

AI Planning for Autonomy

5. Width Based Planning

Searching for Novelty
and How to Plan with Simulators

Tim Miller and Nir Lipovetzky



Winter Term 2018

Agenda

- 1 Width-Based Search
- 2 Balancing Exploration and Exploitation
- 3 Models and Simulators
- 4 Classical Planning with Simulators
- 5 Conclusion

Structure

Planning is **PSPACE-complete**, but current planners can **solve most of benchmarks in a few seconds**

Question:

- Can we explain why planners perform well?
- Can we characterize the line that separates ‘easy’ from ‘hard’ domains?

Our Answer

A new **width** notion and planning **algorithm exponential in problem width**:

- Benchmark domains have **small width** when **goals** restricted to **single atoms**
- Joint goals **easy to serialize** into a sequence of single goals

Our Answer

A new **width** notion and planning **algorithm exponential in problem width**:

- Benchmark domains have **small width** when **goals** restricted to **single atoms**
- Joint goals **easy to serialize** into a sequence of single goals

Do you want **Hard** Problems?

- problems with **high atomic width** (apparently no benchmark in this class)
- **multiple goal** problems that are **not easy to serialize** (e.g. Sokoban)

Agenda

- 1 Width-Based Search
- 2 Balancing Exploration and Exploitation
- 3 Models and Simulators
- 4 Classical Planning with Simulators
- 5 Conclusion

Definition: Novelty

Key definition: the **novelty** $w(s)$ **of a state** s is the size of the smallest subset of atoms in s that is true for the first time in the search.

- e.g. $w(s) = 1$ if there is **one** atom $p \in s$ such that s is the first state that makes p true.
- Otherwise, $w(s) = 2$ if there are **two** different atoms $p, q \in s$ such that s is the first state that makes $p \wedge q$ true.
- Otherwise, $w(s) = 3$ if there are **three** different atoms...

Iterated Width (IW)

Algorithm

- $IW(k)$ = breadth-first search that prunes newly generated states whose $novelty(s) > k$.
- IW is a sequence of calls $IW(k)$ for $i = 0, 1, 2, \dots$ over problem P until problem solved or i exceeds number of variables in problem

Properties

$IW(k)$ expands at most $O(n^k)$ states, where n is the number of atoms.

Is IW any good in Classical Planning?

- *IW*, while simple and blind, is a pretty **good algorithm** over benchmarks when goals restricted to **single atoms**
- This is no accident, **width** of benchmarks domains is **small** for such goals

We tested domains from previous IPCs. For **each instance** with N goal atoms, we **created N instances** with a single goal

- Results quite remarkable:

# Instances	<i>IW</i>	<i>ID</i>	<i>BrFS</i>	<i>GBFS</i> + h_{add}
37921	91%	24%	23%	91%

Why IW does so well?

Key theory of $IW(k)$ in terms of **width**:

Properties

For problems $\Pi \in \mathcal{P}$, where $\text{width}(\Pi) = k$:

- $IW(k)$ solves Π in time $O(n^k)$;
- $IW(k)$ solves Π **optimally**
- $IW(k)$ is **complete** for Π

Theorem

*Blocks, Logistics, Gripper, and n -puzzle have a **bounded width** independent of problem **size** and **initial situation**, provided that goals are **single atoms**.*

In practice, $IW(k \leq 2)$ solves **88.3% IPC problems with single goals**:

# Instances	$k = 1$	$k = 2$	$k > 2$
37921	37.0%	51.3%	11.7%

IW in Classical Planning?

Primary question: *IW* solves atomic goals, how do we extend the blind procedure to multiple atomic goals?

Serialized Iterated Width (SIW)

- Simple way to **use IW** for solving real benchmarks P with **joint goals** is by simple form of “**hill climbing**” over goal set G with $|G| = n$, achieving atomic goals one at a time

Serialized Iterated Width (SIW)

- **SIW uses IW** for both **decomposing** a problem into subproblems and for **solving** subproblems
- It's a **blind search** procedure, **no heuristic** of any sort, **IW does not even know next goal G_i** “to achieve”

Serialized Iterated Width (SIW)

- **SIW** uses **IW** for both **decomposing** a problem into subproblems and for **solving** subproblems
- It's a **blind search** procedure, **no heuristic** of any sort, **IW** **does not even know next goal** G_i "to achieve"

Blind **SIW** better than GBFS + h_{add} !

