{v, v'\} \in A$ of the original graph, a structure depicted in Figure 1, where the labels σ_0 represent the initial strategy of Player 0. First, note that $\mathrm{Win}_0(\mathcal{G}) = V$, as the strategy that selects all edges going to the target t is clearly winning from every node of the game.

We now show that the undirected graph G has a vertex cover S' of size at most k if, and only if, the corresponding strategy repair problem admits a solution σ'_0 such that $\mathsf{dist}(\sigma_0, \sigma'_0) \leq k$.

To simplify the notation, vertices of the form $(v, 0)$ are noted v_0 and the ones of the form $(v, 1)$ are denoted v_1.

From the one hand, assume that $S' \subseteq S$ is a vertex cover of G with $|S'| \leq k$ and consider the strategy $\sigma'_0 = \sigma_0[v_0 \mapsto (v_0, t); v_0 \in S']$. Essentially, the strategy diverts directly to the target state t all and only those nodes that belong to the vertex cover S' of G. Clearly, we have that $\mathsf{dist}(\sigma_0, \sigma'_0) = |S'| \leq k$. It remains to prove that it is winning in every node of the game.

Obviously, it holds that $t \in \mathrm{Win}_0(\mathcal{G}, \sigma'_0)$. For the remaining nodes, we distinguish four cases:

1. $v_0 \in V_0$ with $v \in S'$. Therefore, $\sigma'_0(v_0) = (v_0, t)$ and so $v_0 \in \mathrm{Win}_0(\mathcal{G}, \sigma'_0)$.
2. $v_1 \in V_1$ with $v \notin S'$. Therefore, for every successor v'_0 of v_1 in $\mathcal{G}$, it holds that $v' \in S'$, and so that, from case 1 that $v'_0 \in \mathrm{Win}_0(\mathcal{G}, \sigma'_0)$, which implies that $v_1 \in \mathrm{Win}_0(\mathcal{G}, \sigma'_0)$.
3. $v_0 \in V_0$ with $v \notin S'$. Therefore, $\sigma'_0(v_0) = (v_0, v_1)$ with $v \notin S'$, which, from case 2, implies that $v_1 \in \mathrm{Win}_0(\mathcal{G}, \sigma'_0)$ and so that $v_0 \in \mathrm{Win}_0(\mathcal{G}, \sigma'_0)$.
4. $v_1 \in V_1$ with $v \in S'$. Since any successor v'_0 of v_1 belongs to V_0, we fall in either case 1 or case 3, which implies that $v_1 \in \mathrm{Win}_0(\mathcal{G}, \sigma'_0)$.

On the other hand, consider a solution σ'_0 to the strategy repair problem and define $S' = \{v \in S \mid \sigma'_0(v_0) \neq \sigma_0(v_0)\}$. Clearly, $|S'| = \mathsf{dist}(\sigma_0, \sigma'_0) \leq k$. It remains to prove that S' is a vertex cover. By contradiction, assume that there is an edge $\{v, v'\}$ that is not covered by S', i.e., $v, v' \notin S'$. This means that $\sigma'_0(v_0) = (v_0, v_1)$ and $\sigma'_0(v'_0) = (v'_0, v'_1)$. Now, as it is also depicted in Figure 1, Player 1 can select v'_0 from v_1 and v_0 from v'_1, which makes the four nodes not winning, hence a contradiction. □

4 Algorithmic Solutions

In this section, we discuss two algorithms for Strategy Repair, which we called `Opt` and `Greedy`, respectively. The former returns the optimal solution to the problem, but runs in exponential time. The latter, instead, returns a sub-optimal solution but runs in polynomial time. It is important to remark that they both produce correct winning strategies for the game. However, the algorithm `Greedy` does not provide the most optimal one in terms of distance from the originally specified strategy σ_0.

4.1 The algorithm `Opt`

We now proceed with the description of Algorithm `Opt`. In order to do so, we first introduce some useful definition.

For a given game $\mathcal{G}$ and a set $X \subseteq V$ of nodes, the *Frontier* of X, denoted $\text{Frontier}_0(X) = ((V_0 \setminus X) \times X) \cap \text{E}$, is the set of edges that are outgoing from a Player 0 node and incoming to a node in X. Intuitively, the edges in Frontier_0 can be used by Player 0 to enter in a single step the region X of nodes.

