Solving the Station Repacking Problem

Alexandre Frechette et al.

Sankha Narayan Guria

March 27, 2018

University of Maryland

Introduction

Introduction

- The Federal Communications Commission (FCC) organized an *incentive auction*.
- Aim to transfer billions of dollars of radio spectrum from TV broadcasters to mobile network operators.
- Remaining broadcasters are packed into a lower and narrower spectrum.

Design

- The process is designed as a descending clock auction.
- First auction is called a *reverse auction* allows broadcasters to bid for right sell spectrum to the FCC.
- Second auction called a *forward auction* allows networks to bid for the spectrum made available previously.
- If price of *forward auction* doesn't exceed the *reverse auction*, the FCC lowers the clearing target and starts from beginning.
- Otherwise, remaining broadcasters are assigned to new channels. Consequently we have to solve hundreds of thousands of such repacking problems.
- Repacking problem is NP-complete. Performance is important as every failure to solve a feasible repacking problem is a lost opportunity to a lower price offer.

Formulation

Problem

$$\texttt{station}: s \in S$$
$$\texttt{channel}: c_s \in C \subseteq \mathbb{N}$$
$$\texttt{forbidden}: I \subseteq (S \times C)^2$$
$$\{(s, c), (s', c')\} \in I$$
$$\texttt{domain}\ D : S \to 2^{\overline{C}}$$

Problem

Find $\gamma : S \to \overline{C}$ such that

$$\gamma(s) \in D(s) \text{ for all } s \in S$$
$$\gamma(s) = c \implies \gamma(s') \neq c' \forall \{(s,c),(s',c')\} \in I$$

Challenges

- *NP-Complete:* Worst case performance is very bad. But we are interested in good performance in sort of instances generated in reverse auctions.
- *Descending clock:* Repeatedly generate repacking problems by adding s^+ to a set of S^- provable repackable stations $\gamma^- : S^- \to C$. The new problem $(S^- \cup \{s^+\}, C)$ needs to solved each time.

Mixed Integer Programming

$$\texttt{state variable}: x_{s,c} \in \{0,1\}$$
$$\texttt{exactly 1 channel}: \sum_{c \in D(s)} x_{s,c} = 1 \forall s \in S$$
$$\texttt{interference}: x_{s,c} + x_{s',c'} \leq 1 \forall \{(s,c),(s',c')\} \in I$$

SAT Encoding

$$\texttt{boolean variable}: x_{s,c} \in \{\top, \bot\}$$
$$(s, c) \in S \times C$$

$$\texttt{at least 1 channel}: \vee_{d \in D(s)} x_{s,d} \forall s \in S$$
$$\texttt{at most 1 channel}: \neg x_{s,c} \vee \neg x_{s',c'} \forall s \in S, \forall c, c' \neq c \in D(s)$$
$$\texttt{interference}: \neg x_{s,c} \vee \neg x_{s',c'} \forall \{(s, c), (s', c')\} \in I$$

This problem is fed to SAT solvers, with multiple algorithm portfolios.

Optimizations

Incremental Repacking

Local Augmenting

Find neighboring stations of s^+ as $\Gamma(s^+)$. Solve the reduced repacking problem in which all non-neighbors $S \setminus \Gamma(s^+)$ are fixed to assignment in γ^-.

Starting Assignment

Assign stations in γ^- to their channels and assign a random channel to s^+. If solution exists near such an initialization, we'll find it more quickly.

Problem Simplification

Graph Decomposition

The set of stations considered in a particular problem instance usually makes the interference graph disconnected. We can solve for each component separately one by one.

Underconstrained Station Removal

We can delete stations for which no matter how every other station is assigned there will exist one station on which can be packed to a channel. Verifying this is difficult, so a sound but incomplete heuristic is used - comparing a station's available channels to its number of neighboring stations. Deleting the station often increases the number of components, causing a speedup.

Hydra Portfolio

Incremental solvers solve many instances extremely quickly, allowing remaining solvers in the portfolio to concentrate their efforts elsewhere.

