Planning and Acting in Dynamic Environments

Lukáš Chrpa

Intelligent Acting

- Intelligent entities (agents) **reason about how to act** to achieve their goals
- **Reactive** acting
  - Rule based, Reinforcement Learning
  - Fast
  - Aims for short-term goals (rewards)
- **Deliberative** acting
  - Planning
  - Slow
  - Aims for longer-term goals

Automated Planning

- We have **Domain Definition languages** (e.g. PDDL)
- We have **Planning Engines** (e.g., FF, LAMA, LPG, FDSS, BFWS,...)
- So, we can generate **Plans** (quite easily)

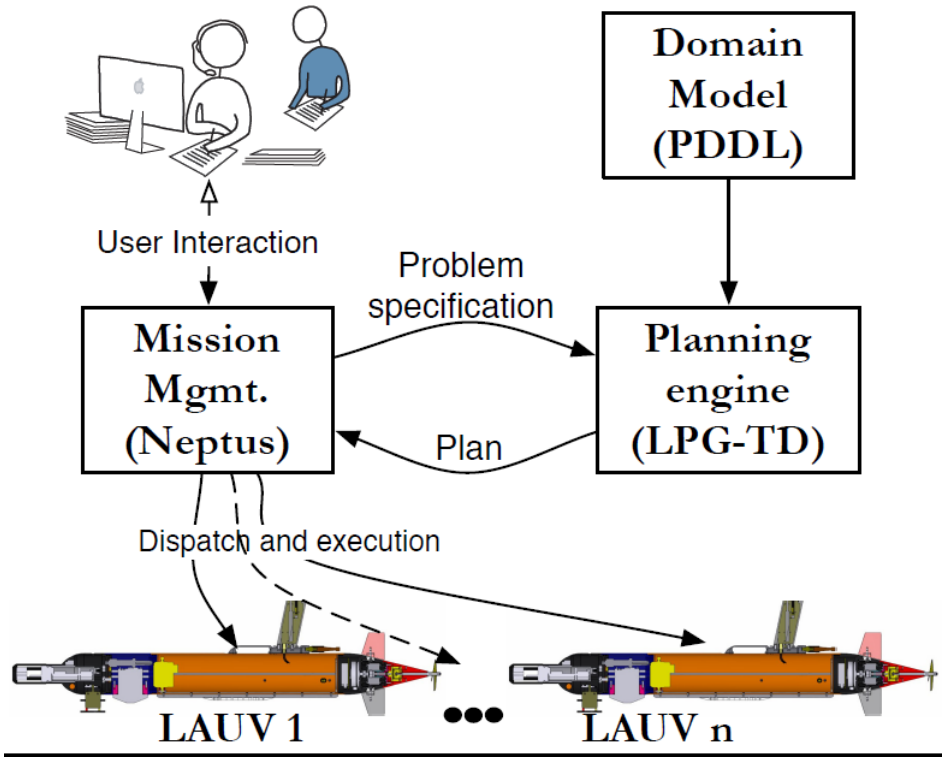
- But what about their **execution**

Task Planning for AUVs

- Necessity to control **multiple heterogeneous AUVs** for fulfilling user-defined tasks (e.g. sampling an object of interest)
- System has to be **flexible** (e.g. a user can add a new task) and **robust** (e.g. handling vehicles' failures)
  - Automatized response on task changes by user and/or exceptional circumstances during plan execution

“One shot” planning Modular Architecture [Chrpa et al., 2015]

- **User specifies tasks** in NEPTUS (the control system developed in LSTS, Univ. of Porto)
- NEPTUS **generates a planning problem** and sends it to the LPG-td planning engine
- LPG-td **returns a plan** to NEPTUS
- NEPTUS **distributes the plan** to each of the vehicles

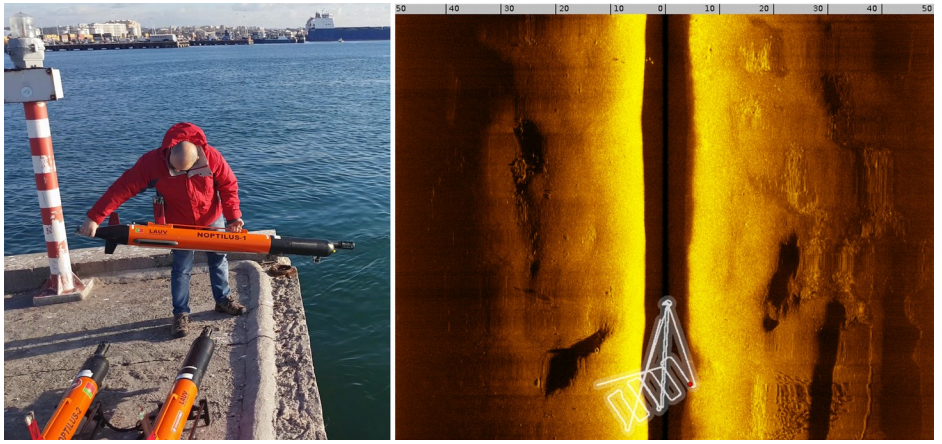


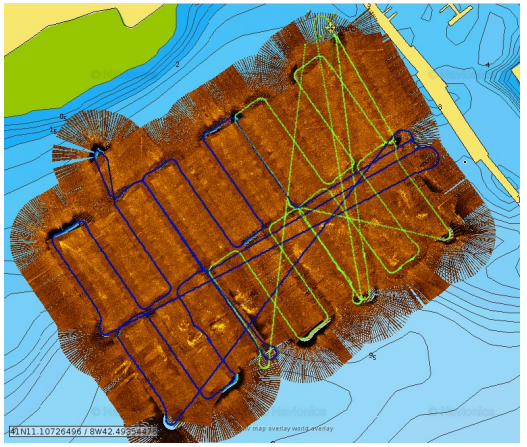
Domain Specification (sketch)

