

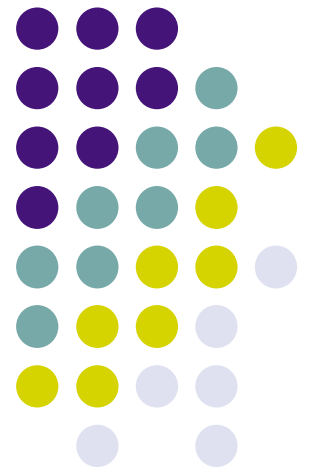
OPIUM: Optimal Package Install/Uninstall Manager

Chris Tucker, David Shuffelton

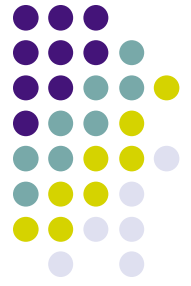
Intellidot, Inc.

Ranjit Jhala, Sorin Lerner

UC San Diego

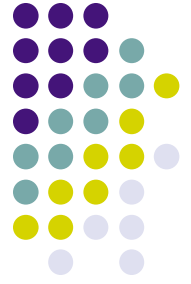


Software Configuration

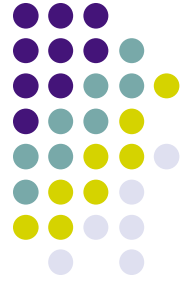


- Dynamic dependencies
 - Widely used in all major operating systems
 - Occur at both library (dll, so, etc.) level and application (perl, gcc, etc.) level
- Poorly managed, prone to error (e.g. dll Hell)

Joe Bloggs Wants OpenOffice

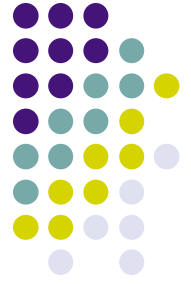


- Joe wants to install OpenOffice
 - OpenOffice requires Java VM
 - OpenOffice requires latest X server
- Q1: What do we need to install?
 - X server has many other dependencies
- Q2: What if there are conflicts?
 - Old version of X server needs to be removed



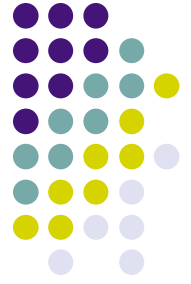
The State of the Art

- APT tool for installing software
- Primary installation mechanism for Debian, Ubuntu, Fink, etc.
- Significant improvement on early RPM
 - Automatic transitive dependency installation!
- Uses simple greedy search algorithm
 - DFS on left-most dependency
 - Limits number of back-tracking steps



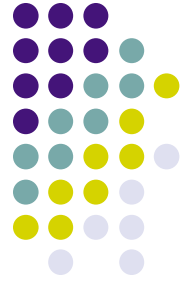
APT Drawbacks

- APT is incomplete
 - Study of 600 real user installation traces
 - 23% encounter incompleteness
- If more than one solution, APT picks arbitrarily
 - Pick smallest size to save download costs
 - Important for users in developing nations
- Arbitrary choice of uninstalls
 - Installing OCaml removed kernel...



OPIUM

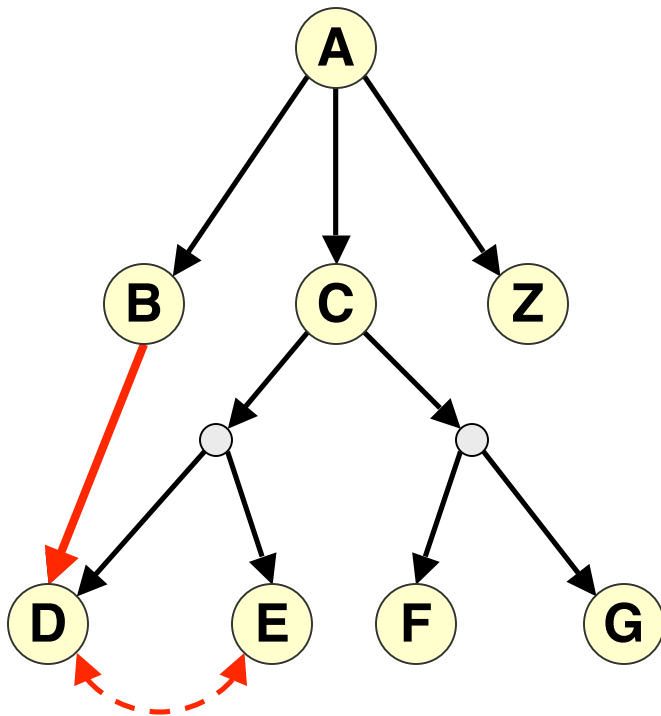
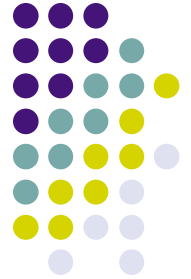
- Uses SAT, PB, and ILP solvers to address limitations of APT
- OPIUM **always** finds a solution if one exists
- OPIUM **optimizes** a user-supplied objective function
- OPIUM **uninstalls** the optimal packages according to another user-supplied objective function



