Reinforcement learning based dependency resolution

DevConf.US Virtual 2020

Fridolin Pokorny <fridolin@redhat.com>
AICoE, Project Thoth

Red Hat

What is Thoth?

- A project in AI Center of Excellence (AICoE), Office of the CTO
- Software stacks are complex and only not-changing when running in production
  - Stacks depend on a lot of components, they keep changing all the time
  - Stacks span a lot of layers
  - Vast amount of platforms available as runtime alternatives
  - Off the shelf builds and upstream libraries do not fit corporate needs
- Make OpenShift a better platform to run AI/ML workloads
  - Python is the driving force for AI/ML applications

Find us on GitHub: http://github.com/thoth-station/

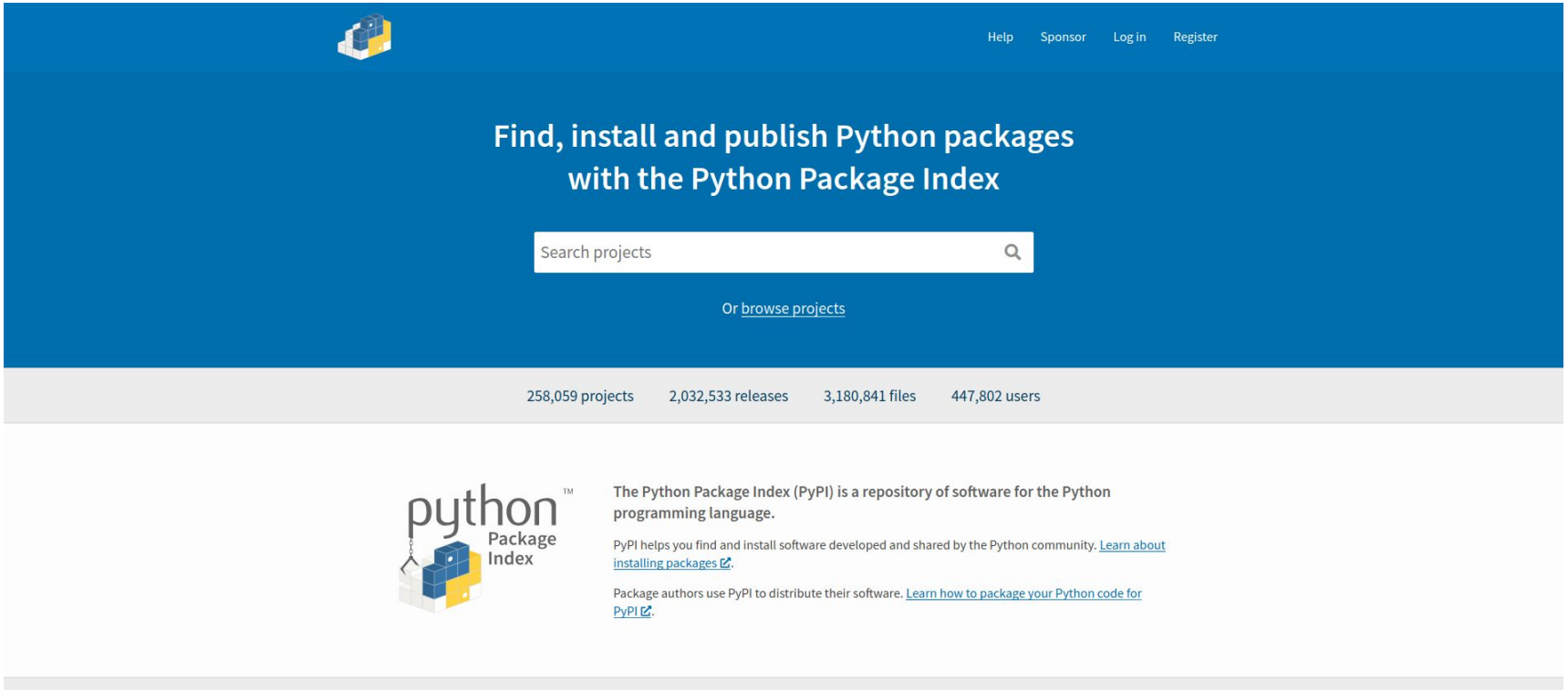
Agenda

1. Dependency resolution in Python
2. Existing solutions and their pros & cons
3. Why another solution?
4. Monte Carlo Tree Search as a way to resolve high quality software stacks
5. Resolution pipelines - reconfigurable resolver

Source:

https://thoth-station.ninja/
https://github.com/thoth-station/

Dependency resolution in Python



Source: pypi.org

Dependency resolution in Python

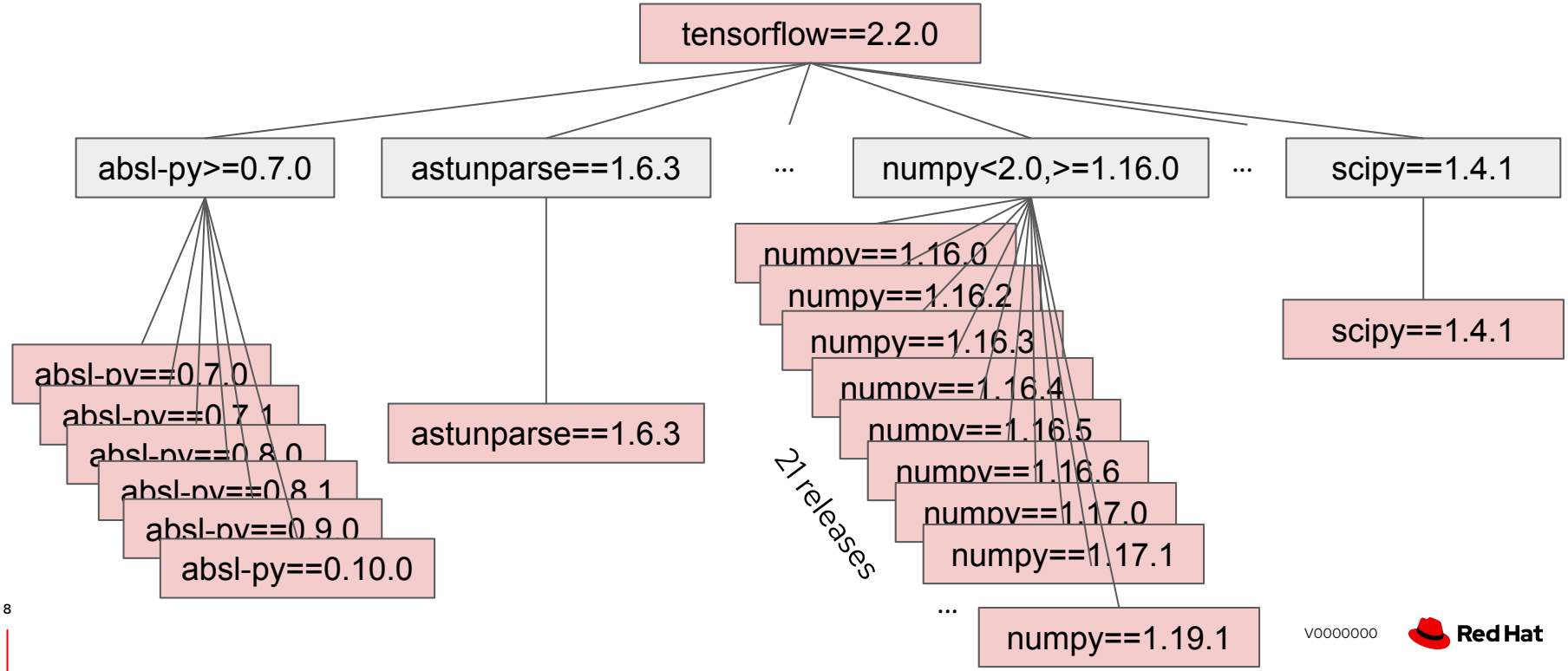
- Python distributions
- package name - package version - index url
  - tensorflow - 2.2.0 - https://pypi.org/simple (upstream TensorFlow)
  - tensorflow - 2.2.0 - http://tensorflow.pypi.thoth-station.ninja/index/manylinux2010/AVX2/simple (AICoE builds of TensorFlow, AVX2 optimized)
- A package can depend on another package given the version range specification and environment markers
  - tensorflow - 2.2.0 - https://pypi.org/simple depends on (~23 packages in total):
    - ...
    - numpy (<2.0,>=1.16.0)
    - enum34 (>=1.1.6) ; python_version < "3.4"
    - ...