Consider a game $\mathcal{G}$ and a strategy σ_0, and let $X = \text{Win}_0(\mathcal{G}, \sigma_0)$ be the set of nodes that are winning for strategy σ_0. Observe that for an edge $(v, v') \in \text{Frontier}_0(X)$, it holds that $\sigma_0(v) \neq (v, v')$, otherwise v would have been winning for σ_0 in the first place. Moreover, it is trivial to show that the strategy $\sigma_0' = \sigma_0[v \mapsto (v, v')]$ is such that $\text{Win}_0(\mathcal{G}, \sigma_0) \subsetneq \text{Win}_0(\mathcal{G}, \sigma_0')$, with the inclusion being proper because $v \in \text{Win}_0(\mathcal{G}, \sigma_0') \setminus \text{Win}_0(\mathcal{G}, \sigma_0)$.

We are now ready to present the algorithm `Opt`, which is reported in Algorithm 1

```
Input: $\mathcal{G}$ a reachability game, $\sigma_0$ a strategy for player 0
Output: Winning strategy for $\mathcal{G}$ minimizing the distance
        from $\sigma_0$
$\mathsf{Fix}(\mathcal{G}, \sigma_0)$ :
$T' \leftarrow \text{Win}_0(\mathcal{G}, \sigma_0)$   // vertices already winning following $\sigma_0$
if $T' = \text{Win}_0(\mathcal{G})$ then
    return $(\sigma_0, 0)$
else
    select $(v, v')$ from Frontier$(T')$
    $(\sigma_0', \beta') \leftarrow \mathsf{Fix}(\mathcal{G}, \sigma_0[v \mapsto (v, v')])$
    $\mathcal{G}' \leftarrow \mathcal{G}_{\sigma_0(v)}$                  // player 0 must follow $\sigma_0$ on $v$
    if $v \in \text{Win}_0(\mathcal{G}')$ then
        $(\sigma_0'', \beta'') \leftarrow \mathsf{Fix}(\mathcal{G}', \sigma_0)$
        if $\beta'' < \beta' + 1$ then
            return $(\sigma_0'', \beta'')$
        end
    end
    return $(\sigma_0', \beta' + 1)$
end
```

Algorithm 1: Pseudocode of Algorithm `Opt`.

The algorithm works as follow. First, it computes the winning region following σ_0 $T' = \text{Win}_0(\mathcal{G}, \sigma_0)$ and compares it with the winning region of the game $\text{Win}_0(\mathcal{G})$. If the two sets are equal, it means that σ_0 is already winning, so it returns the optimal solution $(\sigma_0, 0)$, with the second component denoting the cost of fixing. If that is not the case, the algorithm proceeds by first computing the frontier of $\text{Win}_0(\mathcal{G}, \sigma_0)$, in order to select an edge (v, v') from it, then it compares two possible solutions. The first is obtained by solving the problem where the initial strategy is $\sigma_0[v \mapsto (v, v')]$, obtained from σ_0 by diverting the choice on v with the frontier edge (v, v'). The second is obtained by solving the problem when Player 0 is forced to select edge $\sigma_0(v)$ in v. This is obtained by considering the game $\mathcal{G}' = \mathcal{G}_{\sigma_0(v)}$, where all other outgoing edges from v are removed. Both solutions are computed with their relative costs β' and β'', which are then compared to select the best between the two. Note that the latter solution might not exist, as the choice of σ_0 in v might lead, for instance, out of the winning region. The algorithm then first checks whether such solution is viable before making a useless recursive call on $(\mathcal{G}', \sigma_0)$. Observe that in the first case the total modification cost β' must be increased by 1, as the initial strategy $\sigma_0[v \mapsto (v, v')]$ is at distance 1 from σ_0 itself.

We have the following.