- The user specifies **tasks** by
  - **Locations/areas** of interest
  - Required **payloads** (e.g. camera, sidescan)
- The vehicle can perform the following **actions**
  - **Move** (moving between locations)
  - **Sample/Survey** (sampling the location/surveying the area of interest by a required payload)
  - **Communicate** (communicate task data with control center while being in its “depot”)

Experimental Settings

- Evaluated in Leixões Harbour, Porto
- Mine-hunting scenario was used
- 3 light AUVs, 2 carried sidescan, one carried camera
- In phase one, areas of interest were surveyed
- In phase two, contacts identified in phase one sampled to identify them as mines, or false positives





Planned vs. Execution time

- The plans were **executable**
- **High discrepancies**, especially for move and survey actions
- **Rough time predictions** that were done only on distance and type of vehicle

Vehicle	Action	Time Difference (s)
Noptilus-1	move	47.80 ± 49.11
	survey	23.15 ± 23.26
	sample	1.33 ± 0.58
	communicate	0.16 ± 0.17
Noptilus-2	move	39.57 ± 35.66
	survey	107.88 ± 141.10
	sample	N/A
	communicate	0.25 ± 0.07
Noptilus-3	move	59.90 ± 57.05
	survey	24.00 ± 0.00
	sample	9.57 ± 13.64
	communicate	0.11 ± 0.16

Additional Requirements [Chrpa et al., 2017]

1) Users can **add, remove or modify tasks** during the mission
   - Plans have to by (dynamically) amended

2) Vehicles might **fail to execute an action**
   - Tasks have to be (dynamically) reallocated to another AUV

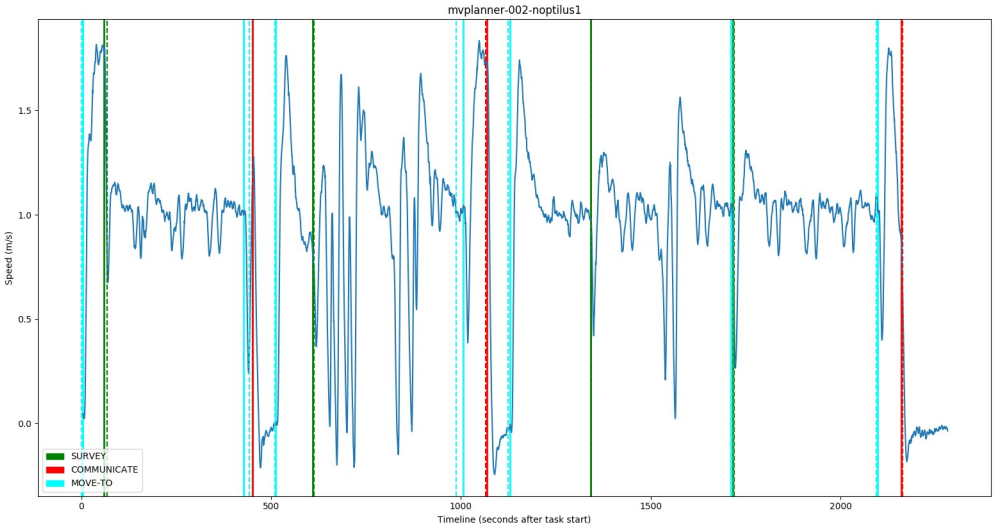
3) **Communication** with the control center is possible only **when a vehicle is in its "depot"**
   - The user defines a **maximum "away" time** for each vehicle (the vehicle has to return to its "depot" in that time)

Execution

- Preprocessing
  - Splitting large surveillance areas into smaller ones
- Planning
  - NEPTUS generates a problem specification in PDDL, runs LPG-td, then processes and distributes the plan among the vehicles
- Execution
  - Each vehicle is responsible for executing its actions
  - Move actions are translate into timed-waypoints for mitigating the differences between planned and actual times
  - When in depots vehicles communicate status of completed tasks (success/failure) – failed tasks are “re-inserted”
- Replanning
  - If a new planning request comes (e.g. a user added a new task), vehicles continue to execute their current plans until they come back to their depots, then they receive new plans

Results of the Field Experiment

- Plans were successfully executed
- During one of the executions one AUV (Noptilus 3) failed (depth sensor fault) – tasks were automatically re-inserted and allocated to a different AUV, which completed them



Most planned/actual differences are quite small (less than 3 seconds).

Around time 1000 a noticeable difference occurred (vehicle had to ascend during the survey). The delay was eliminated by accelerating during the following move action.

Executing Plans

- **In theory** (static environment)
  - Actions in a plan are always applicable (one by one)
  - After all actions are executed the goal is reached
- **In practice** (dynamic environment)
  - Actions might become **inapplicable** (at some point) because of **external factors**
  - **Goal might not be reached** even if all the actions were executed

Planning vs Execution (the AUV case)

- Issues we considered (to some extent)
  - User intervention (e.g. adding tasks)
  - Task failures
  - Vehicles delays
  - Lack of communication
- Issues we didn’t consider
  - Ships passing the area (or other non-deterministic events)
  - Currents, obstacles
  - …….