Dependency graph

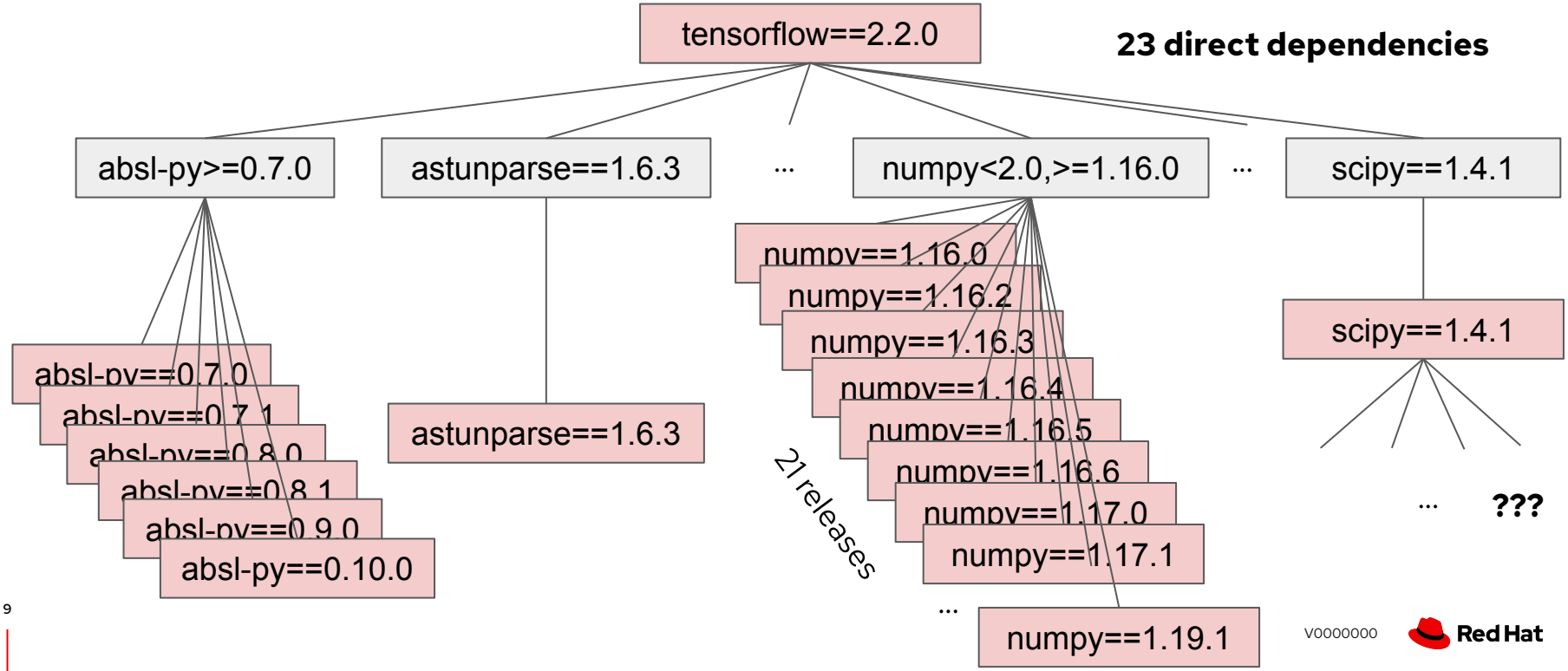
- tensorflow - 2.2.0 - https://pypi.org/simple (~23 dependencies)

absl-py (>=0.7.0), astunparse (==1.6.3), gast (==0.3.3), google-pasta (>=0.1.8), h5py (<2.11.0,>=2.10.0), keras-preprocessing (>=1.1.0), numpy (<2.0,>=1.16.0), opt-einsum (>=2.3.2), protobuf (>=3.8.0), tensorboard (<2.3.0,>=2.2.0), tensorflow-estimator (<2.3.0,>=2.2.0), termcolor (>=1.1.0), wrapt (>=1.11.1), six (>=1.12.0), grpcio (>=1.8.6), wheel ; python_version < "3", mock (>=2.0.0) ; python_version < "3", functools32 (>=3.2.3) ; python_version < "3", scipy (==1.2.2) ; python_version < "3", backports.weakref (>=1.0rc1) ; python_version < "3.4", enum34 (>=1.1.6) ; python_version < "3.4", wheel (>=0.26) ; python_version >= "3", scipy (==1.4.1) ; python_version >= "3"

