Get me to Safety! Escaping from Risks using Automated Planning

Alberto Pozanco, Yolanda E-Martín, Susana Fernández, Daniel Borrajo
Departamento de Informática, Universidad Carlos III de Madrid
Avda. de la Universidad, 30. 28911 Leganés (Madrid), Spain
apozanco@pa.uc3m.es, yescuder@inf.uc3m.es, sfarregu@inf.uc3m.es, dborrajo@ia.uc3m.es

Abstract

This paper focuses on the task of generating *escape plans*. These plans take an agent both in the safest way along the path and as far as possible at the end from states that are considered risks. We refer to this problem as *escape planning*, and formulate it as an optimal sequential planning problem. In this kind of task, there are no explicit goals given unlike most previous work on automated planning; computing those goals (the safest states) has a high computational cost, given that it requires solving a big number of planning problems. We present optimal and suboptimal algorithms to solve this novel planning task, and provide experimental results that show how our algorithms are effective and efficient for solving it.

Introduction

In automated planning, the most common task consists of finding a sequence of actions (namely *plan*) that achieves a set of some given goals from an initial state. Recent work has explored scenarios where goals could not be given initially and/or they change dynamically (Molineaux, Klenk, and Aha 2010; Aha 2018). In this work, we focus on a different kind of planning tasks; those whose goals are only given implicitly. In particular, we are interested in planning tasks where some states given as input are considered as risky and we would like our planning task to find a plan with two characteristics: (1) the execution of the plan takes an agent to a state that is as far as possible from those risks; and (2) the states that the execution of the plan visits should also be as far as possible from the risky states.

As an example, consider a natural disaster domain like the one shown in Figure 1, where a volcano (depicted in the top-left corner of the figure) has erupted. Hence, there are some parts of the nearby region that may become risky. Also, there is some risky area where dangerous animals tend to be. These risk zones are depicted in red, with the level of risk being higher as the red color is darker. Given this situation, the person located in the top part of the map would like to move to a safer (even the safest) place of the map. She can cross the river (depicted with blue cells) by taking the canoe. However, the person does not have a explicit goal, i.e., she does not know where she should go. She just wants to be as far as possible from the risks. The questions we address in this paper are: which states represent the safest places?; and what plan do people need to follow to reach such states and, therefore, minimize their proximity to risk zones?

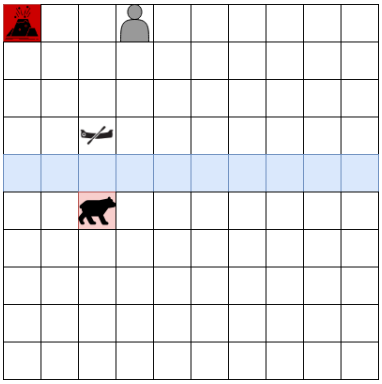


Figure 1: Natural disaster domain. Red cells represent risky areas, with the level of risk being higher as the red color is darker. Blue cells represent a river that the person can cross by taking the canoe.

In addition to any domain related to natural disasters, there are many other real world domains where we are not provided with an explicit goal definition as input. The only objective is to escape from the risks, being as far as possible from them. In these cases, the task consists on finding escape plans and/or safest states. For instance, domains such as: games where we want to avoid or escape from a set of opponents and/or dangerous zones; choosing safe places where to construct buildings with high risk of public safety or health; or choosing the nodes in a network where to store sensitive electronic data to keep them safe from potential threats. Some of these domains do not use a grid to model the problem. Therefore, the distance to a risk is not necessarily determined by the topological distance. In these cases, we can use automated planning to compute (or estimate) distances between states (risks) in order to obtain the set of safest places of a planning problem. The distance between two states is the cost of a plan that reaches one state from the other.

In case we would need to pose this task as a standard planning task, we would need to first compute the set of planning goals. Using a naïve approach, given k risks and being $\mid S \mid$

the size of the state space S, we would have to first solve $k \mid S \mid$ optimization tasks in order to know what the safest states are, and thus our goals. Therefore, the naïve strategy would require a huge computational effort.

Our planning task will consist of finding a plan that visits states with maximum average distance to those risks. Such states can be seen as the *safest* states, and the plans that visit them as *escape* plans. We refer to this task as *escape planning*, which amounts to finding a sequential plan that ends in one of the safest states of the problem. These states are not known a priori, and are obtained through solving an optimization problem that finds those states maximizing the average distance to the risks. Furthermore, the path to those safest states should have minimum cost, where the cost of a path is the sum of the costs of the states along the path, where again the cost of a state is related to its average distance from risky states.

Prior work focuses on finding escape or evacuation plans. However, they often compute the solutions over grids and networks where they solve path finding tasks. Moreover, they are domain-dependent, focusing on specific tasks such as evacuation of people in fire scenarios (Olsen, Granmo, and Radianti 2015) or vehicle routing in critical contexts such as homeland defense preparation (Lu, George, and Shekhar 2005). In contrast, our approach is domain-independent and can find safest states and escape plans in many other scenarios.

Other publications also focus on computing risk-sensitive solutions in a domain-independent way using state space search (Koenig and Simmons 1994; Perny, Spanjaard, and Storme 2007; Jeantet and Spanjaard 2008). These approaches are directed by a specific goal. They have an initial goal and try to find a plan that maximizes the probability of its achievement or minimizes the expected execution cost of the plan. Most of these approaches are based on utility theory (Morgenstern and Von Neumann 1953) or rank-dependent utility (Quiggin 2012). Our approach, by contrast, is not goal-directed; that is, we do not have a initial goal to pursue. Sohrabi *et al.* (2018) also used automated planning to handle risks in their Scenario Planning Advisor decision-support system. Instead, we: (1) consider the set of risks as potential goals rather than as observations; and (2) find plans that avoid those risks and reach the safest states.

