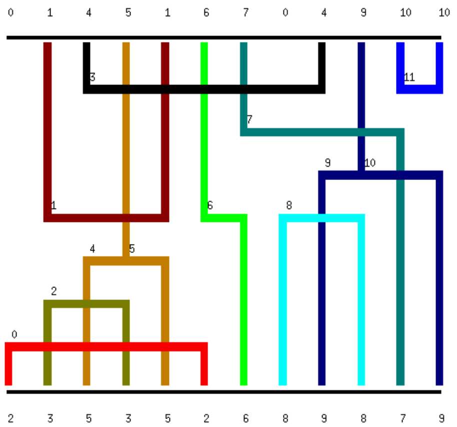
Latest Results with STAGE

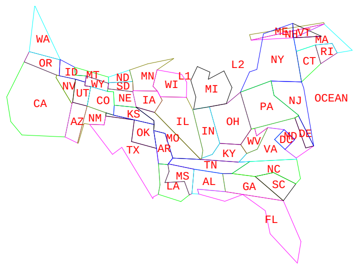
Justin Boyan
jab@cs

COMBINATORIAL OPTIMIZATION

Goal: minimize $Obj : X \rightarrow \Re$
over a configuration space X.



CHANNEL ROUTING
connect nets, minimizing channel width

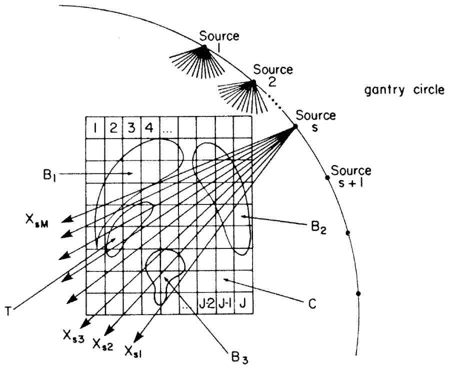


MAP LAYOUT
meet target areas, minimizing distortion

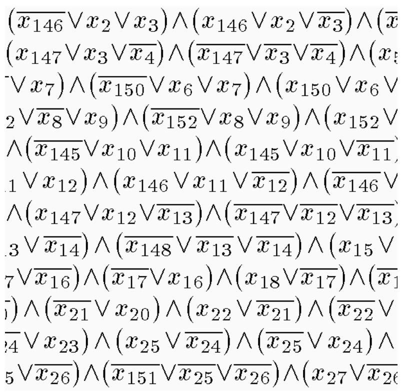
1-D BINPACKING
pack items, minimizing
# of bins

COMBINATORIAL OPTIMIZATION

BAYES NET LEARNING
determine graph structure that best fits data



RADIOTHERAPY
meet target radiation doses, minimizing structural damage

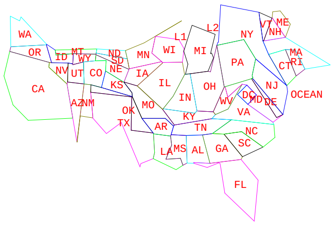


SATISFIABILITY (MAXSAT)
assign 0/1 variables to minimize # unsatisfied clauses

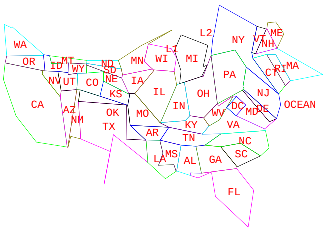
Lacking any cleverer ideas, we resort to stochastic search: HC, SA, GA, TABU, PBIL, WALKSAT, ...

NINE HILLCLIMBING RUNS

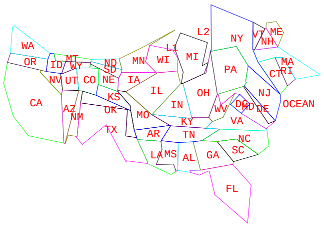
0.1612



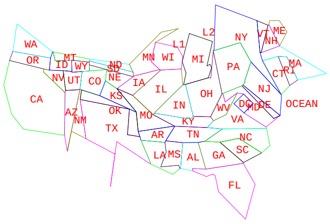
0.1908



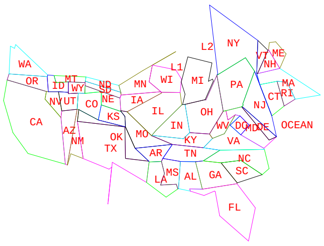
0.1790



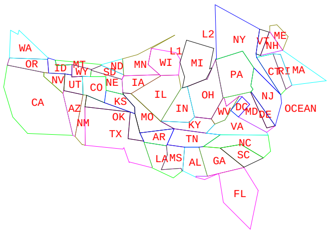
0.2341



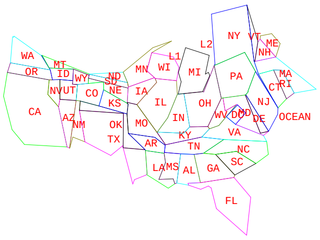
0.2224



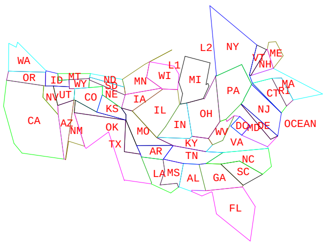
0.1990



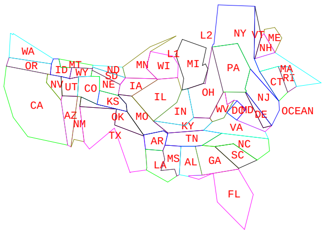
0.1810



0.2114



0.1730

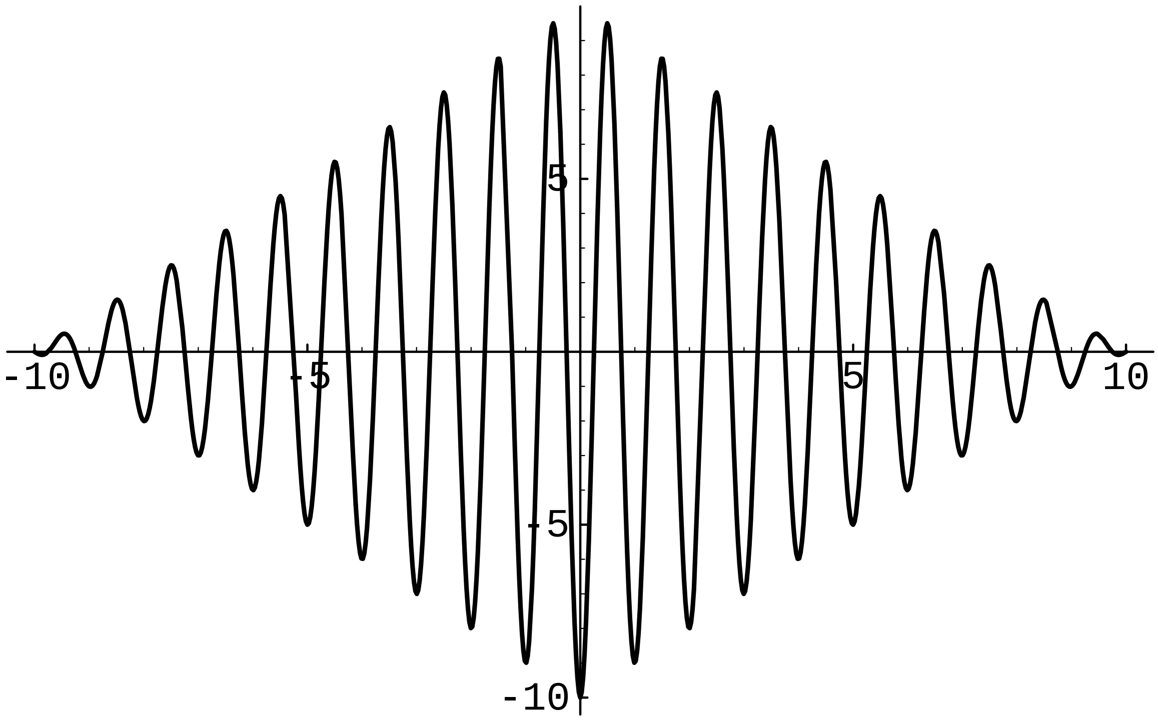


Question:

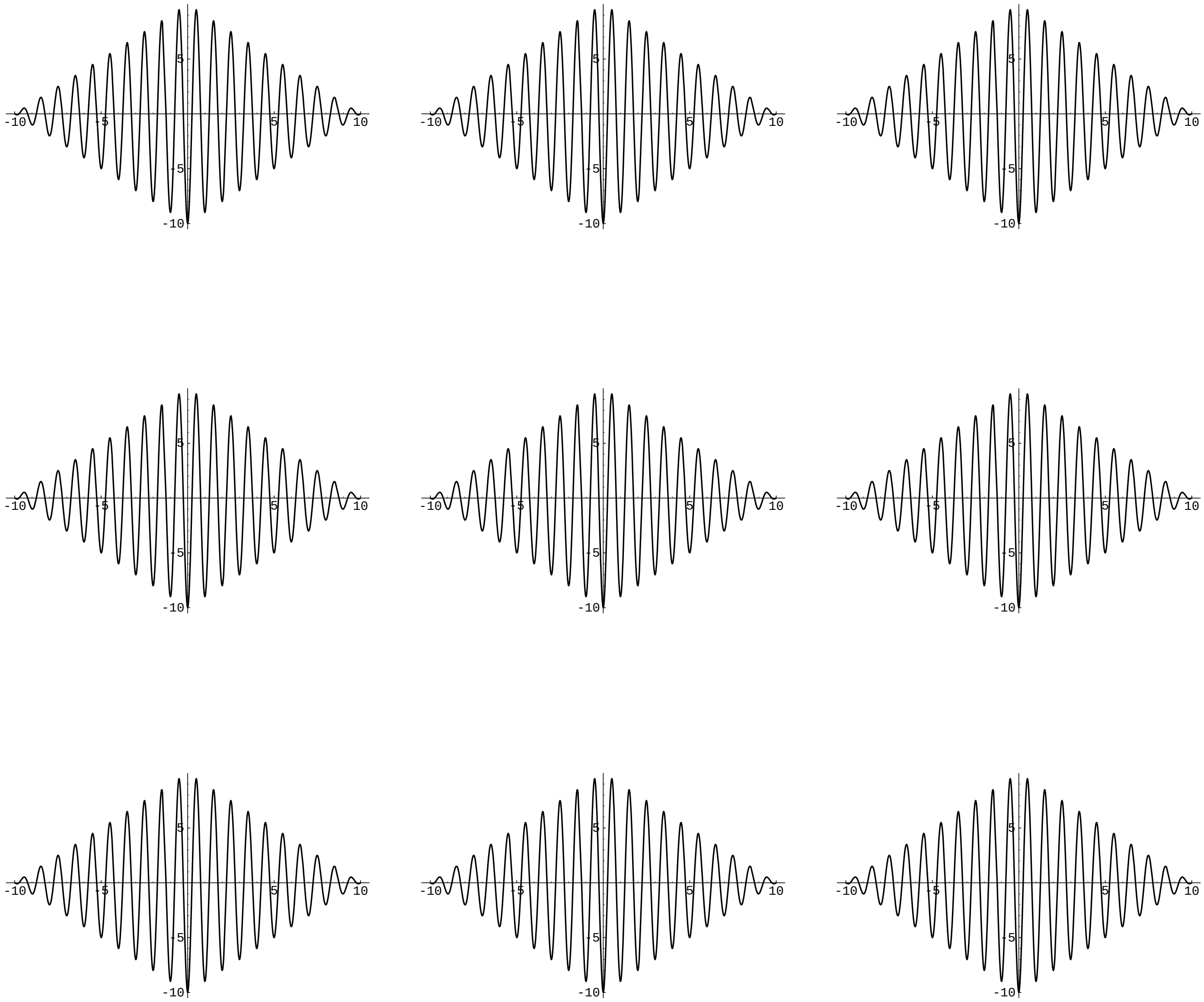
Can we learn from these runs and make the tenth run better?

A SIMPLE (CONTRIVED) EXAMPLE

One-dimensional minimization problem,
tiny discrete moves are allowed.

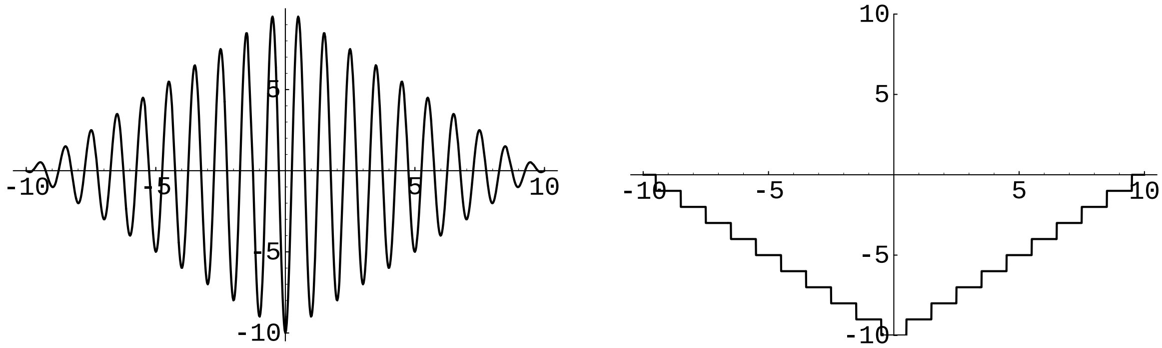


NINE HILLCLIMBING RUNS



WHAT CAN BE LEARNED?

We can learn to predict the results of a search from features of the starting state.

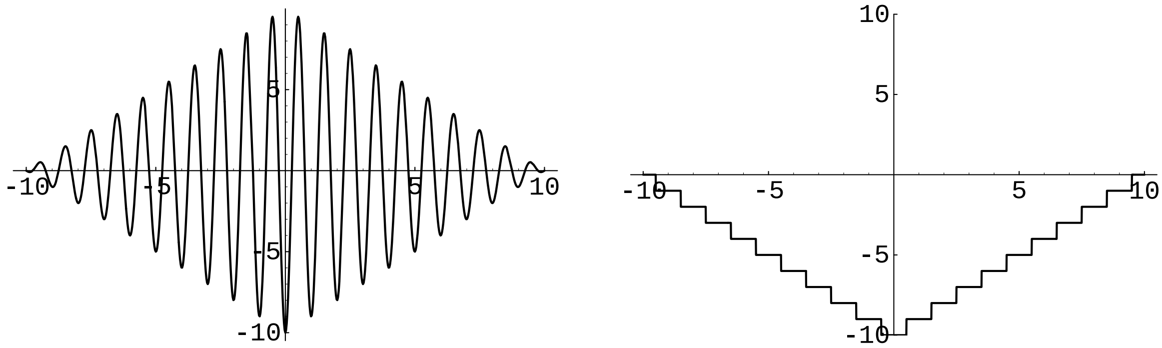


$Obj(x)$ $\Rightarrow$ $V^{HC}(x)$

$V^{HC}(x)$ = expected eventual $Obj(x)$ when starting at state x and doing HC.

