Linköping University

Automated Planning

The Relaxation Principle: A closer look

Jonas Kvarnström
Department of Computer and Information Science
Linköping University

jonas.kvarnstrom@liu.se – 2019

The Problem

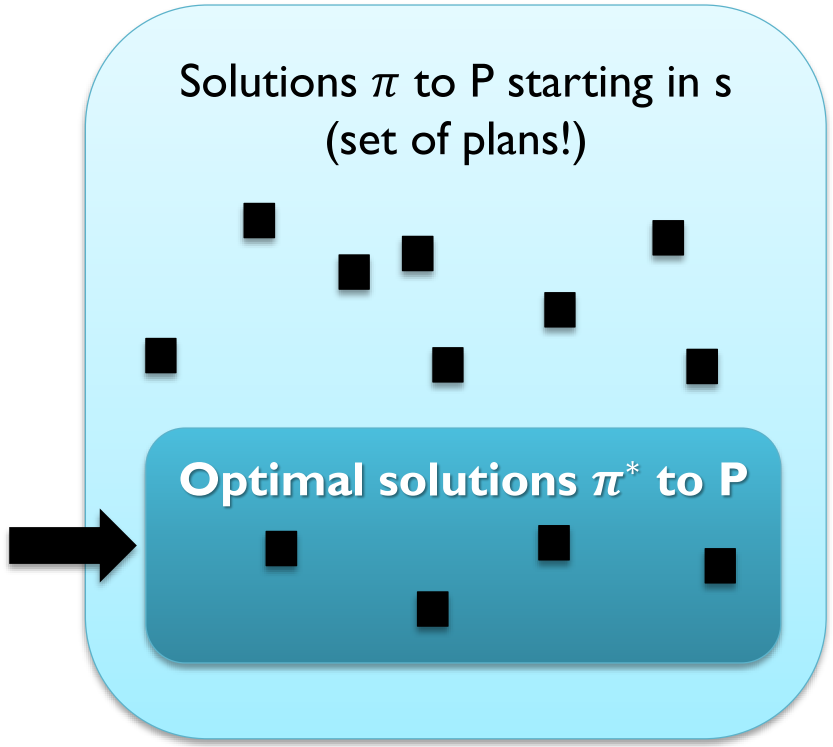
- **We have**:
  - An arbitrary planning problem $P = \langle \Sigma, s_0, S_g \rangle$
- **Suppose we want**:
  - A way to compute an **admissible heuristic h(s)**
    - Given P and some state s in the search space

What do we do?
Where do we start?
How do we think?

Fundamental Ideas (1)

- **<u>One obvious method</u>**:
  Every time we need $h(s)$ for some state s…
  1. **<u>Solve</u>** P **<u>optimally</u>** starting in s, resulting in an *actual* solution $\pi^*(s)$
  2. Let $h(s) = h^*(s) = \text{cost}(\pi^*(s))$
     - Admissible – why?
- Obvious, but stupid
  - If we find $\pi(s)$, we're already done!

Also: These are hard to find (or we wouldn't *need* a heuristic)



Fundamental Ideas (2)

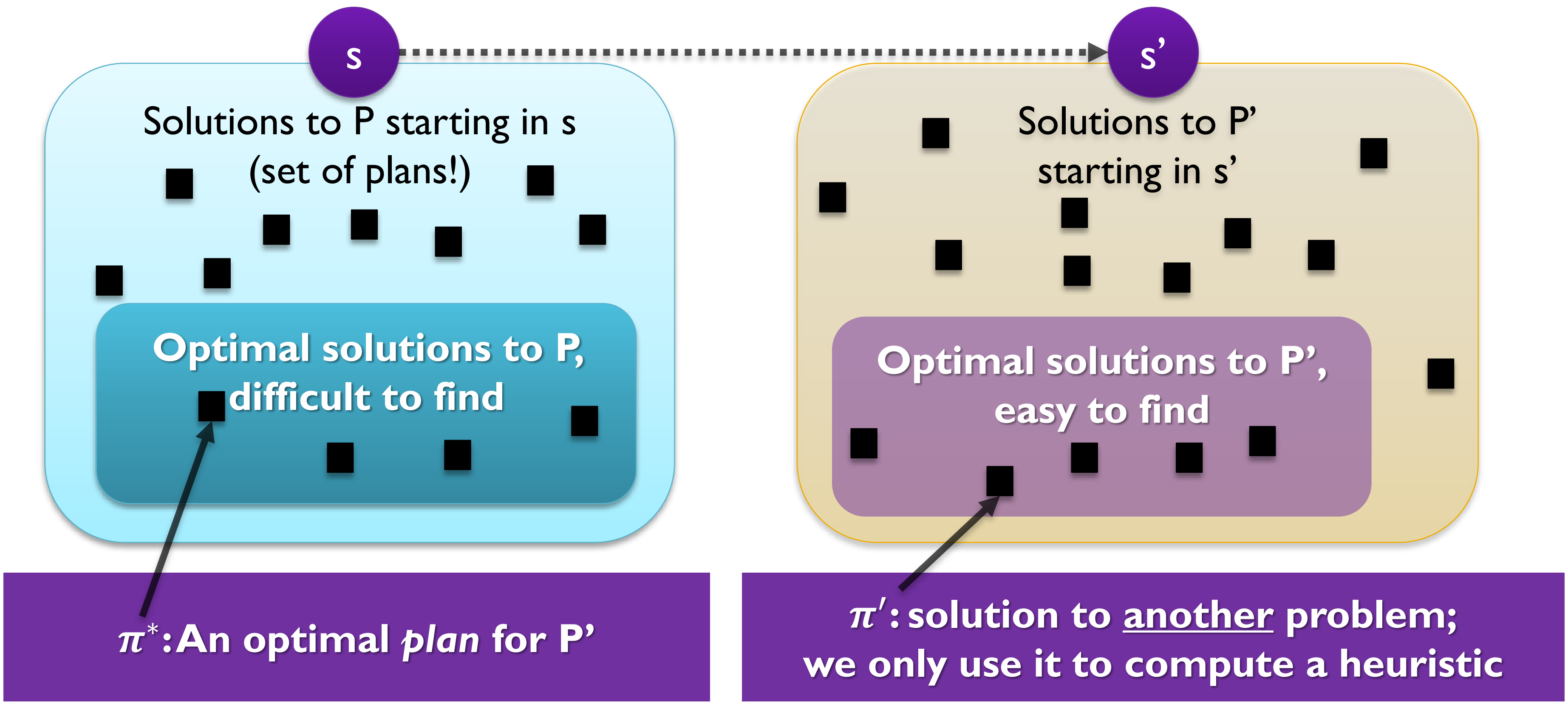
- Let's modify the obvious idea:
  - **Change / transform** P to make it easy (quick) to solve
    - But make sure optimal solutions cannot become more expensive!
    - Example: Add more goal states to S_g → more ways to reach them!

  Relaxation will be **one specific way** of (1) **finding** a simplifying transformation, and (2) **proving** "not-more-expensive"!

  - **Compute** an admissible heuristic:
    - Solve the modified planning problem optimally
    - $h(s)$ = cost of optimal solution for modified problem <= $h^*(s)$ = cost of optimal solution for original problem
    - Definition of admissibility!
  - **Preferably**:
    - Keep $h(s)$ as close as possible to $h^*(s)$ – we want *strong cost information!*

Fundamental Ideas (3)

- More formally:
  - Before planning, **find** a **simpler** problem P', such that in every state s (of P):
    - We can **quickly** transform s into a state s' for P'
    - And we can **quickly** find an optimal solution π' for P' starting in s'
    - And the solution is **never more expensive**: $\text{cost}(\pi') \leq \text{cost}(\pi^*)$