Escape planning also has some relationship with over-subscription planning (Smith 2004; Domshlak and Mirkis 2015), where the objective is to achieve the highest utility subset of a given set of goals, taking into account the cost of achieving them. In our case, we would have to set as soft goals all the reachable states of the planning task, which would turn out to be infeasible even for small problems.

The work on defining and reasoning with path constraints using some kind of linear temporal logic (LTL) is also relevant (Baier, Bacchus, and McIlraith 2009). That would be useful to define that the states visited by the plan should be as far as possible from the risky states. However, we find it difficult to define our constraints in terms of LTL.

Our work is inspired by recent work on finding planning centroids (Pozanco et al. 2019). While its authors were interested in finding a state that minimizes the distance to a given set of goals, we are interested in maximizing the distance to a set of potential risks, not goals as we discussed before. The new task we propose requires the development of new planning algorithms. Besides, we extended their work by (1) assigning a weight to each potential risk; and (2) also focusing on the plans that achieve these states.

The main contributions of this paper are:

- Definition of an escape planning task in terms of a planning model and a set of potential risks with an associated weight.
- Definition of the safest states and escape plans of an escape planning task.
- Definition of a common algorithm with five different parametrizations for computing safest states and escape plans both optimally and suboptimally.
- Experimental results and analysis that show how these algorithms behave in some escape planning tasks.

This paper focuses on escape planning. However, many of the ideas and implementations in this paper would apply to a even more general kind of planning task where no explicit goals are given and computing them would require the computation of a distance metric between each state and those goals before the search starts. Also, the proposed approach would be able to handle path optimization functions relative to the cost of the states in the path to the goals.

The rest of the paper is organized as follows. We begin reviewing some basic notions of classical planning. Then, we formally define planning risks, escape planning tasks, safest states, and escape plans. Subsequently, we introduce the algorithm we use to compute such states and plans. Finally, we present an empirical study in several planning domains and discuss future work.

Background

Classical automated planning is the task of choosing a sequence of actions such that, when applied to a given initial state, it results in a goal state (Ghallab, Nau, and Traverso 2004). Formally, a single-agent STRIPS planning task can be defined as a tuple $\Pi = \langle F, A, I, G \rangle$, where F is a set of propositions, A is a set of instantiated actions, $I \subseteq F$ is an initial state, and $G \subseteq F$ is a set of goals.

A state consists of a set of propositions $s \subseteq F$ that are true at a given time. A state is totally specified if it assigns truth values to all the propositions in F, as the initial state I of a planning task. A state is partially specified (partial state) if it assigns truth values to only a subset of the propositions in F, as the goals G of a planning task.

Each action $a \in A$ is described by a set of preconditions (pre(a)), which represent literals that must be true in a state to execute an action, and a set of effects (eff(a)), which are the literals that become true (add(a) effects) or false (del(a) effects) from the state after the action execution. The definition of each action might also include a cost $c(a)$ (the default cost is one). The execution of an action a in a state s is defined by a function γ such that $\gamma(s, a) = (s \backslash \text{del}(a)) \cup \text{add}(a)$ if pre(a)$\subseteq s$, and s otherwise (it cannot be applied). The output of a planning task is a sequence of actions, called a plan,

$\pi = (a_1, \ldots, a_n)$. The execution of a plan π in a state s can be defined as:

$$\Gamma(s, \pi) = \begin{cases} \Gamma(\gamma(s, a_1), (a_2, \ldots, a_n)) & \text{if } \pi \neq \emptyset \\ s & \text{if } \pi = \emptyset \end{cases}$$

A plan π is valid for solving Π iff $G \subseteq \Gamma(I, \pi)$. The plan cost is commonly defined as $c(\pi) = \sum_{a_i \in \pi} c(a_i)$, where $c : A \rightarrow \mathbb{R}_{>0}$ is a non-negative action cost function. A plan with minimal cost is called optimal.

Definitions

In this section, we will define planning risks, safest states and escape plans. We define a planning risk as follows:

Definition 1 *Given a planning task Π, a **planning risk** R is a tuple $R = \langle T, w_T \rangle$, where $T \subseteq F$ is a partial state (risk) and $w_T \in \mathbb{R}\{0, 1\}$ is a number reflecting the weight of T. $\mathcal{R}$ is the set of all planning risks of a problem.*

As an example, in the problem in Figure 1, $\langle$`at(person,t4-10)`,0.8$\rangle$ would reflect the fact that being the person at the tile in the fourth column tenth row, would present a risk with a weight of 0.8.

In this work we focus on the single-agent classical planning setting previously described, where we make the following assumptions: (1) full observability of the state, which can only change due to the actions performed by the agent; (2) deterministic action outcomes; and (3) the set of planning risks $\mathcal{R}$ does not change during the time when the escape plan is computed.

With these assumptions, we formalize an escape planning task as follows:

Definition 2 *A **escape planning task** is a tuple $\mathcal{E} = \langle F, A, I, \mathcal{R} \rangle$, where F, A, and I are equivalent to a standard planning task Π; and $\mathcal{R}$ is a set of planning risks we want to escape from.*

In a geometric space, computing the point in the space with maximum distance with respect to others is straightforward since: (1) every point is reachable from any other; (2) points' coordinates are fully specified; and (3) the distance between points can be simply calculated. In automated planning, there are states instead of points. These states are not necessarily reachable from all other states nor fully specified (we can handle partial states as, for instance, risks), and computing the distance between two states becomes a problem with a PSPACE complexity.

In order to deal with reachability and distance between planning states, we provide the following standard definitions.