Summary (so far)

IW: sequence of novelty-based pruned breadth-first searches

- **Experiments**: excellent when goals restricted to atomic goals
- **Theory**: such problems have low width w and *IW* runs in time $O(n^w)$

SIW: *IW* serialized, used to attain top goals one by one

- **Experiments**: faster, better coverage and much better plans than GBFS planner with h_{add}
- **Intuition**: goals easy to serialize and have atomic low width w

Agenda

- 1 Width-Based Search
- 2 Balancing Exploration and Exploitation
- 3 Models and Simulators
- 4 Classical Planning with Simulators
- 5 Conclusion

Classical Planning

State-of-the-art methods for **satisficing planning** rely on:

- **heuristics** derived from problem
- plugged into **Greedy Best-First Search** (GBFS)
- extensions (like **helpful actions** and **landmarks**)

Shortcoming of GBFS: Exploration and Exploitation

GBFS is **pure greedy “exploitation”**; often gets **stuck in local minima**

- Recent approaches improve performance by adding exploration

Exploration **required** for **optimal behavior** in RL and MCTS

- Such methods perform **flat exploration** that **ignores structure of states**

We study impact of **width-based exploration methods** that take structure of states into account

Best-First Width Search (BFWS)

BFWS(f)

BFWS(f) for $f = \langle w, f_1, \dots, f_n \rangle$ where w is a novelty-measure, is a plain **best-first** search where nodes are **ordered in terms of novelty function w** , with ties broken by functions f_i in that order.

Basic BFWS($\langle w, h \rangle$) scheme obtained with $\mathbf{h} = \mathbf{h}_{\text{add}}$ or \mathbf{h}_{ff} , and novelty-measure $w = w_h$, where

- $w_h(s)$ = size of **smallest** new tuple of atoms generated by s for the first time in the search **relative to previously generated states s' with $h(s) = h(s')$** .

→ BFWS($\langle w, h \rangle$) much better than purely greedy BFS(h)

Some BFWS(f) variants yield **state-of-the-art performance**:

- **1st place** in Agile track, **Runner-Up** Satisficing track IPC-2018
- more info: <https://ipc2018-classical.bitbucket.io/>

Agenda

- 1 Width-Based Search
- 2 Balancing Exploration and Exploitation
- 3 Models and Simulators
- 4 Classical Planning with Simulators
- 5 Conclusion

Classical Planning

The status quo:

- Model usually represented in compact form (STRIPS, PDDL)



Introduction & Motivation

For more than 40 years, research focused on **exploiting** information about **action preconditions and effects** in order to plan efficiently

- ▶ GPS, POP, GraphPlan, SAT, OBDD, heuristic-search planning, ...

We showed that same level of efficiency can be obtained with **simulators**: **without** a representation of **action preconditions and effects**

Introduction & Motivation

This has been shown by:

- Developing a planner that uses action structure **only** to define
 - the set $A(s)$ of **applicable actions** in state s , and
 - state **transition function** $f(a,s)$

The **planner does not see action preconditions and effects** but just the functions $A(s)$ and $f(a,s)$

- We showed that its **performance matches** the performance of state of the art **planners that make use of PDDL** representations, over the existing PDDL benchmarks

Many consequences follow from this radical departure

Modeling



Modeling

Many problems fitting **classical planning** model but **difficult to describe in PDDL** are easily modeled now: Pacman, Tetris, Pong, etc.

- **Expressive language features** easily supported: functions, conditional effects, derived predicates, state constraints, quantification, ...
- Any element of the problem can be modeled through logical symbols attached to **external procedures** (e.g. C++).
- Action effects can be given as **fully-black-box procedure** taking the state as input.

Introduction & Motivation

Declarative languages also have their **downsides**:

- **Model \neq Language**. Many problems fit Classical Planning model, but hard to express in PDDL-like languages.
- Recent development of **simulation platforms** such as
 - ▶ Atari Learning Environment,
 - ▶ GVG-AI,
 - ▶ Universe, etc.

Need for planners that work **without complete declarative** representations.

Algorithm: Simulated BFWS

Framework **Best-first width search (BFWS)**:

- Novelty measures w also used in best-first algorithms (BFWS)
- Best results when w -measures combined with goal directed heuristics h
(Lipovetzky and Geffner, 2017)
- BFWS(h) picks node from OPEN with least w_h measure, breaking ties with h
 - $w_h(s) = k$ if s is first state to make some set of k atoms true, among those with heuristic $h = h(s)$

BFWS(h) much better than standard BFS(h)

Use of heuristics couples algorithm with *declarative representations*.

Algorithm: Simulated BFWS

Framework **Best-first width search (BFWS)**:

- Novelty measures w also used in best-first algorithms (BFWS)
- Best results when w -measures combined with goal directed heuristics h
(Lipovetzky and Geffner, 2017)
- BFWS(h) picks node from OPEN with least w_h measure, breaking ties with h
 - $w_h(s) = k$ if s is first state to make some set of k atoms true, among those with heuristic $h = h(s)$

BFWS(h) much better than standard BFS(h)

Use of heuristics couples algorithm with *declarative representations*.

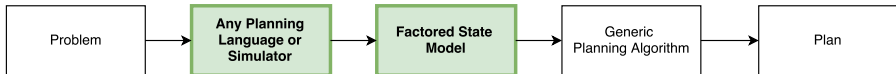
- In (Frances et al. 2017) we lift this requirement

Implications: Modeling and Control

► Traditional toolchain



► Results suggest **alternative**



► No **need** for planning languages that reveal **structure** of actions (e.g. action preconditions and effects)

► **Not much** efficiency appears to be **lost** in **second** pathway

These algorithms open up **Exciting possibilities** for **modeling beyond PDDL**

Width-Based planning over Simulators



Challenges:

- Non-linear dynamics
- Perturbation in flight controls
- Partial observability
- Uncertainty about opponent strategy

Agenda

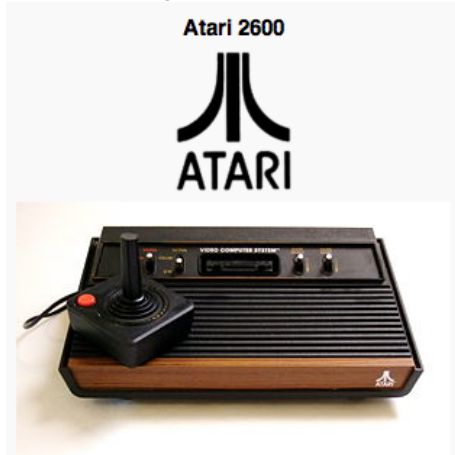
- 1 Width-Based Search
- 2 Balancing Exploration and Exploitation
- 3 Models and Simulators
- 4 Classical Planning with Simulators
- 5 Conclusion

Classical Planning with Simulators

Can classical planners work without PDDL?

Classical Planning with Simulators

Can classical planners work without PDDL?



Arcade Learning Environment

<http://www.arcadelearningenvironment.org/>

The Arcade Learning Environment (ALE) is a simple object-oriented framework that allows researchers and hobbyists to develop *AI agents for Atari 2600 games*.

Recent AI agents:

- Reinforcement Learning and Deep Learning trained to learn a controller
- Search algorithm as a lookahead for action selection

Motivation

While planning is the **model-based approach** to control, planning research is heavily fragmented

- Many models (classical; MDPs, POMDPs; "logical" variants FOND and POND; time, resources, ..)
- Modeling languages vs. Use of Simulators
- Different communities (ICAPS-AAAI; NIPS-UAI-RL ..)

Best way to get communication across and to build on each others' work is **common benchmarks** and **environments** such as *ALE*

ALE and Classical Planning

Planning setting in ALE is **deterministic and initial state fully known**

Yet classical planners can't be used

- **no PDDL** encoding
- **no goals** but rewards

→ *Bellemare et al.* consider Breadth-first Search (*BrFS*) and *MCTS (UCT)*

→ Still, "classical" planning algorithms such as *IW* can be applied almost off-the-shelf!

IW in the Atari Games

- *IW*(1) used with the 128 variables (bytes) of 256 values each
- *IW*(1) generates then up to $128 \times 256 \times 18$ (i.e, 589,824) states
 - Children in *IW*(1) generated in random order
 - Discount factor used $\gamma = 0.995$
- Action leading to most rewarding *IW*(1)-path is executed

/W Playing Atari!

Freeway

Experimental Results

Same setting from Bellemare et al:

- Games are played for 5 minutes maximum (18,000 frames)
- 2BFS and *IW* have a maximum lookahead budget of 150,000 simulated frames
- UCT has same budget by running 500 rollouts of depth 300
- Score is averaged among 5 runs per game

	<i>IW</i> (1)	2BFS	<i>BrFS</i>	UCT
# Times Best (54 games)	26	13	1	19
# Times Better than <i>IW</i>	–	16	1	19
# Times Better than 2BFS	34	–	1	25
# Times Better than UCT	31	26	1	–

Search Tree Depths

- *BrFS* search tree results in a lookahead of 0.3 seconds
- *IW*(1) and 2BFS result in lookahead of up to 6–22 seconds

Atari in the Press

IW vs DeepMind

Lookahead Agents VS Learning Agents:

↪ Still open how to compare them best as they solve different control problems, and different inputs (RAM vs Screen)

But, taking into account gameplay score:

↪ **IW outperforms DeepMind's algorithm in 45 out of 49 games**

↪ Similar results have been reported recently over Screen Inputs [AAAI 2018]

ALE Wrap up

- *IW* makes use of the **state structure** (atoms) to order exploration
- *IW*(1) is a *BrFS* that **keeps states that generate new atoms**
- **Exploitation of this structure pays off in classical planning and ALE**
- First **classical planners using simulators**
- Youtube videos: [▶ Link http://bit.ly/1EuCb9x](http://bit.ly/1EuCb9x)

Agenda

- 1 Width-Based Search
- 2 Balancing Exploration and Exploitation
- 3 Models and Simulators
- 4 Classical Planning with Simulators
- 5 Conclusion

The Width Conspirators: What's next?

- Explore new **applications**: Social Sciences, Computational Sustainability, and any other fields rely on simulators
- Width-based for other **planning computational models**: Uncertainty, Beliefs, Multi-agent
- Devise new **algorithms** for real-time behavior
- Bridge connection between **Control, Planning and Learning**

Help us grow the boundaries of AI planning research!

Resources

Literature:

- <https://nirlipo.github.io/Width-Based-Planning-Resources/>

LAPKT stands for the Lightweight Automated Planning ToolKit:

- <http://lapkt.org>

IW-ALE, BFWS source code:

- <http://lapkt.org/index.php?title=Projects>

Width-based in action featured in ICAPS-19:

- [https://icaps19.icaps-conference.org/!](https://icaps19.icaps-conference.org/)