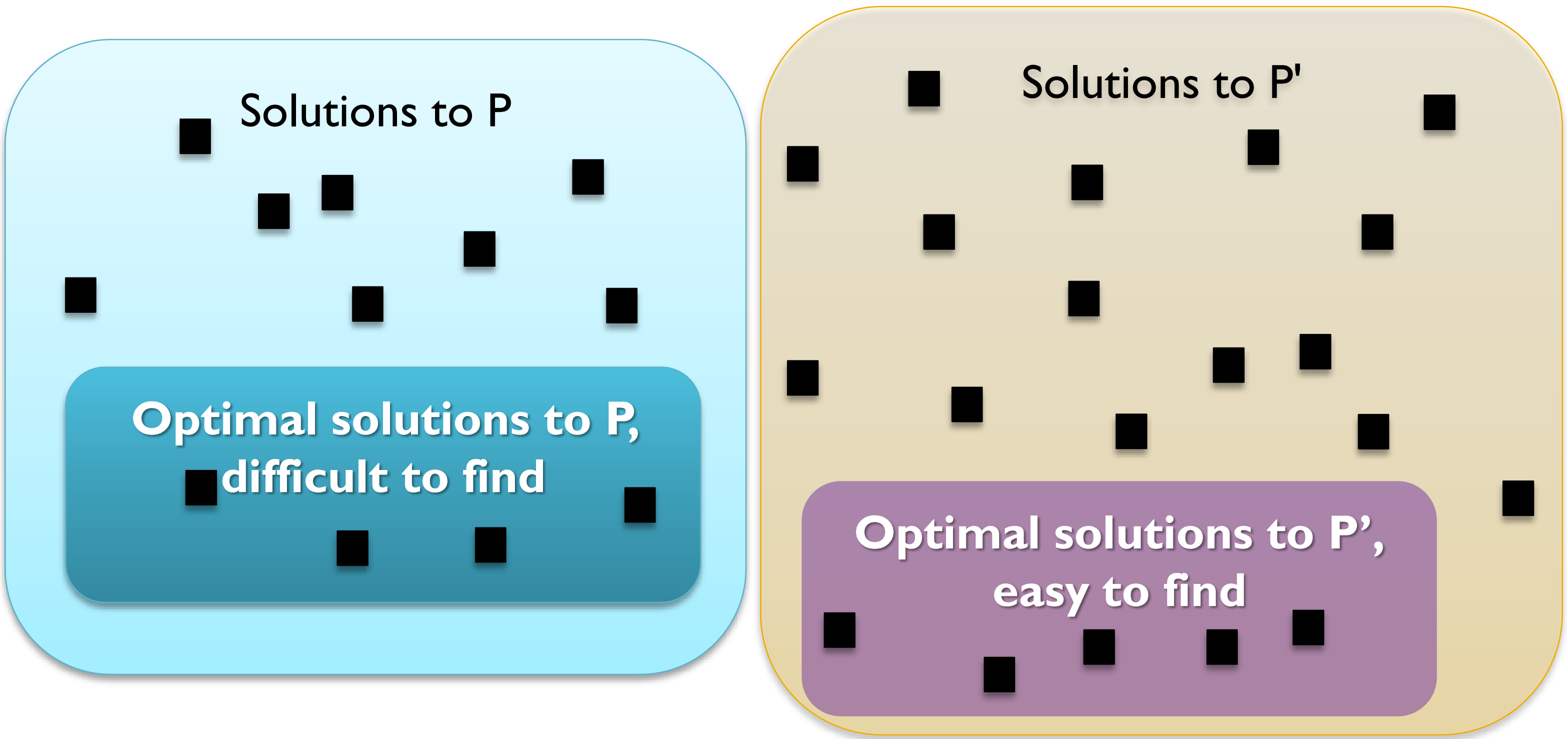


Fundamental Ideas (4)

- **<u>During</u>** planning:
  - Every time we need $h(s)$ for some state s:
    - Transform s to s'
    - **<u>Quickly solve</u>** problem P' **<u>optimally</u>** starting in s', resulting in solution π' – for the *transformed* problem
    - Let $h(s) = \text{cost}(\pi')$
    - Throw away π': It isn't interesting in itself
- We then know:
  - h(s) = cost($\pi'(s)$) = cost(optimal-solution(P')) $\leq$ cost(optimal-solution(P))
  - h(s) is admissible

Fundamental Ideas (5)

- Important:
  - What we **need**: cost(optimal-solution(P')) $\leq$ cost(optimal-solution(P))
  - **Could** use **any** transformation, even with ***completely disjoint*** solution sets, **if** we just have a ***proof*** that optimal solutions to P' are not more expensive

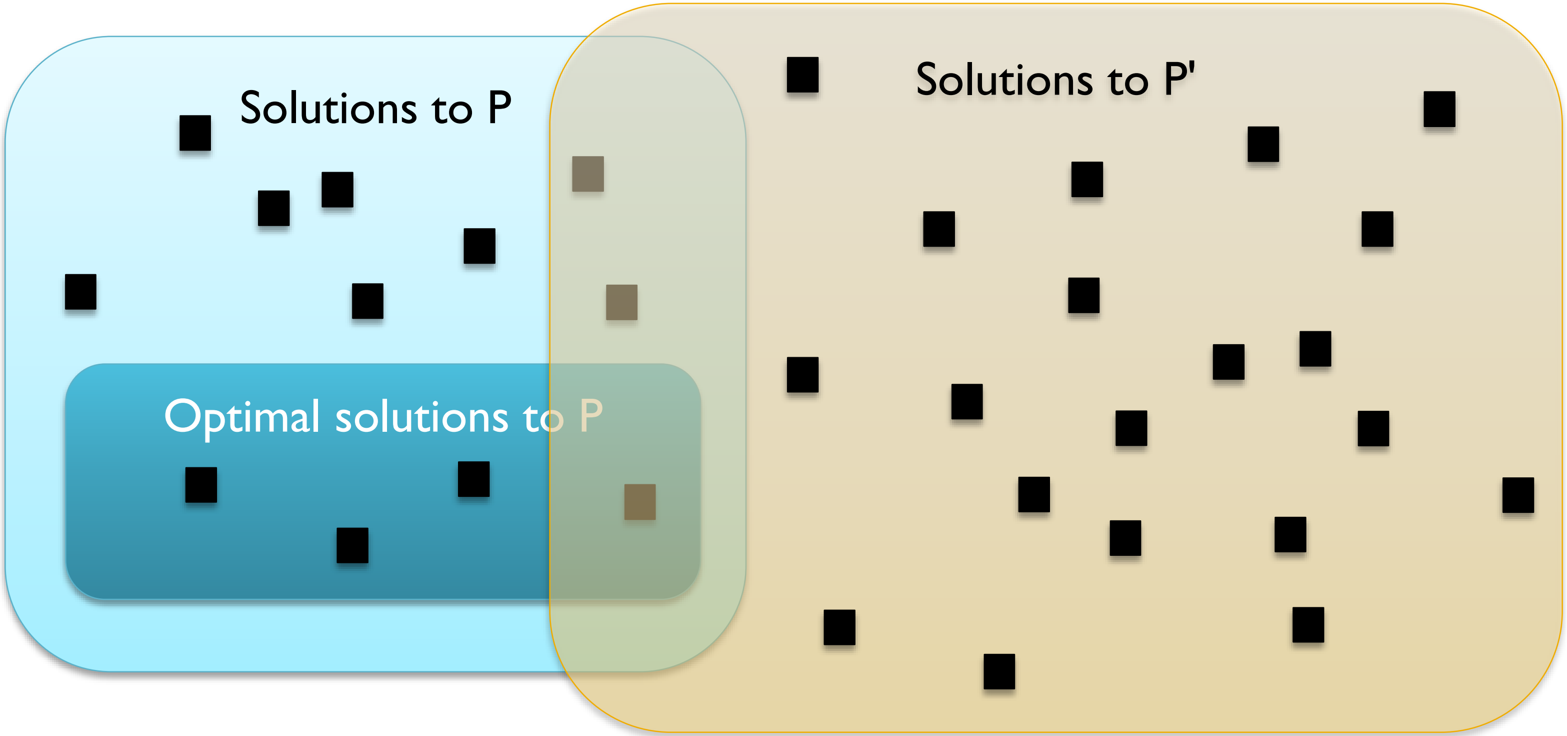


Difficult to find transformations, prove correctness – we need a *method*

Fundamental Ideas (6)

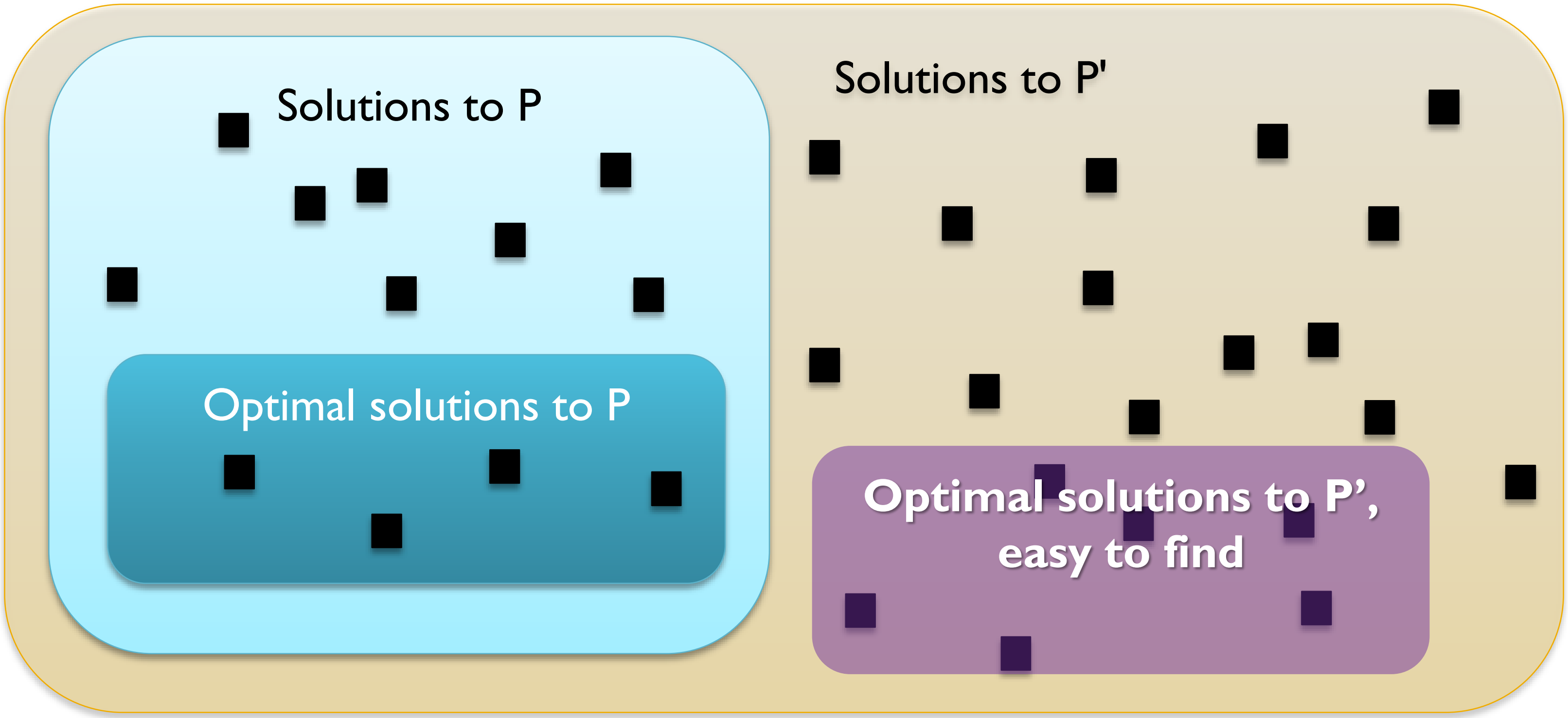
- How to prove cost(optimal-solution(P')) $\leq$ cost(optimal-solution(P))?
  - ***Sufficient*** criterion: **One optimal solution** to P **remains** a solution for P'
    - cost(optimal-solution(P')) = min { cost(π) | π is any solution to P' } <= cost(optimal-solution(P))

Includes the optimal solutions to P, so min {…} cannot be greater



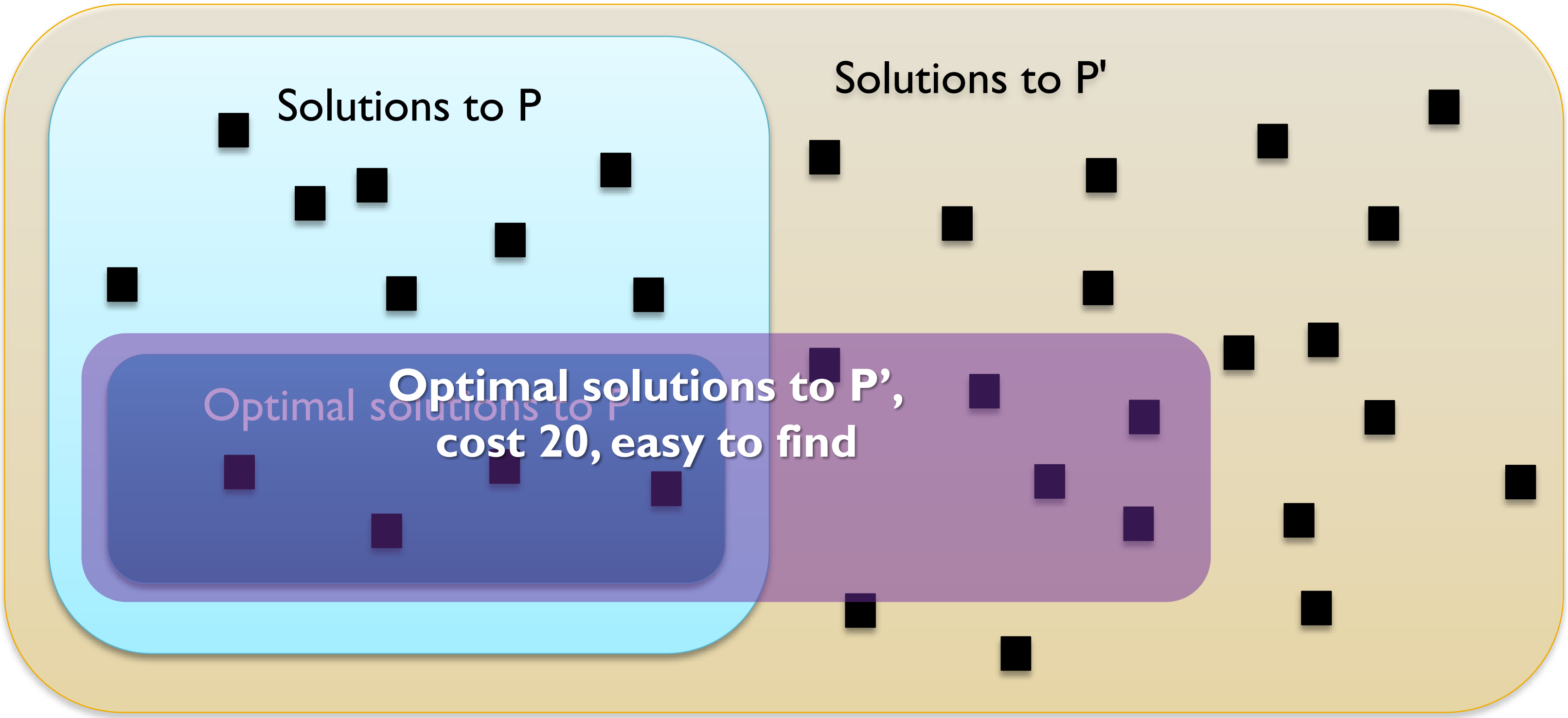
Fundamental Ideas (7)

- Another sufficient criterion: **<u>All solutions</u>** to P **<u>remain</u>** solutions for P'
  - Stronger, but often **<u>easier to prove</u>**
  - **<u>This</u>** is called **<u>relaxation</u>**: P' is a relaxed version of P
  - **<u>Relaxes</u>** the constraint on what is accepted as a solution:
    The **is-solution(plan)?** test is "expanded, relaxed" to cover additional plans



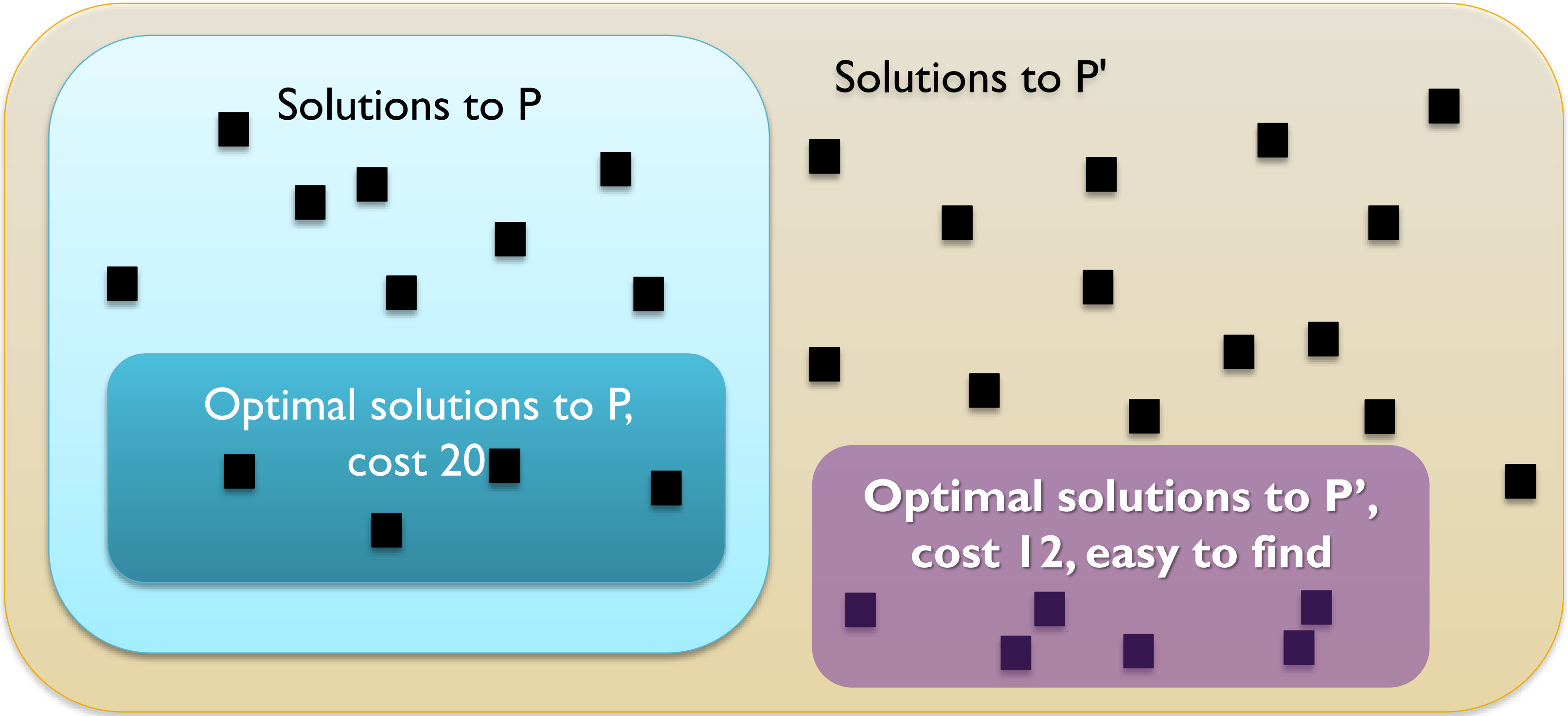
Fundamental Ideas (8)

- Case 1: P' has identical cost (for some starting state s)
  - Unlikely!



Fundamental Ideas (9)

- Case 2: P' has lower cost (for some starting state s)



Relaxation:
Definition and Examples

Relaxation for Planning Problems

- A classical planning problem $P = (\Sigma, s_0, S_g)$ has a **<u>set of solutions</u>**
  - *Solutions*(P) = { π : π is an executable action sequence leading from s_0 to some state in S_g }
- Suppose that:
  - $P = (\Sigma, s_0, S_g)$ is a classical planning problem
  - $P' = (\Sigma', s_0', S_g')$ is another classical planning problem
  - *Solutions*(P) $\subseteq$ *Solutions*(P')
- Then (and only then): P' is a relaxation of P

Solutions for P:

Sol1, cost 10 — Optimal in P
Sol2, cost 12
Sol3, cost 27

Solutions for P':

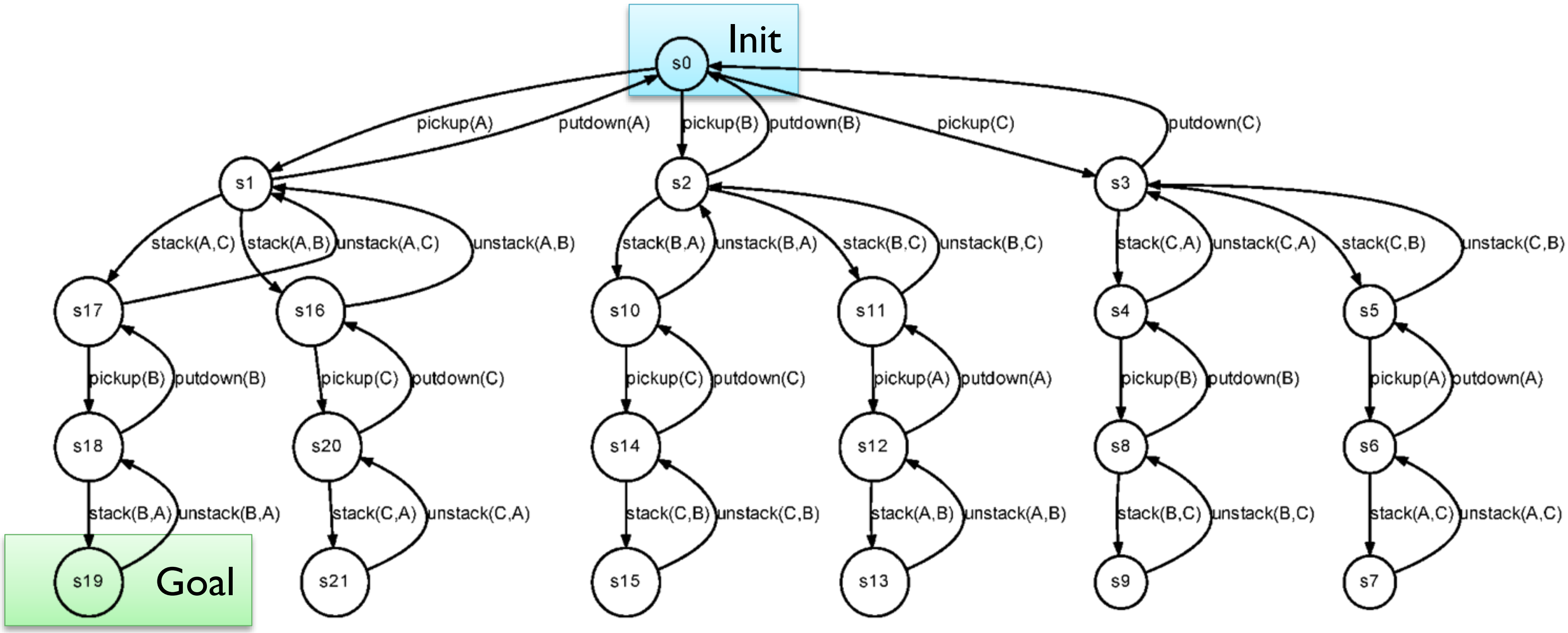
Sol1, cost 10
Sol2, cost 12
Sol3, cost 27
Sol4, cost 8
Sol5, cost 42

All old solutions remain solutions!

Now **sol4** is optimal

Relaxation Example: Basis

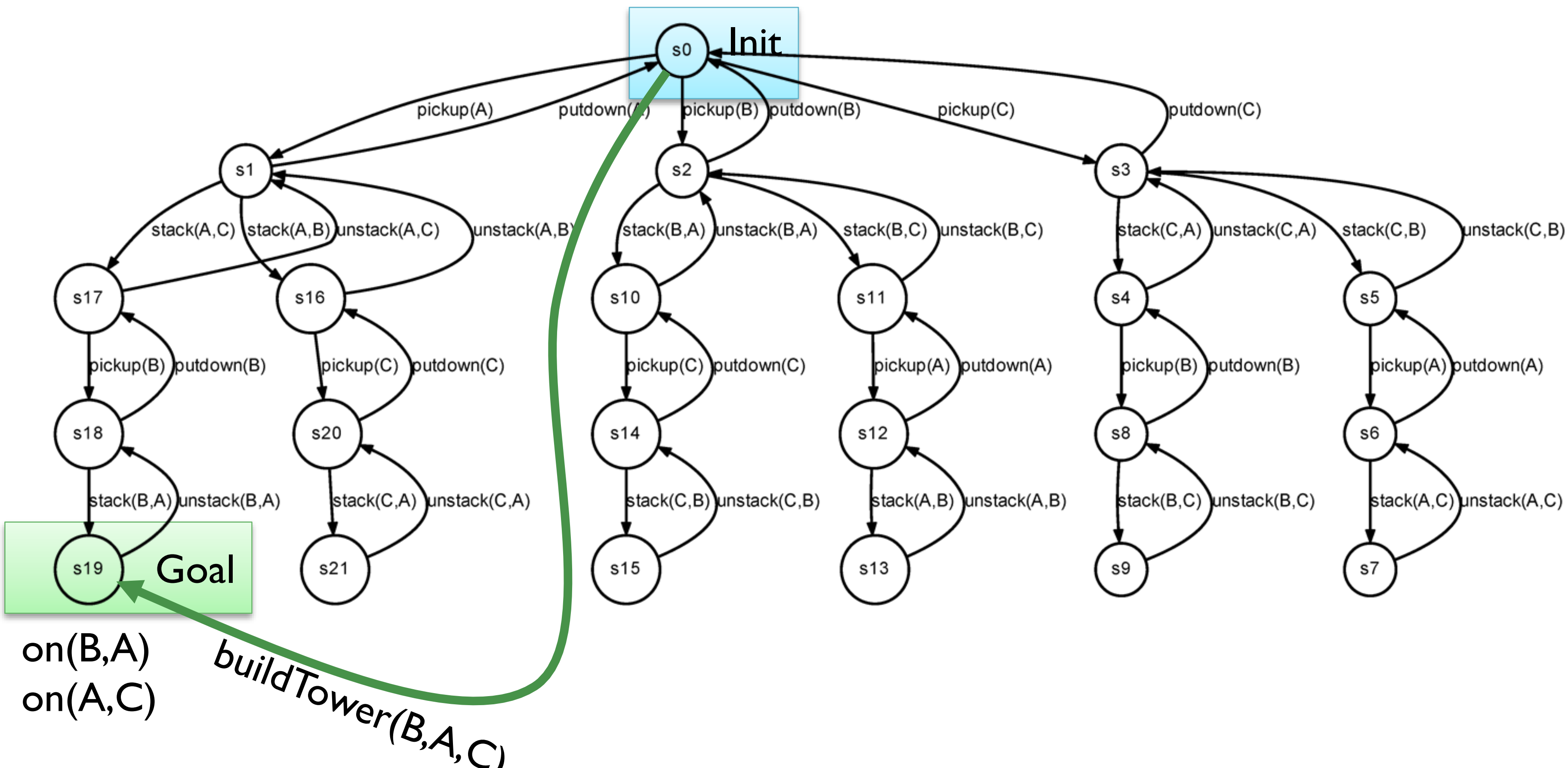
- A simple planning problem (domain + instance)
  - Blocks world, 3 blocks
  - Initially all blocks on the table
  - Goal: (and (on B A) (on A C)) (only satisfied in s19)
  - Solutions: **<u>All</u>** paths from init to goal (infinitely many – can have cycles)



Relaxation Example 1

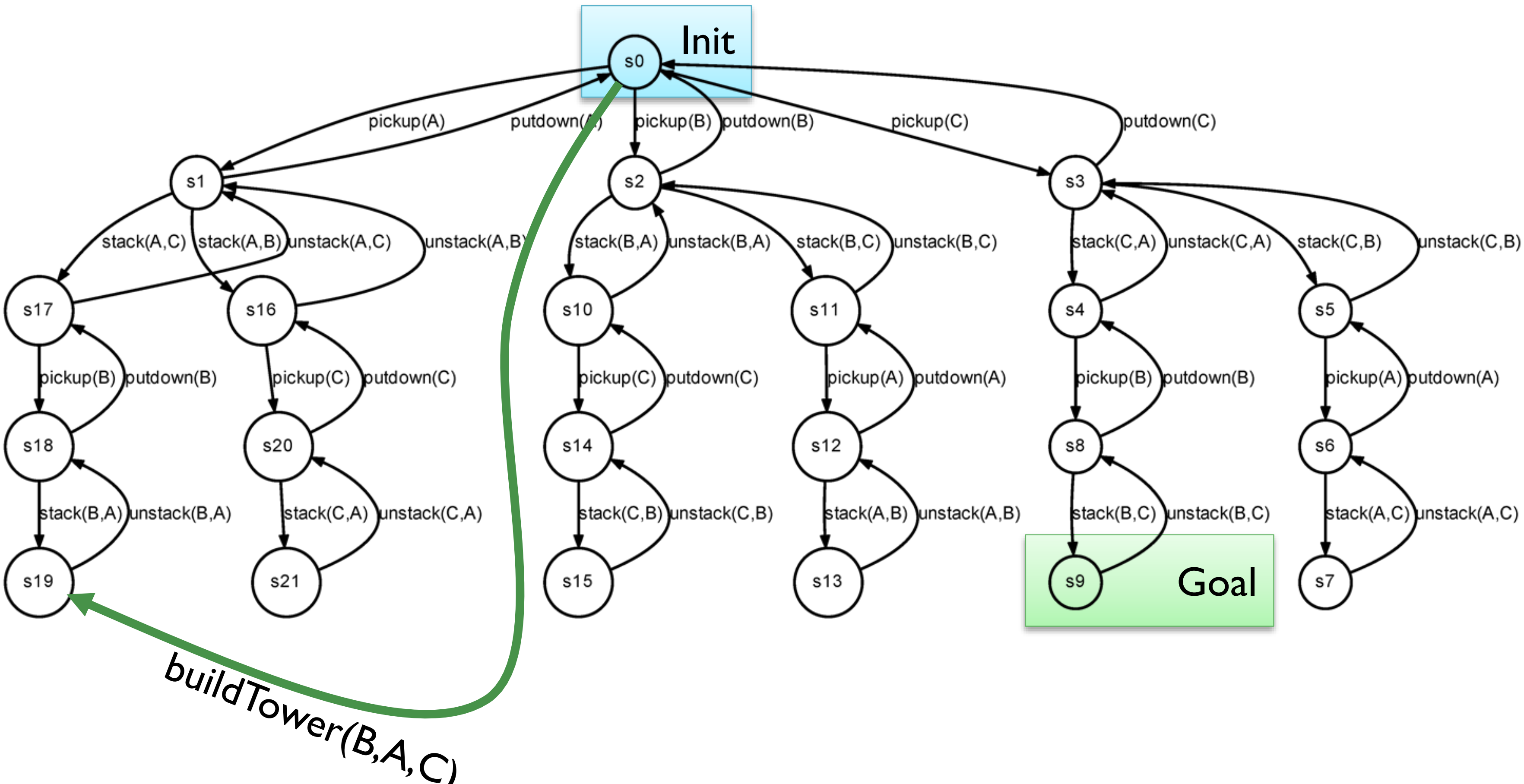
jonkv@ida

- Example 1: **<u>Adding new actions</u>**
  - All old solutions still valid, but new solutions may exist
  - Modifies the STS by **<u>adding new edges / transitions</u>**
  - This particular example: *shorter* solution appears



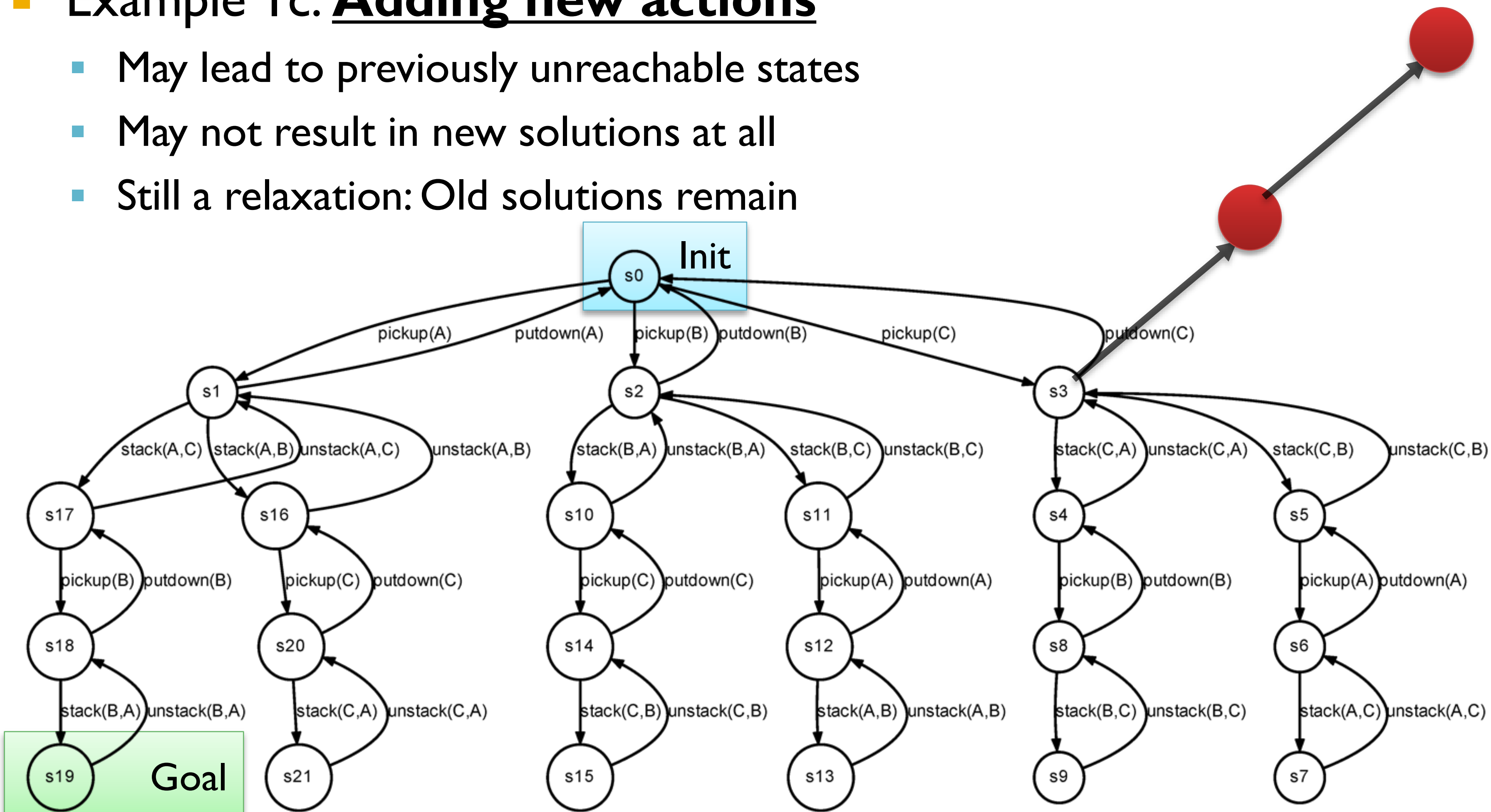
Relaxation Example 1b

- Example 1b: **Adding new actions**
  - In other cases, the new actions may not "help"
  - New solutions ($s_0 \rightarrow s_{19} \rightarrow s_9$) are *longer* as well as *more expensive*
  - Still a relaxation!



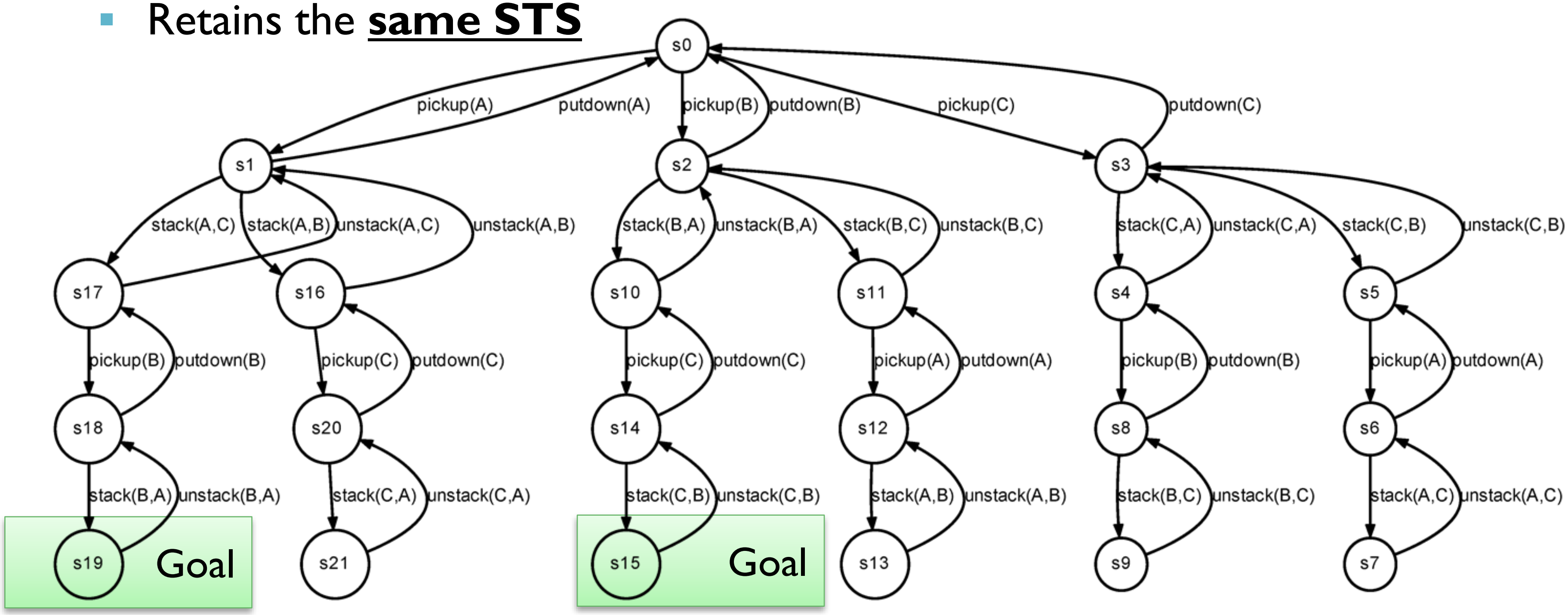
Relaxation Example 1c

- Example 1c: **<u>Adding new actions</u>**
  - May lead to previously unreachable states
  - May not result in new solutions at all
  - Still a relaxation: Old solutions remain



Relaxation Example 2

- Example 2: **Adding goal states**
  - New goal formula: (and (on B A) **(or (on A C) (on C B))**)
  - All old solutions still valid, but new solutions may exist
  - This particular example: Optimal solution **from s_0** retains the same length
  - Retains the **same STS**

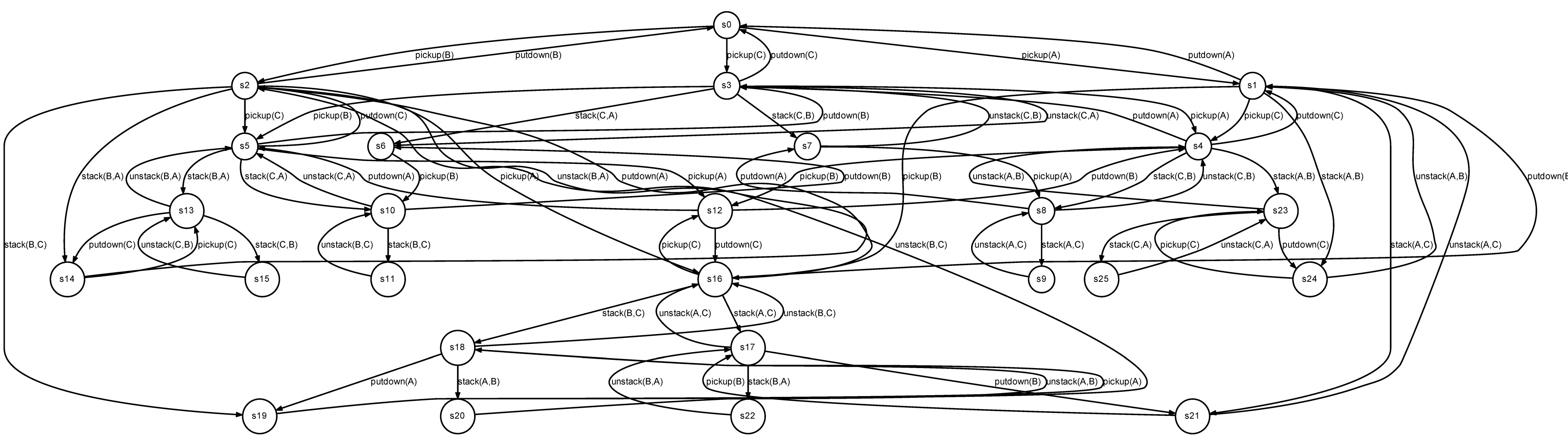


on(B,A)
on(A,C) or on(C,B)

on(B,A)
on(A,C) or on(C,B)

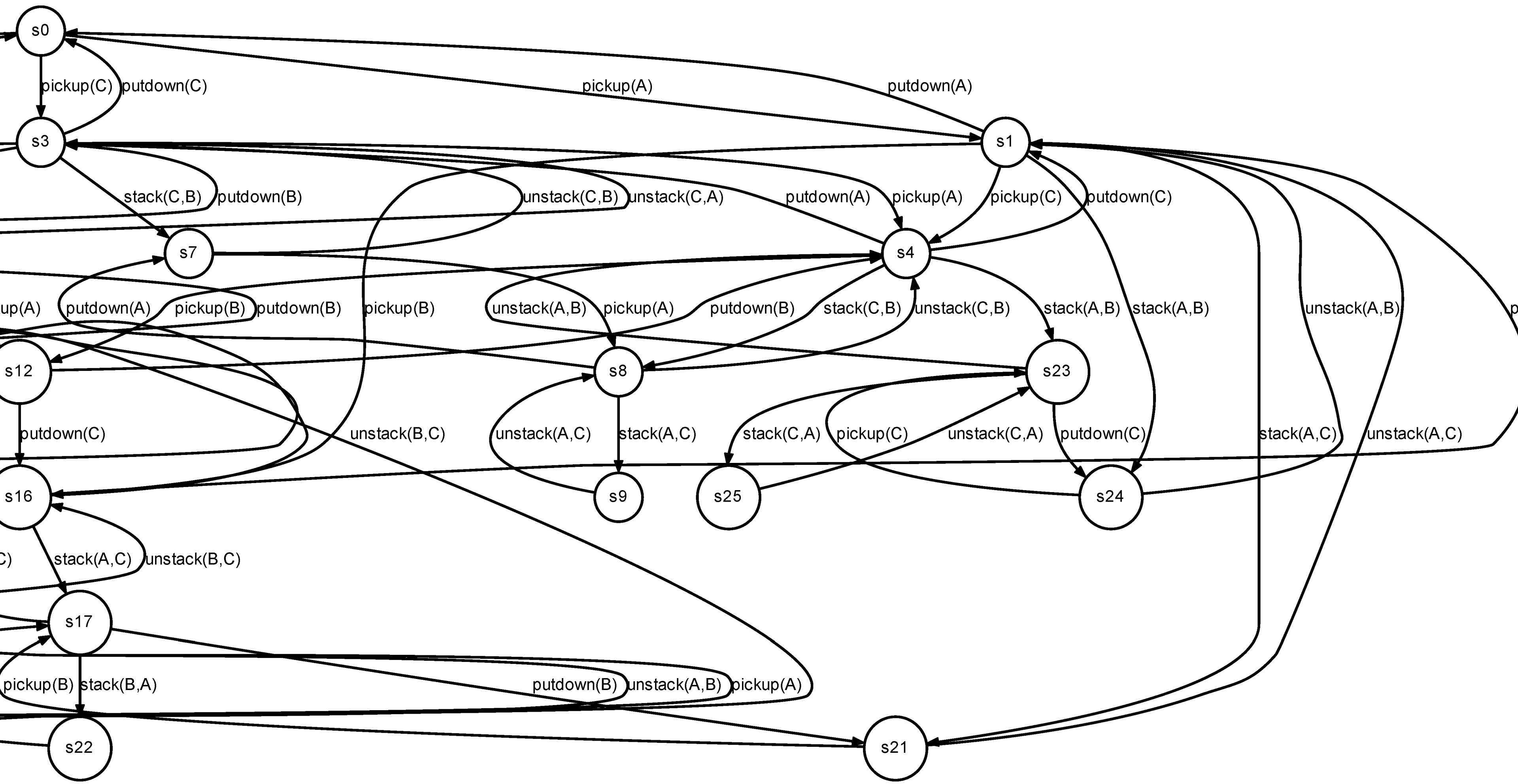
Relaxation Example 3

- Example 3: **Ignoring** state variables
  - Ignore the *handempty* fact in preconditions and effects
  - **Different** state space, no simple addition or removal,
    but all the old solutions (action sequences)
    lead from s_0' to new goal states in s_g'!
    - 22 reachable states ➔ 26
    - 42 transitions ➔ 72



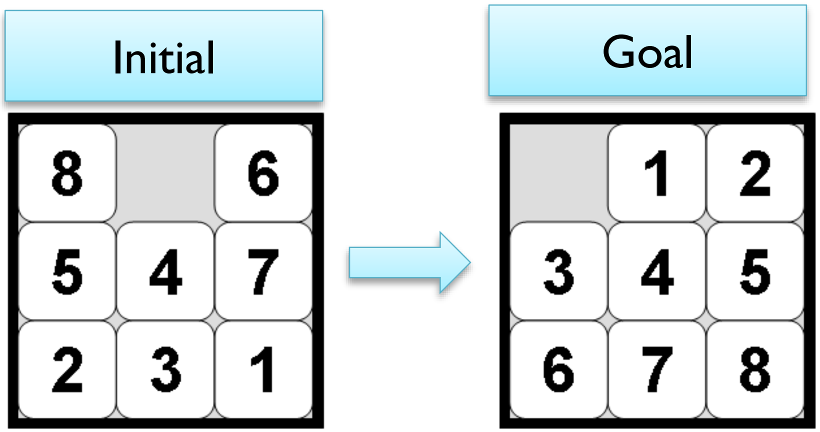
Relaxation Example 3b

- Example 3, enlarged



Relaxation Example 4

- Example 4: **Weakening preconditions** of existing actions



Possible first moves:
Move 8 right
Move 4 up
Move 6 left

- Precondition relaxation: **Tiles can be moved across each other**
  - Now we have 21 possible first moves: **New transitions** added to the STS
- All **old solutions are still valid**, but new ones are added
  - To move "8" into place:
  - Two steps to the right, two steps down, ends up in the same place as "1"

Can still be **solved** through **search**
The **optimal** solution for the *relaxed 8-puzzle*
can **never** be more expensive than the optimal solution for *original 8-puzzle*

Essentially the same as adding actions: Results in new transitions!

Relaxation Heuristics: Summary

- **Relaxation**: **One general principle** for designing **admissible** heuristics for **optimal** planning
  - Find a way of transforming planning problems, so that given a problem instance P:
    - **Computing its transformation** P' is easy (polynomial)
    - **Finding an optimal solution** to P' is easier than for P
    - **All solutions to P are solutions to P'**, but the new problem can have additional solutions as well
  - Then the cost of an optimal solution to P' is an admissible heuristic for the original problem P

This is only *one* principle!
There are others, *not* based on relaxation

Relaxation:
Search or Direct Computation?

Search or Direct Computation (1)

- As stated:
  - Compute an actual solution π' for the relaxed problem P'
  - Compute cost(π')

- Example: The **8-puzzle**…
  - Ignore **blank(x,y)** in preconditions and effects
  - Run the problem through an optimal planner
  - Compute the cost of the resulting plan π'

```
(:action move-up
    :parameters (?t ?px ?py ?by)
    :precondition (and
                    (tile ?t) (position ?px) (position ?py) (position ?by)
                    (dec ?by ?py) (blank ?px ?by) (at ?t ?px ?py))
    :effect (and (not (blank ?px ?by)) (not (at ?t ?px ?py))
                 (blank ?px ?py) (at ?t ?px ?by)))
```

Search or Direct Computation (2)

- But we only use π' to compute its cost!
  - Let's **analyze** the problem…
    - Each piece has to be moved to the intended row
    - Each piece has to be moved to the intended column
    - These are **exactly** the required actions given the relaxation!
  - ➔ **optimal cost** for relaxed problem = sum of Manhattan distances
  - ➔ **admissible heuristic** for *original* problem = sum of Manhattan distances
  - ➔ **Cost** of any optimal solution π' can be computed efficiently *without* π':

$$\sum_{p \in pieces} xdistance(p) + ydistance(p)$$

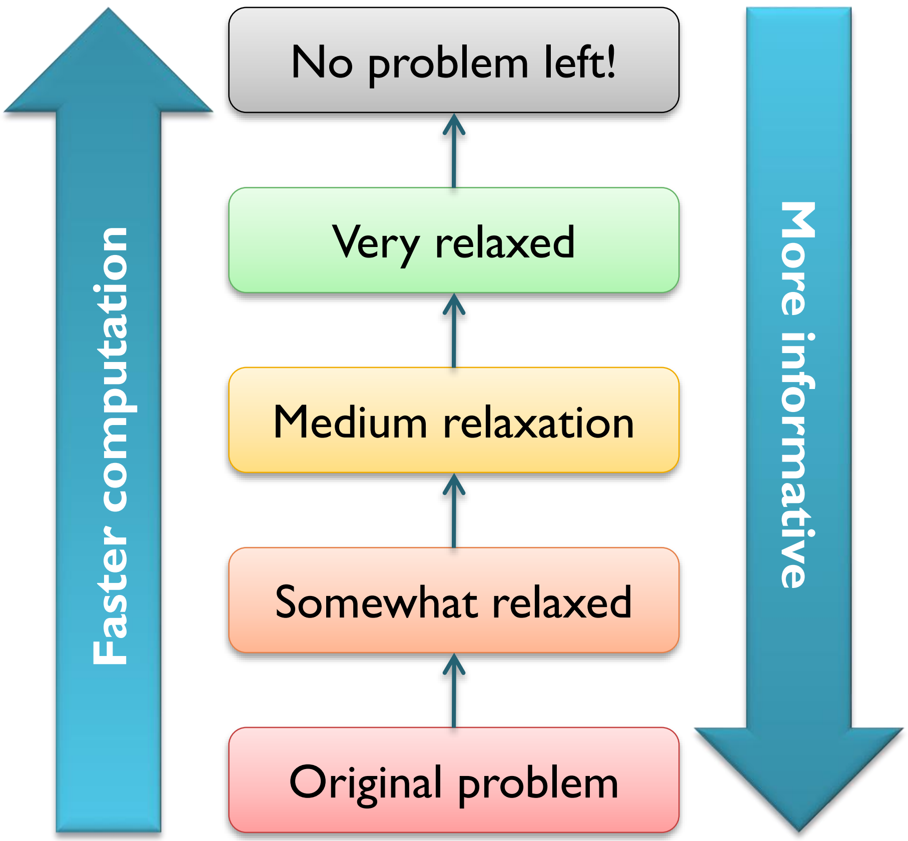
But now we had to **analyze** the problem:
(1) Decide to ignore "blank"
(2) Find "sum of manhattan distances"

Soon: How do we *automatically* find good relaxations + computation methods?

Relaxation: Essential Facts

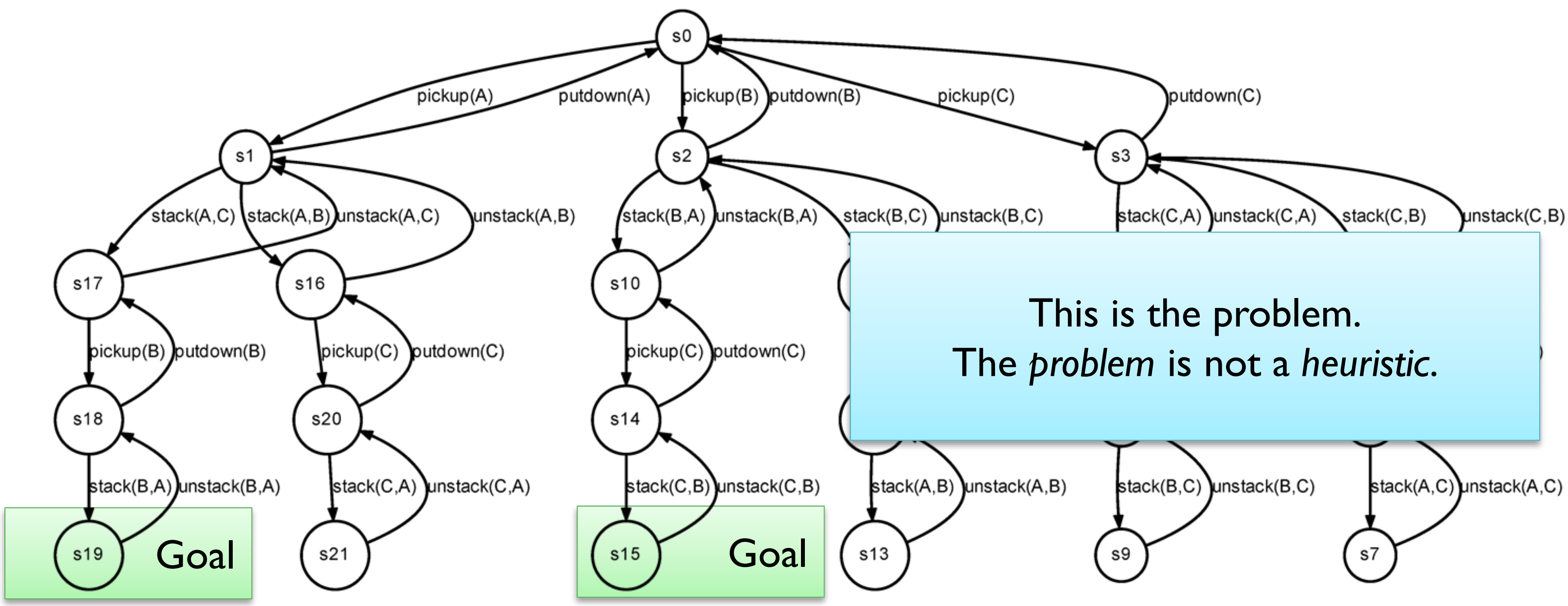
Relaxation Heuristics: Balance

- The **reason** for relaxation is **rapid calculation**
  - **Shorter solutions** are an *unfortunate side effect*: Leads to less informative heuristics
  - Relax too much ➔ not informative
    - Example: Any piece can teleport into the desired position ➔ $h(n)$ = *number* of pieces left to move



Relaxation Heuristics: Important Issues!

You **cannot** "use a relaxed problem as a heuristic".
What would that mean?
You use the **cost** of an **optimal** **solution** to the relaxed problem as a heuristic.



This is the problem.
The *problem* is not a *heuristic*.

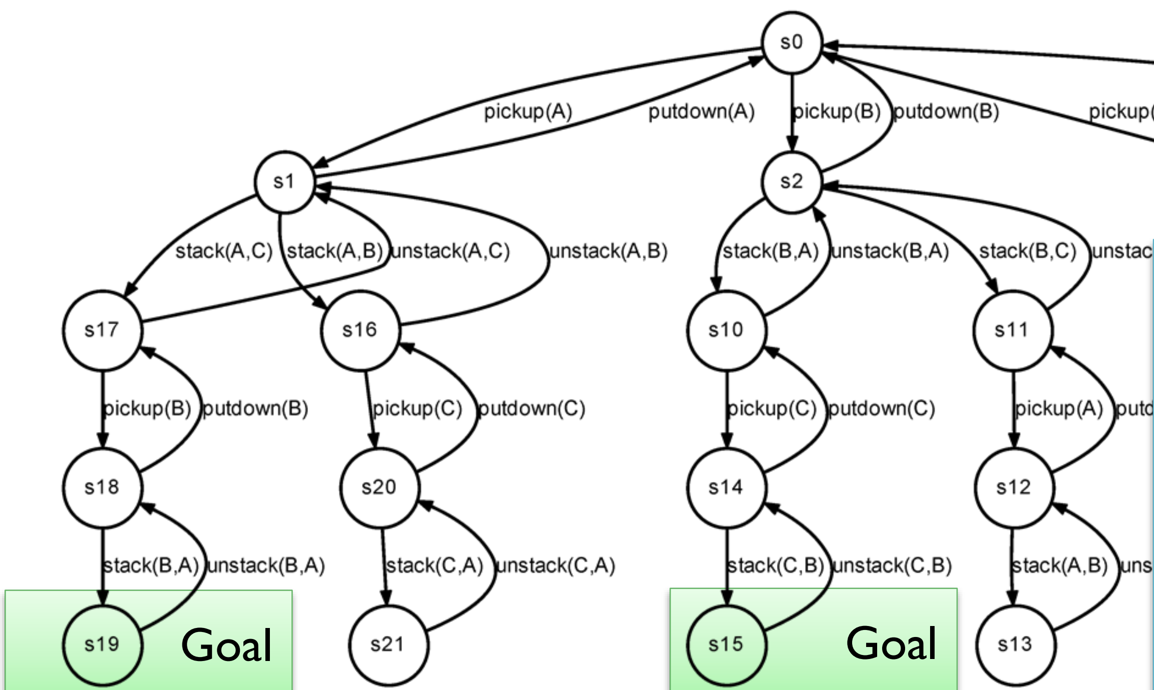
on(B,A)
on(A,C) or on(C,B)

on(B,A)
on(A,C) or on(C,B)

Relaxation Heuristics: Important Issues!

<u>Solving</u> the relaxed problem
<u>can</u> result in a more expensive solution
➔ inadmissible!