Definition 3 *A **state** s is **reachable** from I (or simply reachable) iff there is at least one valid plan π such that its execution from I, $\Gamma(I, \pi)$, achieves s; i.e. $s \subseteq \Gamma(I, \pi)$. $\mathcal{B}_\mathcal{E}$ is the set of all reachable states from I in $\mathcal{E}$. The distance between both states is the plan's cost $c(\pi)$.*

In automated planning, we cannot compute the distance between two partial states, such as risks. However, heuristics allow us to compute the distance between a regular (full) state s and a partial state (representing a risk), T. We use here the function cost(s, T, h) to compute the cost of achieving T from s using the cost estimator h. This function can compute either the actual optimal cost h^* of achieving T from s, or an estimation of that cost (as the one returned by a heuristic function).

Our aim is to find states that maximize the weighted average distance to reach a set of risks, with the aim of avoiding them. Therefore, we will define the distance of a state to a set of risks as its safety by computing that weighted average.

Definition 4 *Given an escape planning task $\mathcal{E}$ with an associated set of risks $\mathcal{R} = \{\langle T_1, w_{T_1} \rangle, \ldots, \langle T_n, w_{T_n} \rangle\}$, and a cost estimator h, the **safety of a state** $s \in \mathcal{B}_\mathcal{E}$, $\mu_{\mathcal{E},h}(s)$, is the weighted average cost of reaching the risks from s:*

$$\mu_{\mathcal{E},h}(s) = \frac{\sum_{i=1}^{n} w_{T_i} \times cost(s, T_i, h)}{\sum_{i=1}^{n} w_{T_i}}$$

We will refer as ComputeSafety$(\mathcal{E}, s, h)$ to the procedure that computes the safety of a state. Let us clarify this by computing the safety of the initial state depicted in Figure 1. Let us assume that the weighted risk of the volcano is 1 and the weighted risk of the bear is 0.5. The heuristic h we employ is equivalent to the Manhattan distance, and hence we assume the agent can cross the river without taking the canoe. In this case, $\mu_{\mathcal{E},h}(s) = ((1 \times 3) + (0.5 \times 6))/(1 + 0.5) = 4$. With this measure of distance, we can compare states of an escape planning task in terms of their safety:

Definition 5 *Given two states $s_1, s_2 \in \mathcal{B}_\mathcal{E}$, we say that s_1 is a **safer state** than s_2 iff $\mu_{\mathcal{E},h}(s_1) > \mu_{\mathcal{E},h}(s_2)$.*

Or, in words, a state is safer than another state if it is further away from the risks. This comparison depends on the heuristic we employ to compute the distance to the risks (see Definition 4). If we want to find the safest state of a escape planning task, we must use the perfect heuristic estimator h^* and show that the given state is safer than all the other reachable states. Formally:

Definition 6 *A state $s \in \mathcal{B}_\mathcal{E}$ is a **safest state** of an escape planning task $\mathcal{E}$ iff $\forall s' \in \mathcal{B}_\mathcal{E}$, $\mu_{\mathcal{E},h^*}(s) \geq \mu_{\mathcal{E},h^*}(s')$. The set of safest states of an escape planning task $\mathcal{E}$ is denoted as $\mathcal{S}^*(\mathcal{E})$.*

Computing these states optimally requires generating an optimal plan from each reachable state to each risk, i.e., it will be necessary to optimally solve $|\mathcal{B}_\mathcal{E}| \times |\mathcal{R}|$ planning tasks, which are itself PSPACE.

Definition 7 *A plan is a **escape plan** of a planning task $\mathcal{E}$, denoted as $\pi_\mathcal{E}$, iff $s = \Gamma(I, \pi)$ and $\mu_{\mathcal{E},h^*}(s) > \mu_{\mathcal{E},h^*}(I)$.*

So, any plan that takes an agent to a state safer than the initial state would be an escape plan. However, this definition is quite loose since a plan consisting of just one action that greedily pushes the agent away of the risks would be considered as an escape plan. We are interested in escape plans that have more safety guarantees. Better escape plans will then reach safest states, and we say they are end-safe.

Definition 8 *an escape plan $\pi_\mathcal{E}$ is an **end-safe escape plan** iff $\exists s_i \in \mathcal{S}^*(\mathcal{E})$ such that $s_i \subseteq \Gamma(I, \pi_\mathcal{E})$.*

Safest states, like every other state in planning, can be reached via multiple plans that meet different criteria such as being the shortest or the cheapest. In this case, we aim to find escape plans that maximize the cost (safety) to a set of risks across all the path to one of the safest states. In other words, we are interested in paths that reach safest states through the least risky states possible, understanding risk as the inverse of safety. We define the cost of an escape plan as follows.

Definition 9 *The **cost** of a **escape plan** $c(\pi_\mathcal{E})$ is the sum of all the inverse safety (risk) values of the states traversed by $\pi_\mathcal{E}$:*

$$c(\pi_\mathcal{E}) = \sum_{a_i \in \pi_\mathcal{E}} (\kappa - \mu_{\mathcal{E},h^*}(\gamma(s_{i-1}, a_i)))$$

where κ is a very large constant number to transform the safety of a state into risk. $\pi_\mathcal{E}$ will be optimal if there is no other end-safe plan $\pi'_\mathcal{E}$ with lower cost.

By using this cost function, we are not only preferring to visit low risk states, but also reaching a safest state as soon as possible (by performing the minimum number of actions). We believe that this is a valid optimization function in a good number of domains. In other cases, one could prefer longer paths visiting many very low risk states. In those cases, it would be enough to divide $c(\pi_\mathcal{E})$ by the plan length. As we will show later, our algorithm is agnostic with respect to the cost function employed, so any other function could be easily incorporated. Due to lack of space, we cannot include a proper comparison among different cost functions, leaving it as a possible future work.

For the sake of clarity, let us summarize all these concepts. For a plan to be considered as an escape plan, it must reach a state safer than the initial state; i.e. farther away from the risks. If an escape plan reaches a safest state (the state which is furthest from the risks), we say it is end-safe. And such end-safe escape plans may be optimal iff they minimize the sum of the risks of all the states along the path.

