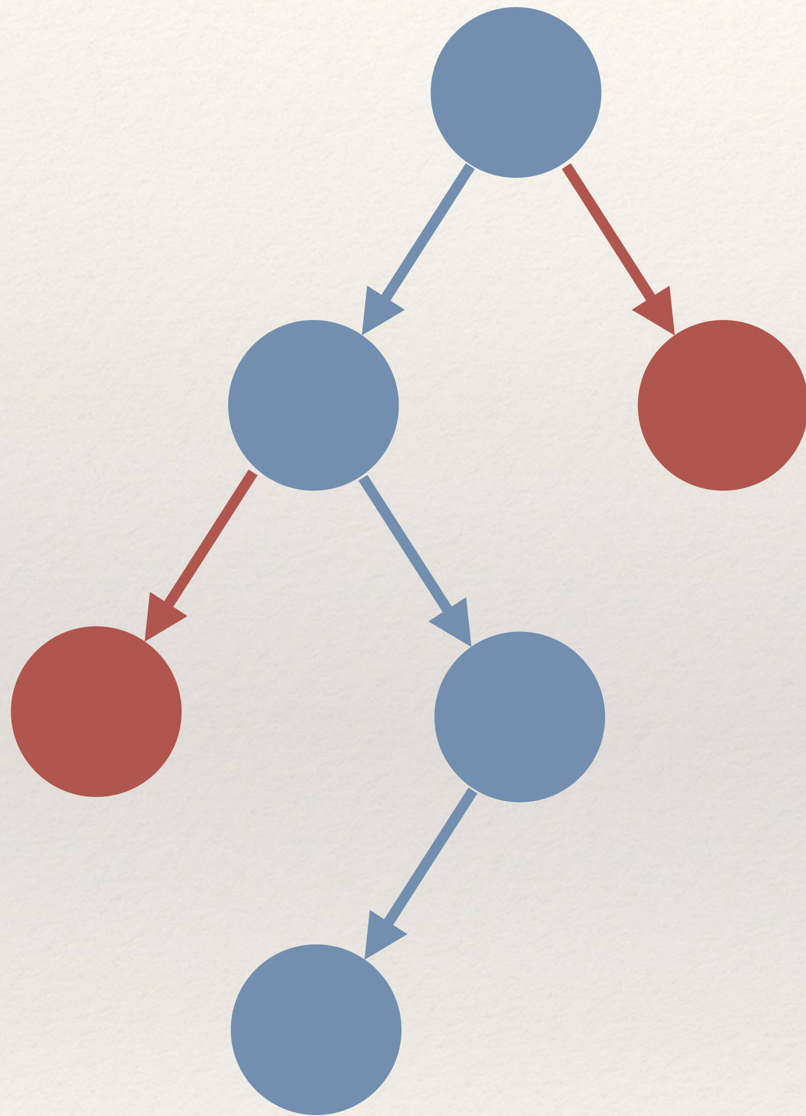
A fix-propagate-repair heuristic for MIP

Domenico Salvagnin
Matteo Fischetti
Roberto Roberti

DEI, University of Padova

Fix-and-propagate



- Iterative constructive heuristic that simulates a dive in B&B tree
- Constraint propagation after each fixing, no LPs *[except for continuous at the end of dive]*
- 1-level backtrack very easy to support
- Limited DFS **not** much more difficult to implement
- Cheap **but** very sensitive to variable/value fixing strategy

Fixing Strategies

- Many different approaches for fixing strategies:
  - static vs dynamic
  - emphasis on good quality vs feasibility only
  - LP-free vs LP-dependent

Variable Strategies

Name	Description
LR	left to right as they appear in the formulation
type	group by type (binaries, integers, continuous)
typecl	by type + use clique cover to rearrange binaries
random	by type + random shuffle within each type
locks	decreasing variable locks within each type
cliques	based on clique cover and relaxation solution
cliques2	based on clique cover and relaxation solution

Greedy Clique Cover

- first process equality cliques (must be disjoint)
- then *greedily* cover remaining binaries:
  - count how many binaries are covered by each clique
  - assign each binary to the largest clique covering it
  - count again (using only the selected ones)
  - local adjustments (e.g., switch variable to a larger clique)
  - finally sort used cliques by size: that gives the cover

