A Survey on Hierarchical Planning –
One Abstract Idea, Many Concrete Realizations

Pascal Bercher\textsuperscript{1} and Ron Alford\textsuperscript{2} and Daniel Höller\textsuperscript{1}

\textsuperscript{1}Institute of Artificial Intelligence, Ulm University, Germany
\textsuperscript{2}MITRE, McLean, Virginia, USA

August 14, 2019
A classical problem description consists of:

- Given an initial state $s_i$,
- a portfolio of the system’s available actions, and
- a goal description $g$. 
A classical problem description consists of:

- Given an initial state $s_I$,
- a portfolio of the system’s available actions, and
- a goal description $g$.

What are we looking for?
Non-hierarchical Planning in a Nutshell

A classical problem description consists of:

- Given an initial state \( s_I \),
- a portfolio of the system’s available actions, and
- a goal description \( g \).

What are we looking for?

→ A plan that transforms \( s_I \) into \( g \).

\[
\begin{align*}
\text{description of the initial world situation} & \quad \text{description of desired world properties} \\
\text{plan} & \quad \text{intermediate states}
\end{align*}
\]
Planning Applications: Intelligent Factories

The Scanalyzer Domain
– automatic greenhouse logistic management

Pictures with kind permission from LemnaTec GmbH
Planning Applications: Intelligent Assistants

Sink devices:
- Television (requires video)
- Amplifier (requires audio)

Source devices:
- Blu-ray player
- Satellite receiver
  (both produce audio & video)
Planning problems are usually defined in terms of a description language based on a first-order predicate logic.

- States are sets of (ground) propositions, e.g.,

\[ s = \{ \text{HasPort}(\text{AMPLIFIER}, \text{HDMI}), \]
\[ \text{HasPort}(\text{AMPLIFIER}, \text{CINCH}), \]
\[ \text{HasPort}(\text{CABLE} \_\text{HDMI}, \text{HDMI}) \} \]
Planning problems are usually defined in terms of a description language based on a first-order predicate logic.

- States are sets of (ground) propositions, e.g.,
  \[ s = \{ \text{HasPort(AMPLIFIER, HDMI)}, \]
  \[ \text{HasPort(AMPLIFIER, CINCH)}, \]
  \[ \text{HasPort(CABLE, HDMI, HDMI)} \}\]

- Actions are defined by preconditions and effects, e.g.,
  \[ \text{plugin}(?\text{cable}, ?\text{device}, ?\text{port}) \]

  
  \[
  \begin{align*}
  \text{precondition: } & \text{HasPort}(?\text{device}, ?\text{port}) \land \\
  & \text{HasPort}(?\text{cable}, ?\text{port}) \land \\
  & \neg \exists \text{cable}' : \text{IsConnected}(?\text{device}, ?\text{cable}', ?\text{port}) \\
  \text{effect: } & \text{IsConnected}(?\text{device}, ?\text{cable}, ?\text{port})
  \end{align*}
  \]

  (Signal flow not shown for the sake of simplicity)
Hierarchical planning extends classical planning by hierarchies:

- on the tasks, which need to be decomposed, or
- on the state features, which need to be achieved (or decomposed)
Hierarchical planning extends classical planning by hierarchies:

- on the tasks, which need to be decomposed, or
- on the state features, which need to be achieved (or decomposed)

Why should we incorporate task or state hierarchies?
Hierarchical planning extends classical planning by hierarchies:

- on the tasks, which need to be decomposed, or
- on the state features, which need to be achieved (or decomposed)

**Why should we incorporate task or state hierarchies?** E.g., because:

- Domain experts might want to model it that way.
- Plans can be presented *more abstract* by relying on task hierarchies.
- Standard solutions or search advice can be modeled.
- More control on the generated solutions using structures similar to formal grammars.
Hierarchical planning extends classical planning by hierarchies:

- on the tasks, which need to be decomposed, or
- on the state features, which need to be achieved (or decomposed)

*Why should we incorporate task or state hierarchies?* E.g., because:

- Domain experts might want to model it that way.
- Plans can be presented *more abstract* by relying on task hierarchies.
- Standard solutions or search advice can be modeled.
- More control on the generated solutions using structures similar to formal grammars.