- DCCA-presat
- DCCA+
- clasp-h1
- clasp-h2

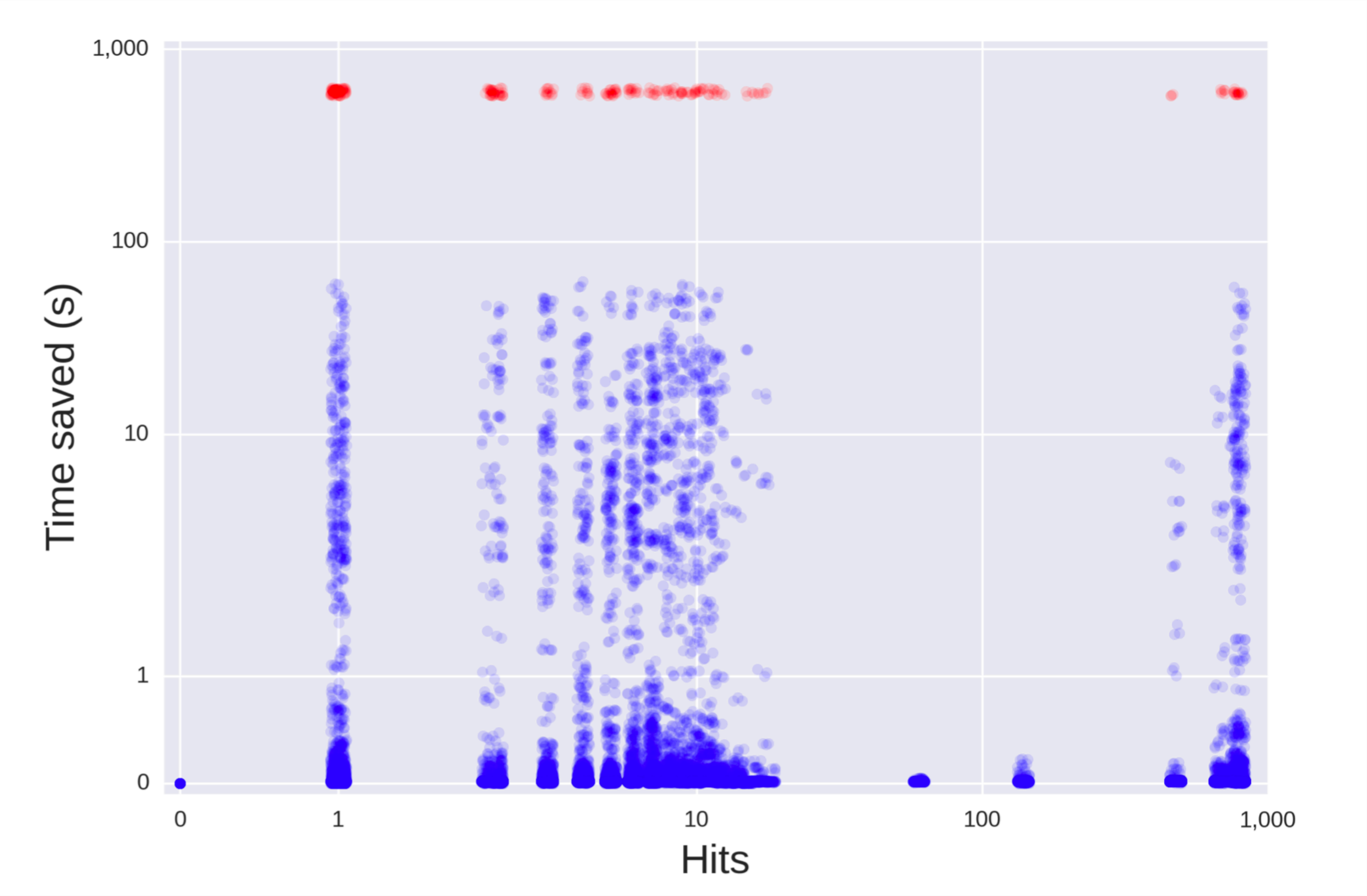
Caching

- If S is packable, $S' \subseteq S$ is packable.
- If S is unpackable, $S' \supseteq S$ is unpackable.