Computing Safest States and Escape Plans

Optimal, and more generally end-safe escape plans, need to reach safest states, which are not known a priori. Computing these states requires exploring all the reachable state space using a perfect heuristic estimator h^* (see Definition 6). Since this is a very expensive task, we want to: compute escape plans at the same time we explore the state space; and relax our definitions to compute suboptimal states and plans.

Following Pozanco *et al.* (2019) approach, we use a common algorithm to compute safest states and escape plans both optimal- and suboptimally, depending on the input parameters. We call it SEPAP, which stands for Safe Escape Plans using Automated Planning. Algorithm 1 details the full procedure. It consists of a Best-First Search algorithm that receives as input: an escape planning task $\mathcal{E}$; the function h used to compute the safety of the states; a parameter o that determines whether we are re-opening nodes or not; and the stopping condition of the algorithm τ.

The algorithm expands nodes until the stopping condition τ is met. For each search node, we store the planning state, the current g, the risk value, and the path that reaches the node. The risk value of a node is computed as the difference between κ and the safety value of the state (see Definition 4). Therefore, states with lower safety values (and hence, closer to the risks) will have higher risk values. g stores the sum of the risk values of all the states that lead to the node (see Definition 9). In lines 11-12, the algorithm sets the value of g and the path for the new generated nodes. The algorithm also performs these two operations in case we allow for re-opening (o parameter) and we found a safer path (line 10). Note that in case of preferring a different cost function, it will suffice to properly change these lines. Then the algorithm checks if the risk value of the current state is equal to or lower than the lowest risk value found so far, stored in $\text{risk}_\mathcal{S}$. If so, $\mathcal{S}$ is updated. In the case where the risk value is lower than $\text{risk}_\mathcal{S}$, we also update $\text{risk}_\mathcal{S}$ (lines 15-17). When τ is met, the algorithm returns S, the safest state found (i.e., the state with maximum safety), and $\pi_\mathcal{E}$, the best escape plan found (i.e., the plan that reaches that state with lower accumulated risk).

Algorithm 1 SEPAP

Inputs: $\mathcal{E} = \langle F, A, I, \mathcal{R} \rangle$, h, o, τ
Outputs: $S, \pi_\mathcal{E}$

```
1: κ ← large constant
2: g_I ← 0
3: risk_S ← κ−ComputeSafety(E, I, h)
4: open, S ← {⟨I, g_I, risk_S, ∅⟩}
5: while not τ do
6:     ⟨n, g_n, risk_n, path_n⟩ ←Pop(open, min(risk))
7:     successors ←GenerateSuccessors(n)
8:     for s, a in successors do
9:         risk_s ← κ−ComputeSafety(E, s, h)
10:        if IsNew(s) or (o and g_s > g_n + risk_s) then
11:            g_s ← g_n + risk_s
12:            path_s ← path_n + a
13:            if IsNew(s) then
14:                open ← open ∪ {⟨s, g_s, risk_s, path_s⟩}
15:        if risk_s < risk_S then
16:            S ← {⟨s, g_s, risk_s, path_s⟩}
17:            risk_S ← risk_s
18:        else if risk_s = risk_S then
19:            S ← S ∪ {⟨s, g_s, risk_s, path_s⟩}
20: ⟨S, g_S, risk_S, π_E⟩ ← argmin_{x∈S} g(x)
21: return S, π_E
```

Evaluation

We implemented SEPAP on top of FastDownward (Helmert 2006). By varying the parameters h, o, and τ we obtain five different algorithms, as shown in Table 1. These algorithms can also be seen as five different parametrizations of the same algorithm. However, since these parametrizations have completely different theoretical properties, we use the term algorithms instead.

We use the FF heuristic, h_{FF} (Hoffmann and Nebel 2001), to estimate the distance to the risks in the suboptimal versions of SEPAP, since it is a fast and rather accurate heuristic in many domains. In the optimal version, h^* is computed from the cost of optimal plans to the risks at each search node using A^* with the LMCUT admissible heuris-

tic (Helmert and Domshlak 2009).

Version	h	o	τ
SEPAP*	h^*	Yes	$\mathcal{B}_\mathcal{E}$
SEPAP^f	h_{FF}	Yes	$\mathcal{B}_\mathcal{E}$
SEPAP^{f-}	h_{FF}	No	$\mathcal{B}_\mathcal{E}$
SEPAP^{hc}	h_{FF}	Yes	Greedy + Hill Climbing
SEPAP^g	h_{FF}	Yes	Greedy

Table 1: Five versions of SEPAP obtained by varying: (1) the heuristic function h; (2) the node re-opening policy o; and (3) the stoping condition τ.

We employ three different stopping conditions τ. The first one, $\mathcal{B}_\mathcal{E}$, explores all the reachable state space. The second one, Greedy, stops the search when there is no state in Open with a better (lower) risk value than the best state visited so far. This is used by the greediest version of the algorithm, SEPAP^g. The third condition, Greedy+Hill Climbing, employed by SEPAP^{hc}, also performs a greedy search, but it tries to escape from local minima by performing 50 random actions[1] from the state found by SEPAP^g. It follows a stochastic hill climbing approach that: (1) generates the successors of a state; (2) randomly selects one of them; (3) evaluates it to check if it is better than the best state visited so far – if so, it updates $\mathcal{S}$, which contains the best state observed –; and (4) goes back to (1) until it reaches the maximum number of iterations (50).

Most versions of the algorithm re-open nodes to find a safer path to the states. However, this computation may be too expensive, especially in the versions that explore all the reachable state space. That is why we include SEPAP^{f-}, which behaves like SEPAP^f, but does not re-open nodes (parameter o). This alternative may potentially reduce the search effort when computing suboptimal solutions (Chen et al. 2019).