⇝ Our focus: Survey of recent hierarchical problem classes.
Problem Class: Hierarchical Task Network (HTN) Planning

\[ \mathcal{P} = (V, P, \delta, C, M, s_I, c_I) \]

- \( V \) a set of state variables.
Problem Class: Hierarchical Task Network (HTN) Planning

\[ \mathcal{P} = (V, P, \delta, C, M, s_I, c_I) \]

- \( V \) a set of state variables.
- \( P \) a set of primitive task names.
- \( \delta : P \rightarrow (2^V)^3 \) the task name mapping.
- \( C \) a set of compound task names.
Problem Class: Hierarchical Task Network (HTN) Planning

\[ \mathcal{P} = (V, P, \delta, C, M, s_I, c_I) \]

- \( V \) a set of state variables.
- \( P \) a set of primitive task names.
- \( \delta : P \rightarrow (2^V)^3 \) the task name mapping.
- \( C \) a set of compound task names.
- \( c_I \in C \) the initial task.

A solution task network \( tn \) must:

- be a refinement of \( c_I \),
Problem Class: Hierarchical Task Network (HTN) Planning

\[ P = (V, P, \delta, C, M, s_I, c_I) \]

- \( V \) a set of state variables.
- \( P \) a set of primitive task names.
- \( \delta : P \rightarrow (2^V)^3 \) the task name mapping.
- \( C \) a set of compound task names.
- \( c_I \in C \) the initial task.
- \( M \subseteq C \times 2^{TN} \) the methods.

A solution task network \( tn \) must:
- be a refinement of \( c_I \),
- only contain primitive tasks, and
Problem Class: Hierarchical Task Network (HTN) Planning

\[ \mathcal{P} = (V, P, \delta, C, M, s_I, c_I) \]

- \( V \) a set of state variables.
- \( P \) a set of primitive task names.
- \( \delta : P \rightarrow (2^V)^3 \) the task name mapping.
- \( C \) a set of compound task names.
- \( c_I \in C \) the initial task.
- \( M \subseteq C \times 2^{TN} \) the methods.

A solution task network \( tn \) must:
- be a refinement of \( c_I \),
- only contain primitive tasks, and
Problem Class: Hierarchical Task Network (HTN) Planning

\[ P = (V, P, \delta, C, M, s_I, c_I) \]

- \( V \) a set of state variables.
- \( P \) a set of primitive task names.
- \( \delta : P \rightarrow (2^V)^3 \) the task name mapping.
- \( C \) a set of compound task names.
- \( c_I \in C \) the initial task.
- \( M \subseteq C \times 2^{TN} \) the methods.

A solution task network \( tn \) must:
- be a refinement of \( c_I \),
- only contain primitive tasks, and
Problem Class: Hierarchical Task Network (HTN) Planning

\[ \mathcal{P} = (V, P, \delta, C, M, s_I, c_I) \]

- \( V \) a set of state variables.
- \( P \) a set of primitive task names.
- \( \delta : P \rightarrow (2^V)^3 \) the task name mapping.
- \( C \) a set of compound task names.
- \( c_I \in C \) the initial task.
- \( M \subseteq C \times 2^{TN} \) the methods.

A solution task network \( tn \) must:
- be a refinement of \( c_I \),
- only contain primitive tasks, and
Problem Class: Hierarchical Task Network (HTN) Planning

\[ \mathcal{P} = (V, P, \delta, C, M, s_I, c_I) \]

- \( V \) a set of state variables.
- \( P \) a set of primitive task names.
- \( \delta : P \rightarrow (2^V)^3 \) the task name mapping.
- \( C \) a set of compound task names.
- \( c_I \in C \) the initial task.
- \( M \subseteq C \times 2^{TN} \) the methods.

A solution task network \( t_n \) must:
- be a refinement of \( c_I \),
- only contain primitive tasks, and
Problem Class: Hierarchical Task Network (HTN) Planning

\[ \mathcal{P} = (V, P, \delta, C, M, s_I, c_I) \]

- \( V \) a set of state variables.
- \( P \) a set of primitive task names.
- \( \delta : P \rightarrow (2^V)^3 \) the task name mapping.
- \( C \) a set of compound task names.
- \( c_I \in C \) the initial task.
- \( M \subseteq C \times 2^{TN} \) the methods.

A solution task network \( tn \) must:
- be a refinement of \( c_I \),
- only contain primitive tasks, and
Problem Class: Hierarchical Task Network (HTN) Planning

\[ \mathcal{P} = (V, P, \delta, C, M, s_I, c_I) \]

- \( V \) a set of state variables.
- \( P \) a set of primitive task names.
- \( \delta : P \rightarrow (2^V)^3 \) the task name mapping.
- \( C \) a set of compound task names.
- \( c_I \in C \) the initial task.
- \( M \subseteq C \times 2^{TN} \) the methods.

A solution task network \( tn \) must:
- be a refinement of \( c_I \),
- only contain primitive tasks, and
Problem Class: Hierarchical Task Network (HTN) Planning

\[ \mathcal{P} = (V, P, \delta, C, M, s_I, c_I) \]

- \( V \) a set of state variables.
- \( P \) a set of primitive task names.
- \( \delta : P \rightarrow (2^V)^3 \) the task name mapping.
- \( C \) a set of compound task names.
- \( c_I \in C \) the initial task.
- \( M \subseteq C \times 2^{TN} \) the methods.
- \( s_I \subseteq V \) the initial state.

A solution task network \( tn \) must:
- be a refinement of \( c_I \),
- only contain primitive tasks, and
Problem Class: Hierarchical Task Network (HTN) Planning

\[ \mathcal{P} = (V, P, \delta, C, M, s_I, c_I) \]

- \( V \) a set of state variables.
- \( P \) a set of primitive task names.
- \( \delta : P \to (2^V)^3 \) the task name mapping.
- \( C \) a set of compound task names.
- \( c_I \in C \) the initial task.
- \( M \subseteq C \times 2^{TN} \) the methods.
- \( s_I \subseteq V \) the initial state.

A solution task network \( tn \) must:
- be a refinement of \( c_I \),
- only contain primitive tasks, and
- have an executable linearization.
In contrast to classical planning, there is no state-based goal.
The objective is to decompose the initial abstract tasks into an executable primitive action sequence.
In contrast to classical planning, there is no state-based goal.
The objective is to decompose the initial abstract tasks into an executable primitive action sequence.
In contrast to classical planning, the planner is not allowed to insert actions arbitrarily.
In contrast to classical planning, there is no state-based goal.
The objective is to decompose the initial abstract tasks into an executable primitive action sequence.
In contrast to classical planning, the planner is *not* allowed to insert actions arbitrarily.
Still, every classical problem can be encoded as an HTN problem, but HTN planning is more expressive.
In the general case, it is undecidable to determine whether an HTN problem has a solution.
HTN Planning Problems – Some Notes

- In contrast to classical planning, there is no state-based goal.
- The objective is to decompose the initial abstract tasks into an executable primitive action sequence.
- In contrast to classical planning, the planner is not allowed to insert actions arbitrarily.
- Still, every classical problem can be encoded as an HTN problem, but HTN planning is more expressive.
- In the general case, it is undecidable to determine whether an HTN problem has a solution.
- The paper also – very briefly – explains all standard algorithms for solving HTN problems.
What to do when plans are not executable?

- Allowing Task Insertion means that actions can be inserted arbitrarily.
Optional Feature: Task Insertion

- Allowing *Task Insertion* means that actions can be inserted arbitrarily.
- This gives greater flexibility: certain parts do not need to be modeled via the hierarchy.

What to do when plans are not executable?
Optional Feature: Task Insertion

- Allowing *Task Insertion* means that actions can be inserted arbitrarily.
- This gives greater flexibility: certain parts do not need to be modeled via the hierarchy.
- Allowing Task Insertion in HTN planning results in the problem class called *TIHTN planning*.

What to do when plans are not executable?
**Optional Feature: Task Sharing**

*Task Sharing* allows to eliminate duplicates that might just be modeling artifacts:

- `connect(DVD-player, Adapter)`
- `connect(Adapter, TV)`
- `connect(Blu-ray-player, Adapter)`
- `connect(Adapter, TV)`

→ actions stem from some method

→ actions stem from *another* method
Task Sharing allows to eliminate duplicates that might just be modeling artifacts:
In many formalizations, abstract tasks show preconditions and effects.

\[
\text{move}(\text{?obj,?from,?to})
\]

decomposes to

\[
egin{align*}
\text{pick}(\text{?obj,?from}) & \quad \neg\text{handopen} \\
\neg\text{at}(\text{?obj,?from}) & \quad \text{holding(\text{?obj})} \\
\text{handopen} & \quad \text{holding(\text{?obj})} \\
\neg\text{at}(\text{?obj,?from}) & \quad \text{place}(\text{?obj,?to}) \\
\text{holding(\text{?obj})} & \quad \text{at}(\text{?obj,?to})
\end{align*}
\]
• In many formalizations, abstract tasks show preconditions and effects.

\[ \text{at}(\text{?obj}, \text{?from}) \quad \text{move}(\text{?obj}, \text{?from}, \text{?to}) \quad \text{at}(\text{?obj}, \text{?to}) \]

decomposes to

\[ \text{at}(\text{?obj}, \text{?from}) \quad \text{handopen} \quad \text{pick}(\text{?obj}, \text{?from}) \quad \text{handopen} \quad \text{holding}(\text{?obj}) \quad \text{at}(\text{?obj}, \text{?from}) \]

\[ \text{holding}(\text{?obj}) \]

\[ \text{at}(\text{?obj}, \text{?to}) \quad \text{handopen} \quad \text{place}(\text{?obj}, \text{?to}) \quad \text{handopen} \quad \text{at}(\text{?obj}, \text{?to}) \]
Problem Class: Decompositional and Hybrid Planning

- In many formalizations, abstract tasks show preconditions and effects.

- They are e.g. used for checking implementation or legality criteria of methods,
• In many formalizations, abstract tasks show preconditions and effects.

- They are e.g. used for checking implementation or legality criteria of methods,
- or to insert abstract tasks during planning.
• In many formalizations, abstract tasks show preconditions and effects.

• They are e.g. used for checking implementation or legality criteria of methods,

• or to insert abstract tasks during planning.

• Some of these formalizations feature an initial task network, others do not.
Problem Class: HGN Planning

- Instead of restricting *what to do* (i.e., which action to perform), we can restrict *what to achieve* (i.e., which state features).
• Instead of restricting *what to do* (i.e., which action to perform), we can restrict *what to achieve* (i.e., which state features).

• For this, *Hierarchical Goal Network (HGN) planning* defines a hierarchy on *goals*. 
Problem Class: HGN Planning

- Instead of restricting what to do (i.e., which action to perform), we can restrict what to achieve (i.e., which state features).
- For this, Hierarchical Goal Network (HGN) planning defines a hierarchy on goals.
- Methods thus refine goals rather than tasks.
Problem Class: HGN Planning

- Instead of restricting *what to do* (i.e., which action to perform), we can restrict *what to achieve* (i.e., which state features).
- For this, *Hierarchical Goal Network (HGN) planning* defines a hierarchy on *goals*.
- Methods thus refine goals rather than tasks.
- Actions can be applied to a current state only if they contribute towards a goal without predecessors.
• Instead of restricting *what to do* (i.e., which action to perform), we can restrict *what to achieve* (i.e., which state features).

• For this, *Hierarchical Goal Network (HGN) planning* defines a hierarchy on *goals*.

• Methods thus refine goals rather than tasks.

• Actions can be applied to a current state only if they contribute towards a goal without predecessors.

• Goals are achieved if they hold in the current state.
We provided a survey on hierarchical planning.
### Core Contributions of the Survey

- We provided a survey on *hierarchical planning*.
- We focused on various problem classes, most of which were developed or formalized within the last 10 years.
Core Contributions of the Survey

- We provided a survey on *hierarchical planning*.
- We focused on various problem classes, most of which were developed or formalized within the last 10 years.
- We discussed the impact of the different problem classes/features on the set of solutions.
### Core Contributions of the Survey

- We provided a survey on *hierarchical planning*.
- We focused on various problem classes, most of which were developed or formalized within the last 10 years.
- We discussed the impact of the different problem classes/features on the set of solutions.
- We provided a brief explanation of all standard algorithms for solving HTN problems.