Theorem 2 *The algorithm* `Opt` *returns the optimal solution to the Strategy Repair problem.*

Proof (sketch) For the sake of space, we provide a high-level intuition of it. The proof proceeds by induction on the number of recursive calls that are made by `Opt`. In particular, the base case trivially holds when the input to the algorithm is a pair $(\mathcal{G}, \sigma_0)$ such that σ_0 is already winning for $\mathcal{G}$. For the induction case, the proof essentially shows that one of the two solutions computed by the two recursive calls must return a winning strategy at the minimum distance from the input σ_0. □

4.2 *The algorithm* `Greedy`

The algorithm `Opt` presented in the previous section is of exponential complexity, as it requires two recursive calls at each iteration to compare the distances between the initial strategy and two candidate best solutions. Also, notice that the recursive call that makes use of the selected edge in the frontier always computes a correct solution, although it might not be the optimal one. Therefore, a sub-optimal but polynomially computable solution could be found by just selecting the one obtained from such call, disregarding the other. This is how the algorithm `Greedy` is conceived. However, in order to improve the quality of the solution, i.e., the accuracy w.r.t. the optimum, we employ a selection criterion for the edge in the frontier set. Indeed, consider an instance $(\mathcal{G}, \sigma_0)$ of Strategy Repair, and an edge $(v, v') \in \text{Frontier}_0(\text{Win}_0(\mathcal{G}, \sigma_0))$. First, note that $\sigma_0(v) \neq (v, v')$, otherwise, the node v would be winning for σ_0 and (v, v') would not be in the frontier. Therefore, consider the set $\mathsf{Repair}_{\sigma_0}(v, v') = \text{Win}_0(\mathcal{G}, \sigma_0[v \mapsto (v, v')]) \setminus \text{Win}_0(\mathcal{G}, \sigma_0)$, that is, the set of nodes that are indirectly repaired by using the frontier edge (v, v') in the solution.

Therefore, when selecting the frontier edge, one might decide to greedily maximize the number of nodes that are indirectly repaired by such a selection. This is how the algorithm `Greedy` works, as it is presented in Algorithm 2.

```
Input: $\mathcal{G}$ a reachability game, $\sigma_0$ a strategy for player 0
$\mathsf{Fix}(\mathcal{G}, \sigma_0)$ :
$T' \leftarrow \text{Win}_0(\mathcal{G}, \sigma_0)$   // vertices already winning following $\sigma_0$
if $T' = \text{Win}_0(\mathcal{G})$ then
    return $(\sigma_0, 0)$
end
$F \leftarrow \text{Frontier}_0(T')$
$(v, v') \leftarrow \mathsf{argmax}\{|\mathsf{Repair}_{\sigma_0}(v, v')|; (v, v') \in F\}$
$(\sigma_0', \beta') \leftarrow \mathsf{Fix}(\mathcal{G}, \sigma_0[v \mapsto (v, v')])$
return $(\sigma_0', \beta' + 1)$
```

Algorithm 2: Pseudocode of Algorithm `Greedy`.

4.3 *A Better Selection Method of Frontier Edges*

The algorithm `Greedy` employs a selection method for the frontier edges that maximizes the number of nodes that enter the winning area of the fixing in progress. Note that this method can always be applied. Indeed, the fact that a given strategy σ_0 needs to be repaired is equivalent to the fact that $\text{Frontier}_0(\text{Win}_0(\mathcal{G}, \sigma_0)) \neq \emptyset$ and so at least one edge can be selected.

Instead, we now present another method, which we called $\mathsf{MustFix}$, that aims at selecting frontier edges that *must* be used to repair the current strategy.

Consider an instance $(\mathcal{G}, \sigma_0)$ and a frontier edge $(v, v') \in \text{Frontier}_0(\text{Win}_0(\mathcal{G}, \sigma_0))$. Assume that $v \notin \text{Win}_0(\mathcal{G}_{\sigma_0(v)})$, that is, there is no strategy consistent with $\sigma_0(v)$ that is winning for v. This means that edge $\sigma_0(v)$ must be replaced and the best way to do so is by using the frontier (v, v'). Thus, the selection method MustFix first checks whether some of the frontier edges satisfy this condition and, if so, selects them to the recursive call.

Observe two things. First, there is no guarantee that at least one frontier edge satisfies the MustFix condition, therefore, it is still necessary to employ other selection methods, as in the Algorithm Greedy. Second, the MustFix method can be applied also to Opt as a preprocessing mechanism. Indeed, it identifies edges that necessarily must be used even in the optimal solution so when such an edge is selected, only one recursive call is made by Opt. In the next section, we discuss also on the performance of such method applied to Opt and Greedy, both in terms of running time and accuracy.