The experimental evaluation is divided as follows. First, we perform an evaluation of suboptimality of plans in small problems taken from two domains to test how the suboptimal versions compare to the optimal one. Second, we increase the size of the problems in a controlled way to test how all versions scale. Third, we continue our quantitative evaluation in large planning instances taken from different International Planning Competitions (IPCs)[2]. Finally, we perform a qualitative assessment of the safest states and escape plans returned by each algorithm.

Reproducibility. The experiments were run on Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz machines with a time limit of 3600s and a memory limit of 8GB. All the domains and problems mentioned from now on are already available on-line (`blind`).

Suboptimality Evaluation

Our first set of experiments aims at comparing the different versions of SEPAP. We perform a set of experiments in small problems where the optimal version can explore all the reachable state space computing optimal plans to the risks. In those problems, the size of the reachable state space is small, $|\mathcal{B}_\mathcal{E}| \leq 1000$. We selected the following two domains, generating 10 random escape planning tasks for each one:

- GRID: a path-planning like domain where an agent can move to adjacent cells. The map size is 20×20 and 20% of the cells are obstacles that the agent cannot pass through. The risks are four random cells that the agent should stay away from. The weight of each risk is randomly generated.
- BLOCKS WORDS: a variation of the well-known Blocksworld domain, where an agent can build words using five available blocks (Ramírez and Geffner 2009). Each problem has three potential risks, which are risky words that the agent wants to avoid, i.e., she wants to be as far as possible from building any of the risky words. The weight of each risk is randomly generated.

Domain	Version	μ	$\Delta(\mu)$	$\lvert\pi_\mathcal{E}\rvert$	r	t
GRID	SEPAP*	25.7±2.9	11.4±5.8	19.3±8.9	19951.1±8814.5	2103.7±421.9
	SEPAP^f	25.7±2.9	11.4±5.8	19.3±8.9	19951.1±8814.5	1.2±0.5
	SEPAP^{f-}	25.7±2.9	11.4±5.8	25.9±14.2	26410.3±13961.7	0.6±0.0
	SEPAP^{hc}	18.7±5.9	4.4±4.2	5.2±4.5	6096.6±4404.4	0.5±0.0
	SEPAP^g	18.6±6.0	4.3±4.3	4.8±4.7	5800.8±4569.0	0.5±0.0
BLOCKS WORDS	SEPAP*	14.7±0.7	5.7±3.6	8.0±2.7	8934.1±2658.2	3435.2±161.6
	SEPAP^f	11.8±0.9	2.8±4.0	10.0±1.7	10928.6±1716.1	0.5±0.1
	SEPAP^{f-}	11.8±0.9	2.8±4.0	18.6±8.2	19483.9±8121.1	0.3±0.0
	SEPAP^{hc}	11.3±1.8	2.3±3.4	6.7±4.6	7643.5±4593.2	0.2±0.0
	SEPAP^g	10.5±1.8	1.5±3.1	3.1±2.3	4268.8±2040.0	0.2±0.0

Table 2: Comparison of different versions of the algorithm on easy problems. For each domain, each row shows the results obtained by each version. Columns show averages and standard deviations over the set of problems for: safety of the returned state (μ); safety improvement with respect to the initial state ($\Delta(\mu)$); length of the escape plan ($\pi_\mathcal{E}$); accumulated risk of the escape plan (r); and running time (t).

Table 2 summarizes the results of the quantitative evaluation. For each domain, each row shows the average and standard deviation over 10 problems of the results obtained by each of the five versions. Each column represents different measures of quality and performance:

- μ: weighted average cost to the risks (safety), for the state S returned by the algorithm. This safety value is computed using h^*. Higher values are better, since they imply a safer state has been reached.
- $\Delta(\mu)$: percentage of improvement of μ for the state S returned by the algorithm with respect to the initial state I. Again, higher values are better.
- $|\pi_\mathcal{E}|$: number of actions in the escape plan.
- r: accumulated risk of the escape plan. This value is computed using $\kappa = 1000$ and h^* in Algorithm 1.
- t: time in seconds to return a solution, S and $\pi_\mathcal{E}$.

The optimal version of the algorithm, SEPAP* obtains the best results, as expected. However, its execution times are at least three orders of magnitude higher than the ones of the suboptimal versions. It explores all the reachable state space from the initial state, computing an optimal plan to each risk

[1] Best value found during our experimental evaluation.

[2] `http://ipc.icaps-conference.org`

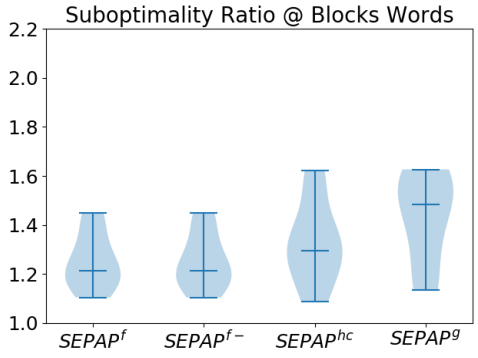


Figure 2: Degradation of SEPAP^{f}, SEPAP^{f-}, SEPAP^{hc}, and SEPAP^{g} with respect to SEPAP* in BLOCKS WORDS. The horizontal line indicates the median of the distribution, while the other lines indicate the maximum and minimum values. A wider blue shadow indicates that more points have that value.

in every state. SEPAP^{f} and SEPAP^{f-} also explore the reachable state space, but computing a heuristic rather than an optimal plan to estimate the distance to the risks. These algorithms obtain results in about one second that are close to or even optimal. As we can see, avoiding node re-opening can speed-up the computation, obtaining the same safest states with the counterpart of having some cases where the plans that achieve them are riskier and have more actions. The greediest versions of the algorithm, SEPAP^{hc} and SEPAP^{g} are faster, but the safest states they return are far from the optimum in some cases. Their associated escape plans tend to be shorter, therefore having lower accumulated risk compared to the optimal version. We can also see how the hill climbing version of the algorithm improves the results of the greedy search in both domains. Overall, we can say that suboptimal versions present a good balance between quality and time, obtaining average results that are close to the optimum in some cases, and very fast.

