Optimal Metric Planning with State Sets in Automata Representation [3]

Björn Ulrich Borowsky Stefan Edelkamp

Fakultät für Informatik,
Technische Universität Dortmund, Germany

2008

(Slides by Alex Klinkhamer)

Goal and Method

- **Goal:** Algorithm which finds optimal plan for infinite state planning problem.
  - Allow variables to span $\mathbb{Z}$ (integers).
- Represent (possibly infinite) state sets with DFAs.
- Search using state sets instead of individual states.
  - Breadth-first search (BFS).
  - Dijkstra's shortest path algorithm (DijkstraSearch).
  - Similar algorithms exist using BDDs to represent state sets [4].

Ex: PDDL with Integer Domains

```
(define (domain Boxes)
  (:types Block Box - Object)
  (:predicates (in ?b - Block ?x - Box))
  (:functions (weight ?o - Object)
              (num ?x - Box))
  (:action move
   :parameters (?bl - Block ?from ?to - Box)
   :precondition (in ?bl ?from)
   :effect
   (and (in ?bl ?to)
        (not (in ?bl ?from))
        (decrease (num ?from) 1)
        (increase (num ?to) 1)
        (decrease (weight ?from) (weight ?bl))
        (increase (weight ?to) (weight ?bl)))))
```

Planning Problem and Cost Models

Uniform Cost

$\mathcal{R} = (\mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{G})$

- $\mathcal{S}$: State space.
- $\mathcal{A}$: Actions.
- $\mathcal{I}$: Initial states. $\mathcal{I} \subseteq \mathcal{S}$ and usually $|\mathcal{I}| = 1$.
- $\mathcal{G}$: Goal states. $\mathcal{G} \subseteq \mathcal{S}$.

Additive Cost

$\mathcal{R} = (\mathcal{S}, \mathcal{A}, \mathcal{C}, \mathcal{I}, \mathcal{G})$

- $\mathcal{C}$: Action costs. $\mathcal{C} = \{c_A \,|\, A \in \mathcal{A}\}$ and $c_A : \mathcal{S} \rightarrow \mathbb{Z}^+$.

Metric Cost

$\mathcal{R} = (\mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{G}, m)$

- m: Metric cost function. $m : \mathcal{S} \cup \{\perp\} \rightarrow \mathbb{Z}$.
- $m(\perp)$ is *total-cost*.

Representing State Sets

- A state is a valuation of variables.
  - In PDDL, these are predicates and functions on objects.
- Can represent a set of states with a predicate formula.
  - Ex: $x = 1 \vee y = 0$ represents 3 states of 2 bits x and y.
  - Equivalently:
    $(x = 0 \wedge y = 0) \vee (x = 1 \wedge y = 0) \vee (x = 1 \wedge y = 1)$.
- A predicate formula can be represented by...
  - Binary decision diagram (BDD) if all variables are Boolean.
  - Multi-valued decision diagram (MDD) if all variables are finite.
  - But what about infinite domains like $\mathbb{Z}$?

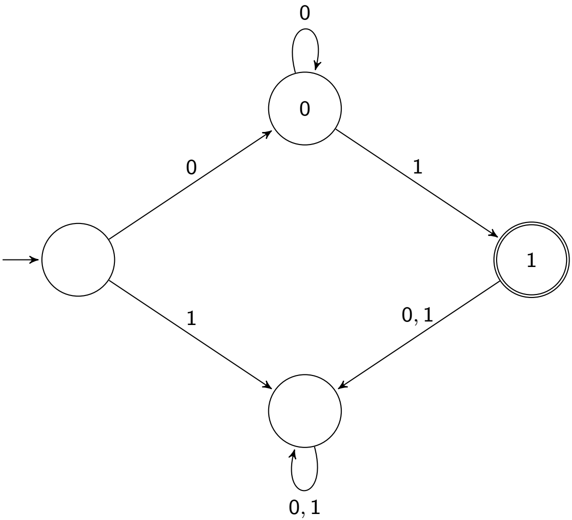
Presburger Arithmetic

- A decidable first-order theory of integers with addition.
  - $73x - 52y + 30z \leq 778 - u$
  - $\forall x : ((\exists k : 2k = x) \vee (\exists k : 2k + 1 = x))$
  - $(\exists y : x = 3y) \wedge (\exists y : x = 7y)$
- Can express these formulas as DFAs.
  - Give an order to the n variables involved.
  - Alphabet is $\Sigma = \{0, 1\}^n$ (bit vectors of length n).
  - Transitions read one bit of each variable at a time.
  - Example:
    - Let Φ be some formula involving x, y, and z.
    - Consider the valuation $x = 5$, $y = 2$, and $z = -2$.
    - In binary: $x = 0101_2$, $y = 0010_2$, and $z = 1110_2$.
    - The DFA for Φ, with variable order (x, y, z) and alphabet $\Sigma = \{0, 1\}^3$, would read the string of length 4:

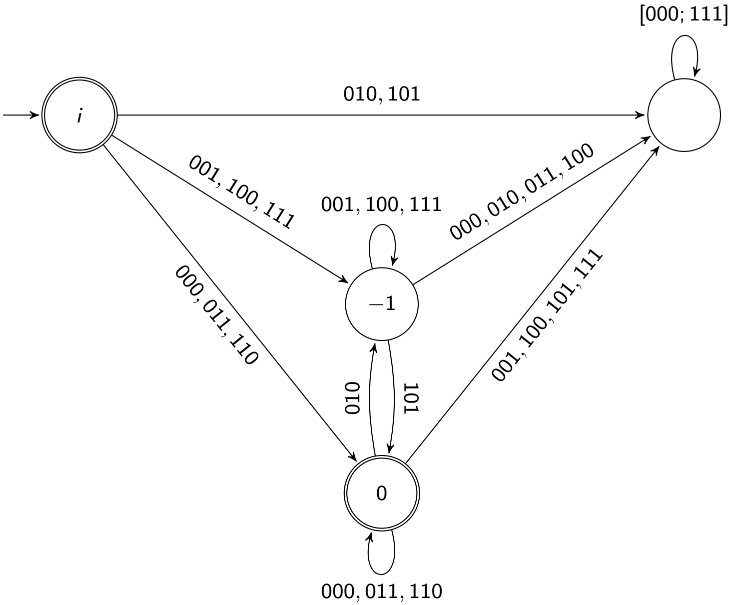
$$(001, 101, 011, 100)$$

.

Ex: DFA for $x = 1$



Ex: DFA for $w = x - y$ with variable order (y, x, w)



Representing Transition Sets (Actions)

- Introduce primed variables for destination variables.
- Example:
  - Consider an action A.

    Precondition: $y < 20$.
    Effect: $x := x - y$.
  - As a formula: $A = (y < 20) \wedge (x' = x - y) \wedge (y' = y)$

State Set Operations

- Formula conjunction by DFA intersection.
- Formula disjunction by DFA union.
- Formula negation by DFA complement.
- Variable substitution by reordering bits of transition symbols.
  - Change unprimed variables to primed.
  - Change primed variables to unprimed.
- Existential quantification by DFA projection.
  - An efficient implementation is described in [2].
- Unique representation by minimized DFA.

Image

- The *image* function finds all states which transitions T map from S.
  - $image(T, S) = \{s_1 \in \mathcal{S} \mid \exists s_0 \in S : (s_0, s_1) \in T\}$
- Compute using standard operations.
  - Conjunct the formulas of S and T.
  - Project out unprimed variables with existential quantification.
  - Change primed variables to unprimed.
- Example:
  - Let action $A = (y < 20) \wedge (x' = x - y) \wedge (y' = y)$.
  - $image(A, (y > 3) \wedge (x = 2))$
    $= unprime(\exists x, y : (y > 3) \wedge (x = 2) \wedge A)$
    $= unprime((y' > 3) \wedge (y' < 20) \wedge (x' = 2 - y'))$
    $= (y > 3) \wedge (y < 20) \wedge (x = 2 - y)$

Preimage

- The *preimage* function finds all states which are mapped to states in S by transitions T.
  - $preimage(T, S) = \{s_0 \in \mathcal{S} \mid \exists s_1 \in S : (s_0, s_1) \in T\}$
- Compute using standard operations.
  - Prime all variables in formula of S.
  - Conjunct with formula of T.
  - Project out primed variables with existential quantification.
- Example:
  - Let action $A = (y < 20) \wedge (x' = x - y) \wedge (y' = y)$.
  - $preimage(A, (y > 3) \wedge (x = 2))$
    $= \exists x', y' : prime((y > 3) \wedge (x = 2)) \wedge A$
    $= \exists x', y' : (y' > 3) \wedge (x' = 2) \wedge A$
    $= (y > 3) \wedge (y < 20) \wedge (x = y + 2)$

Idea for BFS

- Start with initial states I as the frontier.
- Initialize stack of explored state sets with one element I.
- While the frontier does not contain a goal state,
  - Find the next frontier by applying each action to the current frontier.
  - If the next frontier does not contain any new states, return failure.
  - Otherwise, assign the current frontier to be the next and push it onto the stack.
- Build plan by working back through layers (stack) of explored state sets.

Note: This finds a plan for *just one* initial state.

BFS to Solve Uniform Cost Instance

Input: Planning task $\mathcal{R} = (\mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{G})$, set I with $\emptyset \neq I \subseteq \mathcal{I}$
Output: A shortest sequential plan and a set $I' \subseteq I$ or "No plan."
$L := R := stack := I$;
while $L \cap \mathcal{G} = \emptyset$ **do**
    $L' := \emptyset$;
    forall the $A \in \mathcal{A}$ **do**
        $L' := L' \cup image(A, L)$;
    if $L' \subseteq R$ **then**
        return "No plan.";
    $R := R \cup L'$;
    $stack.push(L')$;
    $L := L'$;
$G := stack.pop() \cap \mathcal{G}$;
return $ExtractPlan(\mathcal{R}, stack, G)$;

Algorithm 1: *BFS*

Extract Plan from Explored Layers

Input: Planning task $\mathcal{R} = (\mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{G})$, stack *stack*, $G \subseteq \mathcal{G}$
Output: Sequential plan of length $|stack|$, $I' \subseteq \mathcal{I}$

$\pi := []$; `// Initialize empty sequence.`
$post := G$;
while $stack \neq \emptyset$ **do**
  $pre := stack.pop()$;
  $post' := \emptyset$;
  forall the $A \in \mathcal{A}$ **do**
    $post' := preimage(A, post) \cap pre$;
    if $post' \neq \emptyset$ **then**
      $a := A$;
      break;
  $post := post'$;
  $\pi := [a].\pi$; `// Push action onto front of sequence.`
return $(\pi, post)$;

Algorithm 2: *ExtractPlan*

BFS Analysis

- Does find the optimal solution.
- Terminates when a plan exists (eventually reaches goal states).
- Terminates when state space is finite (no room to grow).
- Does not terminate when all of the following hold...
  - No plan exists.
  - Infinite number of reachable states.
  - Finite number of initial states.
- Otherwise, no general rule for termination.

Optimal Solutions for Other Cost Models

- Additive cost model.
  - Reduces to weighted shortest path problem.
  - Adapt Dijkstra's shortest path algorithm to work with state sets.
  - Same terminating conditions as BFS.
- Metric cost model.
  - Adapt BFS to work with metric costs (MetricBFS).
  - Finds a plan but may continue searching.
    - Guaranteed to terminate when optimal plan exists.
    - Plans may exist without there being an optimal one since cost can be negative.
  - Can "manually" terminate to get the current best solution.

Conclusions

- Algorithm which finds optimal solution to the planning problem with integer variables.
  - Not guaranteed to terminate when no optimal plan exists.
- No performance results given.
  - No follow-up work found.
  - Size of the minimal DFA representation of a Presburger arithmetic formula can be up to *triple exponential* ($\Theta(2^{2^{2^n}})$) in the length of the formula [5]!
- Authors were working on a more efficient implementation.
  - Adapt A^* to use state sets represented by DFAs.
  - Use fast automata libraries such as LIRA [1].
  - Support real-valued variables.

References I

[1] B. Becker, C. Dax, J. Eisinger, and F. Klaedtke.
Lira: handling constraints of linear arithmetics over the integers and the reals.
In *Proceedings of the 19th international conference on Computer aided verification*, CAV'07, pages 307–310, Berlin, Heidelberg, 2007. Springer-Verlag.

[2] B. Boigelot and P. Wolper.
Representing arithmetic constraints with finite automata: An overview.
In *Proceedings of the 18th International Conference on Logic Programming*, ICLP '02, pages 1–19, London, UK, UK, 2002. Springer-Verlag.

References II

[3] B. U. Borowsky and S. Edelkamp.
Optimal metric planning with state sets in automata representation.
In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*, AAAI'08, pages 874–879. AAAI Press, 2008.

[4] A. Cimatti, F. Giunchiglia, E. Giunchiglia, and P. Traverso.
Planning via model checking: A decision procedure for ar.
In *Proceedings of the 4th European Conference on Planning: Recent Advances in AI Planning*, ECP '97, pages 130–142, London, UK, UK, 1997. Springer-Verlag.

[5] F. Klaedtke.
Bounds on the automata size for presburger arithmetic.
ACM Trans. Comput. Logic, 9(2):11:1–11:34, Apr. 2008.