We conclude this section by showing that Greedy is not optimal, even when MustFix is adopted. Indeed, consider the instance depicted in Figure 2 where strategy σ_0 is represented with labels over the edges. We have that $\text{Frontier}_0(\text{Win}_0(\mathcal{G}, \sigma_0)) = \{(v_1, t), (v_2, t)\}$ but MustFix does not select any of its elements. Indeed, if $\sigma_0(v_1) = (v_1, v_3)$ is imposed, then the path $v_1 \cdot v_3 \cdot v_2 \cdot t$ is winning, and if $\sigma_0(v_2) = (v_2, v_4)$ is imposed, then the path $v_2 \cdot v_4 \cdot v_1 \cdot t$ is winning. Therefore, the algorithm Greedy enters the computation of $\text{Repair}(v_1, t) = \{v_1, v_0\}$ and $\text{Repair}(v_2, t) = \{v_2\}$, and chooses to fix the strategy on v_1 as it repairs two vertices instead of one. Next, the two choices are between (v_4, v_1) and (v_2, t) but again, MustFix does not apply, and repairing v_2 would not indirectly repair any other vertex while repairing v_4 would also repairs v_2, which means that Greedy selects (v_4, v_1). Finally, the algorithm repairs v_3, making it a total of three modifications. However, note that $\sigma_0[v_2 \mapsto (v_2, t), v_3 \mapsto (v_3, v_2)]$ is winning and its distance from σ_0 is only two.

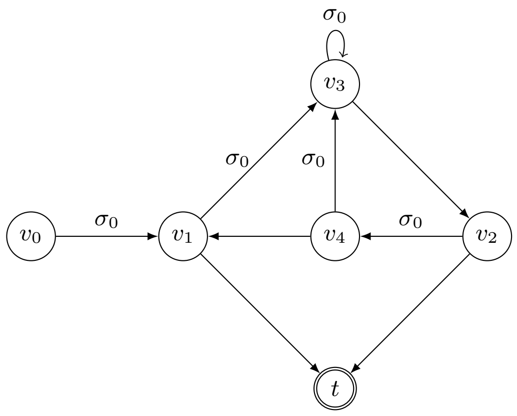

Figure 2: Counter example for Greedy.

5 Experimental setup

The algorithms Opt and Greedy, together with the MustFix heuristics, have been implemented in Python and tested on a variety of benchmarks. The implementation allows to tune the battery test under a variety of parameters that are used to randomly generate the game instances, which are introduced at each experiment description below.

We run four kinds of experiments, each of them designed to test a specific feature. First, we tested the effectiveness of MustFix. In particular, we compared two versions of Opt, one with and one without the MustFix heuristics enabled, to compare their performance. The results and comments on this experiment are presented in Section 5.1. Second, in Section 5.2 we evaluated the performances of Opt and Greedy, both with MustFix activated. In particular, we evaluated both the running time and accuracy of Greedy, which was comparably performing well with respect to Opt, in both terms. Third, we run an additional comparison among Opt and Greedy in Section 5.3, this time by generating benchmark instances specifically designed to push the boundaries on scalability. In particular, we made use of the reduction from Vertex Cover of Theorem 1 to identify instances where Opt supposedly performs poorly. Finally, in Section 5.4 we investigate on a specific setting where the initial strategy of a Strategy Repair instance comes as a winning strategy of a previously instantiated game, where the target set has been slightly deviated. In this case, we evaluate how convenient would be to employ Strategy Repair (both with Opt or Greedy) against recomputing a new winning strategy from scratch, after the underlying graph is changed for any reason.

All tests have been run on an Intel Core i7-8700 CPU @ 3.20GHz, with 32GB of RAM and running Ubuntu 20.04.06 LTS.

5.1 *Testing the Effectiveness of* MustFix

We first tested the effectiveness of MustFix, running experiments on randomly generated games and solving them with two versions of Opt, one making use of MustFix, the other randomly selecting frontier edges and proceeding with recursive calls.