Variable Strategies: cliques

- *greedly* compute clique cover for all variables
- within each clique in the cover:
  - weight variables based on the zero-objective corepoint solution
  - sort them according to a *weighted discrete random distribution*

Variable Strategies: cliques2

- based on a clique cover constructed based on the zero-objective LP solution
- loop over cliques in the problem:
  - skip fixed variables and non-tight cliques
  - pick most positive literal in the clique
  - then the remaining uncovered binaries

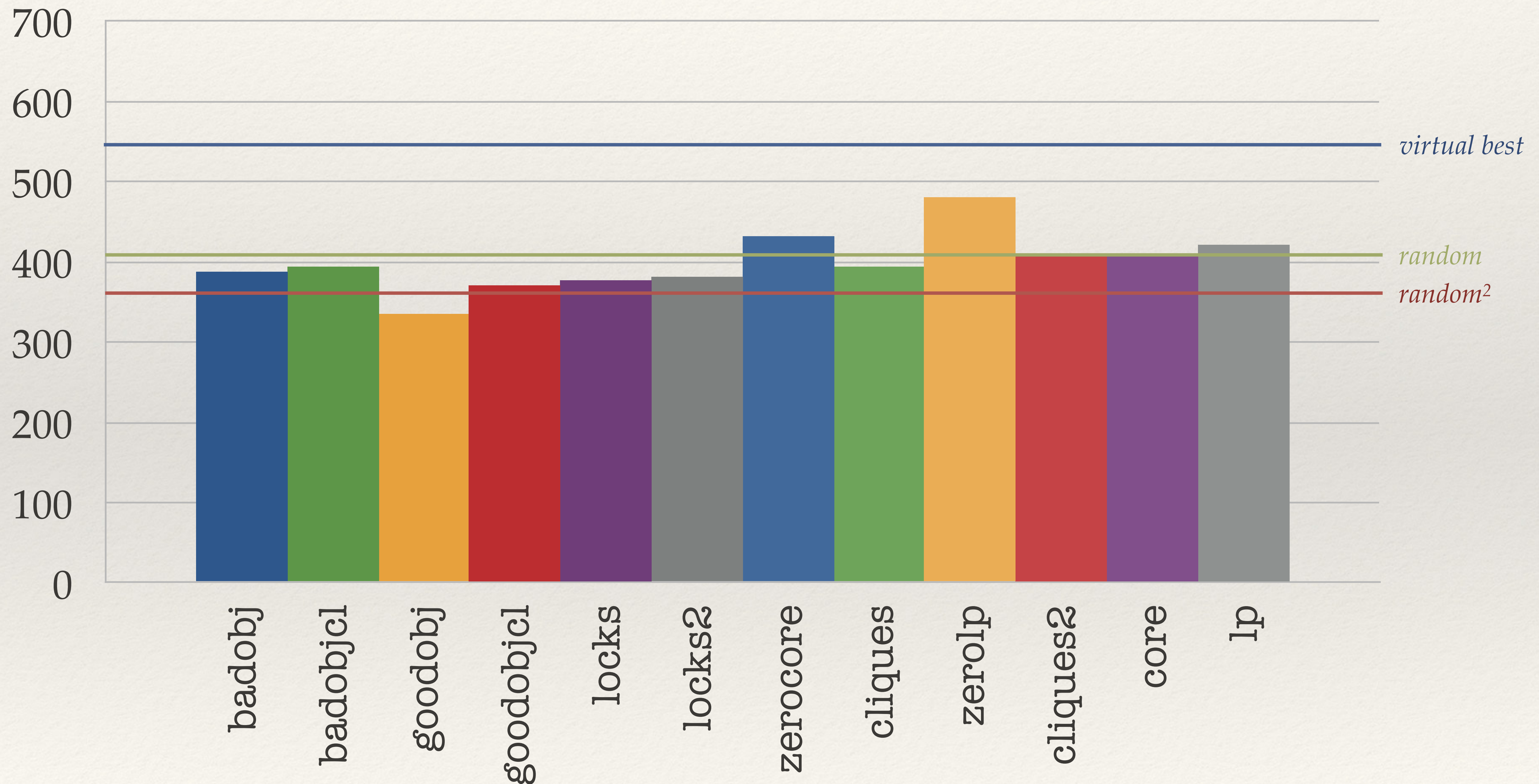
Value Strategies

Name	Description
goodobj	fix toward the objective
badobj	fix against the objective
random	random
loosedyn	compute dynamic locks based on current activities
LP-based	use fractional part of LP value as probability to pick upper bound

Which LP solution?

