

Planning and Optimization

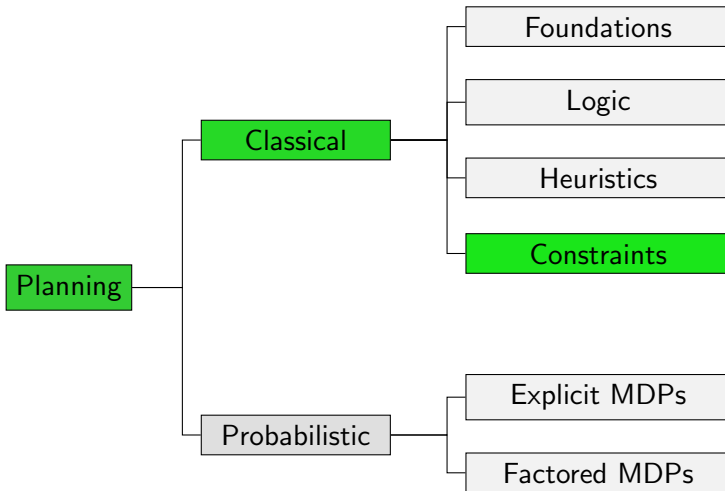
E8. Network Flow Heuristics

Malte Helmert and Thomas Keller

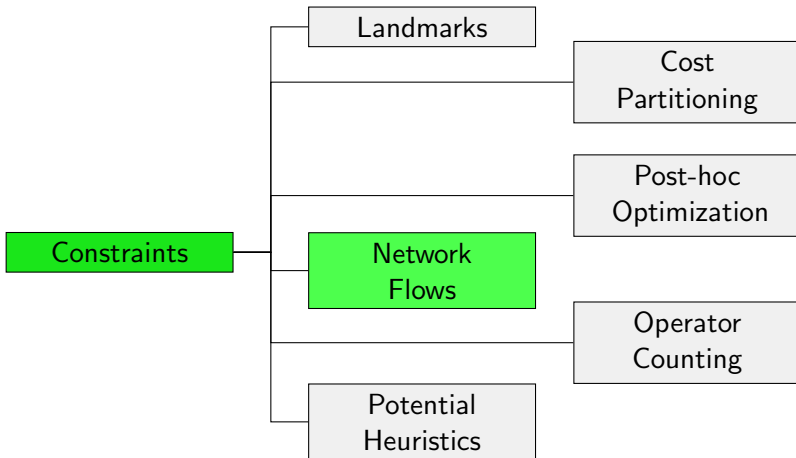
Universität Basel

November 20, 2019

Content of this Course



Content of this Course: Constraints



Introduction

Reminder: SAS⁺ Planning Tasks

For a SAS⁺ planning task $\Pi = \langle V, I, O, \gamma \rangle$:

- V is a set of **finite-domain state variables**,
- Each **atom** has the form $v = d$ with $v \in V, d \in \text{dom}(v)$.
- Operator **preconditions** and the **goal** formula γ are **satisfiable conjunctions of atoms**.
- Operator **effects** are **conflict-free conjunctions of atomic effects** of the form $v_1 := d_1 \wedge \dots \wedge v_n := d_n$.

Example Task (1)

- One package, two trucks, two locations
- Variables:
 - $pos-p$ with $\text{dom}(pos-p) = \{loc_1, loc_2, t_1, t_2\}$
 - $pos-t-i$ with $\text{dom}(pos-t-i) = \{loc_1, loc_2\}$ for $i \in \{1, 2\}$
- The package is at location 1 and the trucks at location 2,
 - $I = \{pos-p \mapsto loc_1, pos-t-1 \mapsto loc_2, pos-t-2 \mapsto loc_2\}$
- The goal is to have the package at location 2 and truck 1 at location 1.
 - $\gamma = (pos-p = loc_2) \wedge (pos-t-1 = loc_1)$

Example Task (2)

- Operators: for $i, j, k \in \{1, 2\}$:

$$\begin{aligned} \text{load}(t_i, \text{loc}_j) = \langle & \text{pos-}t\text{-}i = \text{loc}_j \wedge \text{pos-}p = \text{loc}_j, \\ & \text{pos-}p := t_i, 1 \rangle \end{aligned}$$

$$\begin{aligned} \text{unload}(t_i, \text{loc}_j) = \langle & \text{pos-}t\text{-}i = \text{loc}_j \wedge \text{pos-}p = t_i, \\ & \text{pos-}p := \text{loc}_j, 1 \rangle \end{aligned}$$

$$\begin{aligned} \text{drive}(t_i, \text{loc}_j, \text{loc}_k) = \langle & \text{pos-}t\text{-}i = \text{loc}_j, \\ & \text{pos-}t\text{-}i := \text{loc}_k, 1 \rangle \end{aligned}$$

Example Task: Observations

Consider some atoms of the example task:

- $pos-p = loc_1$ initially true and must be false in the goal
 - ▷ at location 1 the package must be loaded one time more often than unloaded.
- $pos-p = loc_2$ initially false and must be true in the goal
 - ▷ at location 2 the package must be unloaded one time more often than loaded.
- $pos-p = t_1$ initially false and must be false in the goal
 - ▷ same number of load and unload actions for truck 1.