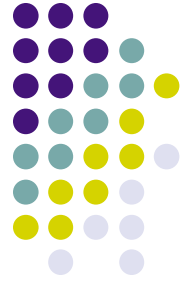
Problem Statement

- To develop OPIUM we studied three problems
 - Install problem: **can** a package be installed?
 - Optimal install problem: what is the **optimal** way to install a package
 - Optimal uninstall problem: what is the **optimal** set of packages to remove

A Simple Distribution

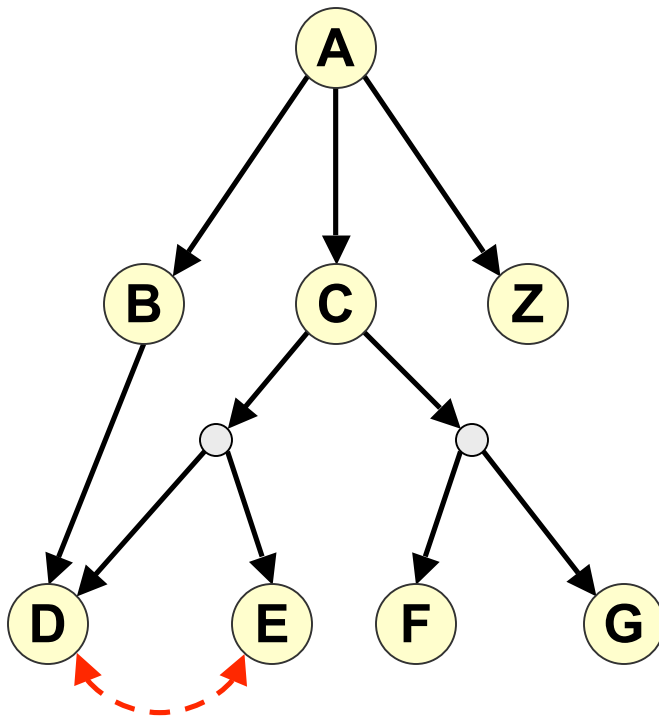


Distribution rule	Constraint
Package: a Depends: b, c, z	$(A \rightarrow B)$ $(A \rightarrow C)$ $(A \rightarrow Z)$
Package: b Depends: d	$(B \rightarrow D)$
Package: c Depends: d e, f g	$(C \rightarrow (D \vee E))$ $(C \rightarrow (F \vee G))$
Package: d Conflicts: e	$(D \rightarrow \neg E)$



Installation Example

- Install *A* onto a system with *Z* already installed



$(A \rightarrow B) \wedge (A \rightarrow C) \wedge (A \rightarrow Z)$	
$\wedge (B \rightarrow D)$	
$\wedge (C \rightarrow (D \vee E))$	Distribution constraints
$\wedge (C \rightarrow (F \vee G))$	
$\wedge (D \rightarrow \neg E)$	