PREDICTING OPTIMIZATION RESULTS

$$V^\pi(x) \ = \ \text{expected eventual} \ \ Obj(x) \ \ \text{when starting at state } x \text{ and following search procedure } \pi.$$



$Obj(x)$ $\Rightarrow$ $V^\pi(x)$

- V^π reveals the global structure of the local optima
- V^π can make use of high-level state-space features
- V^π can be approximated from sample trajectories by supervised learning.

  Insight: if π is Markov (memoryless), we can train on *all* states of the trajectory.

- Extrapolation from function approximator can indicate promising unvisited regions!

USING THE PREDICTIONS

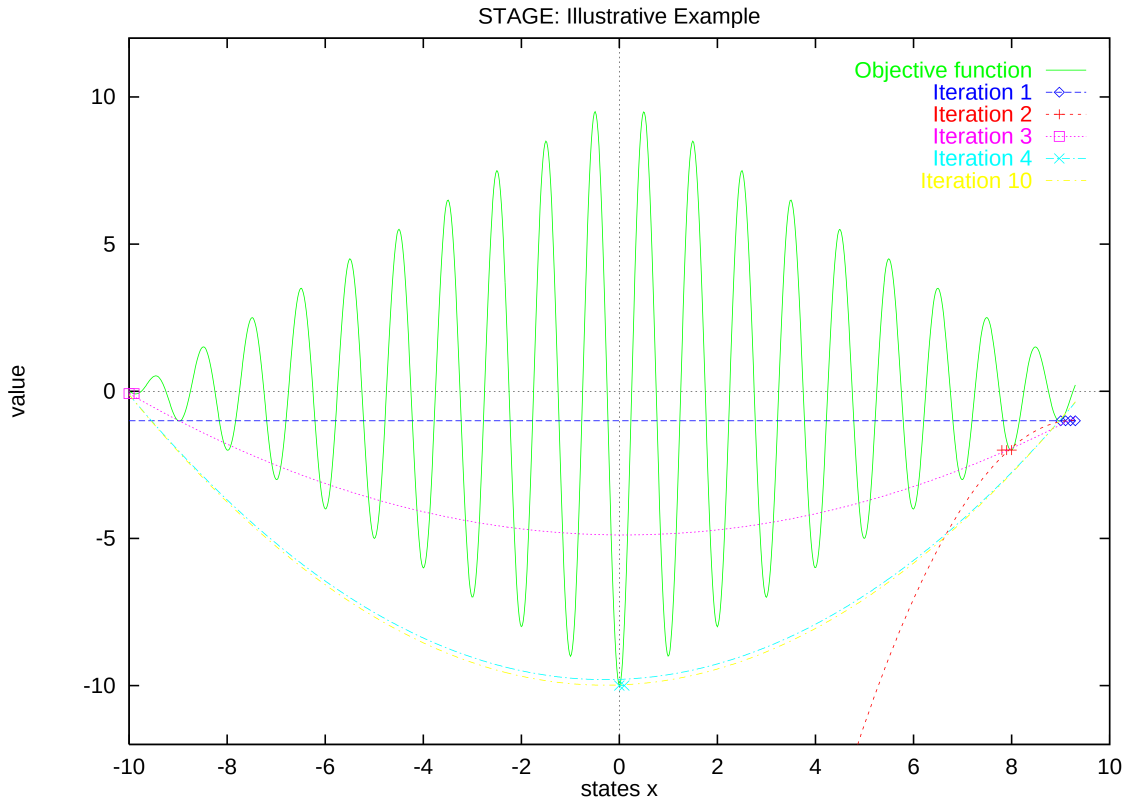
STAGE alternately optimizes $Obj(x)$ and V^π:

Run π to optimize Obj.

produces new training data for V^π; we retrain our fitter

Perform greedy search on V^π

produces good new starting state for π



STAGE(π) ALGORITHM

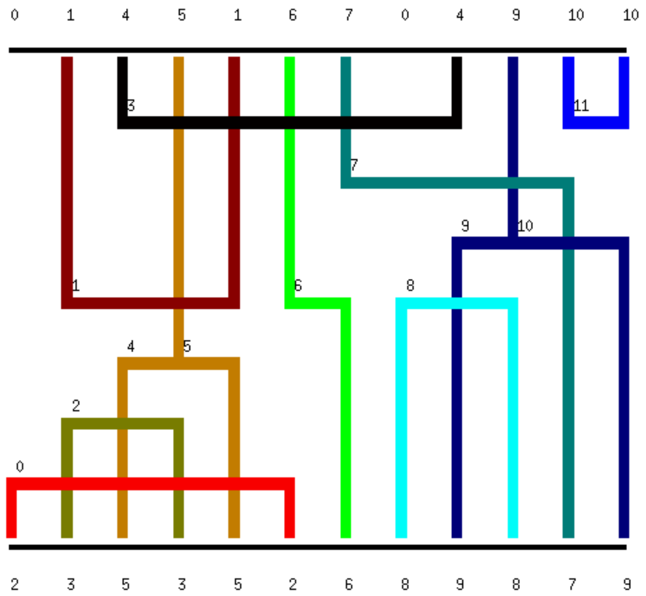
1. **Initialize** the function approximator; let x be a random starting state for search.
2. **Loop** until time runs out:
   - **Optimize Obj using π.** From x, run search algorithm π, producing a search trajectory that ends at a local optimum z.
   - **Train $\mathbf{V}^\pi$.** For each point x_i on the search trajectory, use $\{[\text{features of } x_i] \mapsto Obj(z)\}$ as a new training pair for our function approximator.
   - **Optimize $\mathbf{V}^\pi$ using hillclimbing.** Continuing from z, perform a hillclimbing search on the learned objective function V^π. This results in a new state x which should be a good starting point for π. (If this search accepted no moves, so that $x = z$, then reset x to a new random starting state.)
3. **Return** the best state found.

STAGE IN PRACTICE

To apply STAGE:

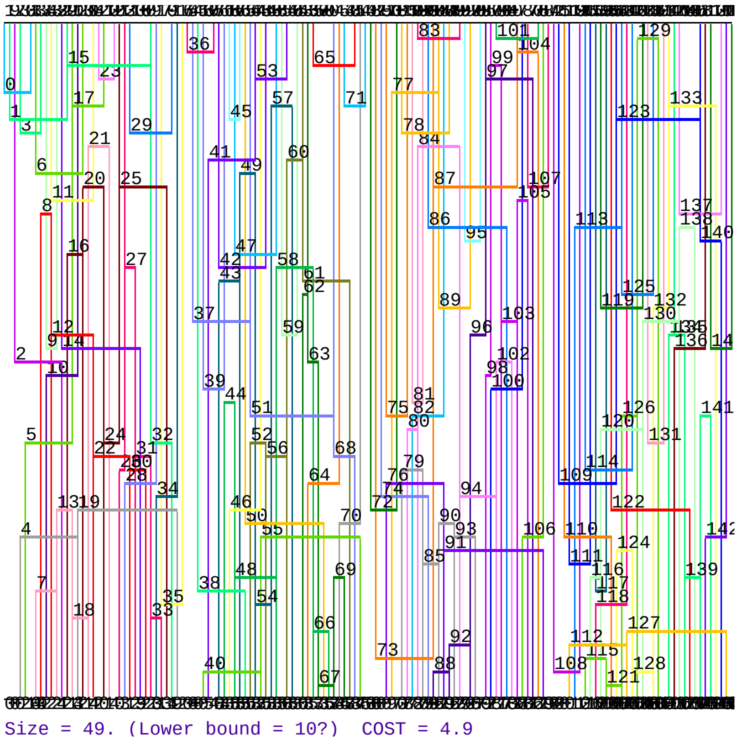
- invent features
  - I use: just a few simple, coarse features
  - Subcomponents of $Obj(x)$ are a natural choice
- pick a function approximator
  - I use: global quadratic regression
- set parameters
  - parameters of π
    (e.g., hillclimbing patience, WALKSAT noise level)
  - method for optimizing V^π
    I use: stochastic hillclimbing, patience=200 or 1000

