Automatic Dependency Removal

A Graph Theory Project

by Đoàn Bá Cường, Phí Đỗ Hải Long, Nguyễn Gia Phong,
Nguyễn Hồng Quang and Trần Minh Vương

January 15, 2021

1 Introduction

1.1 Brief Description

Since the dawn Python packaging ecosystem, proper package management has been missing, partly due to the lack of package dependency resolution tools that are widely compatible. Instead, it is often seen that developers preferring the use of dedicated virtual environments specific for each and every task. In production, such boilerplate introduces additional complexity as well as latency, which negatively affect productivity.

As a proper dependency resolver is now baked into `pip` (since release 20.3), the package installer for Python, it is no longer non-trivial to support common package management use cases such as autoremoval of orphan dependencies. This project aimed to provide a proof of concept for future carry-out of such feature.

1.2 Authors and Credits

The work has been undertaken by group number 4 in the course of Graph Theory at the University of Science and Technology of Hà Nội, whose members are listed in the following table.

Full name	Student ID
Đoàn Bá Cường	BI9-062
Phí Đỗ Hải Long	BI9-149
Nguyễn Gia Phong	BI9-184
Nguyễn Hồng Quang	BI9-194
Trần Minh Vương	BI9-239

We would like to express our special thanks to Dr. Sebastian Basterrech, whose lectures brought us interests and inspirations in graph theory to work on this project in particular.

2 Objective

In this project, we wanted to develop as package pinning and automatic dependency removal for Python packaging, which is commonly found in package managers for Unix-like operating systems. The resulting software would be able to perform the following tasks:

1. Mark manually installed packages
2. When a package is to be removed, automatically look for its dependencies to uninstall at the same time, except for those manually installed or installed as a dependency of other manually installed packages that are not specified for removal

3 Theoretical Background

3.1 Problem Definition

3.1.1 States and Constraints

The states include a dependency graph G and a set M of manually installed distribution packages[*].

Let $G = (V, E)$ where $\forall v \in V$, v is a distribution package, $M \subset V$ and $\forall (u, v) \in E$, u depends on v. For G to be a valid digraph,

$$\forall u \in V, \forall v \notin V, (u, v) \notin E \tag{1}$$

Define direct dependencies of a package as

$$d(v) = \{u \in V \mid (v, u) \in E\}$$

implicit dependencies of a package as

$$i(v) = \{v\} \cup \bigcup_{u \in d(v)} i(u)$$

[*]As defined in the Glossary for Python packaging, a distribution package is

> A versioned archive file that contains Python packages, modules, and other resource files that are used to distribute a release. The archive file is what an end-user will download from the Internet and install.
>
> A distribution package is more commonly referred to with the single words "package" or "distribution", but [it might be confused] with an import package (which is also commonly called a "package") or another kind of distribution (e.g. a Linux distribution or the Python language distribution), which are often referred to with the single term "distribution".

and implicit dependencies of a set of packages as

$$I(S) = \bigcup_{v \in S} i(v)$$

G and M is said to form a valid state when G is valid and all installed distribution packages are implicit dependencies of the manually installed ones:

$$I(M) = V \tag{2}$$

3.1.2 Input

The input is a set R of distribution packages to be removed, where $R \subset V$.

3.1.3 Output

The output is a set K of distribution packages to be kept[†] after uninstallation, where $K \subset V$, $K \cap R = \varnothing$ and that $M' = M \cap K$, $G' = (K, E')$ is a new valid state, where $E' = \{(u, v) \in E \mid u \in K\}$.

3.2 Algorithm

Define direct dependents of a package as

$$b(v) = \{u \in V \mid (u, v) \in E\}$$

implicit dependents of a package as

$$j(v) = \{v\} \cup \bigcup_{u \in b(v)} j(u)$$

and implicit dependents of a set of packages as

$$J(S) = \bigcup_{v \in S} j(v)$$

Let $K = I(M \setminus J(R))$, K is expected to meet the problem's output requirements.

[†]In practice it is often more useful to get $V \setminus K$, but it is easier to formulate reasonings with K.