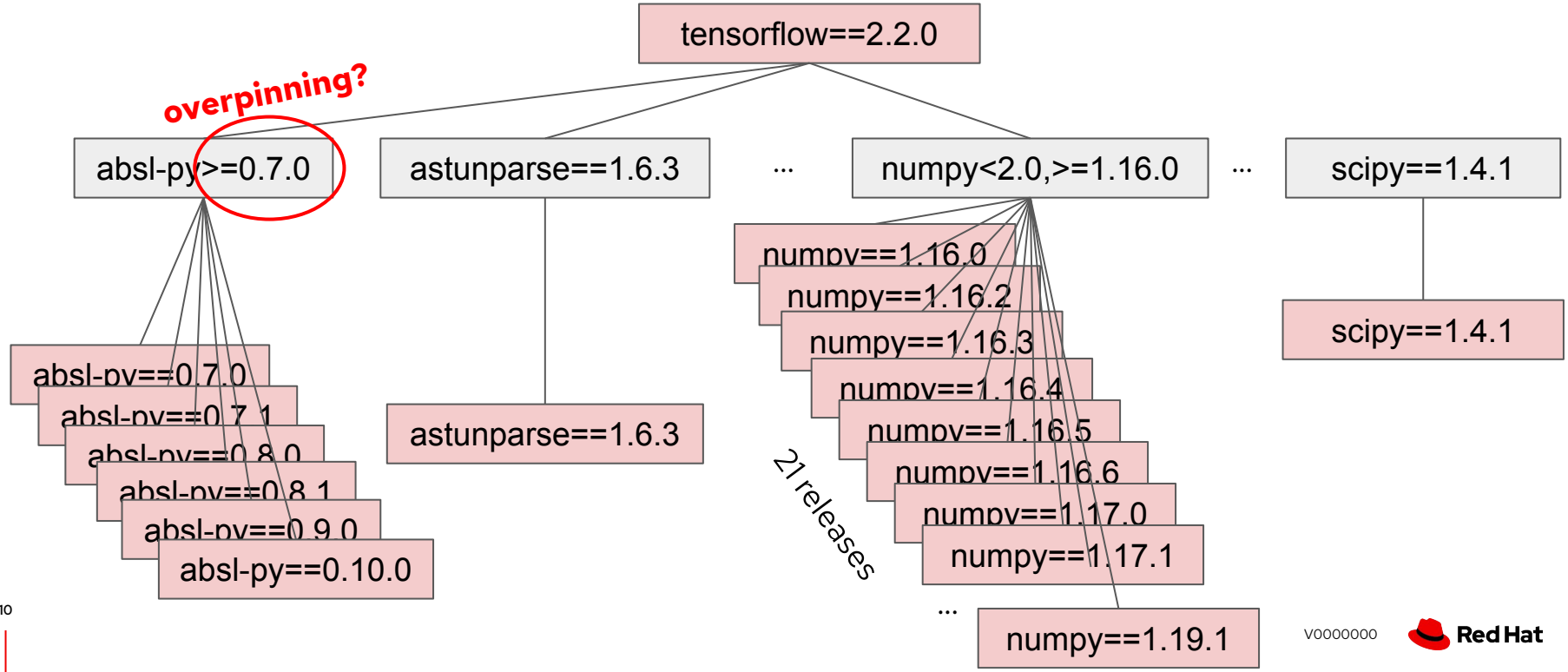
Dependency graph



Dependency graph

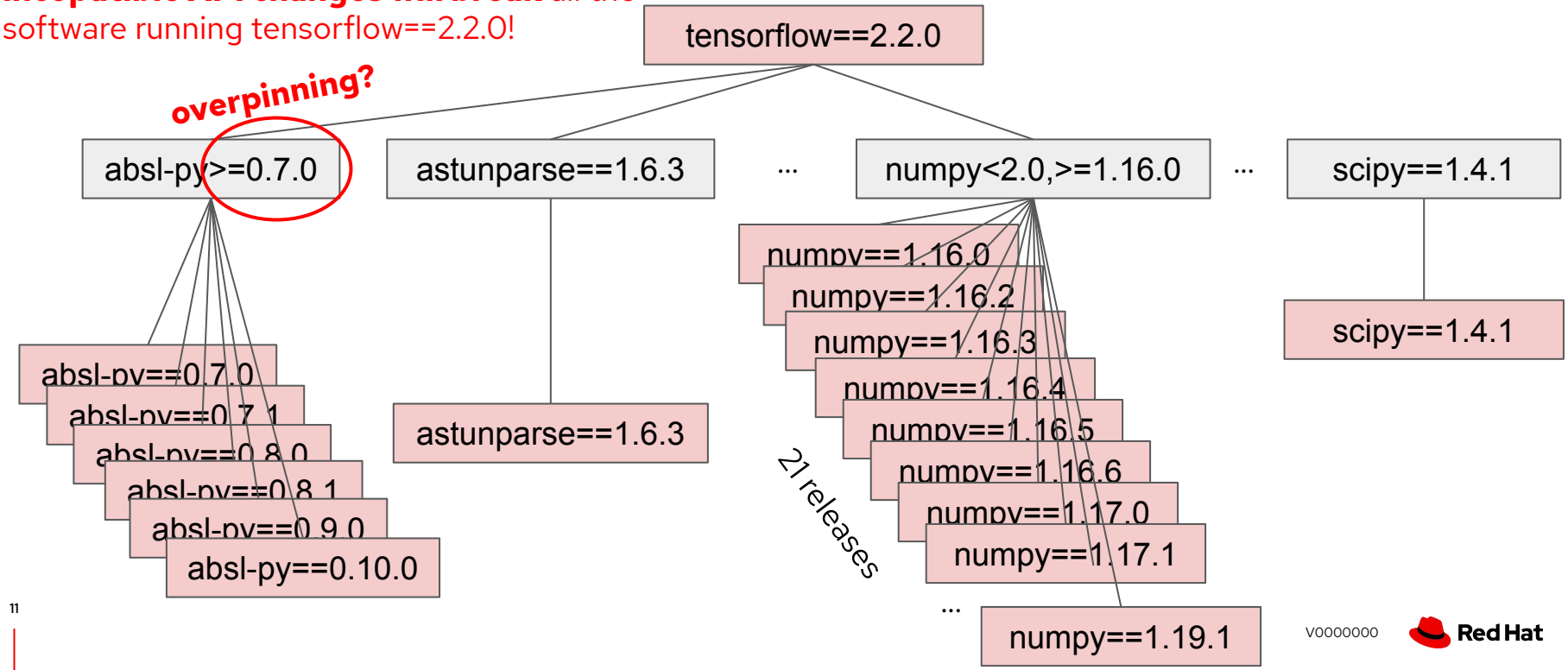


Dependency graph



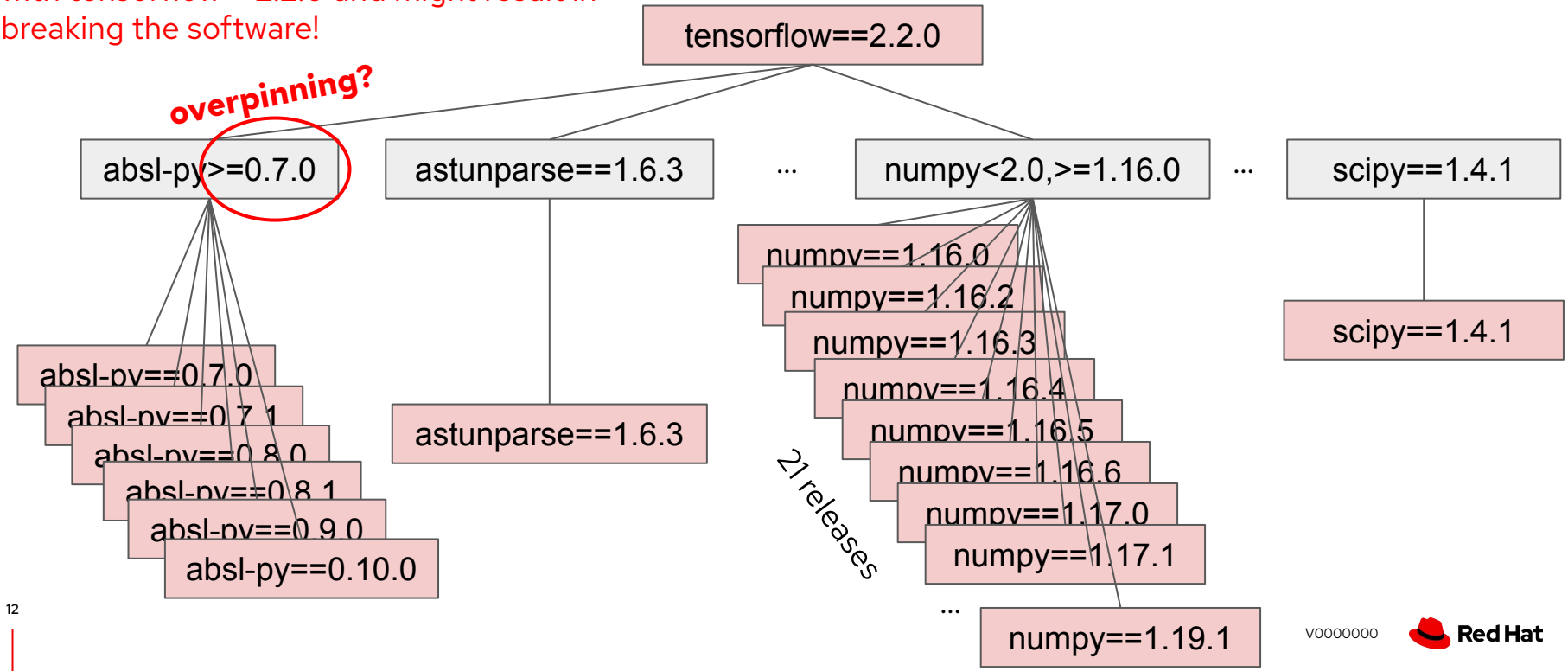
Dependency graph

A new release of absl-py==1.0.0 with **incopatible API changes will break** all the software running tensorflow==2.2.0!