Can we derive a heuristic from this kind of information?

Example: Flow Constraints

Let π be some arbitrary plan for the example task and let $\#o$ denote the **number of occurrences** of operator o in π . Then the following holds:

- $pos-p = loc_1$ initially true and must be false in the goal
 - ▷ at location 1 the package must be loaded one time more often than unloaded.

$$\begin{aligned} \#load(t_1, loc_1) + \#load(t_2, loc_1) = \\ 1 + \#unload(t_1, loc_1) + \#unload(t_2, loc_1) \end{aligned}$$

- $pos-p = t_1$ initially false and must be false in the goal
 - ▷ same number of load and unload actions for truck 1.

$$\begin{aligned} \#unload(t_1, loc_1) + \#unload(t_1, loc_2) = \\ \#load(t_1, loc_1) + \#load(t_1, loc_2) \end{aligned}$$

Network Flow Heuristics: General Idea

- Formulate **flow constraints** for each atom.
- These are satisfied by **every plan** of the task.
- The cost of a plan is $\sum_{o \in O} cost(o) \#o$
- The objective value of an integer program that minimizes this cost subject to the flow constraints is a lower bound on the plan cost (i.e., an admissible heuristic estimate).
- As solving the IP is NP-hard, we solve the LP relaxation instead.

How do we get the flow constraints?

How to Derive Flow Constraints?

- The constraints formulate how often an atom can be produced or consumed.
- “Produced” (resp. “consumed”) means that the atom is false (resp. true) before an operator application and true (resp. false) in the successor state.
- For general SAS^+ operators, this depends on the state where the operator is applied: effect $v := d$ only produces $v = d$ if the operator is applied in a state s with $s(v) \neq d$.
- For general SAS^+ tasks, the goal does not have to specify a value for every variable.
- All this makes the definition of flow constraints somewhat involved and in general such constraints are inequalities.

Good news: easy for tasks in transition normal form

Transition Normal Form

Variables Occurring in Conditions and Effects

- Many algorithmic problems for SAS⁺ planning tasks become simpler when we can make two further restrictions.
- These are related to the **variables** that **occur** in conditions and effects of the task.

Definition ($\text{vars}(\varphi)$, $\text{vars}(e)$)

For a logical formula φ over finite-domain variables V , $\text{vars}(\varphi)$ denotes the set of finite-domain variables occurring in φ .

For an effect e over finite-domain variables V , $\text{vars}(e)$ denotes the set of finite-domain variables occurring in e .

Transition Normal Form

Definition (Transition Normal Form)

A SAS⁺ planning task $\Pi = \langle V, I, O, \gamma \rangle$
is in **transition normal form (TNF)** if

- for all $o \in O$, $\text{vars}(\text{pre}(o)) = \text{vars}(\text{eff}(o))$, and
- $\text{vars}(\gamma) = V$.

In words, an **operator** in TNF must mention the same variables in the precondition and effect, and a **goal** in TNF must mention all variables (= specify exactly one goal state).

Converting Operators to TNF: Violations

There are two ways in which an operator o can violate TNF:

- There exists a variable $v \in \text{vars}(\text{pre}(o)) \setminus \text{vars}(\text{eff}(o))$.
- There exists a variable $v \in \text{vars}(\text{eff}(o)) \setminus \text{vars}(\text{pre}(o))$.

The **first case** is easy to address: if $v = d$ is a precondition with no effect on v , just add the effect $v := d$.

The **second case** is more difficult: if we have the effect $v := d$ but no precondition on v , how can we add a precondition on v without changing the meaning of the operator?

Converting Operators to TNF: Multiplying Out

Solution 1: multiplying out

- 1 While there exists an operator o and a variable $v \in \text{vars}(\text{eff}(o))$ with $v \notin \text{vars}(\text{pre}(o))$:
 - For each $d \in \text{dom}(v)$, add a new operator that is like o but with the additional precondition $v = d$.
 - Remove the original operator.
- 2 Repeat the previous step until no more such variables exist.