$\wedge (Z)$	System constraint

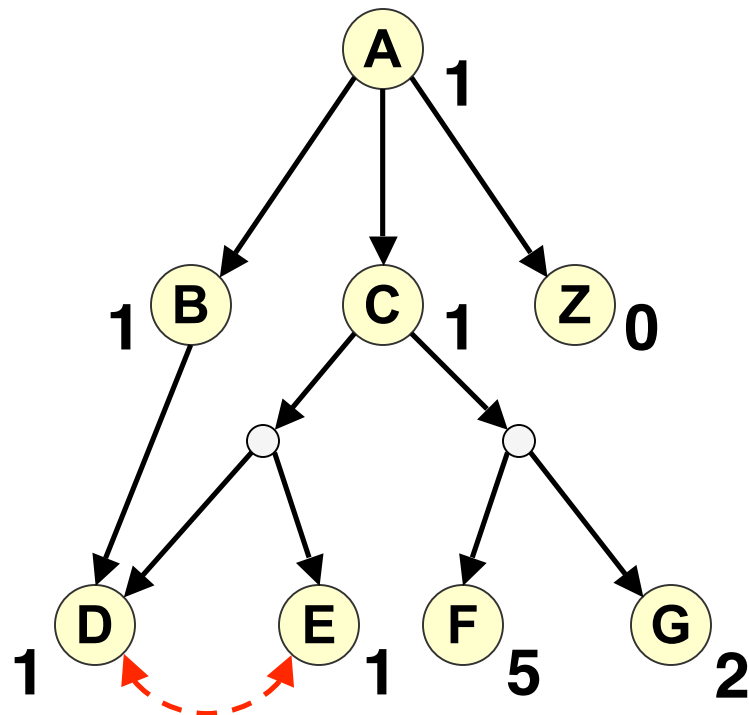
$\wedge (A)$	Installation constraint

SAT Solver



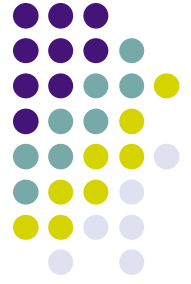
Minimum Install

- Install **A** onto a system with **Z** already installed

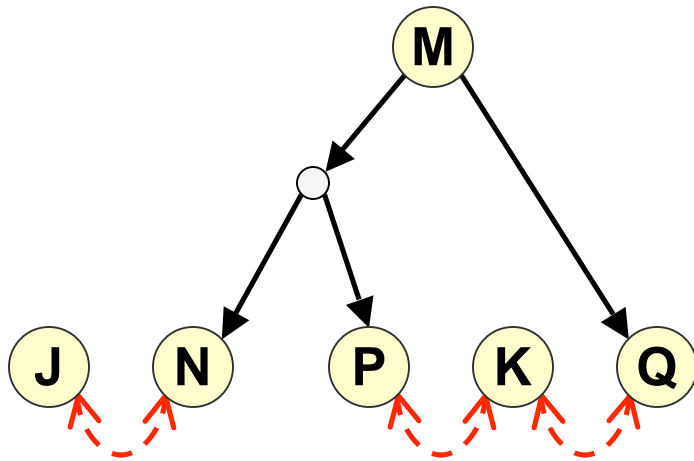


$$\begin{aligned} \min & A + B + C + D + E \\ & + 5F + 2G + 0Z && \text{Objective} \\ \text{s.t} & (A \rightarrow B) \wedge (A \rightarrow C) \\ & \wedge (A \rightarrow Z) \wedge (B \rightarrow D) \\ & \wedge (C \rightarrow (D \vee E)) && \text{Distribution} \\ & \wedge (C \rightarrow (F \vee G)) && \text{constraints} \\ & \wedge (D \rightarrow \neg E) \\ & \wedge (Z) && \text{System constraint} \\ & \wedge (A) && \text{Installation constraint} \end{aligned}$$

PB Solver



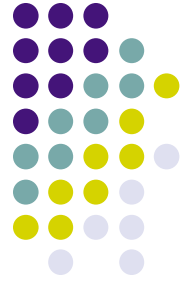
Uninstall



- **J, K** installed, now install **M**
- Naïve approach removes both **J** and **K**
- Optimize for number of packages removed

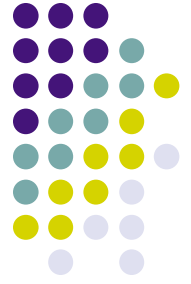
max $J + K$
s.t. $(M \rightarrow (N \vee P)) \wedge (M \rightarrow Q)$
 $\wedge (N \rightarrow \neg J) \wedge (P \rightarrow \neg K) \wedge (Q \rightarrow \neg K)$
 $\wedge (M)$

PB Solver



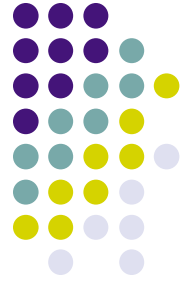
Optimization: Slicing

- Package graphs not fully connected
 - Large parts unreachable
 - Eliminate unreachable packages
 - Does not affect correctness
- Solver performance necessitates slicing
 - Slice walks graph adding packages it encounters
 - Typical slice is about 2,000 (of 20,000) packages



Evaluating OPIUM

- Compare OPIUM to APT
 - Quantify benefits of OPIUM
- Evaluated three measures
 - Running time of OPIUM vs. APT
 - Amount of benefit provided by completeness
 - Amount of benefit provided by optimality

