Sprawozdanie z eksperymentów STRIPS

Bartosz Zieliński, Sergey Zeliuk
Projekt 2

31 marca 2025

1 Wprowadzenie

Niniejszy raport przedstawia eksperyment z planowaniem w przód (ang. *forward planning*) w dziedzinie inspirowanej problemem *Air Cargo*, wykorzystującym uproszczoną reprezentację typu STRIPS. Kod generuje wiele instancji problemu z różną liczbą lotnisk, samolotów i ładunków, a następnie rozwiązuje je zarówno bez heurystyki, jak i z prostą heurystyką (polegającą na przycinaniu pewnych niepotrzebnych akcji).

1.1 Cel

Celem było sprawdzenie, jak sprawuje się proste przeszukiwanie wszerz (BFS) w przestrzeni stanów przy różnych wielkościach problemu, a także porównanie czasu i długości znalezionych planów w dwóch wariantach:

- bez heurystyki,
- z heurystyką

2 Opis kluczowych fragmentów kodu

W naszym eksperymencie bazowaliśmy na uproszczonej implementacji akcji w stylu STRIPS. Poniżej przedstawiono najważniejsze fragmenty kodu w języku Python:

Listing 1: Definicja akcji STRIPS i dziedziny problemu.

```
class Strips(object):
    def __init__(self, name, preconds, effects, cost=1):
        self.name = name
        self.preconds = preconds
        self.effects = effects
        self.cost = cost

    def __repr__(self):
        return self.name

class STRIPS_domain(object):
    def __init__(self, feature_domain_dict, actions):
        self.feature_domain_dict = feature_domain_dict
        self.actions = actions
```

Pierwsza klasa `Strips` opisuje pojedynczą akcję (nazwę, prewarunki i efekty). Druga klasa, `STRIPS_domain`, przechowuje informacje o dziedzinie planowania: wszystkich dopuszczalnych cechach stanu i zbiorze akcji.

Listing 2: Funkcja tworząca akcje *LOAD*, *UNLOAD* i *FLY* w dziedzinie Air Cargo.

```
def create_air_cargo_domain(cargos, planes, airports):
    feature_domain_dict = {}
    for c in cargos:
        feature_domain_dict[f"At_{c}"] = set(airports) | set(planes)
    ...
    actions = set()

    # LOAD
    for c in cargos:
```

```
        for p in planes:
            for a in airports:
                name = f"LOAD_{c}_onto_{p}_at_{a}"
                preconds = {...}
                effects = {...}
                actions.add(Strips(name, preconds, effects))

    # UNLOAD
    ...

    # FLY
    ...
    return STRIPS_domain(feature_domain_dict, actions)
```

Funkcja `create_air_cargo_domain` tworzy wszystkie akcje *LOAD*, *UNLOAD* oraz *FLY* we wszystkich możliwych kombinacjach (ładunek–samolot–lotnisko).

Listing 3: Fragment BFS.

```
CURRENT_GOAL = {}

def do_heuristics(state, action):
    name_parts = action.name.split("_")
    action_type = name_parts[0]
    ...

def plan_with_timeout(problem, use_heuristics, max_time=300):
    global CURRENT_GOAL
    CURRENT_GOAL = problem.goal
    ...
    while frontier:
        if time.time() - start_time > max_time:
            return None
        state, path = frontier.popleft()
        if goal_satisfied(state, problem.goal):
            return path
        for action in problem.prob_domain.actions:
            new_state = apply_action(state, action, use_heuristics)
            if new_state is not None:
                frontier.append((new_state, path + [action]))
    return None
```

Widzimy tu prosty mechanizm BFS posługujący się kolejką (`frontier`). Wariant z heurystyką odrzuca zbędne akcje, np. *LOAD* ładunku, który jest już w docelowym miejscu. W razie przekroczenia limitu czasu zwracana jest wartość `None` (*timeout*).

3 Wyniki eksperymentów

Poniższa tabela zawiera skrócone wyniki wygenerowane przez kod i zapisane w pliku `air_cargo_results.csv`. Dla każdego problemu pokazano m.in. długość planu, liczbę akcji w domenie, a także czas znalezienia planu *bez* oraz *z* heurystyką:

problem_name	plan_length	domain_actions	time_no_h	time_h
Normal Problem #1	7	36	0.00000000	0.01332521
Normal Problem #2	6	20	0.00261188	0.00000000
Normal Problem #3	9	28	0.00175166	0.00241542
Subgoals Problem #1	7	36	0.00259566	0.00268722
Subgoals Problem #2	9	36	0.00351882	0.00258493
Subgoals Problem #3	8	36	0.00193691	0.00273800
Bigger Problem #7	20	204	Timeout/No plan	237.48214030
Bigger Problem #8	20	160	110.81333447	85.28159785
Bigger Problem #9	24	136	117.81604290	78.79018116

Tabela 1: Wybrane pola z `air_cargo_results.csv`.

Dla przykładu, w Problemie #1 plan bez heurystyki znaleziono natychmiast (czasy okrągłe do 0.00000000s), a z heurystyką plan był identyczny, choć czas nieznacznie wzrósł przez dodatkowe sprawdzanie reguł. W większych problemach heurystyka w niektórych przypadkach skraca czas (Problemy #8, #9), a w Problemie #7 w ogóle umożliwia znalezienie planu.

4 Wnioski

Na podstawie przeprowadzonych eksperymentów można stwierdzić, że:

- Proste przeszukiwanie wszerz szybko znajduje plany w mniejszych problemach,
- Zastosowana heurystyka (przycinanie pewnych akcji) najczęściej przynosi korzyść w większych instancjach, czasem jest nieznacznie wolniejsza w mniejszych,
- Przekroczenie limitu czasu (np. w Problemie #7 bez heurystyki) pokazuje, że podejście BFS w pewnych sytuacjach jest zbyt kosztowne obliczeniowo bez dodatkowych ograniczeń.