Hierachical Task Network Planning

1 Einführung

- **Ansatz**:
  Aufgaben durchführen (anstatt "Ziele" zu erreichen)
- **HTN-Prinzip**:
  rekursive Zerlegung komplexer Aufgaben in kleinere Unteraufgaben - bis einfache Aufgaben erreicht werden, die dann direkt ausgeführt werden können

2 Simple Task Network (STN) Planning

- Spezialfall des HTN Planning
- dabei sind Zustände und Operatoren (states and operators) genauso wie beim "classical planning" definiert
- zusätzlich gibt es noch tasks, methods und task networks

2.1 Aufgaben und Methoden

Definition 2.1 *(Task) Ein Task ist ein Ausdruck der Form* t(r_1,...,r_n) *mit:*

- t *ist das* task symbol*, wobei man zwischen Operatoren (primitive) und Methoden (nonprimitive) unterscheidet,*
- *jedes r_i bezeichnet einen Term.*

Definition 2.2 *(Simple Task Network) Ein Simple Task Network ist ein azyklisch gerichteter Graph* w = (U,E) *mit:*

- *U ist die Menge der Knoten, wobei jeder Knoten einen task t_u enthält,*
- *E ist die Menge der Kanten, welche eine partielle Ordnung der Knoten festlegen.*

Beispiel (Simple Task Network):

In einer DWR-Domain gibt es drei Tasks:

- t_1 = *take(cran2,loc1,c1,c2,p1)* (primitive task),
- t_2 = *put(cran2,loc2,c3,c4,p2)* (primitive task),
- t_3 = *move-stack(p1,q)* (nonprimitive task),

und zwei Task Networks (∀ Knoten u_i gilt $t_{u_i} = t_i$):

- $w_1 = (\{u_1, u_2, u_3\}, \{(u_1, u_2), (u_2, u_3)\})$,
- $w_2 = (\{u_1, u_2\}, \{(u_1, u_2)\})$.

Da w_2 total geordnet ist, schreibt man auch $w_2 = \langle t_1, t_2 \rangle$. Wenn w_2 auch *ground* und *primitive* ist, entspricht dies dem Plan π = ⟨take(cran2,loc1,c1,c2,p1),put(cran2,loc2,c3,c4,p2)⟩.

Definition 2.3 *(STN-Methode) Eine STN-Methode ist ein 4-Tupel*

$$\mathrm{m}=(name(\mathrm{m}),task(\mathrm{m}),precond(\mathrm{m}),network(\mathrm{m}))$$

mit:

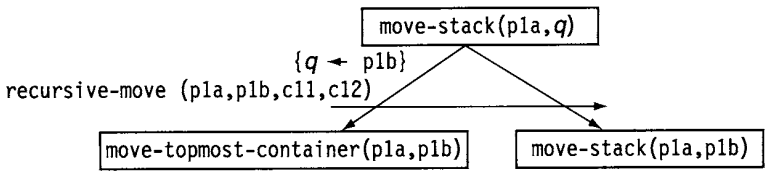
- *name*(m)*: der Name der Methode, z.B. ein Ausdruck der Form* $\mathrm{n}(x_1,...,x_n)$*, wobei* n *ein eindeutiges Methoden-Symbol ist und* $x_1,...,x_n$ *die Parameter von* n *sind,*
- *task*(m)*: ein nonprimitive Task,*
- *precond*(m)*: Vorbedingungen der Methode,*
- *network*(m)*: Task-Network mit den Unteraufgaben von* m*.*

Definition 2.4 *(Anwendbare Methode) Eine Methode* m *ist anwendbar in einem Zustand* s*, wenn gilt:* $precond^+(\mathrm{m}) \subseteq s$ *und* $precond^-(\mathrm{m}) \cap s = \emptyset$*.*

Definition 2.5 *(Passende Methode) Sei* t *ein Task und* m *eine Methode.* m *ist dann passend für* t*, wenn es eine Substitution* σ *gibt, so dass gilt* $\sigma(\mathrm{t}) = task(\mathrm{m})$*. Die Ersetzung von* t *durch* m *unter* σ *ist* $\delta(t,m,\sigma) = network(\mathrm{m})$*. Wenn* m *total geordnet ist, können wir auch schreiben* $\delta(\mathrm{t},\mathrm{m},\sigma) = subtasks(\mathrm{m})$*.*

Beispiel (Methodenanwendung):

- Ausgangszustand *s* entspricht dem Beispiel 1, *t* ist der *nonprimitive Task move-stack(p1a,q)* und *m* ist die *Methode recursive-move(p1a,p1b,c11,c12)*
- dann ist *m* anwendbar in *s* und passend für *t* (wenn q durch p1b ersetzt wird)
- t wird in folgende *subtasks* zerlegt: $\delta(t,m,\sigma) = \langle$move-topmost-container(p1a,p1b),move stack(p1a,p1b)$\rangle$



2.2 Probleme und Lösungen

Definition 2.6 *(STN Planning Domain) Eine STN Planning Domain ist ein Paar* $\mathcal{D} = (\mathrm{O},\mathrm{M})$ *mit:*

- O *ist eine Menge von Operatoren,*
- M *ist eine Menge von Methoden.*

$\mathcal{D}$ *ist eine Total-Order Planning Domain, wenn jedes* $m \in M$ *total geordnet ist.*

Definition 2.7 *(STN Planning Problem) Ein STN Planning Problem ist ein 4-Tupel* $\mathcal{P} = (s_0,\mathrm{w},\mathrm{O},\mathrm{M})$ *mit:*

- s_0 *ist der Ausgangszustand,*
- w *ist ein Task Network (initial task network),*
- $\mathcal{D} = (\mathrm{O},\mathrm{M})$ *ist eine STN Planning Domain.*

$\mathcal{P}$ *ist ein Total-Order Planning Problem, wenn* w *und* $\mathcal{D}$ *total geordnet sind.*

Definition 2.8 *(Solution Plan) Sei $\mathcal{P}$ = (s_0,w,O,M) ein Planungsproblem. Dann gibt es 3 Fälle, in welchem ein Plan $\pi = \langle a_1,...,a_n \rangle$ eine Lösung für $\mathcal{P}$ ist:*

- ***Case 1:*** w *ist leer. Dann ist π eine Lösung für $\mathcal{P}$, wenn π leer ist (z.B.* n = *0),*
- ***Case 2:*** *Es gibt einen* primitive task node u ∈ w, *der keinen Vorgänger in* w *besitzt. Dann ist π eine Lösung für $\mathcal{P}$, wenn a_1 anwendbar auf t_u in s_0 ist und der Plan $\pi = \langle a_2, ..., a_n \rangle$ eine Lösung für das Planungsproblem $\mathcal{P}'$ ist:*

$$\mathcal{P}' = (\gamma(S_0, a_1), w - \{u\}, O, M),$$

- ***Case 3:*** *Es gibt einen* nonprimitive task node u ∈ w, *der keinen Vorgänger besitzt in* w, *und es existiert eine Methode* m, *die passend für t_u und in s_0 anwendbar ist. Dann ist π eine Lösung für $\mathcal{P}$, wenn es ein task network* w' ∈ δ(w,u,m,σ) *gibt, so dass π eine Lösung für (s_0,w',O,M) ist.*

3 Total-Order STN Planning

TFD (Total-order Forward Decomposition):

```
TFD(s, ⟨t_1, ..., t_k⟩, O, M)
    if k = 0 then return ⟨⟩ (i.e., the empty plan)
    if t_1 is primitive then
        active ← {(a, σ) | a is a ground instance of an operator in O,
                           σ is a substitution such that a is relevant for σ(t_1),
                           and a is applicable to s}
        if active = ∅ then return failure
        nondeterministically choose any (a, σ) ∈ active
        π ← TFD(γ(s, a), σ(⟨t_2, ..., t_k⟩), O, M)
        if π = failure then return failure
        else return a.π
    else if t_1 is nonprimitive then
        active ← {m | m is a ground instance of a method in M,
                      σ is a substitution such that m is relevant for σ(t_1),
                      and m is applicable to s}
        if active = ∅ then return failure
        nondeterministically choose any (m, σ) ∈ active
        w ← subtasks(m).σ(⟨t_2, ..., t_k⟩)
        return TFD(s, w, O, M)
```

TFD Vergleiche:

- Wie Vorwärtsplaner beachtet TFD nur Aktionen, deren Vorbedingungen im aktuellen Zustand erfüllt sind. Außerdem werden, wie bei Rückwärtsplanern, nur Aktionen berücksichtigt, die nützlich sind, um das Ziel zu erreichen. Diese Kombination führt zu einer enormen Leistungssteigerung der Suche.
- Wie Vorwärtsplaner erzeugt TFD nur Aktionen in der Reihenfolge, wie sie ausgeführt werden, d.h. TFD weiß immer über den aktuellen Zustand Bescheid.

4 Partial-Order STN Planning

- nicht alle Planungsprobleme können in Total-Order Planning umgeschrieben werden
- ein erweiterter Algorithmus wird benötigt
- der Plan legt die Reihenfolge, in welcher die *Tasks* ausgeführt werden müssen nur teilweise fest

PFD (Partial-order Forward Decomposition):

```
PFD(s, w, O, M)
    if w = ∅ then return the empty plan
    nondeterministically choose any u ∈ w that has no predecessors in w
    if t_u is a primitive task then
        active ← {(a, σ) | a is a ground instance of an operator in O,
                           σ is a substitution such that name(a) = σ(t_u),
                           and a is applicable to s}
        if active = ∅ then return failure
        nondeterministically choose any (a, σ) ∈ active
        π ← PFD(γ(s, a), σ(w − {u}), O, M)
        if π = failure then return failure
        else return a.π
    else
        active ← {(m, σ) | m is a ground instance of a method in M,
                           σ is a substitution such that name(m) = σ(t_u),
                           and m is applicable to s}
        if active = ∅ then return failure
        nondeterministically choose any (m, σ) ∈ active
        nondeterministically choose any task network w' ∈ δ(w, u, m, σ)
        return(PFD(s, w', O, M)
```

5 HTN Planning

5.1 Einführung

HTN planning ist eine Extension zu STN planning, die es ermöglicht Task networks mit mehr als nur TFD oder PFD zu benutzen.

Um dies zu ermöglichen, wird das Task network verallgemeinert, um noch nicht erfüllte Constraints explizit im Task Network auszudrücken.

5.2 Task Networks

Definition 5.1 (Task network) *Ein task network ist ein Tupel $w = (U, C)$, U ist eine Menge von task nodes und C ist eine Menge von Constraints.*

Wobei Constraints Anforderungen sind, die von jedem Plan, der eine Lösung für das Planungsproblem ist, erfüllt werden müssen. Für alle Constraintbeschreibungen sei:

- π eine Lösung für w
- $U' \subseteq U'$ eine Menge von task nodes in w
- A die Menge aller Aktionen in $a_i \in \pi$, so dass in πs Dekomposition a_i ein Nachfolger eines Knoten U' ist.
- first(U', π) und last(U', π) sind die letzten und ersten Aktionen

Mögliche Constraints sind:

- precedence
- before
- after
- between

5.3 HTN Methoden

Definition 5.2 (HTN Methode) *Eine HTN Methode ist ein 4-Tupel*

$$m = (name(m), task(m), subtasks(m), constr(m)).$$

- *name*(m) *ist ein Ausdruck der Form:* $n(x_1, ..., x_k)$*, wobei n ein einzigartiges Methodensymbol ist und* $x_1, ...x_k$ *alle Variablensymbole sind, die in m auftreten.*
- *task*(m) *ist ein nicht-primitive task.*
- (*subtasks*(m), *constr*(m)) *ist ein task network.*

Eine Instanz m zerlegt ein task network $w = (U, C)$ mit $u \in U$ und den, zu u und m gehörigen, Task t_u in subtasks(m') und produziert das task network:

$$\delta(w, u, m) = ((U - \{u\}) \cup \text{subtasks}(m'), C' \cup \text{const}(m'))$$

wobei C' aus C gewonnen wird durch:

- Für jedes precedence constraint, das u beinhaltet, ersetze es mit precedence contraints, welche die Knoten von subtasks(m') enthalten.
- Für jedes before-, after, oder between constraint, in dem eine Menge von task nodes U', die u enthalten, vorkommt, ersetze U' mit $(U' - \{u\}) \cup$ subtasks(m').