Dependency graph

A new release of absl-py==0.11.0 is **untested** with tensorflow==2.2.0 and might result in breaking the software!



Dependency graph

absl-py: 6
astunparse: 1
gast: 1
google-pasta: 2
h5py: 1
keras-preprocessing: 3
numpy: 21
opt-einsum: 6
protobuf: 14
tensorboard: 3
tensorflow-estimator: 1
termcolor: 1
wrapt: 4
six: 4
grpcio: 38
wheel: 57
mock: 10
functools32: 3
scipy: 1
backports.weakref: 2
enum34: 4
wheel: 21
scipy: 1

- Number of combinations for direct dependencies in case of tensorflow==2.2.0 from pypi.org: 3.3×10^{13}
  - to this date!
  - only the ones available on PyPI, only direct dependencies and only tensorflow==2.2.0 from pypi.org

Dependency resolution in Python

- Dependency resolution is dependent on the environment used
  - enum34 (>=1.1.6) ; python_version < "3.4"
  - thoth-solver - a tool that checks dependencies
  - Invest CPU time to pre-compute dependencies for a dependency graph construction
- Dusting Ingram: Why PyPI Doesn't Know Your Project Dependencies
- How to beat Python's pip: Solving Python dependencies

Existing solutions and their pros&cons

Dependency resolution in Python

- pip
  - The PyPA recommended tool for installing Python packages
- Pipenv
  - Uses a lock file and manages virtual environment
- pip-tools
  - A package locking mechanism with requirements.in/requirements.txt
- Poetry
  - A community effort similar to Pipenv
- micropipenv
  - Not a resolver
  - Installs dependencies as described by pip/Pipenv/pip-tools/Poetry

Source: https://packaging.python.org/tutorials/managing-dependencies/

Why another solution?

Why another solution?

- pip, Pipenv, pip-tools, Poetry install “latest software”
- What if latest is not greatest?
  - Install packages that:
    - … install into my environment (e.g. no setup.py issues)
    - … run in my environment correctly (e.g. no Python 3 vs Python 2 syntax errors)
    - … produce correct results (no bugs on application level)
    - … perform well (e.g. optimized builds of TensorFlow for AVX2 instruction set available on my CPU)
    - … are not prone to known vulnerabilities (e.g. CVE-2019-9635: NULL pointer dereference in Google TensorFlow before 1.12.2 could cause a denial of service via an invalid GIF file)
    - …
  - Recall the absl-py==1.0.0 release seen earlier in slides

Monte Carlo Tree Search as a way to resolve high quality software stacks

Why Monte Carlo Tree Search?

- we already had other efforts
  - Performing computations directly on the dependency graph
    - PyCon US: Thoth - how to recommend the best possible libraries for your application
  - Neural Combinatorial Optimization with Reinforcement Learning
  - Adaptive Simulated Annealing
    - FOSDEM 2020 Thoth - a recommendation engine for Python applications
  - Reinforcement Learning
    - Temporal Difference leaning
    - Monte Carlo Tree Search

Monte Carlo Tree Search

In computer science, Monte Carlo tree search (MCTS) is a heuristic search algorithm for some kinds of decision processes, most notably those employed in software that plays board games. In that context MCTS is used to solve the game tree.

Source: https://en.wikipedia.org/wiki/Monte_Carlo_tree_search

Monte Carlo Tree Search

- Resolver has no real opponent
  - A variation of MCTS which uses adaptive simulated annealing principles to balance exploration and exploitation
  - CPU time is our opponent as the state space is too large
- The resolution process can still be seen as a Markov decision process (MDP)
  - Try to resolve so that the cummulative reward is the highest possible
  - The final state with the highest possible cummulative reward is the best software stack
- MCTS is a type of “predictor” in Thoth’s adviser, other predictors:
  - “Latest software stack”
  - Adaptive Simulated Annealing
  - Temporal Difference learning
  - …

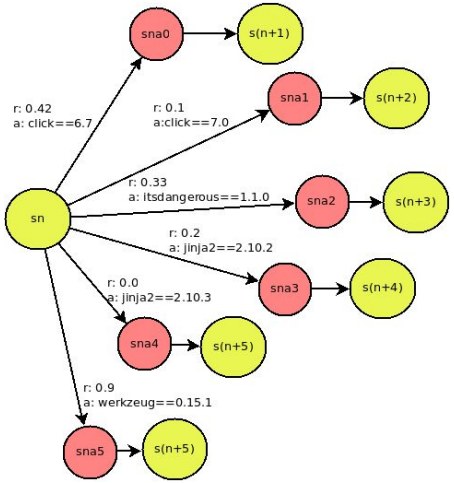
Markov decision process

In mathematics, a Markov decision process (MDP) is a discrete-time stochastic control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying optimization problems solved via dynamic programming and reinforcement learning.

Source: https://en.wikipedia.org/wiki/Markov_decision_process

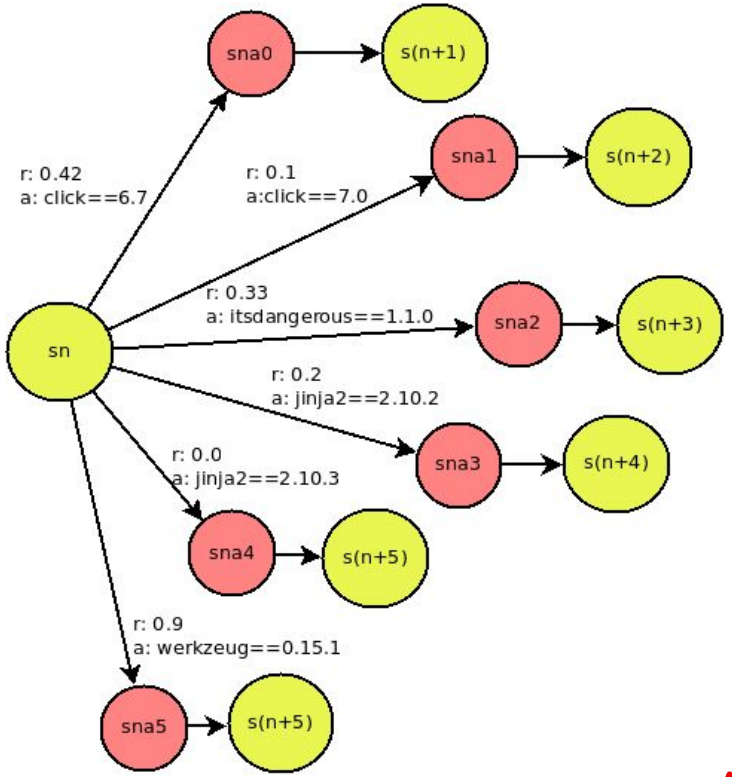
Monte Carlo Tree Search based resolution process

- Maintain a beam of states
- Maintain a set of resolved and unresolved dependencies
- A dependency is resolved by expanding it's direct dependencies which are added to the unresolved set
  - Respects Python dependency resolution
  - Corresponds to an an action ("resolver steps") in an MDP
  - Compute cummulative reward signal
  - Backpropagate information about the reward computed