3.3 Proof of Correctness

The algorithm is correct if and only if the new state of G' and M' satisfies (1) and (2):

$$\forall u \in K, \forall v \notin K, (u,v) \notin E' \tag{3}$$

$$I(M') = K \tag{4}$$

as well as all specified distribution packages are removed:

$$K \cap R = \varnothing \tag{5}$$

Proof. It is trivial that

$$\begin{cases} S \subset X \Longrightarrow I(S) \subset I(X) \\ I(S) = X \Longrightarrow X = I(X) \end{cases}$$

and therefore

$$K = I(M \setminus J(R)) \Longrightarrow K = I(K) \supset I(M \cap K) = I(M') \tag{6}$$

In addition, it is easy to see that

$$\begin{aligned} &M \setminus J(R) \subset I(M \setminus J(R)) \land M \setminus J(R) \subset M \\ &\quad\Longrightarrow M \setminus J(R) \subset M \cap I(M \setminus J(R)) \iff M \setminus J(R) \subset M \cap K \\ &\qquad\Longrightarrow I(M \setminus J(R)) \subset I(M \cap K) \iff K \subset I(M') \end{aligned} \tag{7}$$

From (6) and (7) we have (4) proved.
From $K = I(K)$ in (6), we also have (3) proved as follows:

$$\begin{aligned} \forall v \in I(K), v \in K &\Longrightarrow \forall u \in K, \forall v \notin K, (u,v) \notin E \\ &\iff \forall u \in K, \forall v \notin K, (u,v) \notin E \lor u \notin K \\ &\iff \forall u \in K, \forall v \notin K, (u,v) \notin \{(u,v) \in E \mid u \in K\} \\ &\iff \forall u \in K, \forall v \notin K, (u,v) \notin E' \end{aligned}$$

Assume that $R \cap K \neq \varnothing$, we get

$$\begin{aligned} &\exists v \in R, v \in K \iff \exists v \in R, v \in I(M \setminus J(R)) \\ &\quad\Longrightarrow \exists v \in R, \exists u \in j(v), u \in M \setminus J(R) \Longrightarrow \exists u \in J(R), u \in M \setminus J(R) \end{aligned} \tag{8}$$

As (8) is false, the reverse which is (5) is true. □

4 Implementation

4.1 Obtaining Dependency Graph

Through the standard library `importlib.metadata`, local dependency information can be obtained trivially as follows:

```
from collections import defaultdict
from importlib.metadata import distributions

from packaging.requirements import Requirement

def dependency_graph():
    vertices, edges = set(), defaultdict(set)
    for distribution in distributions():
        d = distribution.metadata['Name']
        vertices.add(d)
        for r in distribution.requires or []:
            requirement = Requirement(r)
            marker = requirement.marker
            if marker is None or marker.evaluate({'extra': ''}):
                edges[d].add(requirement.name)
    return vertices, edges
```

We then define the functions I and J above as `dependencies` and `dependents` respectively:

```
from collections import deque

def dependencies(edges, packages):
    result, queue = set(), deque(packages)
    while queue:
        v = queue.popleft()
        if v in result: continue
        result.add(v)
        queue.extend(edges[v])
    return result

def dependents(edges, packages):
    egdes = defaultdict(set)
    for k, v in edges.items():
        for i in v: egdes[i].add(k)
    return dependencies(egdes, packages)
```

Manually installed packages M are stored in a text file specific to the environment. The packages to be removed $V \setminus K = V \setminus I(M \setminus J(R))$ are computed as follows:

```
manual = set(file.read_text().strip().split())    # M
must_remove = dependents(edges, distributions)    # J(R)
must_keep = manual.difference(must_remove)        # M \ J(R)
should_keep = dependencies(edges, must_keep)      # K = I(M \ J(R))
should_remove = vertices.difference(should_keep)  # V \ K
```

Operations with side-effects are outsourced to `pip`. The reference implementation can be found on the Python Package Index under the name anage.

5 Conclusion

Through the abstraction given by graph theory, we was able to deduced a rather unexpectedly simple method of managing automatically installed distribution packages that is mathematically proven. Although the proof-of-concept was not production-ready (as in, it did not fully comply with all packaging specification such as *extras* and was lacking certain package managements features), we are confident about the tool will eventually become helpful with future development.