5.4 Probleme und Lösungen

Definition 5.3 (HTN planning domain) *Eine HTN planning domain ist ein Paar*

$$\mathcal{D} = (O, M),$$

und ein HTN planning problem ist ein 4-Tupel

$$\mathcal{P} = (s_0, w, O, M),$$

wobei s_0 *der Ausgangszustand, w das anfängliche task network, O eine Menge von Operatoren und M eine Menge von HTN Methoden sind.*

Definition 5.4 (Lösung) *Falls* $w = (U, C)$ *primitiv ist, ist ein Plan* π *eine Lösung, falls:*

- *es existiert eine ground instance* (U', C') *von* (U, C)
- *es existiert eine totale Ordnung* $\langle u_1, u_2, ..., u_k \rangle$

So dass:

- *Aktionen in* π *sind von den Knoten genannt*
- π *ist ausführbar in* s_0
- *precedence, before, after und between constraints werden von der Ordnung erfüllt*

Falls $w = (U, C)$ *nicht primitiv ist, dann is* π *eine Lösung, falls eine Sequenz von Dekompositionen existiert, so dass* π *eine Lösung für das entstandene primitive task network ist.*

5.5 Planning Procedures

HTN planning procedures haben die Aufgabe Operatoren zu instanziieren und Tasks zu zerlegen. Da mehrere Wege existieren diese Probleme zu lösen, wird hier nur eine abstrakte Methode vorgestellt, um Planungsprobleme zu lösen.

```
Abstract-HTN(s, U, C, O, M)
  if (U,C) can be shown to have no solution
    then return failure
  else if U is primitive then
      if (U, C) has a solution then
        nondeterministically let π be any such solution
        return π
      else return failure
  else
    choose a nonprimitive task node u ∈ U
    active ← { m ∈ M | task(m) is unifiable with t_u }
    if active ≠ ∅ then
      nondeterministically choose any m ∈ active
      σ ← an mgu for m and t_u that renames all variables of m
      (U',C') ← δ(σ(U,C), σ(u), σ(m))
      (U',C') ← apply-critic(U',C')
      return Abstract-HTN(s, U', C', O, M)
    else return failure
```

5.6 Erweiterungen und erweiterte Ziele

HTN planning ermöglicht, wie auch Classical Planning, verschiedene Erweiterungen in den Planungsalgorithmus zu inkorporieren. Dadurch kann die Expressivität von HTN planning entscheidend erweitert werden. Erweiterungen können unterschiedliche Effekte auf die Vollständigkeit und Korrektheit des Algorithmus haben.

Erweiterte Ziele können mithilfe verschiedener Erweiterungen leicht ausgedrückt werden, ohne das der Formalismus weiter verändert werden muss. Die Fähigkeit von HTN planning leicht erweiterbar zu sein, begünstigt seinen Einsatz in Planern.

Mögliche Erweiterungen für HTN Planer:

- Funktionssymbole
- Axiome
- Attached Procedures
- High-Level Effekte
- External Preconditions
- Zeit

Beispiele für erweiterte Ziele:

- DWR Domäne: Roboter soll niemals zu einem bestimmten Ort (bad-loc). Veränderung der Operatoren nicht nötig, da move-Operator nur verwendet wird, wenn er in einem subtask existiert. Einschränkung über preconditions der Methoden, die move aufrufen.
- Einschränkungen wie "Ziel in 5 Aktionen erreichen" über Einschränkung der Domäne oder Verwendung von Erweiterungen (Funktionssymbole, Attached Procedures), um arithmetische Operationen zu ermöglichen und zu zählen.