- We have 2 independent choices:
    - zero-obj vs original objective
    - simplex vs barrier
- Each has pros and cons
- We test all four combinations

MIPLIB 2017 Results



single threaded run - 10min timelimit - 3 seeds on each instance

WalkSAT

- Repair heuristic starting from a complete infeasible assignment
- Iteratively flips variables with a mix of ***greedy*** and ***random*** strategy:
  - Pick a violated clause (uniformly) at random
  - If possible to fix the clause ***without*** breaking any other, always do it
  - Otherwise, flip a variable in the clause:
    - completely at random with probability p
    - among those of ***minimal damage*** with probability $1-p$
- Much less sensitive to initial mistakes…
- …but difficult to recover wildly infeasible solutions

Fix-Propagate-Repair

- Combine fix-and-propagate with WalkSAT-like repair
- Try to repair infeasibilities as soon as they appear
- Need to generalize WalkSAT logic:
  1. from clauses to general linear constraints (and variables)
  2. from complete to partial assignments
- Also need to make it converge faster!

Extensions to MIP

- Extension to partial assignments and linear constraints:
  - cumulative violation not the same as violated count
  - flipping a binary in a violated constraint does not necessarily fix it (could even do worse): skip those
  - violation of a constraint based on the minimum/maximum activities computed from the current partial assignment
  - non-binary variables (not necessarily fixed) are shifted

WalkMIP

- Iteratively shifts variables:
  - Pick a violated constraint (uniformly) at random
  - Consider variables that can reduce its violation by shifting
  - If possible to reduce violation in the constraint ***without*** increasing violation elsewhere, always do it
  - Otherwise, select a variable to shift:
    - completely at random with probability ***p***
    - among those of ***minimal damage*** with probability ***1-p***

Repair Walk

- Very similar in spirit to regular WalkSAT
- Can take too long to repair a “small” infeasibility even with a lot of hocus-pocus:
  - tabu list of size 3 to avoid short cycles
  - soft restarts every 10 iterations
  - max 100 steps
- Walk-nature good eventually, but not necessarily on a short run
- Also does not exploit constraint propagation during repair :-(

Repair Search

- Embed WalkSAT-like repair moves into a search scheme
- Basically a search over the sequences of repair moves
- Limited DFS again :-)
- In this DFS we can actually propagate the implications of the repair moves we have done so far (and only those!)
- A blend of WalkSAT and shift-and-propagate

Repair Search (details)

- We now have two domains:
  - an infeasible domain associated to the partial solution we are trying to repair
  - a feasible domain associated to the current sequence of repair moves (and their implications)
- Need to figure out how to:
  - Find a repair move based on the current domains
  - Turn a repair move into a repair disjunction (for search)
  - Apply repair move ***implications*** to the current infeasible domain
- Not just for binary variables, but for integers and continuous as well!

How to apply changes across domains

	Partial Solution	Repair Domain	Effect	Description
binary	0	0	0	nothing
	1	0	0	flip
	{0,1}	0	0	fix
non-binary	[1,3]	[4,5]	[4]	shift and fix
	[4,5]	[1,3]	[3]	shift and fix
	[1,4]	[2,5]	[2,4]	intersect

- After sync:
  - repair domain always includes partial solution domain
  - partial solution is minimally modified to achieve this

Repair Search (more details)

- WalkSAT logic still in place (pick violated constraint, etc…)
- Repair moves (= shifts) are computed w.r.t. the current repair domain
- In order to turn a move into a disjunction:
  - trivial for binaries
  - for non-binaries we compute the extremes of the shifted domain and use one of the two for branching (a la spatial branching)