Monte Carlo Tree Search based resolution process

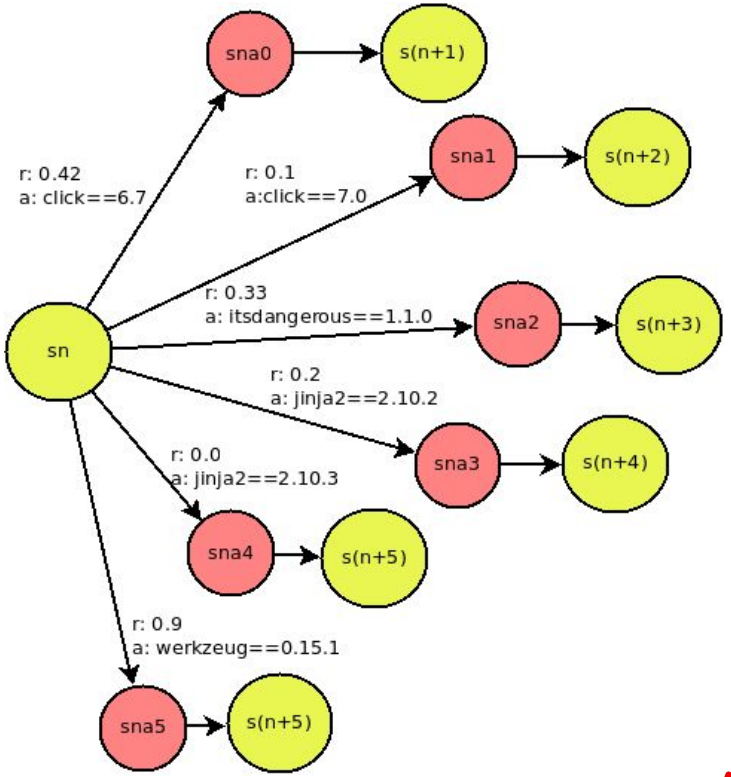
- state *sn*
  - score: 0.3
  - resolved dependencies:
    - flask==1.1.2 from pypi.org
  - unresolved dependencies:
    - click:
      - ==6.7 from pypi.org
      - ==7.0 from pypi.org
    - itsdangerous
      - ==1.1.0 from pypi.org
    - jinja2
      - ==2.10.2 from pypi.org
      - ==2.10.3 from pypi.org
    - werkzeug
      - ==0.15.1 from pypi.org



Monte Carlo Tree Search based resolution process

- state *sn*
  - score: 0.3
  - resolved dependencies:
    - flask==1.1.2 from pypi.org
  - unresolved dependencies:
    - click:
      - ==6.7 from pypi.org
      - ==7.0 from pypi.org
    - itsdangerous
      - ==1.1.0 from pypi.org
    - jinja2
      - ==2.10.2 from pypi.org
      - ==2.10.3 from pypi.org
    - werkzeug
      - ==0.15.1 from pypi.org

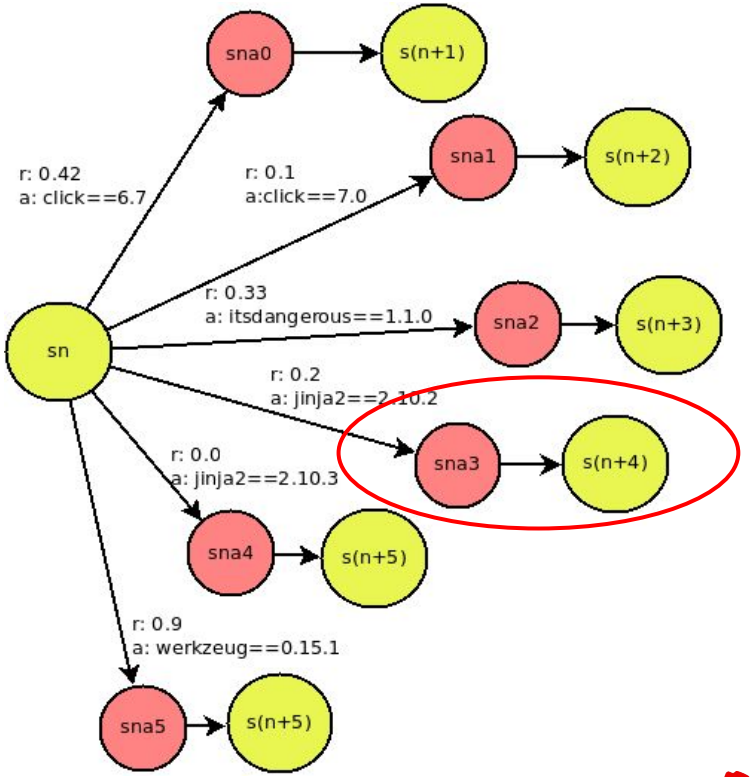
Predictor, which dependency should I choose?



Monte Carlo Tree Search based resolution process

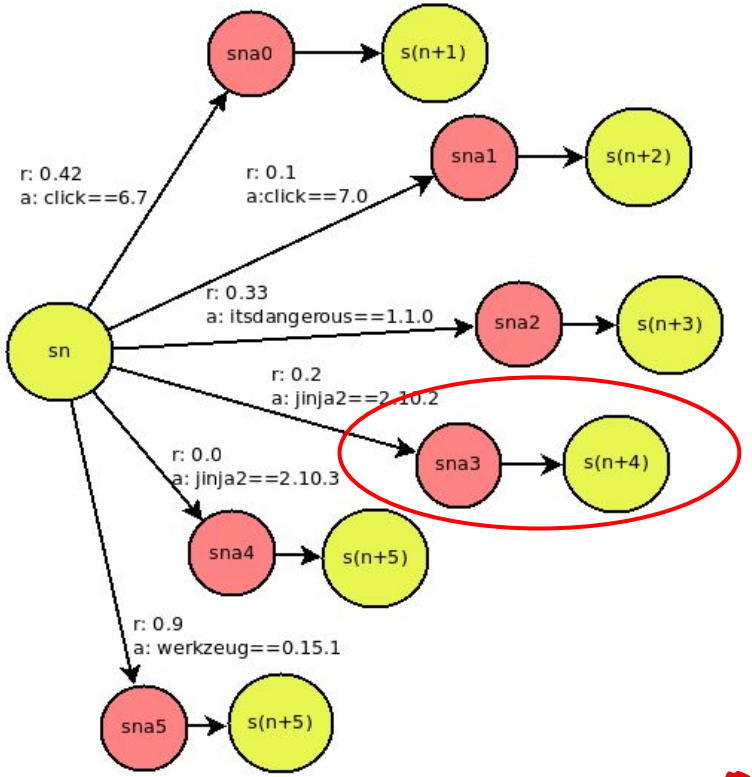
- state *sn*
  - score: 0.3
  - resolved dependencies:
    - flask==1.1.2 from pypi.org
  - unresolved dependencies:
    - click:
      - ==6.7 from pypi.org
      - ==7.0 from pypi.org
    - itsdangerous
      - ==1.1.0 from pypi.org
    - jinja2
      - **==2.10.2 from pypi.org**
      - ==2.10.3 from pypi.org
    - werkzeug
      - ==0.15.1 from pypi.org

Predictor, which dependency should I choose?