The random instances have been generated as follows:

- To input n, the random generator produces a game with n nodes.
- Each node is assigned to P_0 or P_1 with equal probability (0.5 each);
- For each node, a number d between s_{min} and s_{max} is uniformly chosen and then, d successors are randomly assigned to the node. For the experiments in place, we fixed two sets of s_{min} and s_{max}, keeping them between 1% and 5% of n for a first set of experiments, and between 5% and 10% of n for a second set. Here, we report the experiments obtained with the first set, as they produced similar results.
- A number t of target vertices, uniformly selected. In all the experiments reported in the paper, we fixed t to be 5% of n.

Regarding the initial strategy σ_0 that will be repaired, the random generator works as follows. For each node v of P_0, if there are any successors that **do not** belong to the target set T, we uniformly select one of them for σ_0. To our opinion, this increases the chances that a number of edges in σ_0 must be repaired. It is open to establish whether a better generation of strategies is possible.

N. nodes	N. experiments	no MustFix	MustFix
40	1000	0.0043	0.0017
60	100	0.28	0.013
80	20	6.2	0.25
100	20	150	7.6

Table 1: Comparison of running time (in sec.) for Opt with and without the MustFix selection method.

We tested the performance of Opt against a battery of tests, running from 40 nodes, up to 100 nodes. The results are reported in Table 1 and a plot comparison of their running time provided in Figure 3. It can be easily seen that MustFix significantly improves the running time of Opt by 2 order of magnitudes. This was expected, as each application of MustFix avoids the propagation of two recursive call per iteration, making the algorithm much more efficient.

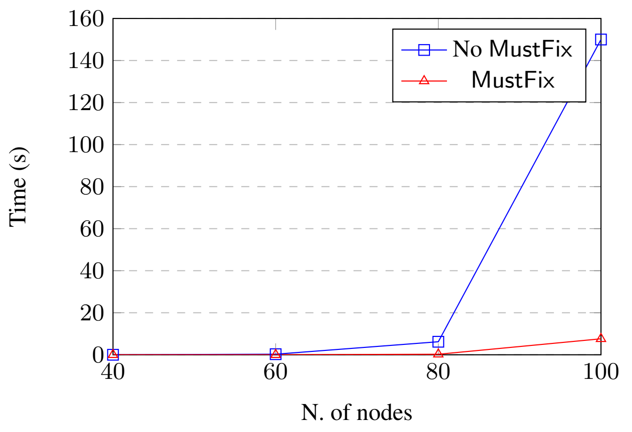

Figure 3: Comparison of `Opt` with and without the MustFix selection method.

5.2 *Comparing* `Opt` *and* `Greedy`

Once established that MustFix significantly improves the performance of our approaches, we proceeded comparing `Opt` and `Greedy` on the same set of randomly generated games. For this case, given that `Greedy` does not provide the optimal solution, we also had to evaluate its accuracy, which is defined by $\mathsf{dist}(\sigma_0, \sigma_0^{\mathtt{Opt}})/\mathsf{dist}(\sigma_0, \sigma_0^{\mathtt{Greedy}})$, where σ_0 being the original (possibly losing) strategy, $\sigma_0^{\mathtt{Opt}}$ the one computed by `Opt`, and $\sigma_0^{\mathtt{Greedy}}$ the strategy computed by `Greedy`.

N. nodes	N. experiments	`Opt`	`Greedy`	Accuracy
40	5000	0.0012	0.001	0.9994
60	5000	0.02	0.0065	0.9952
100	1000	3.8	0.064	0.9904
200	700	3.39	0.79	0.9994
500	300	17.71	17.6	1
700	150	60.9	60.85	1

Table 2: Comparison of `Opt` and `Greedy`.

We tested our approach by running experiments with up to 700 nodes. The results are summarized and plotted in Table 2 and Figure 4, respectively. The running times of the two algorithms are essentially equal, together with a very high accuracy rate. This is due to the fact that the MustFix selection method is triggered most of the time, boiling the two approaches down to the same execution.

5.3 *Stress test on vertex cover*

We run a third batch of tests, reported below, in which we generate games where it is unlikely to trigger MustFix, both for `Opt` and `Greedy`. To do so, we generated instances directly from Vertex Cover, the problem we used to prove that Strategy Repair is NP-complete. The instance of the experiment is then generated starting from a vertex cover problem with n nodes, and then reducing it to Strategy Repair, the same way it is done in the proof of Theorem 1.