Problem:

- If an operator o has n such variables, each with k values in its domain, this introduces k^n variants of o .
- Hence, this is an **exponential** transformation.

Converting Operators to TNF: Auxiliary Values

Solution 2: auxiliary values

- 1 For every variable v , add a new **auxiliary value** u to its domain.
- 2 For every variable v and value $d \in \text{dom}(v) \setminus \{u\}$, add a new operator to change the value of v from d to u at no cost: $\langle v = d, v := u, 0 \rangle$.
- 3 For all operators o and all variables $v \in \text{vars}(\text{eff}(o)) \setminus \text{vars}(\text{pre}(o))$, add the precondition $v = u$ to $\text{pre}(o)$.

Properties:

- Transformation can be computed in linear time.
- Due to the auxiliary values, there are new states and transitions in the induced transition system, but all **path costs** between **original states** remain the same.

Converting Goals to TNF

- The auxiliary value idea can also be used to convert the goal γ to TNF.
- For every variable $v \notin \text{vars}(\gamma)$, add the condition $v = u$ to γ .

With these ideas, every SAS^+ planning task can be converted into transition normal form in linear time.

TNF for Example Task (1)

The example task is not in transition normal form:

- Load and unload operators have preconditions on the position of some truck but no effect on this variable.
- The goal does not specify a value for variable *pos-t-2*.

TNF for Example Task (2)

Operators in transition normal form: for $i, j, k \in \{1, 2\}$:

$$\text{load}(t_i, \text{loc}_j) = \langle \text{pos-}t\text{-}i = \text{loc}_j \wedge \text{pos-}p = \text{loc}_j, \\ \text{pos-}p := t_i \wedge \text{pos-}t\text{-}i := \text{loc}_j, 1 \rangle$$

$$\text{unload}(t_i, \text{loc}_j) = \langle \text{pos-}t\text{-}i = \text{loc}_j \wedge \text{pos-}p = t_i, \\ \text{pos-}p := \text{loc}_j \wedge \text{pos-}t\text{-}i := \text{loc}_j, 1 \rangle$$

$$\text{drive}(t_i, \text{loc}_j, \text{loc}_k) = \langle \text{pos-}t\text{-}i = \text{loc}_j, \\ \text{pos-}t\text{-}i := \text{loc}_k, 1 \rangle$$

TNF for Example Task (3)

To bring the goal in normal form,

- add an additional value \mathbf{u} to $\text{dom}(pos-t-2)$

- add zero-cost operators

$$o_1 = \langle pos-t-2 = loc_1, pos-t-2 := \mathbf{u}, 0 \rangle \text{ and}$$

$$o_2 = \langle pos-t-2 = loc_2, pos-t-2 := \mathbf{u}, 0 \rangle$$

- Add $pos-t-2 = \mathbf{u}$ to the goal:

$$\gamma = (pos-p = loc_2) \wedge (pos-t-1 = loc_1) \wedge (pos-t-2 = \mathbf{u})$$

Flow Heuristic

Notation

- In SAS^+ tasks, states are variable assignments, conditions are conjunctions over atoms, and effects are conjunctions of atomic effects.
- In the following, we use a **unifying notation** to express that an atom is true in a state/entailed by a condition/made true by an effect.
- For **state** s , we write $(v = d) \in s$ to express that $s(v) = d$.
- For a **conjunction of atoms** φ , we write $(v = d) \in \varphi$ to express that φ has a conjunct $v = d$ (or alternatively $\varphi \models v = d$).
- For **effect** e , we write $(v = d) \in e$ to express that e contains the atomic effect $v := d$.

Flow Constraints (1)

A flow constraint for an atom relates how often it can be produced to how often it can be consumed.

Let o be an operator in transition normal form. Then:

- o **produces** atom a iff $a \in \text{eff}(o)$ and $a \notin \text{pre}(o)$.
- o **consumes** atom a iff $a \in \text{pre}(o)$ and $a \notin \text{eff}(o)$.
- Otherwise o is **neutral** wrt. atom a .

↪ State-independent

Flow Constraints (2)

A flow constraint for an atom relates how often it can be produced to how often it can be consumed.

The constraint depends on the current state s and the goal γ .
If γ mentions all variables (as in TNF), the following holds:

- If $a \in s$ and $a \in \gamma$ then atom a must be equally often produced and consumed.
- Analogously for $a \notin s$ and $a \notin \gamma$.
- If $a \in s$ and $a \notin \gamma$ then a must be consumed one time more often than it is produced.
- If $a \notin s$ and $a \in \gamma$ then a must be produced one time more often than it is consumed.