Monte Carlo Tree Search based resolution process

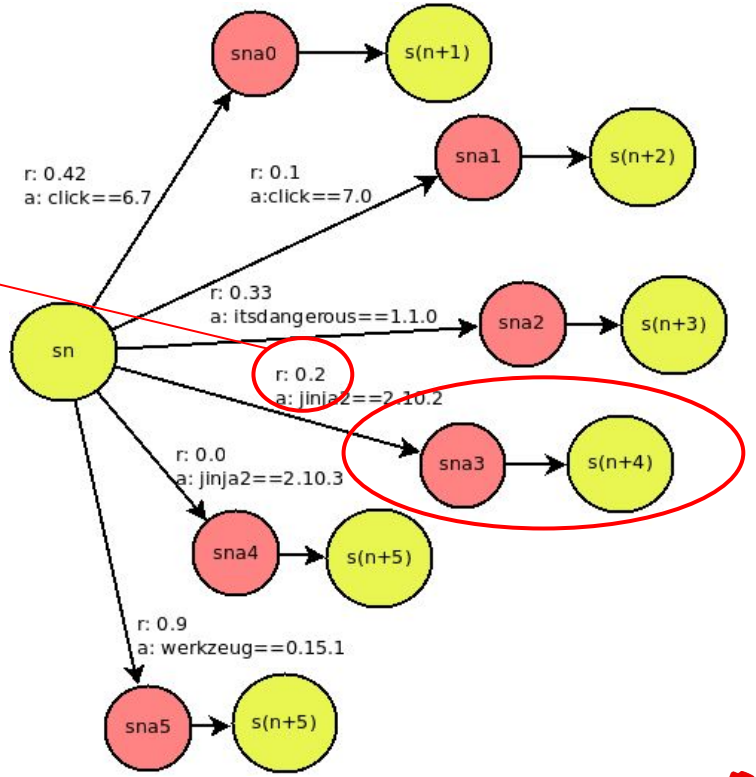
- state *sn*
  - score: 0.3
  - resolved dependencies:
    - flask==1.1.2 from pypi.org
  - unresolved dependencies:
    - click:
      - ==6.7 from pypi.org
      - ==7.0 from pypi.org
    - itsdangerous
      - ==1.1.0 from pypi.org
    - ~~jinja2~~
      - ~~**==2.10.2 from pypi.org**~~
      - ~~==2.10.3 from pypi.org~~
    - werkzeug
      - ==0.15.1 from pypi.org



Monte Carlo Tree Search based resolution process

- state *sn**+4***
  - score: 0.3+0.2
  - resolved dependencies:
    - flask==1.1.2 from pypi.org
    - **jinja2==2.10.2 from pypi.org**
  - unresolved dependencies:
    - click:
      - ==6.7 from pypi.org
      - ==7.0 from pypi.org
    - itsdangerous
      - ==1.1.0 from pypi.org
    - ~~jinja2~~
      - ~~==2.10.2 from pypi.org~~
      - ~~==2.10.3 from pypi.org~~
    - werkzeug
      - ==0.15.1 from pypi.org
    - **+ dependencies of jinja2**

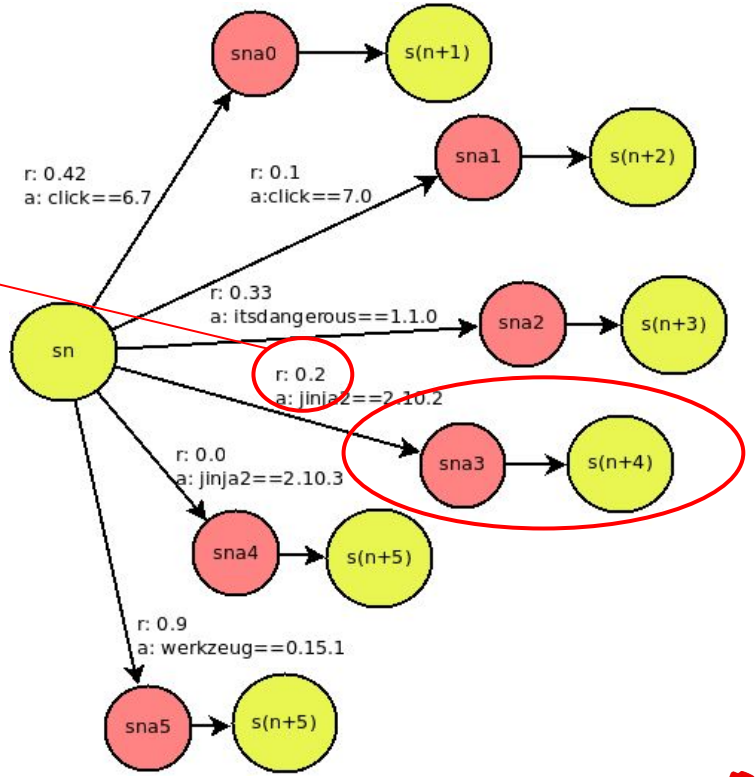
Information about immidiate reward will be propagated to parent states (sn, ...).



Monte Carlo Tree Search based resolution process

- state *sn**+4***
  - score: 0.3+0.2
  - resolved dependencies:
    - flask==1.1.2 from pypi.org
    - jinja2==2.10.2 from pypi.org
  - unresolved dependencies:
    - click:
      - ==6.7 from pypi.org
      - ==7.0 from pypi.org
    - itsdangerous
      - ==1.1.0 from pypi.org
    - ~~jinja2~~
      - ~~==2.10.2 from pypi.org~~
      - ~~==2.10.3 from pypi.org~~
    - werkzeug
      - ==0.15.1 from pypi.org
    - **+ dependencies of jinja2**

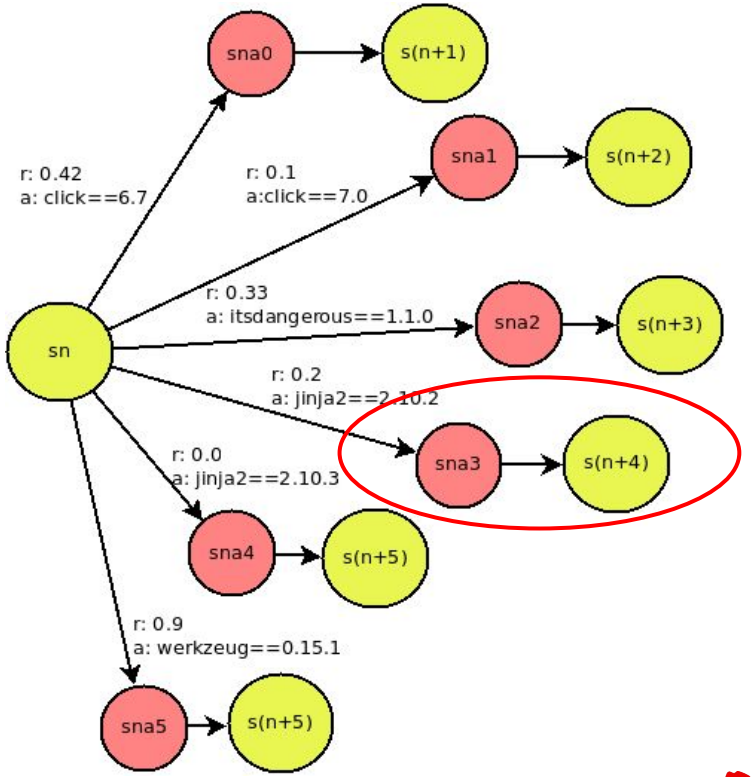
Reward can be also NaN or Inf.



Monte Carlo Tree Search based resolution process

- state *sn**+4***
  - score: **0.5**
  - resolved dependencies:
    - flask==1.1.2 from pypi.org
    - jinja2==2.10.2 from pypi.org
  - unresolved dependencies:
    - click:
      - ==6.7 from pypi.org
      - ==7.0 from pypi.org
    - itsdangerous
      - ==1.1.0 from pypi.org
    - werkzeug
      - ==0.15.1 from pypi.org
    - **markupsafe>=0.23**
    - **babel>=0.8**

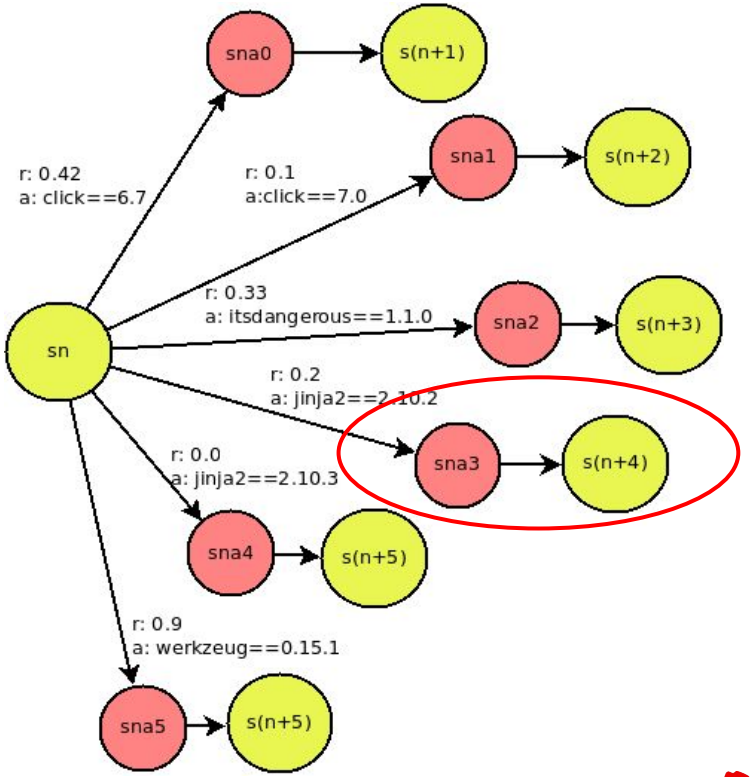
Resolve dependencies of jinja2==2.10.2 from pypi.org.