Non-deterministic events

- **Events** are encoded similarly to actions – they have **preconditions, add** and **delete effects**
- A non-deterministic event can occur if its precondition is met (but doesn't necessarily have to)
- We assume, for simplification, a "two-player" like scenario
  - The **controller** applies an **action** (including "noop")
  - The **environment** applies a set of independent **events** (including "noop")

Reasoning on “dangerous” states [Chrpa & Pilát & Gemrot, 2017;2021]

Handling “danger” locally

- Computing complete policies might not be feasible
- However, the **controller** should still **avoid dead-ends**
- The **controller** needs to know if it is in a **dangerous state**, i.e., a state “close” to a dead-end state, so it can avoid “falling” into it

Dark Dungeon domain: a sample scenario

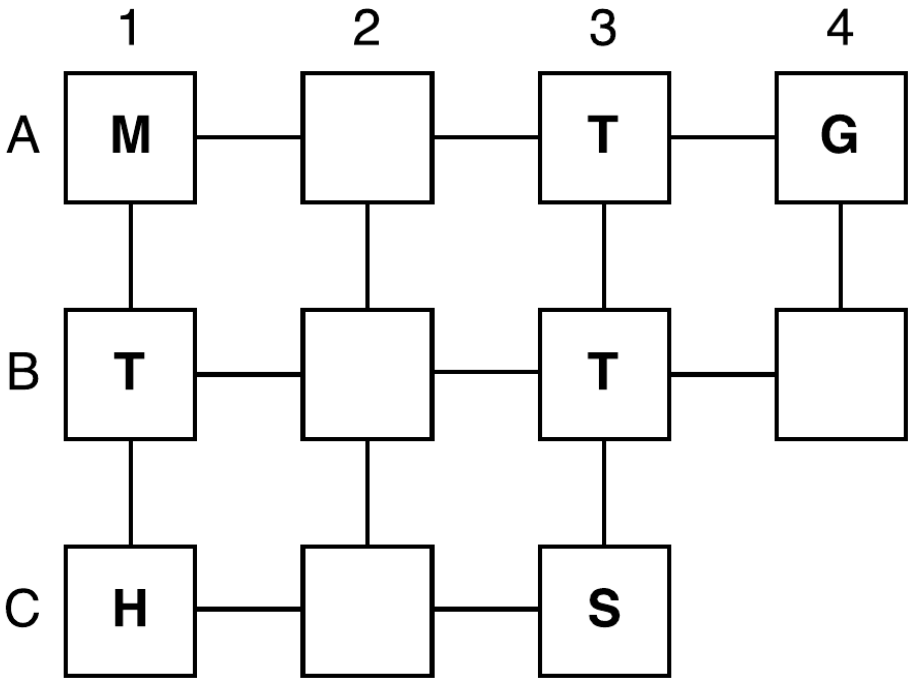
- The hero has to **navigate through the dungeon** full of traps and monsters
- The hero can **use the sword** (if s/he found it) to **eliminate monsters**
- The hero can **disarm traps** but must be **empty handed**
- **Monsters can move** (they cannot be in a room with a trap or another monster) and eventually **eliminate empty handed hero**

Dangerous States

- A state is
  - **0-dangerous** if it's a dead-end state
  - **n-dangerous** if events (without controller's actions) might transform it to a dead-end state in *n* steps
  - **Safe** (∞-dangerous) otherwise
- The **dang** function determines how dangerous the state can be (the worst case scenario) after executing a given sequence of actions

An example of dangerousness

- The initial state (I) is 4-dangerous
- dang(I, 〈right〉) = 2
- dang(I, 〈right,up〉) = 0
- dang(I, 〈right,right〉) = 2
- dang(I, 〈right,right,pickup〉) =∞



Meta-reasoning on Dangerous states

- **When in "dangerous" state** (the value of *dang* less than a given threshold) **the controller**:
  - **Reactively escapes the danger**, i.e, executes actions maximizing the value of *dang*
  - **Plans towards a safe state**
  - **Plans towards eliminating the source of the danger**
- **After escaping the danger** (the value of *dang* is above the threshold), **the controller plans towards the goal**

Considered Agents (baseline)

- **R1 –** behaves reactively according to given rules
- **R2 –** behaves reactively according to given rules but can plan its path (by A*)
- **N1 –** re-plans whenever an event changes the state of the environment
- **N2 –** re-plans when the current action is inapplicable

Considered Agents (clever)

- **C1** – if the value of the *dang* function is small (2 or less), then it reactively escapes to a “safer” state (3-dangerous or better)
- **C2 -** if the value of the *dang* function is small (2 or less), then it plans to eliminate the source of danger
- **C3** – check when the plan gets disrupted and if at that point the value of the *dang* function is small (2 or less), then it plans to eliminate the source of danger

Results

agent	wins	loses	TO	PC	steps	WS	WPC
N1	0.679	0.320	0.000	26.955	43.762	44.975	24.017
N1C	0.808	0.192	0.001	23.632	40.871	43.768	23.415
N2	0.560	0.440	0.000	1.130	30.337	39.648	1.045
N2C	0.753	0.247	0.000	2.257	36.412	42.219	1.281
R1	0.505	0.495	0.000	0.000	90.842	123.557	0.000
R2	0.788	0.212	0.000	0.000	41.073	47.227	0.000
C1	0.479	0.419	0.102	92.650	199.302	80.915	22.441
C1C	0.513	0.394	0.093	85.404	184.327	73.934	20.010
C2	0.823	0.176	0.001	10.525	49.341	54.528	8.963
C2C	0.841	0.158	0.000	8.879	46.033	50.578	7.151
C3	0.803	0.196	0.001	9.567	47.567	50.636	6.586
C3C	0.840	0.160	0.000	8.050	45.573	48.580	5.311

Agents' wins, losses, and time-outs (TO); number of planner calls (PC), steps winning steps (WS) and wining planning time (WPC). "C" suffix considers penalizing "unsafe" actions.