Iverson Bracket

The dependency on the current state and the goal can concisely be expressed with Iverson brackets:

Definition (Iverson Bracket)

Let P be a logical proposition (= some statement that can be evaluated to true or false). Then

$$[P] = \begin{cases} 1 & \text{if } P \text{ is true} \\ 0 & \text{if } P \text{ is false.} \end{cases}$$

Example: $[2 \neq 3] = 1$

Flow Constraints (3)

Definition (Flow Constraint)

Let $\Pi = \langle V, I, O, \gamma \rangle$ be a task in transition normal form.
The **flow constraint** for atom a in state s is

$$[a \in s] + \sum_{o \in O: a \in \text{eff}(o)} \text{Count}_o = [a \in \gamma] + \sum_{o \in O: a \in \text{pre}(o)} \text{Count}_o$$

- Count_o is an LP variable for the number of occurrences of operator o .
- Neutral operators either appear on both sides or on none.

Flow Heuristic

Definition (Flow Heuristic)

Let $\Pi = \langle V, I, O, \gamma \rangle$ be a SAS⁺ task in transition normal form and let $A = \{(v = d) \mid v \in V, d \in \text{dom}(v)\}$ be the set of atoms of Π .

The **flow heuristic** $h^{\text{flow}}(s)$ is the objective value of the following LP or ∞ if the LP is infeasible:

minimize $\sum_{o \in O} \text{cost}(o) \cdot \text{Count}_o$ subject to

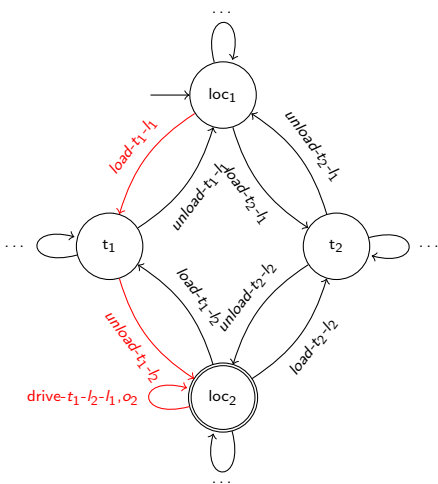
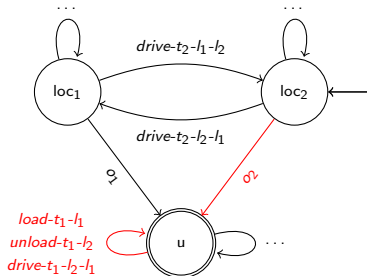
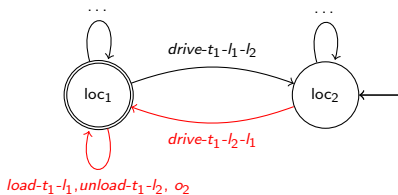
$$[a \in s] + \sum_{o \in O: a \in \text{eff}(o)} \text{Count}_o = [a \in \gamma] + \sum_{o \in O: a \in \text{pre}(o)} \text{Count}_o \text{ for all } a \in A$$

$$\text{Count}_o \geq 0 \quad \text{for all } o \in O$$

Flow Heuristic on Example Task

↪ Blackboard

Visualization of Flow in Example Task



Flow Heuristic: Properties (1)

Theorem

The flow heuristic h^{flow} is goal-aware, safe, consistent and admissible.

Proof Sketch.

It suffices to prove goal-awareness and consistency.

Goal-awareness: If $s \models \gamma$ then $\text{Count}_o = 0$ for all $o \in O$ is feasible and the objective function has value 0. As $\text{Count}_o \geq 0$ for all variables and operator costs are nonnegative, the objective value cannot be smaller. . . .

Flow Heuristic: Properties (2)

Proof Sketch (continued).

Consistency: Let o be an operator that is applicable in state s and let $s' = s[o]$.

Increasing Count_o by one in an optimal feasible assignment for the LP for state s' yields a feasible assignment for the LP for state s , where the objective function is $h^{\text{flow}}(s') + \text{cost}(o)$.

This is an upper bound on $h^{\text{flow}}(s)$, so in total $h^{\text{flow}}(s) \leq h^{\text{flow}}(s') + \text{cost}(o)$. □

Summary

Summary

- A flow constraint for an atom describes how the number of producing operator applications is linked to the number of consuming operator applications.
- The flow heuristic computes a lower bound on the cost of each operator sequence that satisfies these constraints for all atoms.
- The flow heuristic only considers the number of occurrences of each operator, but ignores their order.