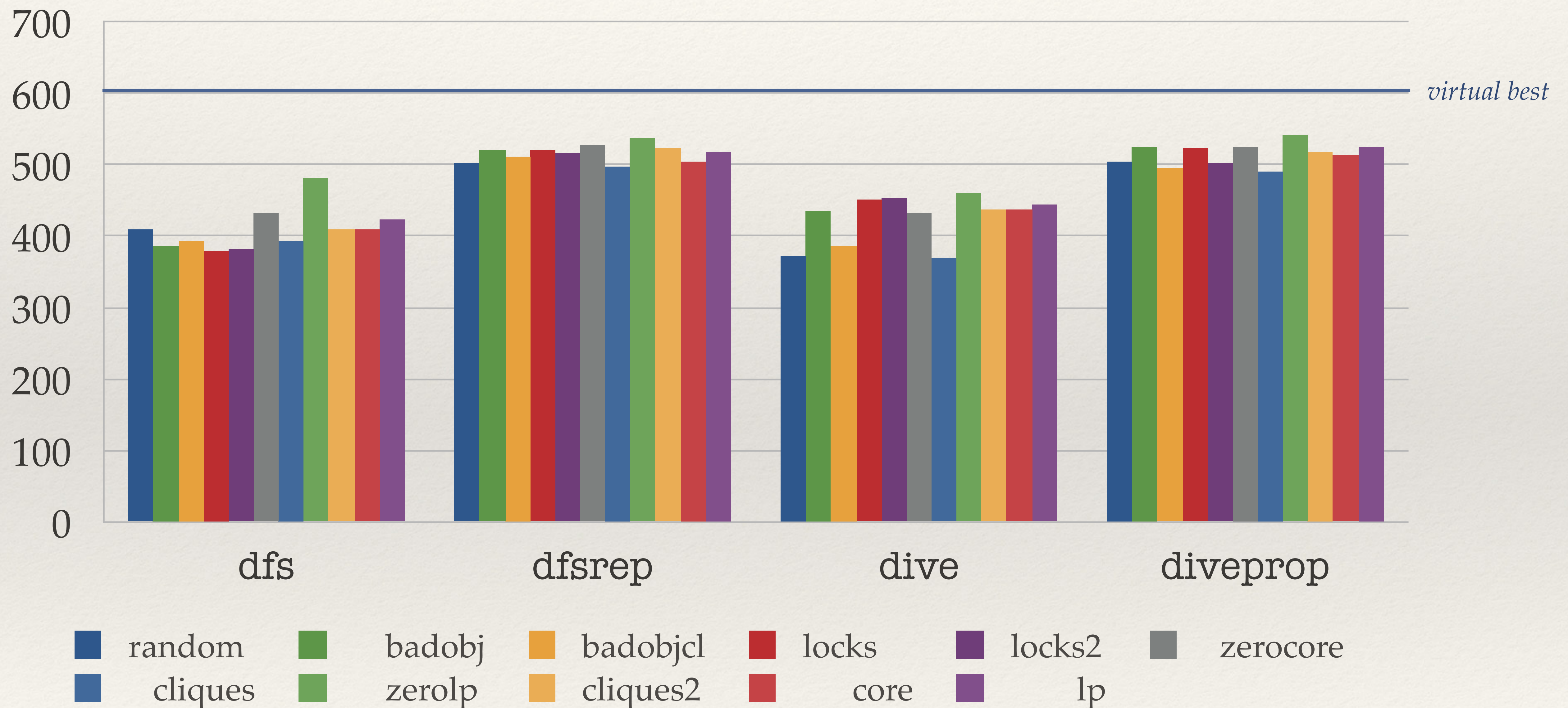
Repair Search (even more details)

- We no longer need a tabu list (DFS takes care of cycles)
- But we need some advanced jumping to mitigate DFS…
- Backtrack to the best open node in case of lack of progress, with the same thresholds as soft restarts in the walk case
- Still strict limits: max 100 repair nodes
- Intuition: if WalkSAT is quickly able to repair infeasibility, repair DFS should do as well

Overall Implementation

- MIP presolve applied at the beginning to get smaller model and tighter (actually, always finite) bounds on all variables
- Both constraint propagation and repairs work on the same data structures and update activities incrementally
- Deterministic work limits based on #nnz for constraint propagation and LP solves
- All tested variants are different parametrizations of the same code:
  - dfs: regular limited DFS with propagate but no repair
  - dfsrep: dfs + repair
  - dive: dive with repair but no propagation nor backtrack
  - diveprop: dive + propagation