Results cont.

	S	D0.1	D0.2	D0.5	K0.1	K0.2	K0.5
N1	0.999	0.816	0.747	0.568	0.716	0.554	0.355
N1C	0.999	0.921	0.869	0.746	0.862	0.726	0.531
N2	1.000	0.644	0.577	0.440	0.603	0.416	0.236
N2C	0.999	0.847	0.765	0.673	0.816	0.684	0.486
R1	1.000	0.664	0.540	0.371	0.459	0.319	0.183
R2	1.000	0.926	0.864	0.741	0.842	0.693	0.453
C1	0.331	0.631	0.640	0.633	0.418	0.379	0.321
C1C	0.389	0.641	0.653	0.651	0.450	0.424	0.381
C2	0.996	0.932	0.881	0.793	0.869	0.750	0.539
C2C	0.999	0.940	0.894	0.804	0.879	0.778	0.596
C3	0.996	0.934	0.889	0.785	0.848	0.704	0.466
C3C	0.999	0.955	0.913	0.816	0.876	0.763	0.556

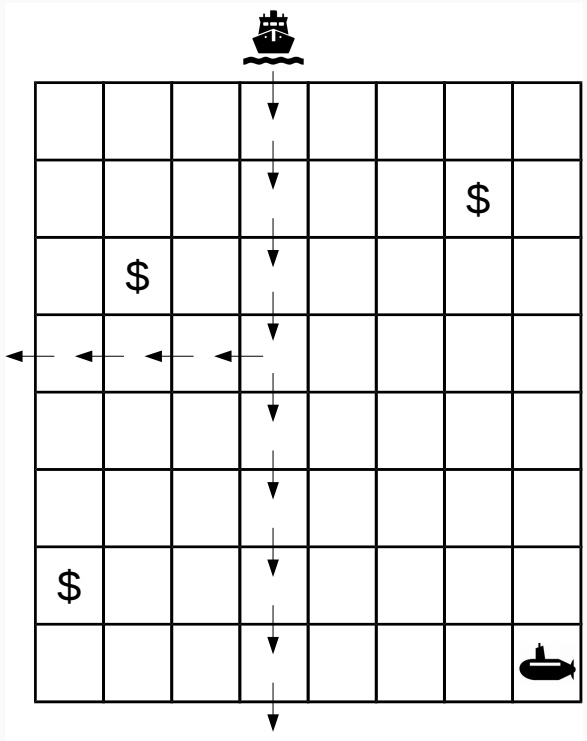
The success rate of the different types of agents in dungeons with different monsters (Static, Dynamic, Killer) and their movement probabilities

“C” suffix considers penalizing “unsafe” actions.

Reasoning on “safe” states
[Chrpa & Pilát & Gemrot, 2020]

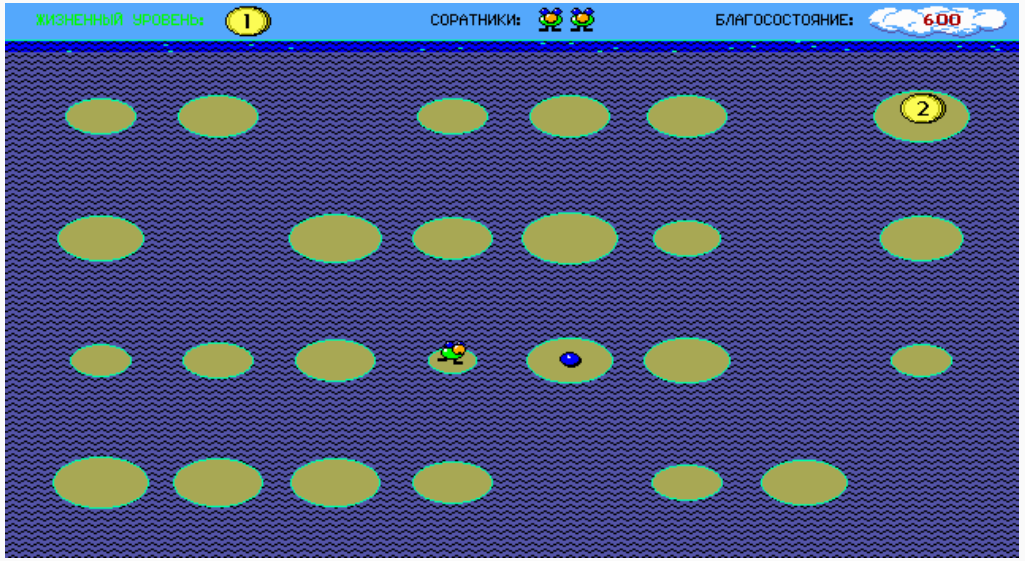
Case Studies: AUV Domain

- An AUV moves and collects resources in a grid-like environment
- Ships can move in certain grid cells
- Ships are not controlled by the agent
- If a ship runs over the AUV, the AUV is destroyed