We can also observe this behavior in Figure 2, where we measure how the suboptimal versions compare to the optimal one with respect to the safety, μ, of the returned state. As we can see, SEPAP^{f} and SEPAP^{f-}, the versions that explore all the reachable state space using a heuristic to compute the distance to the risks, obtain results that are at most 1.5 times suboptimal in BLOCKS WORDS. The greediest version, SEPAP^{g}, has the worst performance, but rarely returns safest states that are more than 1.5 times suboptimal. We can also observe this behavior in the GRID domain, where suboptimal versions obtain results even closer to the optimal one due to the lower number of local minima.

Scalability Evaluation

We generated a set of GRID problems with increasing reachable states and risks to test how the different algorithms scale. We increase the reachable state space by considering grids of 5×5, 10×10, 20×20, and 50×50 with no obstacles; and increase the number of risks, $|\mathcal{R}|$, in powers of 2 from 2 to 16. We generated 10 problems for each combination of grid size and number of risks (16 combinations). The results we report hereafter are average among these problems. The optimal version of the algorithm only solves 9 out of the 16 combinations. The versions that also explore all the reachable state space solve 14 out of 16. They fail in the problems with a 50×50 grid with 8 and 16 risks. The greediest versions are able to solve all problems in all cases.

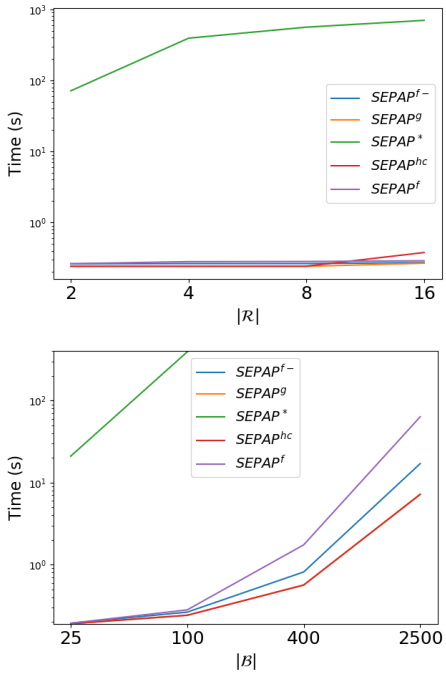


Figure 3: Time in logarithmic scale needed for each algorithm to return a solution. The upper image shows the results in 10×10 problems of GRID when we increase the number of risks. The lower image shows the results when we fix the number of risks to 4 and increase the grid size, where $|\mathcal{B}|$ goes from 25 (5×5) to 2500 ($50x50$).

Figure 3 shows how the different algorithms scale when we fix either the number of risks or the number of reachable states. We measure the time needed for each algorithm to return a solution. The upper image shows the results in a 10×10 problem of GRID when we increase the number of risks. The lower image shows the results when we fix the number of risks to 4 and increase the square size from 5 to 50. As expected, the execution times of the optimal algorithm are various orders of magnitude higher than the ones of the suboptimal versions. In the upper image we can see that only the optimal algorithm gets affected by increasing the number of risks. This is because it computes optimal plans to them. On the other hand, the suboptimal versions use a heuristic estimator to compute the distance to the risks. That is why they are not affected by increasing $|\mathcal{R}|$. We also observed that the length of the escape plans decreases as the number of risks increases. If the state space is full of risks that are randomly distributed, it is more likely that the safest

Domain	Version	μ	$\Delta(\mu)$	$\lvert\pi_\mathcal{E}\rvert$	r	t	$\#s$
LOGISTICS	SEPAP^{f}	4.1±0.9	4.1±0.9	16.3±2.6	17290.8±2612.4	1.8±2.1	3
	SEPAP^{f-}	7.5±2.9	7.5±2.9	40.4±17.3	41165.5±17140.9	11.7±18.2	10
	SEPAP^{hc}	4.7±1.6	4.7±1.6	23.3±10.3	24225.1±10269.6	0.2±0.0	10
	SEPAP^{g}	4.4±1.7	4.4±1.7	16.6±6.2	17544.2±6138.3	0.2±0.0	10
FERRY	SEPAP^{f}	-	-	-	-	-	0
	SEPAP^{f-}	-	-	-	-	-	0
	SEPAP^{hc}	2.7±1.0	2.7±1.0	14.1±7.8	15076.1±7768.0	0.2±0.0	10
	SEPAP^{g}	2.5±1.0	2.5±1.0	9.4±3.2	10383.4±3223.5	0.2±0.0	10
GRIPPER	SEPAP^{f}	3.7±0.3	3.7±0.3	21.0±8.5	22000.0±8485.3	189.4±331.2	5
	SEPAP^{f-}	3.8±0.2	3.8±0.2	31.0±12.9	32000.0±12917.3	320.1±635.8	7
	SEPAP^{hc}	2.2±0.7	2.2±0.7	18.7±7.4	19700.0±7416.9	0.2±0.0	10
	SEPAP^{g}	1.9±0.8	1.9±0.8	11.4±2.0	12400.0±1959.6	0.2±0.0	10
HANOI	SEPAP^{f}	30.3±36.5	30.3±36.5	30.3±36.5	30288.1±34401.6	0.7±0.9	7
	SEPAP^{f-}	53.9±71.1	53.9±71.1	93.8±142.7	86415.1±125379.2	1.1±1.7	8
	SEPAP^{hc}	2.7±0.9	2.7±0.9	2.7±0.9	3794.5±598.5	0.2±0.0	10
	SEPAP^{g}	2.7±0.9	2.7±0.9	2.7±0.9	3794.5±598.5	0.2±0.0	10

Table 3: Comparison of different versions of the algorithm on large problem instances. For each domain and algorithm, we measure the same metrics as in Table 2, plus the number of problems solved ($\#s$) by each version.

states are not far from the initial state. In the lower image we can see that all the algorithms are affected by increasing the number of reachable states. The execution time grows exponentially for the suboptimal versions, but they are still able to solve all the problems.