MIPLIB 2017 Results

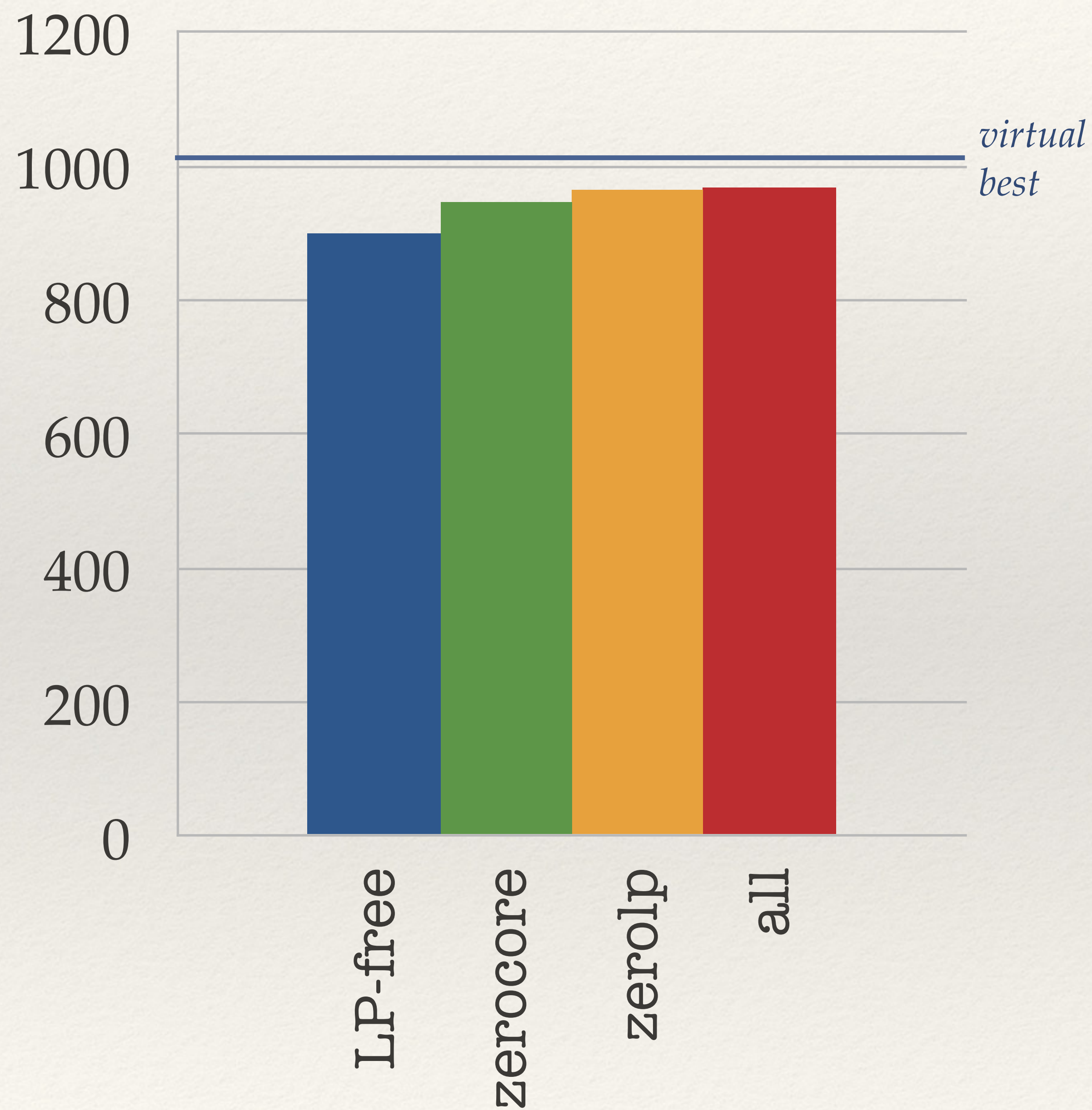


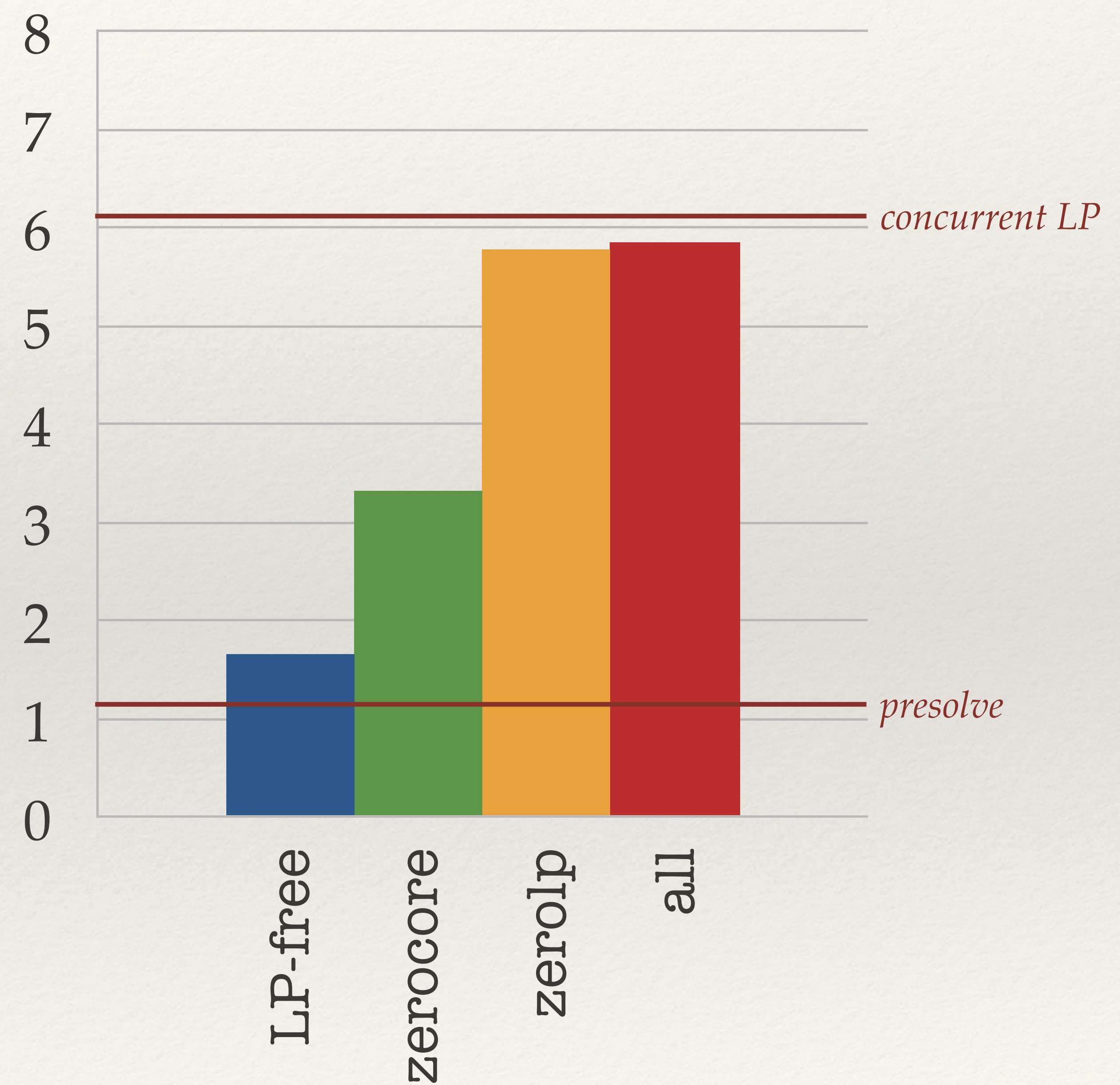
single threaded run - 10min timelimit - 3 seeds on each instance

Overall Strategy

- All methods inherently sequential
- Exploit parallel hardware with simple portfolio approach:
  - First cheap strategies that do not depend on an LP reference solution
  - Then those that depend on the zero-objective core point solution
  - Then those that depend on the zero-objective vertex solution
  - Finally solve LP with original objective (concurrent solve) and run remaining strategies

Final Results





parallel run (4 threads) - 10min timelimit - 5 seeds on each instance