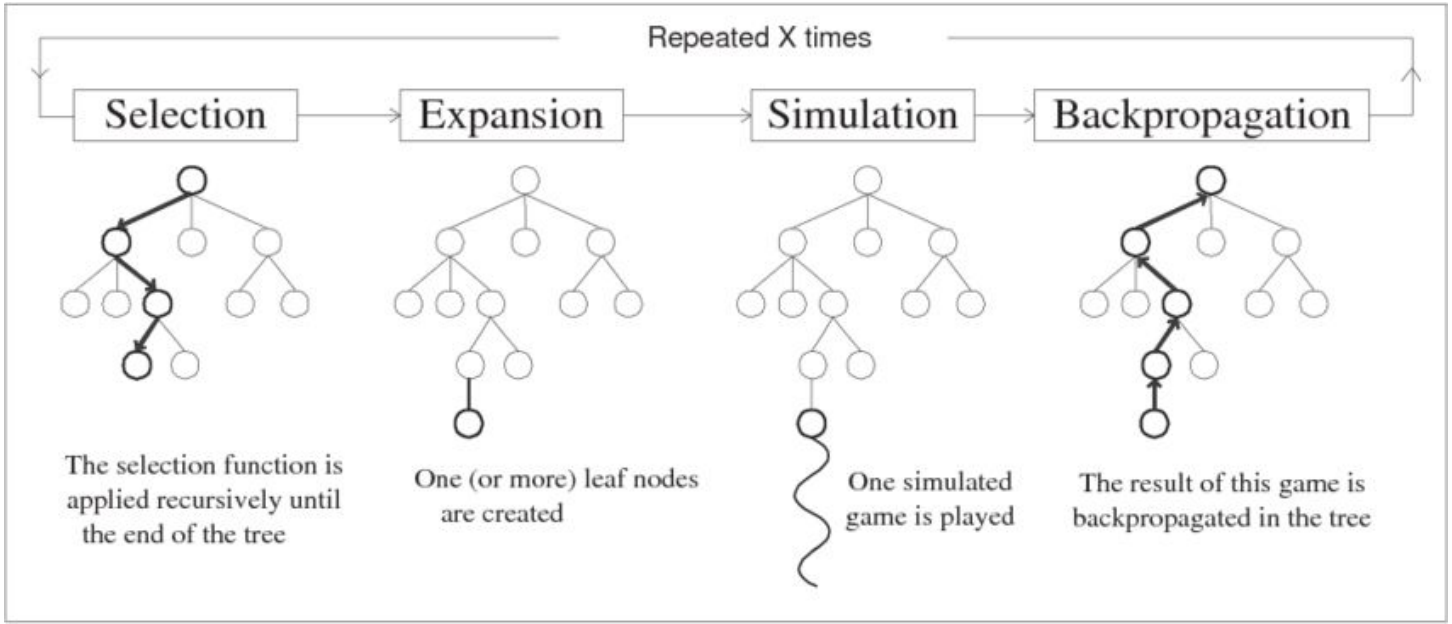
Monte Carlo Tree Search based resolution process

- state *sn**+4***
  - score: **0.5**
  - resolved dependencies:
    - flask==1.1.2 from pypi.org
    - jinja2==2.10.2 from pypi.org
  - unresolved dependencies:
    - click:
      - ==6.7 from pypi.org
      - ==7.0 from pypi.org
    - itsdangerous
      - ==1.1.0 from pypi.org
    - werkzeug
      - ==0.15.1 from pypi.org
    - **markupsafe**
      - **==...**
    - **babel**
      - **==...**

Resolve dependencies of jinja2==2.10.2 from pypi.org.



Monte Carlo Tree Search based resolution process



Monte Carlo Tree Search based resolution process

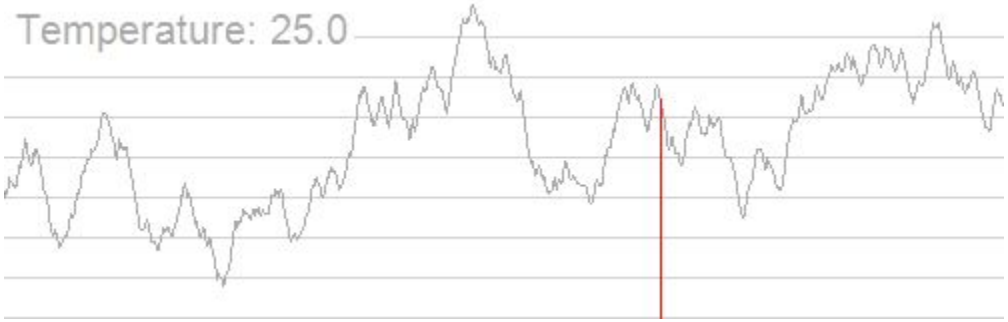
parent

- state *sn*
  - score: 0.3
  - resolved dependencies:
    - flask==1.1.2 from pypi.org
  - unresolved dependencies:
    - click:
      - ==6.7 from pypi.org
      - ==7.0 from pypi.org
    - itsdangerous
      - ==1.1.0 from pypi.org
    - jinja2
      - ~~==2.10.2 from pypi.org~~
      - ==2.10.3 from pypi.org
    - werkzeug
      - ==0.15.1 from pypi.org

- state *s(n+4)*
  - score: 0.5
  - resolved dependencies:
    - flask==1.1.2 from pypi.org
    - jinja2==2.10.2 from pypi.org
  - unresolved dependencies:
    - click:
      - ==6.7 from pypi.org
      - ==7.0 from pypi.org
    - itsdangerous
      - ==1.1.0 from pypi.org
    - jinja2
      - ==2.10.2 from pypi.org
    - werkzeug
      - ==0.15.1 from pypi.org

Balancing exploration and exploitation

- *Predictor, which dependency should I choose?*
  - Exploration
    - explore the state space of available options and observe how it behaves
  - Exploitation
    - based on observations of the state space, maximize reward signal
  - Adopted (Adaptive) Simulated Annealing
  - "Termial" random