Case Studies: Perestroika Domain

- An agent moves and collects coins in a grid-like environment
- There are solid and shrinking platforms (big, medium, small)
- Shrinking platforms can shrink until they disappear; they can reappear as big ones
- If a shrinking platform disappears with the agent on it, the agent dies



High-level Idea of Safe State Reasoning

- A **safe state** is a state in which no sequence of events lead to dead-end
- A **robust plan** is a plan that can always be applied and goal reached despite event occurrence
- A **reference plan** is the initially generated plan
- The idea is that planning and acting concerns of generation and execution of robust plans between safe states
- However, safe states should be "reasonably close" to each other, so the reference plan has to reflect this

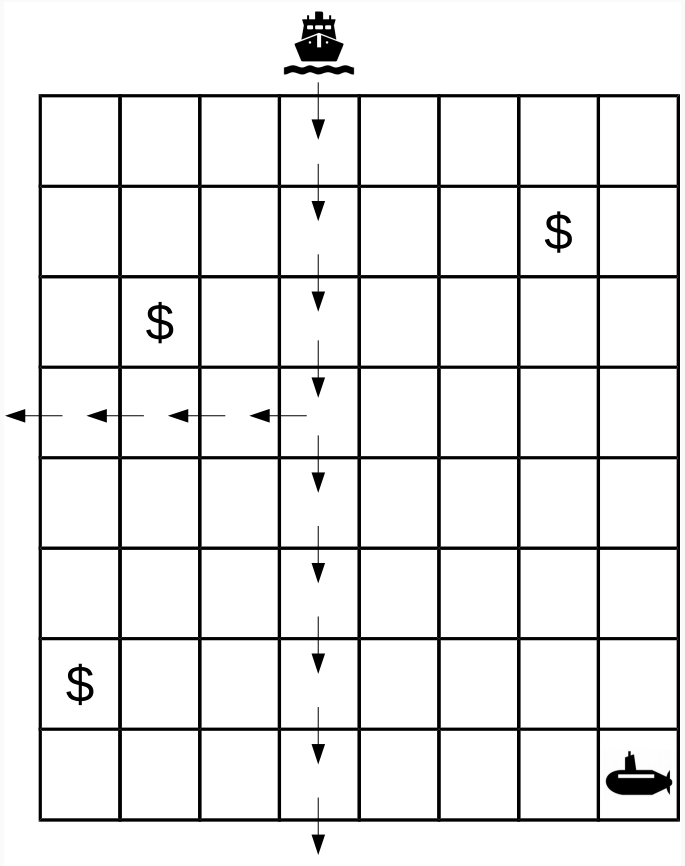
Robust Plans

- Robust plans are generated such that event applicability is **optimistically** assumed while action applicability is **pessimistically** assumed
- $E^0 = p^0_+ = p^0_- = \emptyset$ (no event can occur before agent's first action)
- For each $1 \leq i \leq n$ it is the case that:
  - $E^i = \{e \mid e \in E, pre(e) \subseteq ((s_{i-1} \cup p^{i-1}_+) \setminus del(a_i)) \cup add(a_i)\}$,
  - $p^i_+ = (p^{i-1}_+ \setminus del(a_i) \cup \bigcup_{e \in E^i} add(e))$,
  - $p^i_- = (p^{i-1}_- \setminus add(a_i) \cup \bigcup_{e \in E^i} del(e))$.
  - $pre(a_i) \cap p^{i-1}_- = \emptyset$
- $G \subseteq s_n \setminus p^n_-$

Safe State Reasoning in Planning and Acting

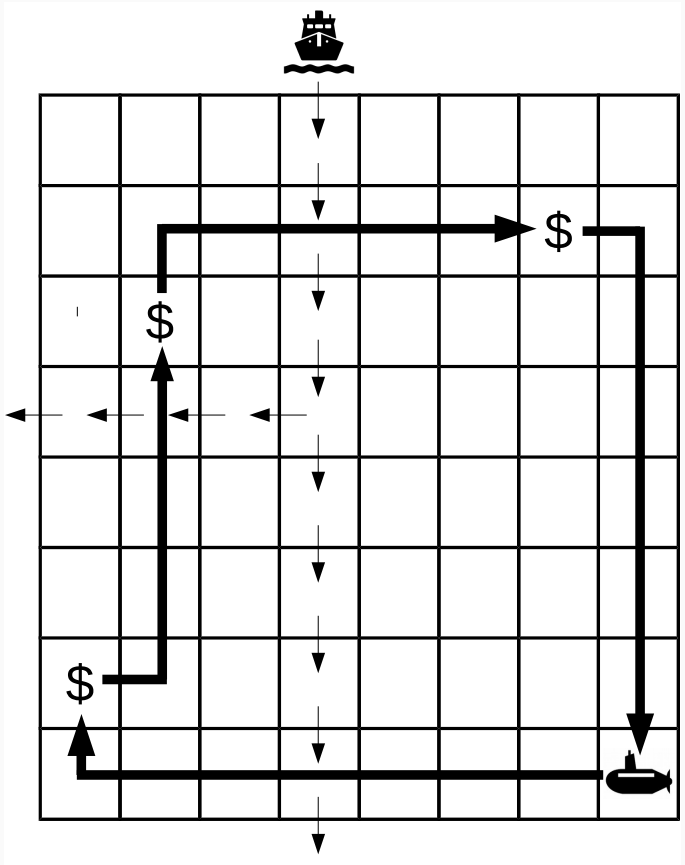
- Try to generate a robust plan (if successful, just execute it)
- Try to generate a reference plan with increasing unsafeness limit (if it fails, stop)
- Iterate until the goal is reached
  - Identify k **safely applicable** actions from the (rest of) reference plan (i.e., a robust plan to the furthest safe state)
  - If $k > 0$, execute k actions for the reference plan and continue
  - If $k = 0$, try to generate a robust plan to the next safe state (if it exists, execute it, otherwise do noop)