Safest States on Large Planning Instances

As we have seen, suboptimal versions can compute safest states and escape plans that are quite close to the optimal one in a reasonable amount of time. However, we used small instances where the optimum can be computed in most cases. The set of reachable states of a standard planning task is much larger. In this section, we evaluate the suboptimal versions of the algorithm in planning domains taken from different IPCs. For each domain, we selected the first 10 problems, which typically have an increasing level of difficulty. Our first attempt was to take the goals from the original problem and turn them into risks, assigning them a random weight. However, the returned escape plans in those cases were only composed by one or two actions; in the IPC, goals tend to be far from the initial state, so that planners generate non-trivial plans. Since we were turning the original goals into risks, the initial state might be quite safe, thus leaving little room for improvement. Finally, we decided to turn the dynamic propositions[3] of the initial state of the problem into risks, with the aim of escaping from the initial state. Table 3 summarizes the results of a quantitative evaluation. We measure the same metrics as in Table 2, adding now the number of problems solved within the given time and memory bounds ($\#s$) .

There are some tasks that SEPAP^{f} and SEPAP^{f-} cannot solve, since they cannot explore all the reachable state space within the given time and memory bounds. On the other hand, SEPAP^{hc} and SEPAP^{g} always return a solution in less than a second. The average results should be read carefully in this set of experiments, since there are versions that solve problems that others do not. For instance, SEPAP^{f-} seems to be slower than SEPAP^{f}. This would contradict our previous analyses, but it is only due to the fact that SEPAP^{f-} is able to solve harder instances. In this case, the Δ values are equal to the safety of the returned state, μ, since we consider the initial state as risky, thus having a safety value of 0.

All versions return escape plans in most domains and instances we tested. A remarkable exception is HANOI, where we could design problem instances where the returned plans reach states that are less safe than the initial state (negative $\Delta(\mu)$). This is because the h_{FF} heuristic ignores delete effects, and might be inaccurate in some domains such as HANOI. This makes the algorithm to prefer states that are actually closer to the risks in some circumstances. This behavior strongly depends on the combination of heuristic and domain (we have only observed it in this domain using h_{FF}). Also, it is not related to the heuristic being admissible or not. Consider any domain and problem with an inaccurate yet admissible heuristic. It might assign higher h values to states that are actually (h^*) closer to the goals (risks in this case), therefore misleading our algorithm, which will think that these states are safer. The possible solutions to this problem would be to: (1) study the use of heuristics that are as close as possible to h^*, regardless if they are admissible or not; and (2) experimentally test which heuristics behave better in certain domains. Although these research lines are interesting, they are out of the scope of this paper, and we leave them as future work.

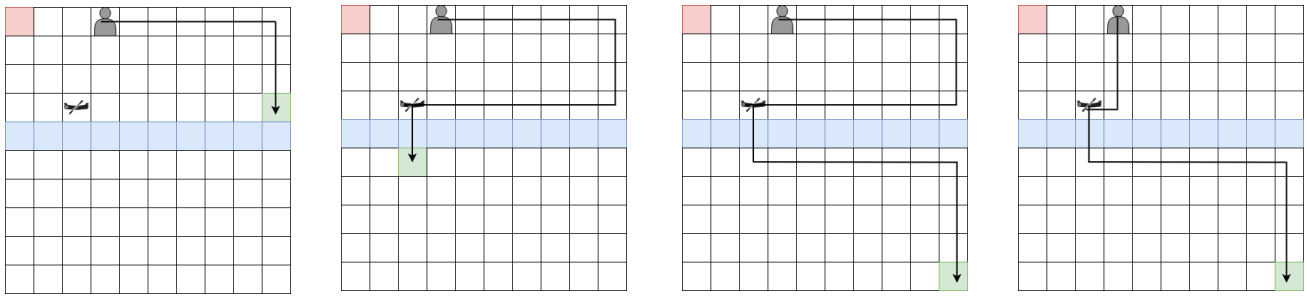
Qualitative Assessment of Escape Plans

Finally, we want to see how safest states and escape plans look like in different domains, and how the different algorithms behave. Figure 4 shows a graphical representation of the states and plans returned by the four suboptimal versions of SEPAP in the natural disaster domain of our running example.

As we can see, the greediest algorithm, SEPAP^{g}, falls into local minima. It is not able to realize that catching the boat can make the person move further away from the risk, and it generates a greedy plan to move her to the farthest place from danger on the north side of the river. SEPAP^{hc} performs a local search from the state returned by SEPAP^{g} and finds a better plan that involves crossing the river with the boat. On the other hand, SEPAP^{f-} and SEPAP^{f}, which explore all the reachable state space, find the safest state of the planning task, thus returning end-safe escape plans. However, SEPAP^{f} computes an optimal escape plan that maximizes the distance to the risk along all the path to the safest state while SEPAP^{f-} returns a suboptimal escape plan. The plan returned by SEPAP^{f} is also a shortest path to the safest state. There exist many other optimal paths in terms of cost in this particular case, but the algorithm is able to return the one that also maximizes the distance to the risk. This is also the case of the optimal version, SEPAP*, which returns the same escape plan as SEPAP^{f}.

Figure 5 shows a safest state in a BLOCKS WORDS instance. The optimal version of the algorithm returns an escape plan composed of 10 actions that groups all blocks in one tower forming the word PETRA. From this state, it is costly to build any of the risky words, so it is a safest state. We observed this behavior of grouping all the blocks into a big tower in most BLOCKS WORDS instances.

[3]Dynamic propositions are the ones whose truth value can be affected by at least one action.



(a) SEPAP^{g} (b) SEPAP^{hc} (c) SEPAP^{f-} (d) SEPAP^{f}

Figure 4: Safest states (in green) and escape plans (black arrow) computed using different versions of SEPAP in the natural disaster domain.

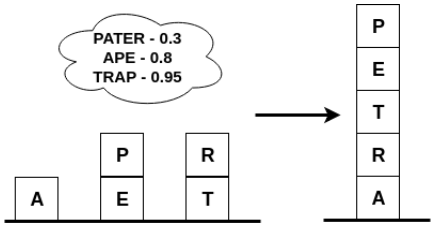


Figure 5: Safest state in a BLOCKS WORDS instance. The left part depicts the initial state and the set of risky words with their associated weight. The right part depicts the safest state returned by SEPAP*.

Discussion and Future Work

This paper presents the formal definition of a new kind of planning task, that of finding the safest states, which are the farthest possible from a set of risks. Also, we defined the task of finding end-safe plans that achieve those safest states with the minimum risk. These new planning tasks required the development of new planning algorithms that provide both outputs. We introduced a common algorithm with five different parametrizations to solve these tasks, by varying suboptimality requirements. Finally, we evaluated our approach in several planning domains, reporting both quantitative and qualitative results.

Experimental results showed that automated planning can effectively solve this kind of planning tasks, that of escaping from risks. Among its virtues, the presented approach: (1) is domain-independent; (2) defines risks as a combination of partial states and weights, which allow for a richer risks' representation, as using for example influence maps from games (Uriarte and Ontañón 2012); and (3) it is straightforward to extract explanations about the returned escape plans and/or safest states, which might be useful in real-world applications (Chakraborti et al. 2019).

Computing optimal escape plans turns out to be infeasible in practice, since it is necessary to explore all the reachable state space, computing an optimal plan to each risk. However, we have shown that suboptimal versions obtain results close to the optimal very fast, also scaling up to larger instances. Thus, we can run a greedy algorithm if we want to rapidly escape from risks, reaching safer states fast. This can be very useful in games where we want to avoid risky states or prepare the defense against an imminent attack. But we can also run either SEPAP^{f-} or SEPAP^{f}, if we have more time and want high quality escape plans. This can be the case of natural disaster management, where we can plan a set of escape plans prior to a disaster.

In future work, we would like to make the algorithm anytime, so users can stop the algorithm at their will, returning the best states/plans found so far. We are also interested in allowing users to insert their preferences into the algorithm. As instance, one could want an escape plan with a maximum level of risk in any action; or safest states that are achievable within a budget/cost bound. Finally, we would like to define different cost functions and compare both theoretical- and experimentally the returned escape plans.

Acknowledgements

This work has been partially funded by the Spanish Government (Ministerio de Economía y Empresa) and FEDER, UE funds under projects TIN2017-88476-C2-2-R and RTC-2017-6753-4.

References

Aha, D. W. 2018. Goal reasoning: Foundations, emerging applications, and prospects. *AI Magazine* 39(2):3–24.

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2009. A heuristic search approach to planning with temporally extended preferences. *Artif. Intell.* 173(5-6):593–618.

Chakraborti, T.; Kulkarni, A.; Sreedharan, S.; Smith, D. E.; and Kambhampati, S. 2019. Explicability? Legibility? Predictability? Transparency? Privacy? Security? The emerging landscape of interpretable agent behavior. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 86–96.

Chen, J.; Sturtevant, N. R.; Doyle, W. J.; and Ruml, W. 2019. Revisiting suboptimal search. In *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*, 18–25.

Domshlak, C., and Mirkis, V. 2015. Deterministic over-

subscription planning as heuristic search: Abstractions and reformulations. *J. Artif. Intell. Res.* 52:97–169.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*.

Helmert, M. 2006. The fast downward planning system. *J. Artif. Intell. Res.* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res.* 14:253–302.

Jeantet, G., and Spanjaard, O. 2008. Rank-dependent probability weighting in sequential decision problems under uncertainty. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, 148–155.

Koenig, S., and Simmons, R. G. 1994. How to make reactive planners risk-sensitive. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, volume 293298.

Lu, Q.; George, B.; and Shekhar, S. 2005. Capacity constrained routing algorithms for evacuation planning: A summary of results. In *Advances in Spatial and Temporal Databases, 9th International Symposium, SSTD 2005, Angra dos Reis, Brazil, August 22-24, 2005, Proceedings*, 291–307.

Molineaux, M.; Klenk, M.; and Aha, D. W. 2010. Goal-driven autonomy in a navy strategy simulation. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*.

Morgenstern, O., and Von Neumann, J. 1953. *Theory of games and economic behavior*. Princeton university press.

Olsen, M. G.; Granmo, O.; and Radianti, J. 2015. Escape planning in realistic fire scenarios with ant colony optimisation. *Appl. Intell.* 42(1):24–35.

Perny, P.; Spanjaard, O.; and Storme, L. 2007. State space search for risk-averse agents. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2353–2358.

Pozanco, A.; E-Martín, Y.; Fernández, S.; and Borrajo, D. 2019. Finding centroids and minimum covering states in planning. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019.*, 348–352.

Quiggin, J. 2012. *Generalized expected utility theory: The rank-dependent model*. Springer Science & Business Media.

Ramírez, M., and Geffner, H. 2009. Plan recognition as planning. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, 1778–1783.

Smith, D. E. 2004. Choosing objectives in over-subscription planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, 393–401.

Sohrabi, S.; Riabov, A. V.; Katz, M.; and Udrea, O. 2018. An AI planning solution to scenario generation for enterprise risk management. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, 160–167.

Uriarte, A., and Ontañón, S. 2012. Kiting in RTS games using influence maps. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*.