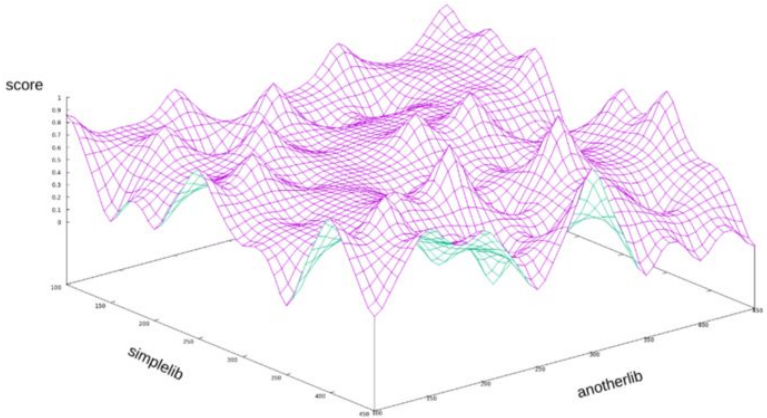


Source: https://en.wikipedia.org/wiki/Simulated_annealing

Resolving software stacks in a large state space

Resolving software stacks in large state space

- Too many possibilities to check
  - Number of combinations of direct dependencies for a tensorflow==2.2.0 stack: 3.3×10^{13}
- Not possible to check all of them for any real-world application stack
  - Have a heuristic to approximate the best possible software stack as close as possible
- Reinforcement learning is the way to learn how to resolve high quality software stacks

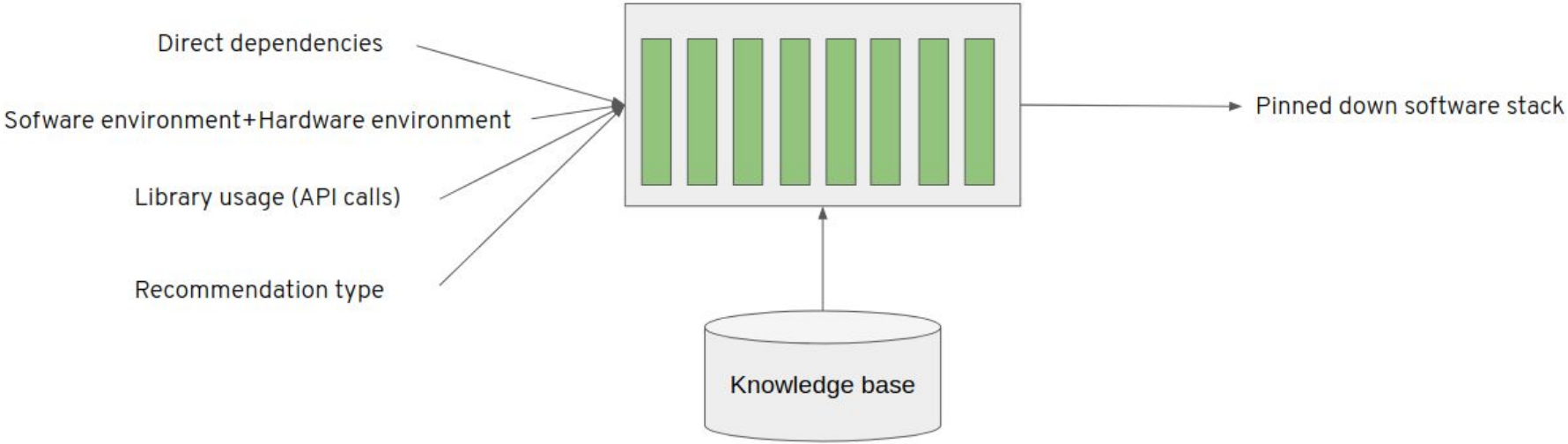


Resolution pipeline

Resolution pipeline

- Reconfigurable resolver
  - Configuration made out of “pipeline units”
  - Pipeline units added based on semantics (e.g. secure software stack, high performance software stack, ...)
- Do not resolve to latest software stack but to “greatest” software stack
  - Greatest based on user’s needs
  - Units included in the pipeline dynamically on each adviser start up
- Different pipeline units based on their semantics
  - Can compute immidiate reward signal
- Units form easily programabble interface to the resolution process

Resolution pipeline



A pipeline unit example

```
class TensorFlowAVX2Step(Step):
  """A step that recommends AICoE TensorFlow builds optimized for AVX2 enabled CPU processors."""

  _AVX2_CPUS = frozenset(
      ...
  )

  @classmethod
  def should_include(cls, builder_context: "PipelineBuilderContext") -> Optional[Dict[str, Any]]:
      """Register this pipeline unit for adviser and stable/performance recommendation types."""

  def run(
      self, state: State, package_version: PackageVersion
  ) -> Optional[Tuple[Optional[float], str]]:
      """Recommend TensorFlow builds optimized for AVX2 enabled CPU processors."""
```

A pipeline unit example

```
class TensorFlowAVX2Step(Step):
  """A step that recommends AICoE TensorFlow builds optimized for AVX2 enabled CPU processors."""


  # A tuple (CPU_FAMILY, CPU_MODEL) of Intel processors supporting AVX2:
  _AVX2_CPUS = frozenset(
    {
      (0x6, 0x5),  # Cascade Lake
      (0x6, 0x6),  # Broadwell, Cannon Lake
      (0x6, 0xA),  # Ice Lake
      (0x6, 0xC),  # Ice Lake, Tiger Lake
      (0x6, 0xD),  # Ice Lake
      (0x6, 0xE),  # Skylake, Keby Lake, Coffee Lake, Ice Lake, Comet Lake
      (0x6, 0xF),  # Haswell
    }
  )
```

A pipeline unit example

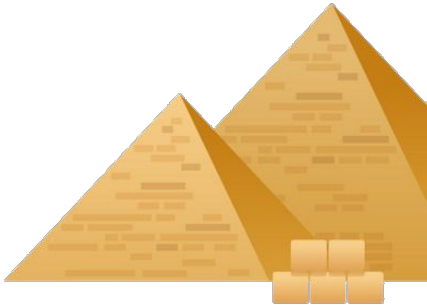
```
class TensorFlowAVX2Step(Step):
    """A step that recommends AICoE TensorFlow builds optimized for AVX2 enabled CPU processors."""

    @classmethod
    def should_include(cls, builder_context: "PipelineBuilderContext") -> Optional[Dict[str, Any]]:
        """Register this pipeline unit for adviser and stable/performance recommendation types."""
        if builder_context.recommendation_type not in (RecommendationType.STABLE, RecommendationType.PERF):
            return None

        cpu_tuple = (
            builder_context.project.runtime_environment.hardware.cpu_family,
            builder_context.project.runtime_environment.hardware.cpu_model,
        )
        if cpu_tuple not in cls._AVX2_CPUS:
            # No AVX2 support for the given CPU or no CPU info.
            return None

        return {}
```

A pipeline unit example

```
class TensorFlowAVX2Step(Step):
  """A step that recommends AICoE TensorFlow builds optimized for AVX2 enabled CPU processors."""

  def run(
      self, state: State, package_version: PackageVersion
  ) -> Optional[Tuple[Optional[float], str]]:
      """Recommend TensorFlow builds optimized for AVX2 enabled CPU processors."""
      if package_version.name != "tensorflow":
          # Not a TensorFlow package.
          return None

      aicoe_config = self.get_aicoe_configuration(package_version)
      if not aicoe_config or aicoe_config["configuration"].lower() != "avx2":
          # Not an AICoE build or not an AVX2 build.
          return None

      return self._REWARD, "AICoE TensorFlow builds are optimized for AVX2 instruction sets supported in the CPU"
```

Project Thoth

- AICoE, Office of the CTO
- Homepage
  - http://thoth-station.ninja/
- GitHub organization
  - http://github.com/thoth-station/
- Twitter account with updates
  - https://twitter.com/thothstation
- YouTube channel
  - https://www.youtube.com/channel/UCIUIDuq_hQ6vlzmqM59B2Lw



Thanks for your attention!

https://github.com/thoth-station

https://twitter.com/thothstation

https://www.youtube.com/channel/UCIUIDuq_hQ6vlzmqM59B2Lw

References

Website https://thoth-station.ninja/

Twitter https://twitter.com/thothstation

GitHub https://github.com/thoth-station

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat