Generalized Planning as Heuristic Search

Javier Segovia-Aguas [1] Sergio Jiménez [2] Anders Jonsson [1]

[1]Universitat Pompeu Fabra [2]Universitat Politècnica de València

Motivation and Contributions

Generalized Planning (GP) is the problem of computing **algorithm-like solutions** that solve a set of planning problems:

1. How to compute solutions "efficiently"? **Heuristic search**
2. How to deal with the *grounded state-space* search? Proposing a **tractable solution space** leveraging a *Random-Access Machine* computational model and *Intel X86* FLAGS registers
3. How the heuristics should be? Evaluation and heuristic functions must be **grounding-free** (do not require to ground states and actions in advance)
4. What about the searching algorithm? Given an enumerated solution space, the model and the evaluation/heuristic functions, a **best-first search** algorithm works

Classical Planning

A **classical planning problem** is a tuple $\langle X, A, I, G\rangle$ where:

- X: set of state variables
- A: set of *lifted* actions
- $I \in S$: initial state (*total variable assignment*)
- G: is a goal condition (*partial variable assignment*) which induces $S_G = \{s | s \models G, s \in S\}$

Planning Programs

Definition 1. A **planning program** is a sequence of n instructions $\Pi = \langle w_0, \ldots, w_{n-1}\rangle$, where each instruction $w_i \in \Pi$ is associated with a *program line* $0 \le i < n$ and is either:
- A *planning action* $w_i \in A$.
- A *goto instruction* $w_i = \texttt{go}(i', !y)$, where i' is a program line $0 \le i' < i$ or $i+1 < i' < n$, and y is a proposition.
- A *termination instruction* $w_i = \texttt{end}$. The last instruction is always $w_{n-1} = \texttt{end}$.

Given a program state (s, i) composed of a planning state $s \in S$ and program counter $i \in [0, n)$, the **execution model** of a programmed instruction w_i is defined as:

- If $w_i \in A$, the new program state is $(s', i+1)$, where $s' = s \oplus w_i$.
- If $w_i = \texttt{go}(i', !y)$, the new program state is $(s, i+1)$ if y holds in s, and (s, i') otherwise.
- If $w_i = \texttt{end}$, program execution terminates.

The Space of Planning Programs

The space of possible planning programs is compactly represented with three bit-vectors:

1. The *action vector* of length $(n-1) \times |A|$, indicating whether action $a \in A$ appears on line $0 \le i < n-1$.
2. The *transition vector* of length $(n-1) \times (n-2)$, indicating whether $\texttt{go}(i', *)$ appears on line $0 \le i < n-1$.
3. The *proposition vector* of length $(n-1) \times \sum_{x \in X} |D_x|$, indicating if $\texttt{go}(*, !\langle x = v\rangle)$ appears on line $0 \le i < n-1$.

A planning program is then encoded as the concatenation of these three bit-vectors. The length of the resulting vector is:

$$(n-1)\left(|A| + (n-2) + \sum_{x \in X} |D_x|\right) \qquad (1)$$

Classical Planning with a RAM

Given a classical planning instance $P = \langle X, A, I, G\rangle$ an extended instance with a RAM of $|Z| + 2$ registers is $P_Z = \langle X'_Z, A'_Z, I'_Z, G\rangle$ where:

- The new set of **state variables** X'_Z comprises:
  - the original set of state variables X,
  - *Boolean* state variables *zero* and *carry* flags $Y = \{y_z, y_c\}$,
  - the *pointers* Z which are state variables of domain $[0, |X|)$.
- The new set of **actions** A'_Z includes:
  - *planning actions* A' from abstracting each $a \in A$ where each $par(a) \subseteq X$ is replaced by pointers in Z,
  - *RAM actions* $\{\texttt{inc}(z_1), \texttt{dec}(z_1), \texttt{cmp}(z_1, z_2), \texttt{cmp}(*z_1, *z_2), \texttt{set}(z_1, z_2) \mid z_1, z_2 \in Z\}$
- The **initial state** I'_Z is I and all pointers set to 0 and flags to *False*.

Thus, $|A'_Z|$ is independent of X and their domain:

$$|A'_Z| = 2|Z|^2 + |A'|. \qquad (2)$$

1. Generalized Planning with a RAM

Definition 2. The **feature language** $\mathcal{L} = \{\neg y_z \wedge \neg y_c, y_z \wedge \neg y_c, \neg y_z \wedge y_c, y_z \wedge y_c\}$ of the four possible joint values for the pair of Boolean variables $Y = \{y_z, y_c\}$.