Example



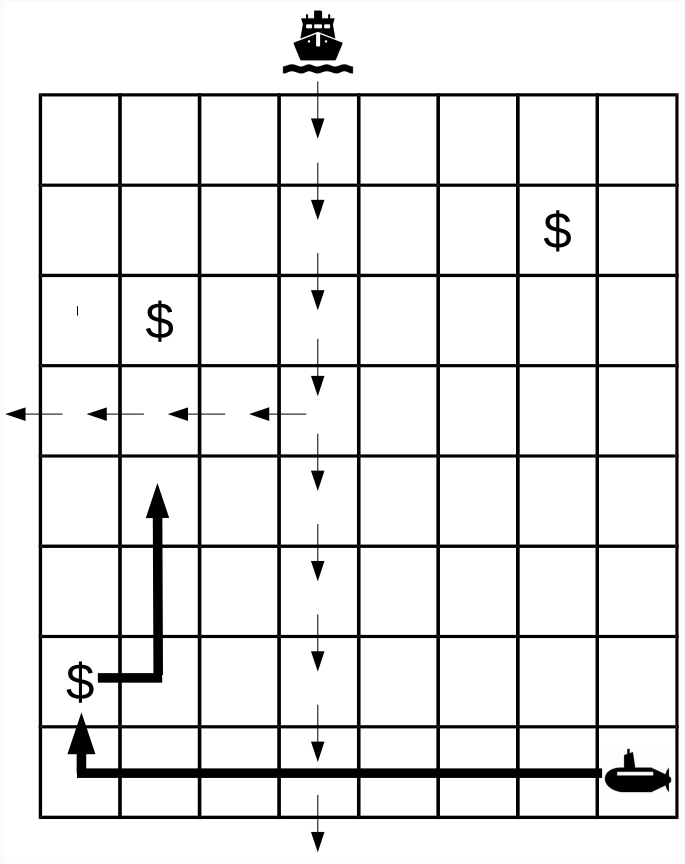
The AUV has to collect all resources and return to the location of origin.

Example



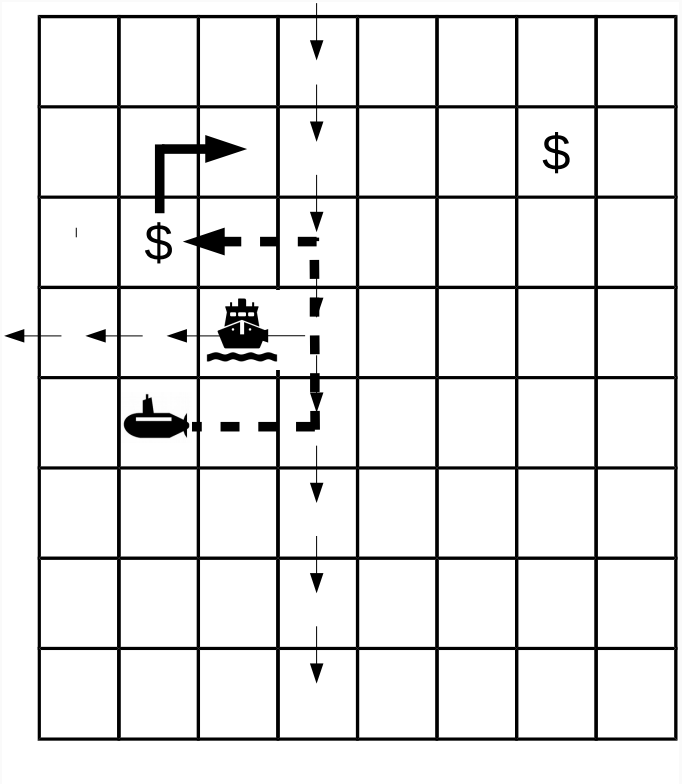
A reference plan (with the unsafeness limit of 1)

Example



A safely applicable sequence of actions (maximum length)

Example



A robust plan around the ship

Experiments - results

Algorithm	PER-APP			PER-EVENT			DANG			LIMIT		
Problem	PT	ACT	SR	PT	ACT	SR	PT	ACT	SR	PT	ACT	SR
AUV-1	226	37.32	76	3307	36.66	88	1364	41.81	100	410	38.60	100
AUV-2	418	39.16	49	3940	35.91	57	1582	45.79	98	678	32.89	100
AUV-3	352	37.64	50	5078	39.68	47	1620	45.13	80	2198	40.44	100
AUV-4	664	75.12	98	32518	109.56	93	1539	75.99	100	1603	75.39	100
AUV-5	943	75.61	85	39642	101.57	67	2946	78.70	98	2416	77.32	100
AUV-6	848	75.02	83	43604	98.87	61	2681	78.10	97	6982	73.79	100
Per-1	544	21.58	24	2714	24.13	15	6497	60.84	100	600	28.78	100
Per-2	425	22.89	18	2661	23.00	10	7694	68.73	90	699	31.20	100
Per-3	488	26.57	14	3143	27.23	13	7322	72.69	95	577	33.88	100
Per-4	889	40.00	1	N/A	N/A	0	15883	118.54	100	2139	56.49	100
Per-5	1327	42.67	3	6327	46.00	1	21479	153.07	83	2198	57.99	100
Per-6	764	22.21	19	3009	14.63	8	14170	111.78	95	633	15.84	100

PT denotes the average time spent by planning (milliseconds), ACT denotes the average number of actions needed to solve the problem, and SR denotes the number of successful runs.

Reasoning on “cyclic phenomena”

[Chrpa & Pilát & Med, 2021]

Our Terminology

- A **cyclic phenomenon** occurs repeatedly in the environment and is dangerous to the agent if it interferes with the phenomenon
- A **safe state** is a state in which no sequence of events lead to dead-end
- A **dead-end event** is an event that might lead to dead-end
- A **robust plan** is a plan that can always be applied and goal reached despite event occurrence
- An **unsafe bridge** is a sequence of actions "crossing" possibly unsafe states
- A **reference plan** is the initially generated plan
- An **eventually applicable plan** is a plan in which all unsafe bridges become robust plans (after a finite number of "noop" actions)

Cyclic Phenomena

- An event e is S-**reversible** if and only if for each $s \in S$, where e is applicable, there is a sequence of events transforming the environment back to s
- We heuristically determine event S-reversibility by exploring an *event only Domain Transition Graph*
- Reversible Events represent **cyclic phenomena**
- Irreversible Events represent potential **dead-end events**

Proposition
Let S^s be a set of states reachable from s by applying only events. If for each $s' \in S^s$ and for each event e applicable in s' it is the case that e is $\{s'\}$-reversible, then (i) $S^{s'} = S^s$ and (ii) if also s is not a dead-end state, then s is a safe state.

Generating Eventually Applicable Reference Plans

- Initially, the initial state is verified for the condition (i) from the previous Proposition
- If none irreversible event is "enabled" we assume we are in a safe state
- We might apply reversible events to modify the environment to a "desirable" state
- Unsafe bridges are eventually applicable in the Theorem below holds

Theorem (Theorem Sketch)

Let s be a safe state and $\pi = \langle a_1, \ldots, a_n \rangle$ be an unsafe bridge. If the following conditions

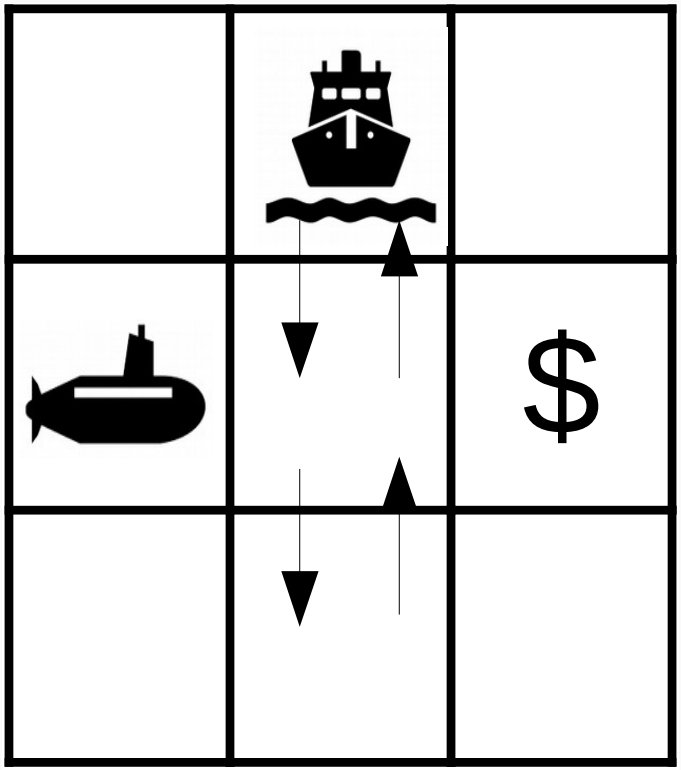
(1) *for each $a_i \in \pi$: minimum distance to an event being a clobberer for $a_i \geq i - 1$*

(2) *for each $a_j \in \pi$: minimum distance to a dead-end (irreversible) event a_j is a clobberer for $\geq j$*

hold, then π is a robust plan.