Results on this are reported in Table 3 and plotted in Figure 5.

In this case, the performances of `Opt` and `Greedy` diverge very quickly. By setting a timeout of 30 minutes, the algorithm `Opt` could solve instances of up to 80 nodes (for a total of 161 nodes of the corresponding Strategy Repair), while `Greedy` was capable of solving the same instances in around 10 seconds, and scaling quite well. Regarding the accuracy, it decreases a little with respect to random games, but still it is maintained at around 0.9.

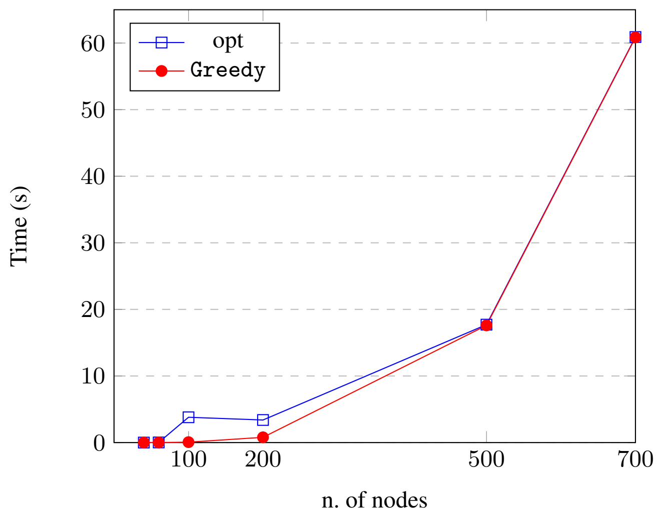

Figure 4: Comparison of `Opt` and `Greedy`.

VC nodes	N. experiments	`Opt`	`Greedy`	Accuracy
5	5,000	0.0009	0.0005	0.9815
10	2,000	0.013	0.004	0.889
20	500	0.32	0.041	0.875
30	100	2.8	0.185	0.8835
50	5	56.8	1.44	0.923
55	5	118.2	2.11	0.911
60	5	206	3.09	0.906
65	3	371	4.32	0.914
70	3	603	6	0.9254
75	2	985	8.28	0.937
80	2	1385	11	0.929
100	10	Timeout	29.7	N/A
120	4	Timeout	66.9	N/A
150	4	Timeout	182	N/A
175	3	Timeout	368	N/A
200	3	Timeout	692	N/A

Table 3: Vertex cover experiments.

5.4 *Target deviation*

We run a last battery experiments that aim at evaluating how convenient in terms of strategy distance to employ Strategy Repair instead of generating a completely new strategy from scratch.

We evaluated the following scenario. First, we generate a game instance $\mathcal{G}$ using the same random generator of the first two batches of experiments. We then found a winning strategy σ_0 using the classic backward induction approach. At this point, we randomly deviated the target set T of $\mathcal{G}$ by a margin of 1%. Given that the target sets are large around 5% of the nodes, this means that 1 out of 5 target nodes are replaced with a new one, obtaining a new game $\mathcal{G}'$. At this point, the instance $(\mathcal{G}', \sigma_0)$ is fed into `Opt` and `Greedy`, which generate two winning strategies $\sigma_0^{\mathtt{Opt}}$, and $\sigma_0^{\mathtt{Greedy}}$, respectively. In addition to them, we also solve $\mathcal{G}'$ from scratch, generating a winning strategy σ_0'. At this point, we evaluate the distance of the original σ_0 with all the three winning strategies of $\mathcal{G}'$ found with these three approaches.

These values are reported in Table 4 in columns "`Opt` dist", "`Greedy` dist", and "scratch dist", respectively. We could observe that the distances obtained from `Opt` and `Greedy` are comparable. This was expected, given the high accuracy of `Greedy` in random games. On the other hand, observe that they improve the distance of the strategy obtained recomputing the solution from scratch by a margin ratio ranging between 2 and 4. This means that the amount of repair needed can be reduced down to 25% when employing Strategy

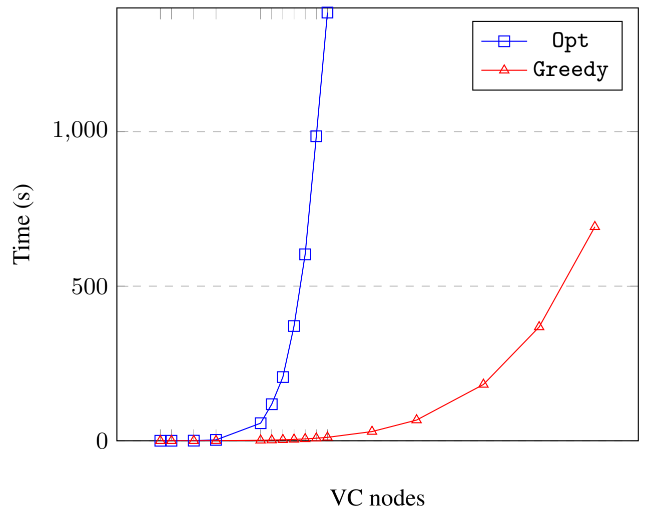

Figure 5: Vertex Cover experiments.

Repair instead of recomputation from scratch.

N. nodes	N. experiments	`Opt` dist	`Greedy` dist	scratch dist
40	5000	0.81	0.811	1.6
60	1000	1.856	1.86	4
100	500	3.5	3.5	10.1
200	100	10.9	10.9	37.2
500	100	55.5	55.5	160
700	5	96	96	257

Table 4: Target deviation experiments.

6 Conclusion

We have introduced *Strategy Repair*, i.e., the problem of turning a losing strategy for a RG into one that is winning, by means of a minimal number of modifications. The work can be seen as transferring the idea of plan repair, originally introduced and studied in Classical Planning, together with its basic principles, to the setting of RGs, which are a more general model, expressive enough to model also FOND Planning. In this way, all the results obtained in the setting of RGs are also immediately available for FOND Planning and, more in general, to all those settings that can be modeled by RGs.

We have devised an algorithmic approach that returns a minimal-distance solution (wrt the number of changes applied to the losing strategy) and characterized its complexity as NP-complete, via reduction from Vertex Cover. To deal with the high-complexity in practice, we have then designed a greedy, polynomial algorithm, together with a very effective heuristic, called MustFix, which has proven extremely efficient in practice, from the point of view of both running time and distance from the initial strategy. The heuristic is also very beneficial to the optimal algorithm, up to the point that on random problems not specifically selected to stress the algorithms, the performance of the optimal approach almost matches that of the polynomial, sub-optimal one. An interesting experimental finding, is that when the set of goal states is subject to small changes, it is more convenient to find a solution by repairing the initial strategy (computed before goal change) than computing one anew. This offers a further powerful tool for efficient RG strategy computation, especially when the initial strategy features properties that are desirable to preserve.

This work is an initial investigation into the problem of Strategy Repair and leaves at least two interesting questions open. Firstly, while the polynomial algorithm, coupled with the MustFix heuristic, exhibits outstanding experimental performance, no approximation guarantee was obtained. For future work, we aim at studying such a property. Secondly, it is interesting to go beyond simple reachability and apply the repair approach to other problems. In particular, one immediate extension would be to investigate applicability and effectiveness of the approach for *strong cyclic* [5] solutions. More in general, the repair approach could be applied to more complex games, such as *parity* or *Büchi* games [11, 15], which would have an immediate impact on more complex forms of planning, such as Classical or FOND Planning for temporally extended goals.

Acknowledgements

The work of Giuseppe Perelli was partially funded by MUR under the PRIN programme, grant B87G22000450001 (PINPOINT), and by the PNRR MUR project PE0000013-FAIR. The work of Fabio Patrizi was partially funded by MUR under the PNRR MUR project PE0000013-FAIR, the ERC Advanced Grant WhiteMech (No. 834228), and the Sapienza Project MARLeN.

References

[1] Henrik Björklund and Sergei G. Vorobyov, 'A Combinatorial Strongly Subexponential Strategy Improvement Algorithm for Mean Payoff Games', *Discret. Appl. Math.*, **155**(2), 210–229, (2007).
[2] Alberto Camacho, Jorge A. Baier, Christian J. Muise, and Sheila A. McIlraith, 'Finite LTL synthesis as planning', in *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018*, eds., Mathijs de Weerdt, Sven Koenig, Gabriele Röger, and Matthijs T. J. Spaan, pp. 29–38. AAAI Press, (2018).
[3] Alberto Camacho, Christian J. Muise, Jorge A. Baier, and Sheila A. McIlraith, 'LTL realizability via safety and reachability games', in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, ed., Jérôme Lang, pp. 4683–4691. ijcai.org, (2018).
[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms, 3rd Edition*, MIT Press, 2009.
[5] Marco Daniele, Paolo Traverso, and Moshe Y. Vardi, 'Strong cyclic planning revisited', in *Recent Advances in AI Planning, 5th European Conference on Planning, ECP'99, Durham, UK, September 8-10, 1999, Proceedings*, eds., Susanne Biundo and Maria Fox, volume 1809 of *Lecture Notes in Computer Science*, pp. 35–48. Springer, (1999).
[6] Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh V. Vazirani, *Algorithms*, McGraw-Hill, 2008.
[7] Giuseppe De Giacomo, Marco Favorito, Jianwen Li, Moshe Y. Vardi, Shengping Xiao, and Shufang Zhu, 'Ltlf synthesis as AND-OR graph search: Knowledge compilation at work', in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, ed., Luc De Raedt, pp. 2591–2598. ijcai.org, (2022).
[8] Giuseppe De Giacomo and Moshe Y. Vardi, 'Synthesis for LTL and LDL on finite traces', in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, eds., Qiang Yang and Michael J. Wooldridge, pp. 1558–1564. AAAI Press, (2015).
[9] Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina, 'Plan stability: Replanning versus plan repair', in *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK, June 6-10, 2006*, eds., Derek Long, Stephen F. Smith, Daniel Borrajo, and Lee McCluskey, pp. 212–221. AAAI, (2006).
[10] Alfonso Gerevini and Ivan Serina, 'Efficient plan adaptation through replanning windows and heuristic goals', *Fundam. Informaticae*, **102**(3-4), 287–323, (2010).
[11] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, eds. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[12] Walter Ludwig, 'A Subexponential Randomized Algorithm for the Simple Stochastic Game Problem', *Inf. Comput.*, **117**(1), 151–155, (1995).

[13] Bernhard Nebel and Jana Koehler, 'Plan reuse versus plan generation: A theoretical and empirical analysis', *Artif. Intell.*, **76**(1-2), 427–454, (1995).

[14] Christos H. Papadimitriou, *Computational complexity*, Academic Internet Publ., 2007.

[15] Dominique Perrin and Jean-Eric Pin, *Infinite words - automata, semigroups, logic and games*, volume 141 of *Pure and applied mathematics series*, Elsevier Morgan Kaufmann, 2004.

[16] Alessandro Saetti and Enrico Scala, 'Optimising the stability in plan repair via compilation', in *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022, Singapore (virtual), June 13-24, 2022*, eds., Akshat Kumar, Sylvie Thiébaux, Pradeep Varakantham, and William Yeoh, pp. 316–320. AAAI Press, (2022).

[17] Enrico Scala and Pietro Torasso, 'Proactive and reactive reconfiguration for the robust execution of multi modality plans', in *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, eds., Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pp. 783–788. IOS Press, (2014).

[18] Enrico Scala and Pietro Torasso, 'Deordering and numeric macro actions for plan repair', in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, eds., Qiang Yang and Michael J. Wooldridge, pp. 1673–1681. AAAI Press, (2015).

[19] Roman van der Krogt and Mathijs de Weerdt, 'Plan repair as an extension of planning', in *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA*, eds., Susanne Biundo, Karen L. Myers, and Kanna Rajan, pp. 161–170. AAAI, (2005).

[20] Jens Vöge and Marcin Jurdziński, 'A discrete strategy improvement algorithm for solving parity games', in *Computer Aided Verification*, eds., E. Allen Emerson and Aravinda Prasad Sistla, pp. 202–215, Berlin, Heidelberg, (2000). Springer Berlin Heidelberg.