Definition 3. A **GP problem with shared features** is a finite and non-empty set of T classical planning instances $\mathcal{P} = \{P_1, \ldots, P_T\}$, where instances share the same set of actions A'_Z, but may differ in the state variables, initial state, and goals. Formally, $P_1 = \langle X'_{1_Z}, A'_Z, I'_{1_Z}, G_1\rangle, \ldots, P_T = \langle X'_{T_Z}, A'_Z, I'_{T_Z}, G_T\rangle$.

- Feature language $\mathcal{L}$ defines a tractable solution space for GP, where the *proposition vector* requires $(n-1) \times 4$ bits. Eq. 1 simplifies to:

$$(n-1)\left(|A'_Z| + (n-2) + 4\right). \qquad (3)$$

- Now the solution space for GP is independent of the number and domain size (e.g integers) of the planning state variables.

Example of Generalized Plans with a RAM

```
0. set(j,tail)
1. swap(*i,*j)
2. dec(j)
3. inc(i)
4. cmp(j,i)
5. goto(1,¬(¬y_z ∧ ¬y_c))
6. end
```

```
0. set(min,i)
1. cmp(*j,*min)
2. goto(5,¬(¬y_z ∧ ¬y_c))
3. set(min, j)
4. swap(*i,*min)
5. inc(j)
6. cmp(length,j)
7. goto(1,¬(y_z ∧ ¬y_c))
8. inc(i)
9. set(j,i)
10. cmp(length,i)
11. goto(0,¬(y_z ∧ ¬y_c))
12. end
```

Figure 1. *Generalized plans*: (left) for reversing a list; (right) for sorting a list with the *selection-sort* algorithm.

2. Evaluation and Heuristic Functions

- *The program structure.* Evaluation functions computed in linear time, traversing the bit-vector representation of Π:
  - $f_1(\Pi)$, the number of *goto* instructions in Π
  - $f_2(\Pi)$, the number of *undefined* program lines in Π
  - $f_3(\Pi)$, the number of repeated actions in Π
- *The empirical performance of the program.* Performance of Π on a GP problem $\mathcal{P} = \{P_t | t \in [1, T]\}$:
  - $h_4(\Pi, \mathcal{P}) = n - PC^{MAX}$, where PC^{MAX} is the maximum undefined program line reached after executing Π over all instances in $\mathcal{P}$
  - $h_5(\Pi, \mathcal{P}) = \sum_{P_t \in \mathcal{P}} \sum_{x \in X_t} (v_x - G_t(x))^2$
  - $f_6(\Pi, \mathcal{P}) = \sum_{P_t \in \mathcal{P}} |exec(\Pi, P_t)|$, is the cost of a GP solution

3. Best First Search for Generalized Planning

The third contributions is the **Best-First Generalized Planning** (BFGP) algorithm, which is a best-first search that uses one or more of the previous evaluation and heuristic functions.

In the first two experiments, we analyze the performance of BFGP with each single function, and select a good combination of two functions (structure and performance-based, one of each). The best results are obtained with BFGP(h_5, f_1) which are compared with a compilation-based approach to PDDL of Planning Programs (PP):

Domain	PP in sec.	BFGP(h_5, f_1) in sec.
Triangular Sum	0.85	**0.1**
Corridor	-	**4.5**
Reverse	87.86	**1.4**
Select	204.20	**80**
Find	274.86	**162**
Fibonacci	3,570	**22**
Gripper	**1**	6.9
Sorting	-	**713**

Table 1. Computing CPU-time (secs) for solving domains in the GP compilation approach (PP) and $BFGP(h_5, f_1)$.

We refer to the paper for further details on the experiments.

Conclusions and Future Work

- We propose a **tractable solution space** and a first **native heuristic** approach for GP.
- **Heuristic search** and **classical planning** may improve the base performance of our approach.
  - **Open list** may be effectively handled with several smaller lists.
  - **Detecting symmetries** in the programs, e.g. any program that can be built with transpositions of the causally-independent instructions.
  - Multi-thread computing to **parallelize the search** in solution spaces, e.g. SATPLAN.
- BFGP starts from an empty program, but nothing prevent us to start from a **partial specification**.
- The goal-driven heuristic $h_5(\Pi, \mathcal{P})$ builds on top of the *Euclidean distance*; better estimates may be obtained by building on top of **better informed planning heuristics**.