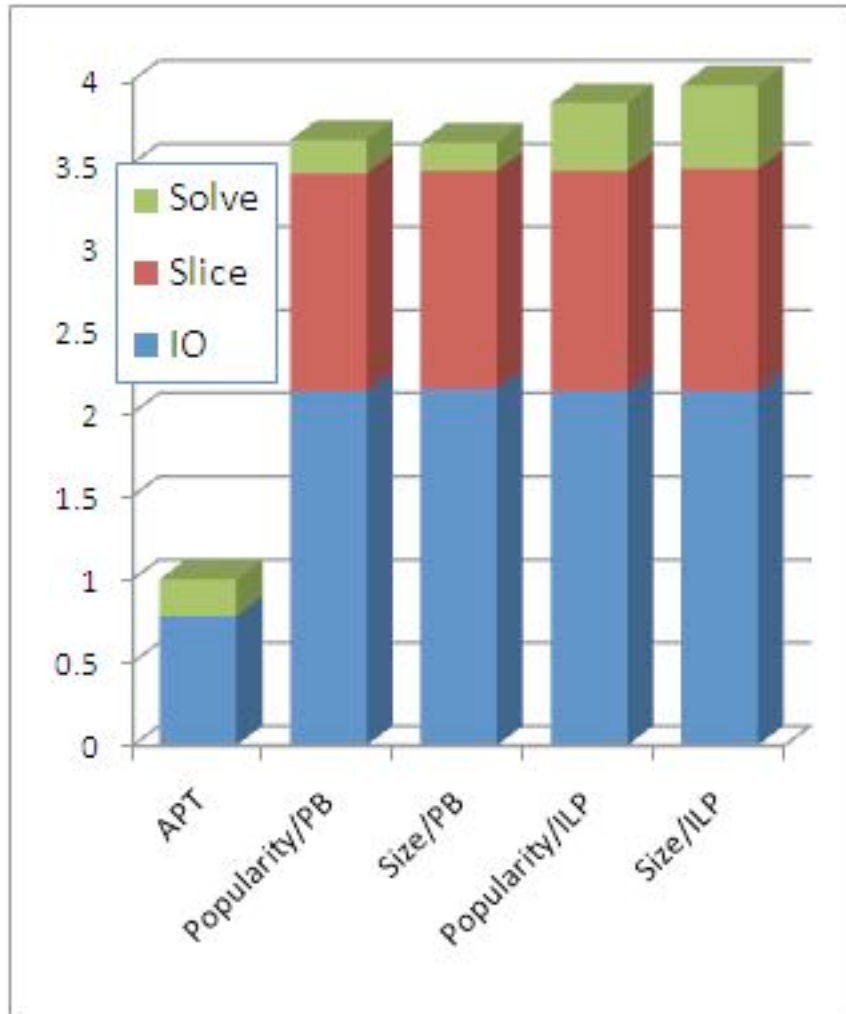


Experiment

- Used the installation histories of 600 real users of the Linspire Linux OS
 - Over 50,000 installation attempts
 - Over 4gb of package descriptions
- Compared APT to OPIUM with different objective functions and back-end solvers
 - Large task, even if running just APT
 - Used 100 machines from the FWGrid cluster

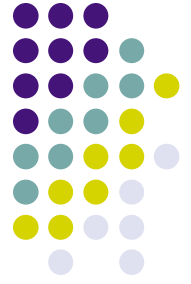


Running Time of OPIUM

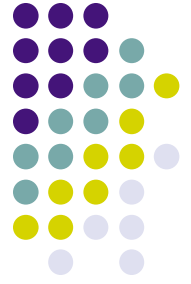


- Cost function doesn't change runtime
- PB solver **faster** than APT, and is optimal!
- Read/slice slow, but not yet optimized
- Smaller downloads offset longer calculation time
 - @ 10kbps, OPIUM is on average faster

Completeness and Optimality



- Completeness eliminates installation failures facing one quarter of APT users
 - Enormous impact on perceived quality
- Install optimality saves user's bandwidth
 - Up to 129mb per install attempt
 - Big issue for users paying per-byte/on slow connection
- Optimality at uninstall dramatically improves stability
 - APT removal of kernel is catastrophic
 - Can weight what to uninstall however we like



Summary

- OPIUM uses SAT/PB/ILP solvers to find best possible solution to dependency problem
- Effectively deals with NP-completeness
 - SAT solvers have real-world applications!
- Allows for guaranteed QoS claims
 - A valid package will install on *any* system
- Application in other dynamic configuration environments
 - Easy to implement
 - Allows for contextualized decision making



Questions?