CHANNEL ROUTING

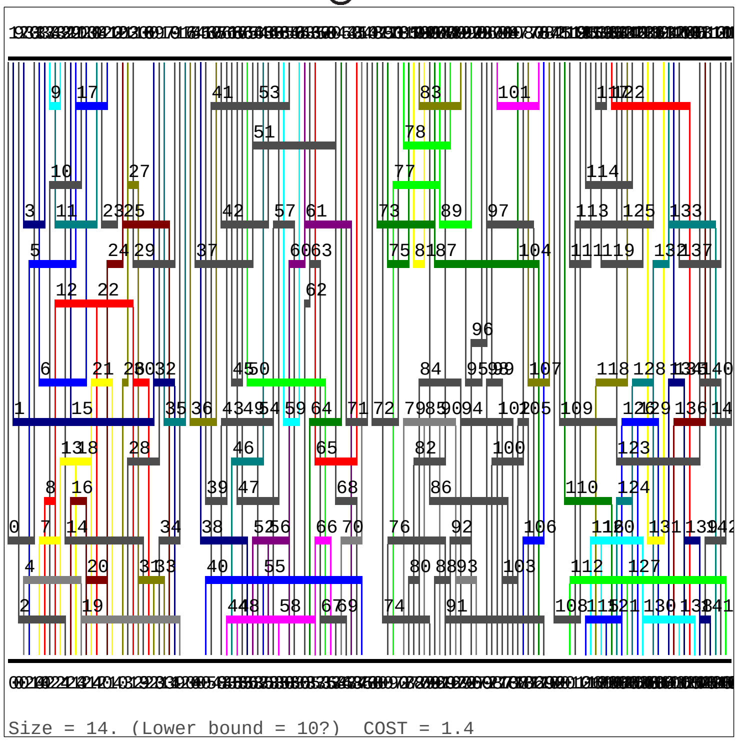


- Configuration space:
  All legal circuit routings (internally represented as a partitioning of vertices in a constraint graph)
- Search moves:
  Perturb the graph partitioning (check for constraint violations)
- Objective function: $Obj(x) =$ circuit size
- Features used:
  $w =$ circuit size, $p =$ length of longest path in constraint graph, $U =$ track sparsity (from [Wong 88])

TYPICAL CHANNEL ROUTING RESULTS

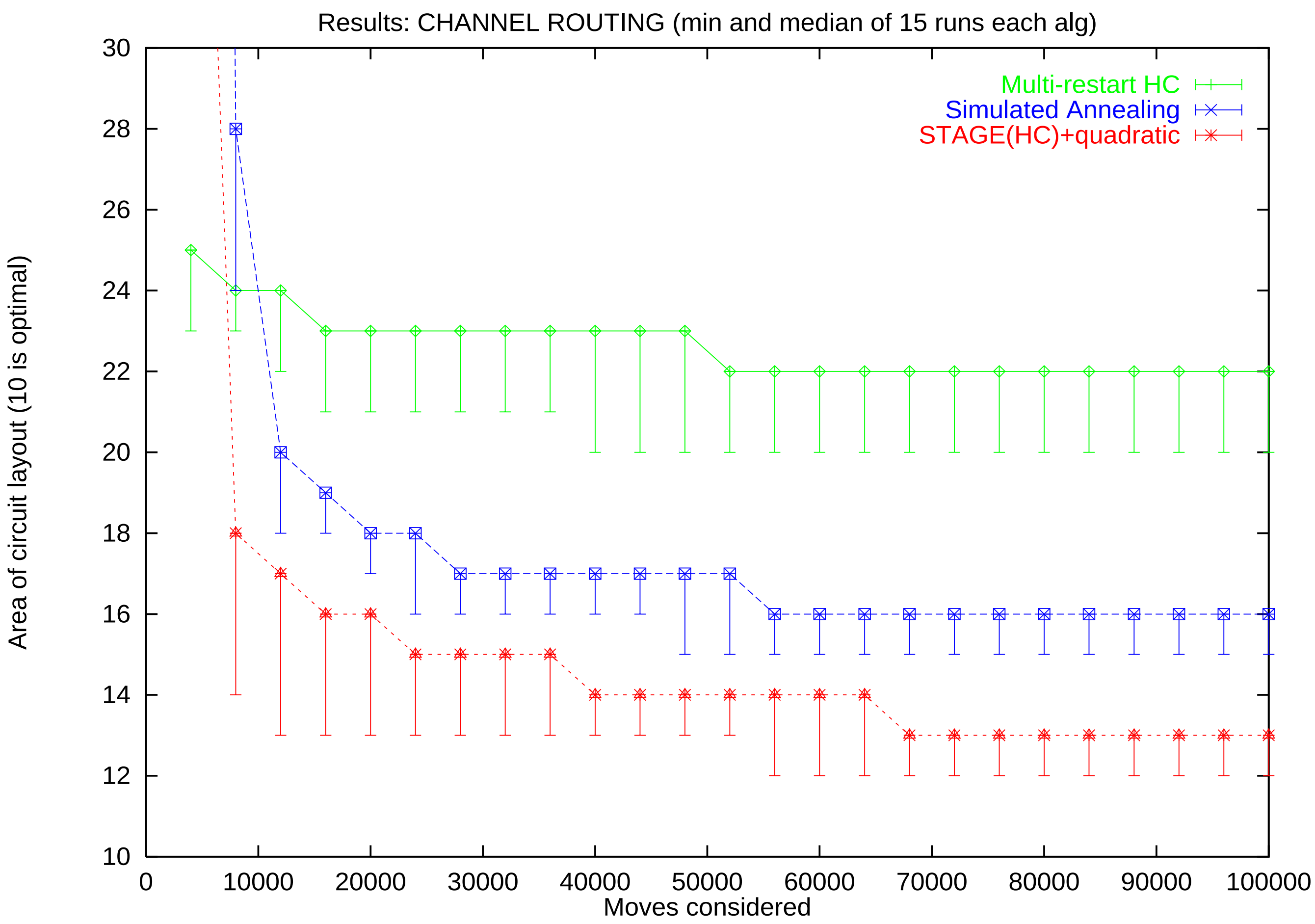


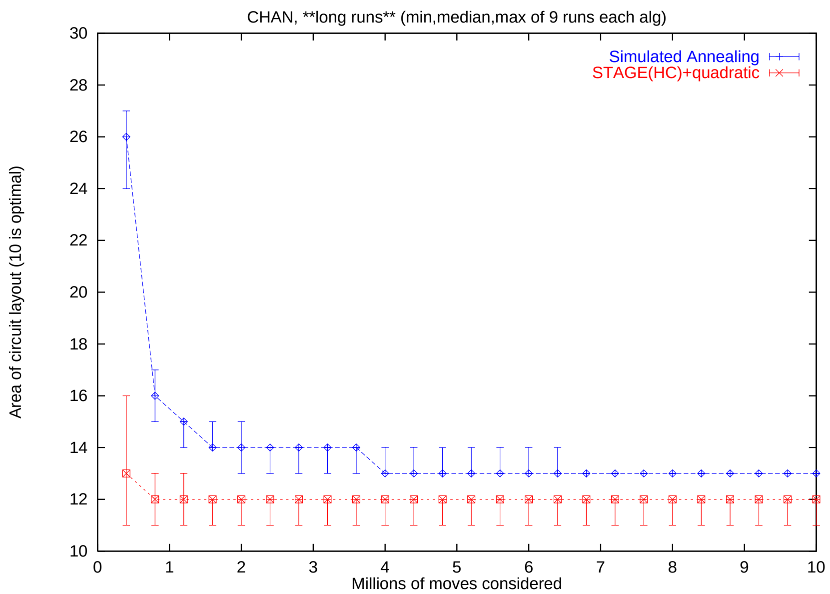
Hillclimbing ⇒ 49 tracks



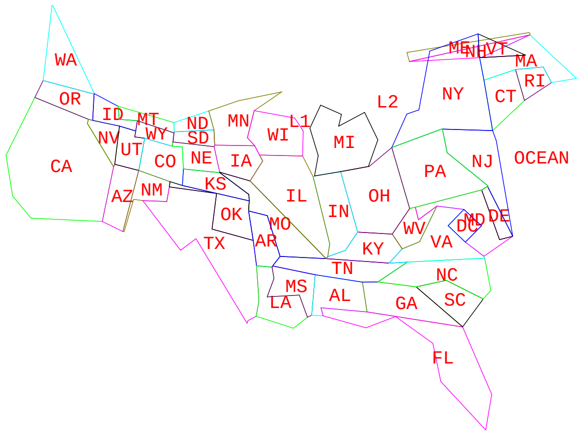
STAGE ⇒ 14 tracks

CHANNEL ROUTING RESULTS





MAP LAYOUT

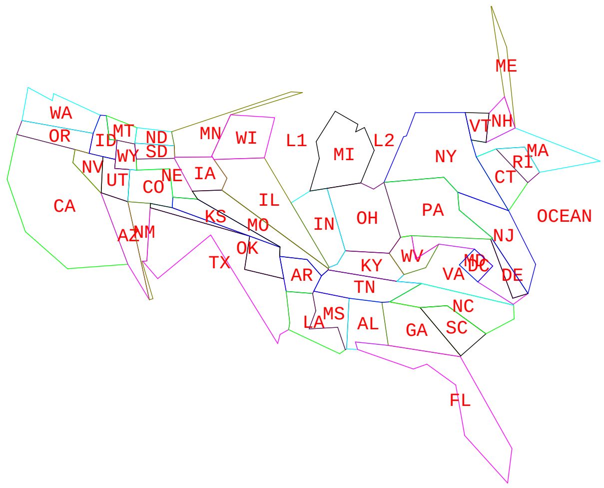


- Configuration space:
  Map $=$ 162 vertices in $\Re^2$; states are polygons
- Search moves:
  Perturb a random vertex (ensuring no constraint violations)
- Objective function:

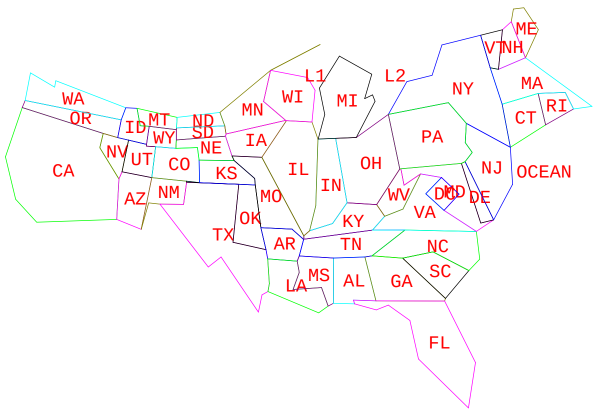
$$\begin{aligned} Obj(x) \;=\; & \sum_{s \in \text{states}} (\Delta\text{area}(s) + \\ & \sum_{v \in \text{vertices(s)}} (\Delta\text{gape}(v) + \Delta\text{orient}(v) + \Delta\text{segfrac}(v)) \end{aligned}$$

- Features used:
  $\Sigma\,\Delta\text{area},\ \Sigma\,\Delta\text{gape},\ \Sigma\,\Delta\text{orient},\ \Sigma\,\Delta\text{segfrac}$

TYPICAL MAP LAYOUT RESULTS

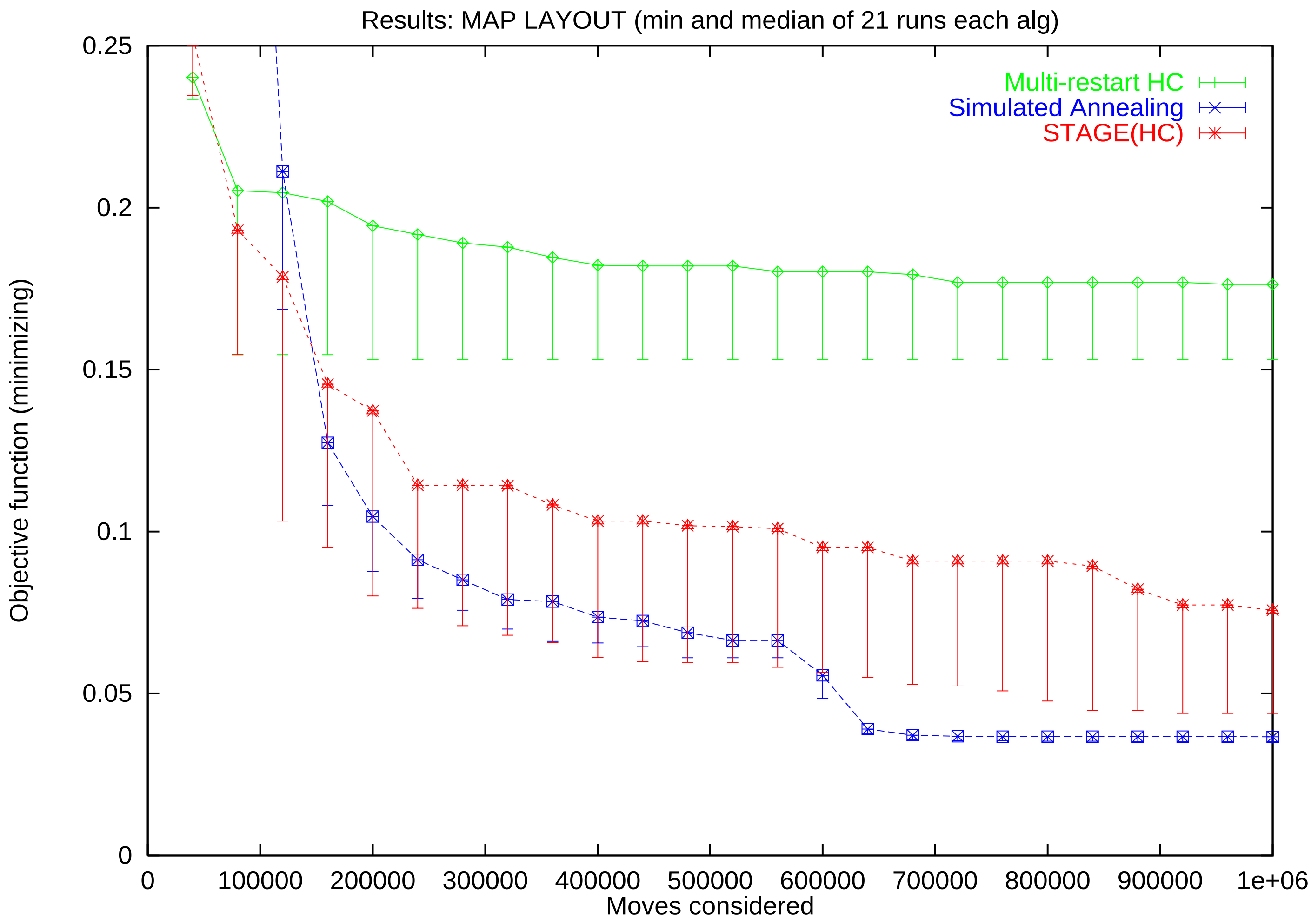


Hillclimbing ⇒ objective=0.115



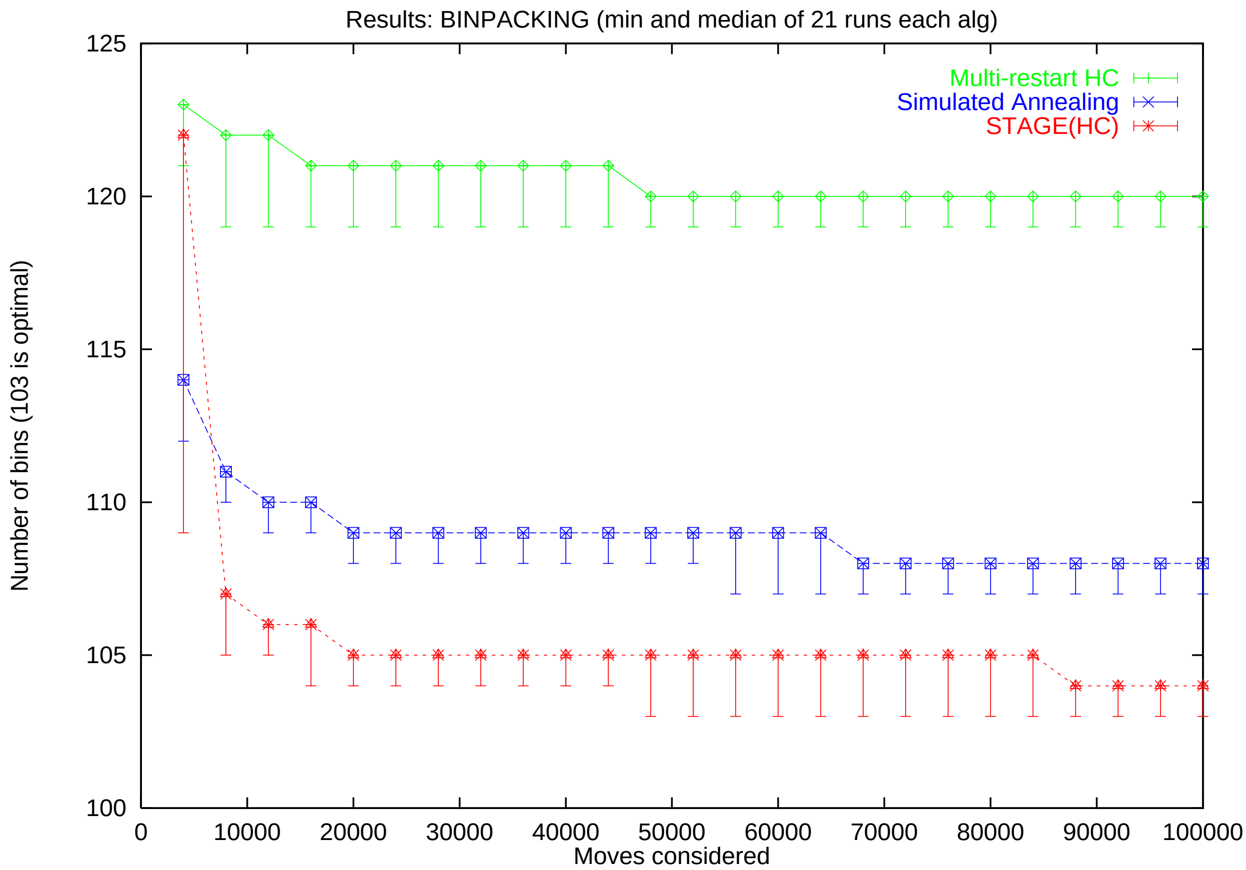
STAGE ⇒ objective=0.043

MAP LAYOUT RESULTS

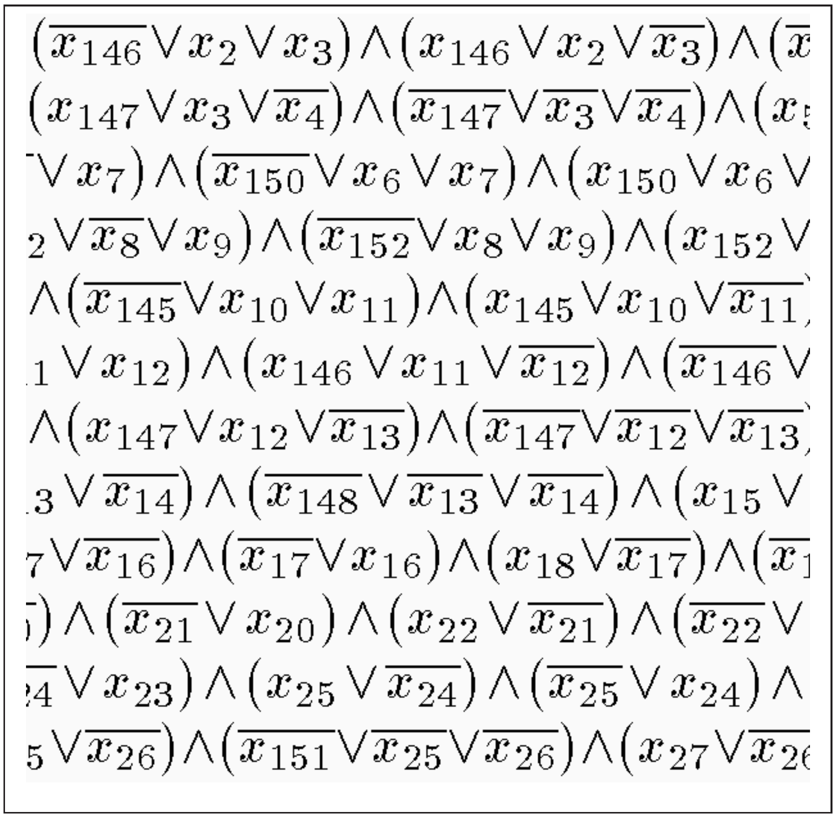


BINPACKING

- Configuration space:
  All legal packings of items into bins
  (benchmark u250-13 from [Falkenauer 94]: 250 items of sizes 20..100, bin size 150, optimal packing = 103 bins?)
- Search moves: Pick up a random item, put it down in a random bin where it fits
- Objective function: $Obj(x) = \#$ bins used
- Features used: Obj(x), sum-squared bin sparsity, fullness of least-full bin



BOOLEAN SATISFIABILITY



- Goal: Given a Boolean formula in CNF, assign 0/1 values to all variables so as to satisfy all the clauses.
- The original NP-complete problem
- Yet: WALKSAT [Selman & Kautz 94] can solve hard instances with over 1000 variables. Hence lots of attention at recent AI meetings.
- Applications: planning, circuit verification, circuit synthesis, scheduling, graph-coloring

WALKSAT

WALKSAT tries to minimize

$$Obj(x) = \# \text{ of unsatisfied clauses}$$

by performing a biased random walk.

Repeatedly:

- Select a random clause that is not satisfied.
- Satisfy that clause by flipping one of its variables. Which one?
  - for each var v_i, calculate improvement Δ_i
  - if all Δ_i are negative, with probability p flip a random v_i
  - otherwise, flip the var with maximum Δ_i

Observations on WALKSAT:

1. It's not greedy. It doesn't get stuck (though it can loop).
2. It's *incomplete*: it can't prove unsatisfiability.
3. It's king of the hill.
4. It's Markov.

STAGE(WALKSAT)

With $\pi =$ WALKSAT, STAGE learns to approximate:

$V^{\text{WALKSAT}}(x) = E[Obj(\text{best state WALKSAT reaches from } x)]$

Since WALKSAT is Markov, we can train STAGE from whole trajectories.*

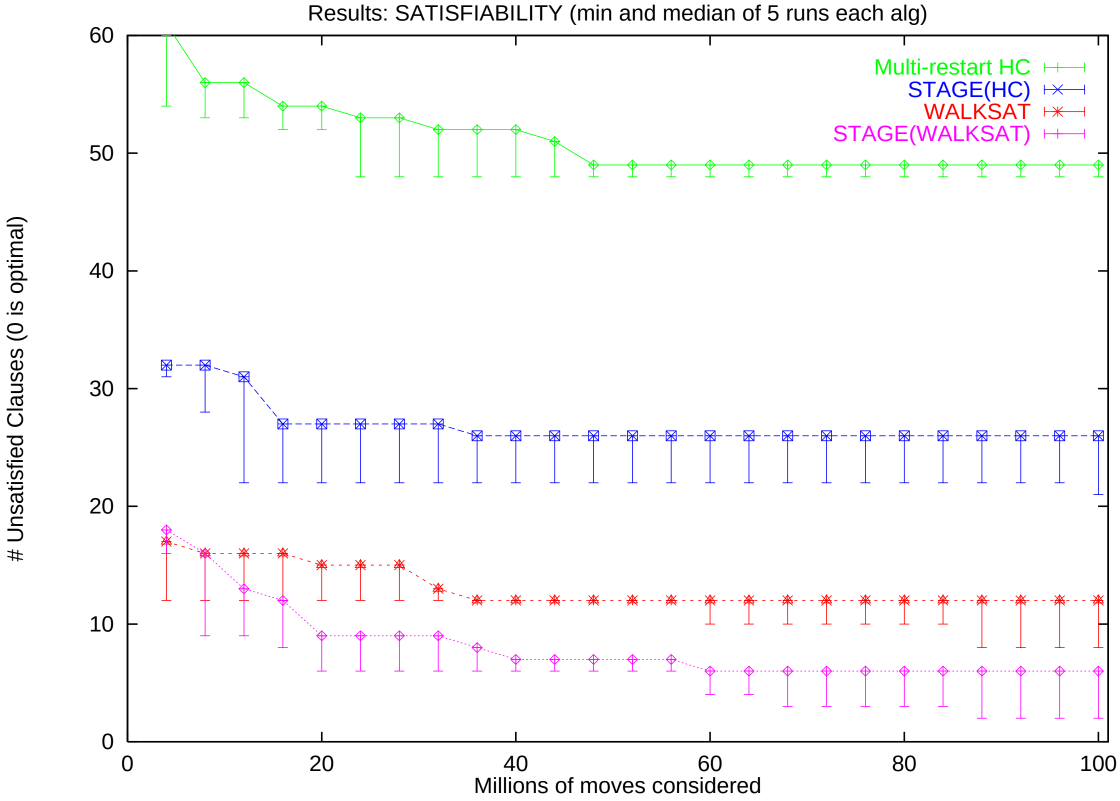
Features:

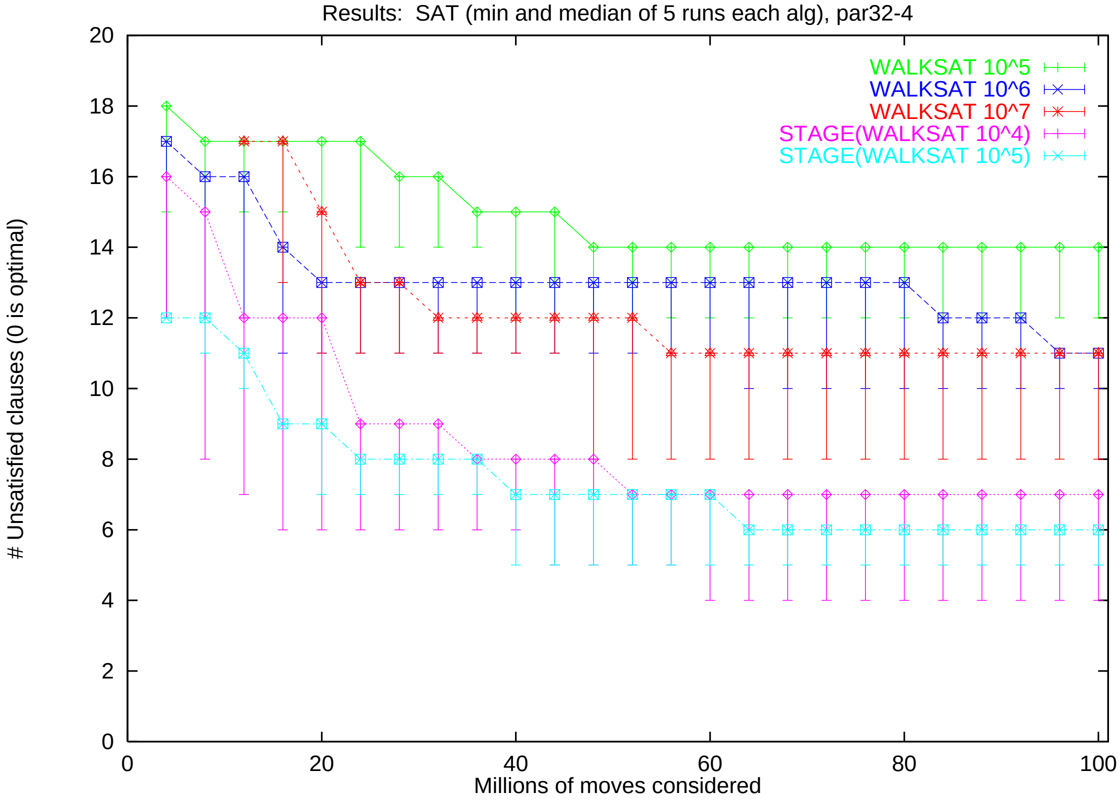
- % clauses with 0 variables right $(= Obj)$
- % clauses with 1 variable right
- % clauses with 2 variables right
- % variables that would break a clause if flipped
- % variables set to their "naive" setting

Benchmarks (satisfiable instances, hard for WALKSAT):

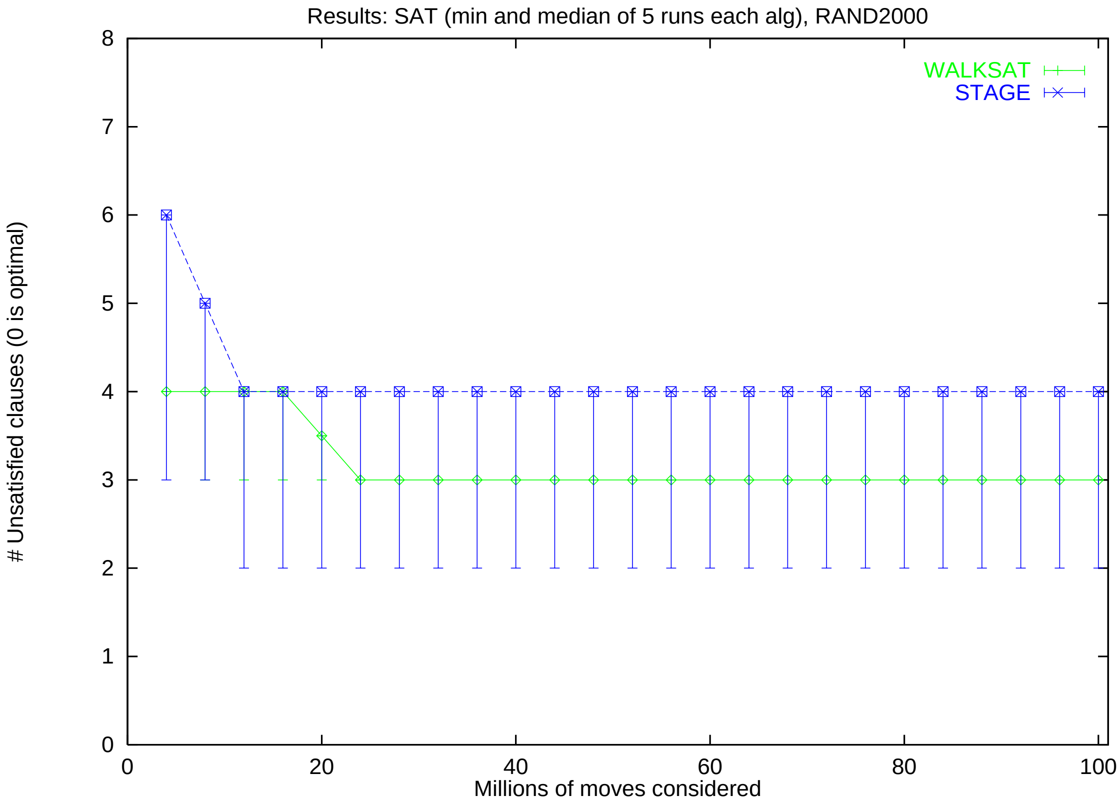
- Boolean function learning. 3176 vars, 10277 clauses; 3176 vars, 10313 clauses. [Crawford]
- Random 3CNF-SAT. 2000 vars, 8500 clauses. [Selman]
- Towers of Hanoi (n=5). 1931 vars, 14468 clauses. [Selman]

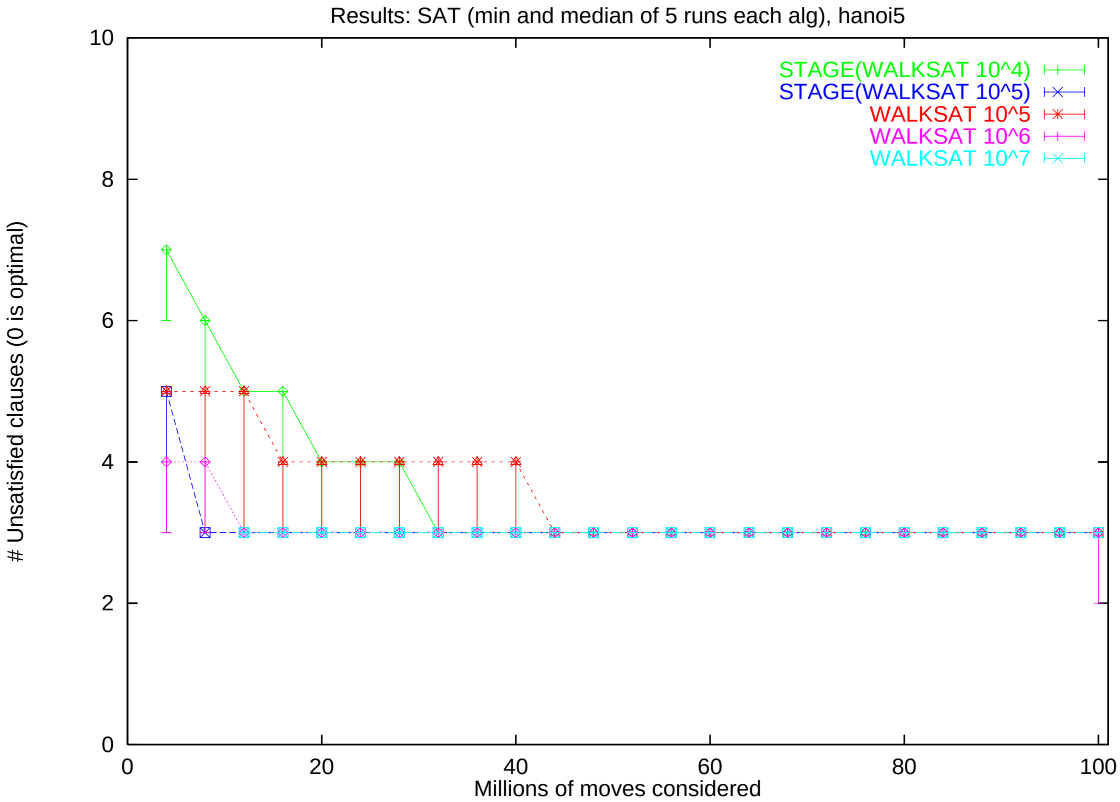
STAGE(WALKSAT) PRELIMINARY RESULTS





STAGE(WALKSAT) PRELIMINARY RESULTS





WHEN WILL STAGE WORK?

- Coherent structure among attracting basins
- Identifying structure depends on features, move operators, regression models
- Boese (1994) gives more reasons for optimism
- Largely an empirical question

ONGOING WORK

- Understanding what V^π has learned.
- Transfer — does the V^π we learned on instance A lead us quickly to a good solution on instance B? (Yes! future RL chat)
- STAGE(steepest-ascent HC)
- STAGE(SA) – Map Layout is a good candidate
- Try better features and function approximators.
- Domains: Radiotherapy and Bayes nets
- Using TD(λ) or ROUT to approximate V^π instead of supervised learning

RELATED WORK

- Boese 94 — found global structure among the local optima in TSP and graph partitioning. Suggests smart multi-restart.
- Martin & Otto 94 — chained local optimization, best heuristic for TSP: from local optimum, apply a “kick” to a nearby basin
- Zhang & Dietterich 95 — use RL to approximate V^* (value function of not fixed policy π, but *optimal* policy)