You have to solve it <u>optimally</u> to get the admissibility guarantee.



on(B,A)
on(A,C) or on(C,B)

on(B,A)
on(A,C) or on(C,B)

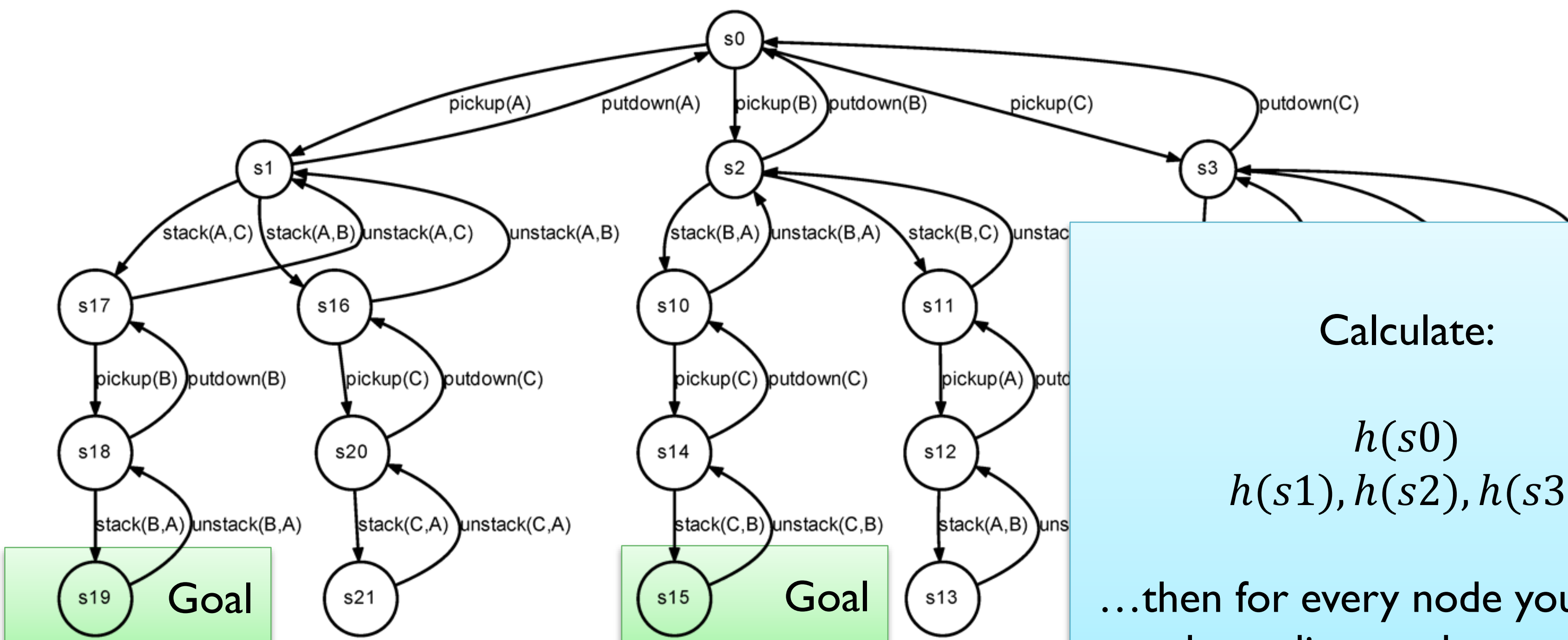
One solution to relaxed problem:

pickup(C)
putdown(C)
pickup(B)
stack(B,A)
pickup(C)
stack(C,B)

Relaxation Heuristics: Important Issues!

jonkv@ida

You don't just solve the relaxed problem once.
Every time you reach a new state and want to calculate a heuristic,
you have to solve the relaxed problem
of getting from **<u>that</u>** state to the goal.



on(B,A)
on(A,C) or on(C,B)

on(B,A)
on(A,C) or on(C,B)

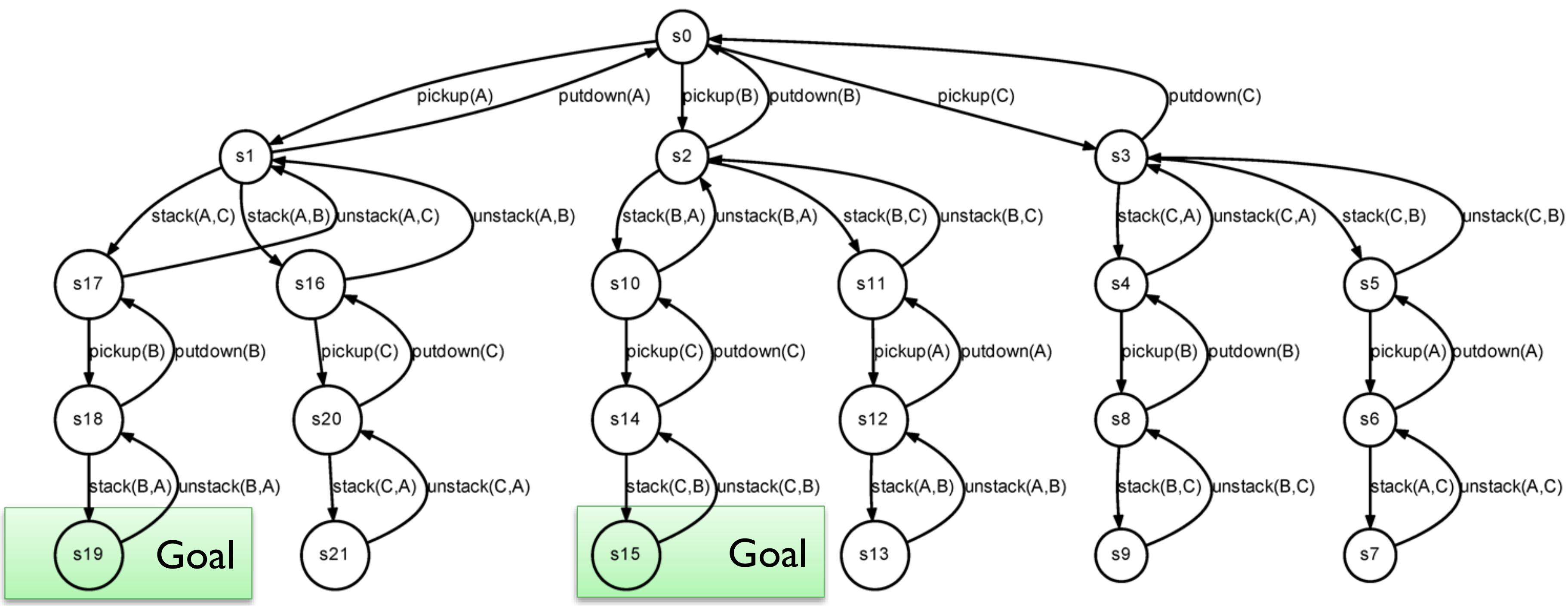
Calculate:

$h(s0)$
$h(s1), h(s2), h(s3)$

…then for every node you create, depending on the strategy

Relaxation Heuristics: Important Issues!

Relaxation does **not** always mean "**removing constraints**"
in the sense of *weakening preconditions* (moving across tiles, removing walls, …)
Sometimes we get new *goals*. Sometimes the entire *state space* is transformed.
Sometimes action *effects* are modified, or some other change is made.
What defines relaxation: **All old solutions are valid, new solutions may exist**.



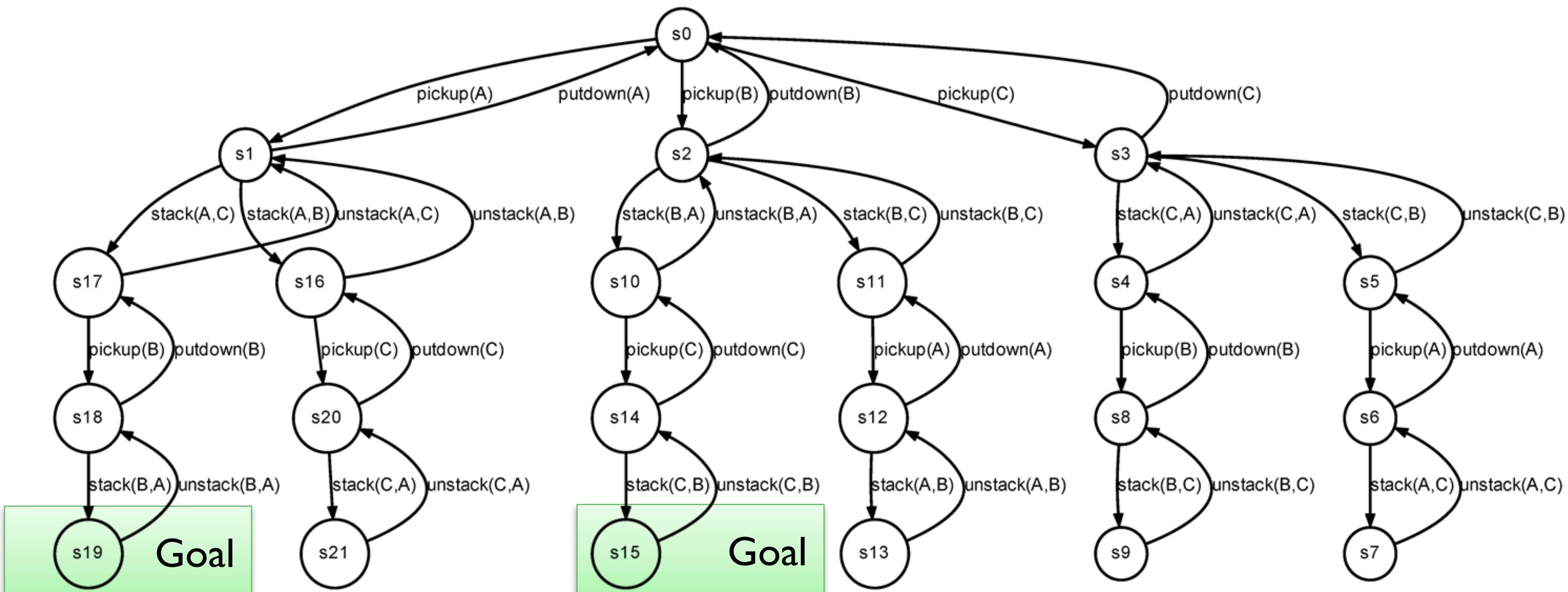
on(B,A)
on(A,C) or on(C,B)

on(B,A)
on(A,C) or on(C,B)

Admissibility: Important Issues!

Relaxation is useful for finding **admissible heuristics**.

A heuristic cannot be **admissible for some states**.
Admissible == does not overestimate costs for *any* state!



on(B,A)
on(A,C) or on(C,B)

on(B,A)
on(A,C) or on(C,B)

Admissibility: Important Issues!

If you are asked "why is a relaxation heuristic admissible?", don't answer "because it cannot overestimate costs". This is the *definition* of admissibility!

"Why is it admissible?" == "*Why* can't it overestimate costs?"

Admissible heuristics *can* "lead you astray" and you *can* "visit" suboptimal solutions.

But with the right search strategy, such as A*,
the planner will eventually get around to finding an optimal solution.
This is not the case with A* + non-admissible heuristics.