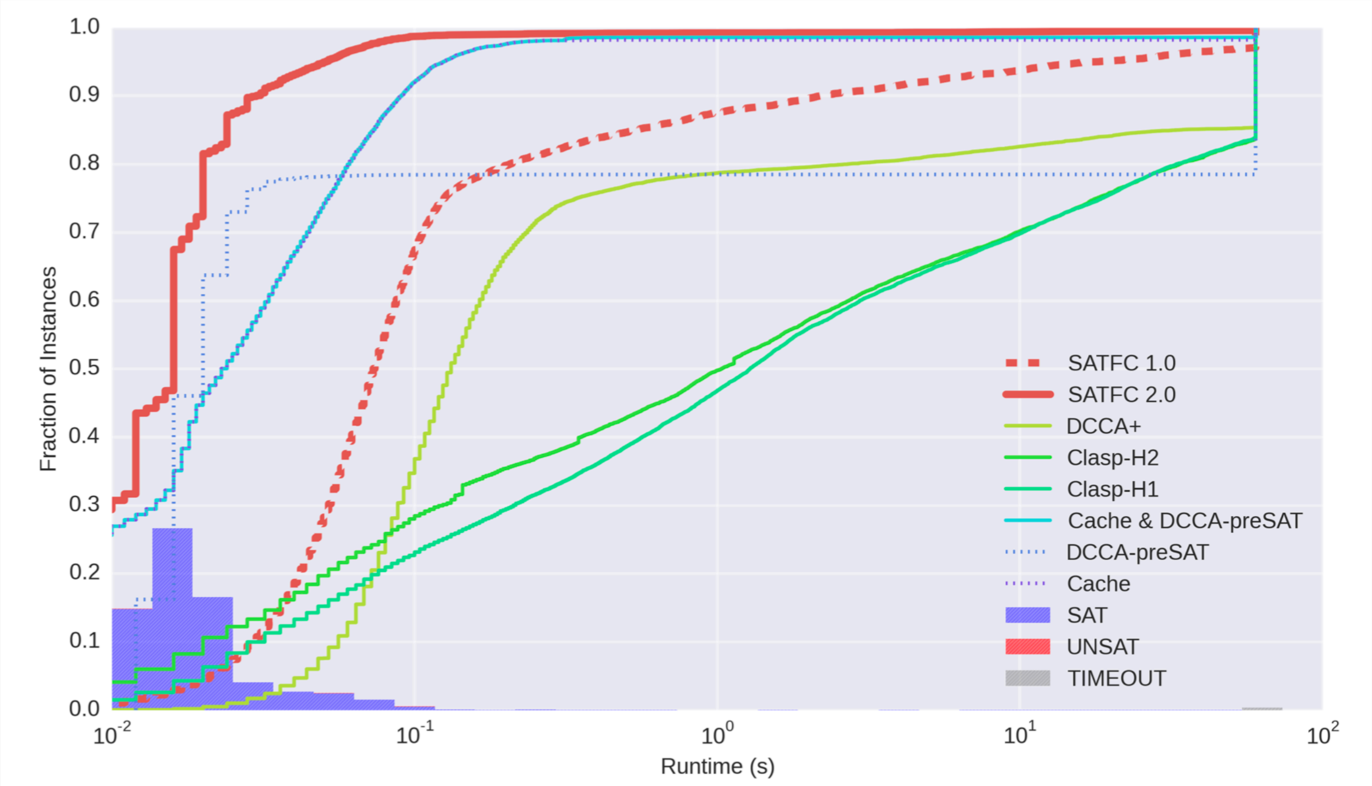
The implement these two principles as non-trivial caching algorithm and implement a *feasible* and *infeasible* containment cache.

Results

Caching



Performance



Questions?