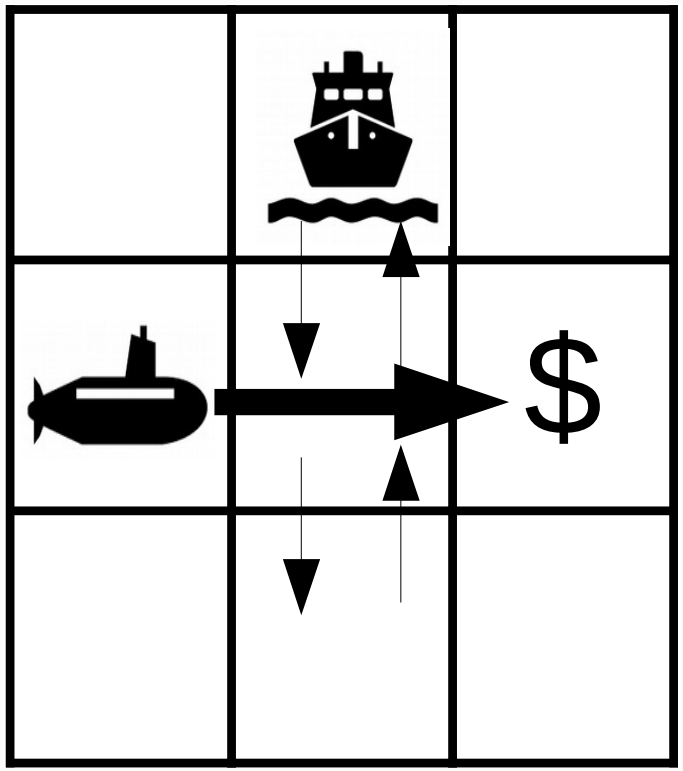
Require: A planning task $\mathcal{P} = (V, A, E, I, G)$, E^i,E^r
Ensure: Eventually Applicable Robust Plan π
  $s \leftarrow I$, $\pi \leftarrow \langle\rangle$, $en = \emptyset$
  while $s \not\models G$ **do**
    if $en = \emptyset$ **then**
      $un \leftarrow 0$
      non-deterministically select $a \in A \cup E^r$ s.t. $s \models pre(a)$
    else
      $un{+}{+}$
      non-deterministically select $a \in A$ s.t. $s \models pre(a)$, $c^1(a, s) \geq (un - 1)$ and $c^2(a, s) \geq un$
    end if
    if no a was selected **then return** fail
    $s = \gamma(s, a)$
    if $a \in A$ **then**
      $en \leftarrow en \cup \{e_i \mid a$ is an achiever for $e_i\}$
      $en \leftarrow en \setminus \{e_i \mid a$ is a clobberer for $e_i\}$
      $\pi \leftarrow \pi.a$
    end if
  end while
  return π

Example



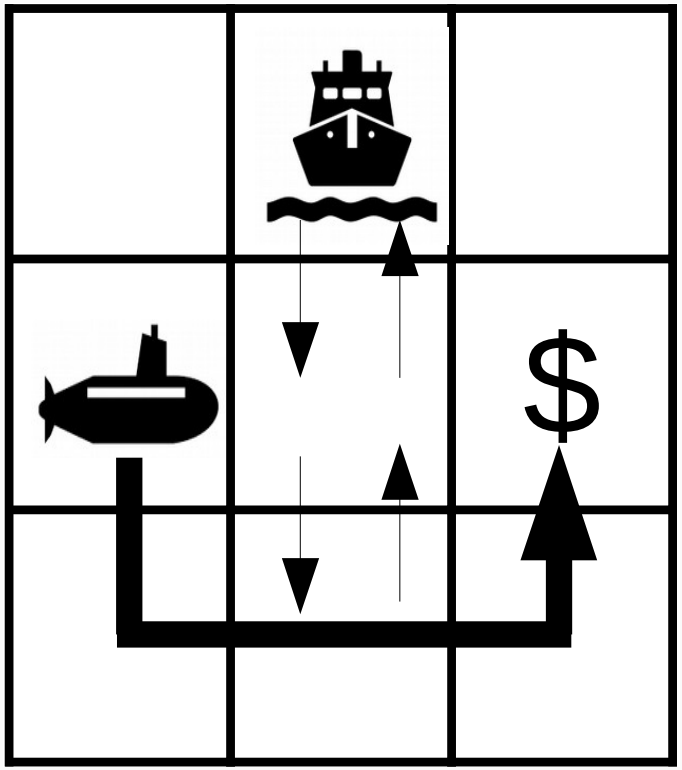
The AUV has to collect the resource on the right.

Example



The reference plan is not eventually applicable (the ship either block or poses a direct threat to the AUV)

Example



The reference plan is eventually applicable (the unsafe bridge from the bottom left to the bottom right cell becomes a robust plan if the ship is in the top row)

Experiments - results

	AUV										
	Structure			**APP**			LIMIT			FOND	
#	N	#S	#R	Pt	RPL	St	Pt	Pl	St	Pt	St
1	3	1W	1	130	9	17	FAIL - Act			18711	26
2	3	1C	1	129	7	11	400	7	11	13827	31
3	5	3C	3	200	29	136	2210	21	144	FAIL - Plan	
4	10	5C	5	1953	41	61	FAIL - Plan			FAIL - Plan	
5	15	7C	7	59073	93	117	FAIL - Plan			FAIL - Plan	
6	5	2W	2	152	16	29	545	16	30	FAIL - Plan	
7	10	6W	5	909	87	154	229401	41	116	FAIL - Plan	
8	15	12W	7	1231	85	114	FAIL - Act			FAIL - Plan	
	Perestroika										
1	5	16E	5	181	17	29	747	17	25	FAIL - Plan	
2	5	16E	8	778	32	52	FAIL - Act			FAIL - Plan	
3	9	56E	14	7179	66	118	FAIL - Act			FAIL - Plan	
4	9	56E	24	7709	114	298	FAIL - Act			FAIL - Plan	
5	5	10R	9	161	33	39	FAIL - Act			FAIL - Plan	
6	5	19R	5	1000	17	24	FAIL - Plan			FAIL - Plan	
7	9	40R	22	3257	74	102	FAIL - Act			FAIL - Plan	
8	9	37R	43	7603	167	223	FAIL - Act			FAIL - Plan	

Structure of the problem consists of N – size of the square grid, $\#R$ – number of resources, and $\#S$ – number of ships/shrinking platforms. Pt – runtime in ms for generating Reference Plan (including preprocessing), RPL – reference plan length, and St – average number of execution steps.