Online Selection, Adaptation, and Hybridization of Algorithms

(Thesis proposal)

Matthew Streeter

Thanks to my committee

Avrim Blum

Stephen Smith (Chair)

Carla Gomes, Cornell

Tuomas Sandholm

John Hooker, CMU Tepper School

Thesis

(i) the performance of algorithms can be improved by adapting them to the sequence of instances they are run on; and

(ii) this adaptation can be accomplished using *black box* techniques

Outline

- Introduction ☆
- Four problems
  - Combining multiple heuristics online (0:25)
  - Online reduction from optimization to decision (0:10)
  - Online tuning of branch and bound algorithms (0:05)
  - The max k-armed bandit problem (0:05)
- Summary & timeline

Common framework

(for first three problems)

- You are given a set Π of problem-solving policies, fed a sequence $I_1, I_2, \ldots, I_n$ of instances to solve
- For j from 1 to n:
    - You select policy $\pi_j \in \Pi$, receive feedback $f_j = F(\pi_j, I_j)$, and incur cost $c_j = C(\pi_j, I_j)$
    - Your decision is a function of history $(\pi_1, f_1), (\pi_2, f_2), \ldots, (\pi_{j-1}, f_{j-1})$ plus private random bits
- $regret = \frac{1}{n}\left(\mathbb{E}\left[\sum_{j=1}^{n} c_j\right] - \min_{\pi\in\Pi}\sum_{j=1}^{n} C(\pi, I_j)\right)$
- A *no-regret* policy-selection strategy has worst-case regret that is o(1) as a function of n

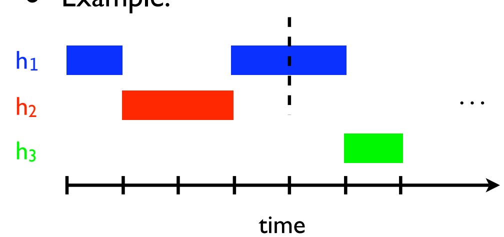
Combining multiple heuristics online

Setup

- Given set $H=\{h_1, h_2, \ldots, h_k\}$ of deterministic heuristics (e.g., SAT solvers)
- Fed sequence of decision problems to solve
- Each heuristic eventually returns "yes" or "no", but CPU time depends on instance
- Solve each problem by interleaving execution of heuristics, stopping as soon as one of them returns an answer

Task-switching schedules

- Mapping $\pi: \mathbb{Z} \mapsto H$ from time slices to heuristics; $\pi(t)$ = heuristic to run from time t to time t+1
- Example:



Completion times

	h_1	h_2	h_3
I_1	2	7	7

Motivations

- Task-switching schedule can be better than any single heuristic
- Evidence that tables like this arise in practice (e.g., for SAT solvers)

Completion times

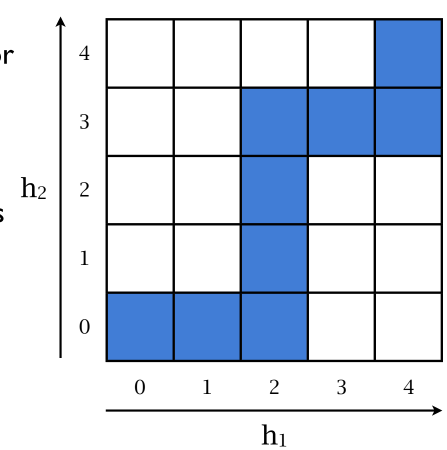
	h_1	h_2
I_1	10^6	1
I_2	1	1
I_3	1	10^6

Solving the offline problem

- Offline problem: given table of completion times, compute task-switching schedule that minimizes sum of CPU time over all instances
- Can solve as shortest path problem

Solving the offline problem

- Assume completion times in {1,2, ..., B} for some known B (here B=4)
- Can think of a task-switching schedule as a path in a k-dimensional grid
- E.g. "run h_1 for 2 seconds, then run h_2 for 3 seconds..."



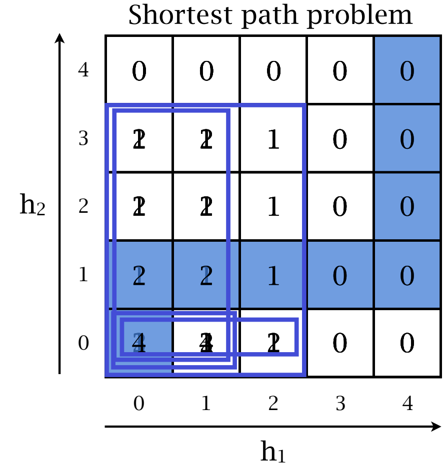
Solving the offline problem

Completion times

	h_1	h_2
I_1	3	4
I_2	2	1
I_3	2	4
I_4	3	1

- Time complexity is $O(n\ B^k)$
- Can get α-approximation in time $O(n(\log_\alpha B)^k)$

Shortest path problem



Solving the online problem

- In online problem, adversary secretly fills in table of completion times. Then:
- For j from 1 to n
  - You select task-switching schedule π_j
  - You incur cost c_j = time it takes to solve I_j using π_j
  - Your feedback is c_j

Solving the online problem

- Can solve using existing no-regret strategies for online shortest paths
  - two-player game where you pick path, adversary picks edge weights
  - various feedback models
- György et al. (2006) assume at the end of each round, you find out weights of all edges you traversed
- Cesa-Bianchi et al. (2005) assume that by paying a price (here Bk), you can find out weights of all edges
- Combining the two gives regret at most

$$O\left(Bk \cdot \min\left\{\sqrt{\frac{k(\ln k)L^k}{n}}, \left(\frac{Lk}{n}\right)^{\frac{1}{3}}\right\}\right)$$

where L = length of sides of grid

Handling a large number of heuristics

(Joint work with Daniel Golovin)

- Bad news:
  - Offline problem generalizes MIN-SUM SET COVER, which is NP-hard to approximate within 4-ϵ for any ϵ >0 (Feige et al., 2002)
- Good news:
  - Simple greedy algorithm gives 4-approximation; seems to behave much better in practice
  - Can use to get 4+o(1) multiplicative regret in *distributional* online setting
  - Proposed work: similar result for adversarial online setting?

Offline experiments

- Each year, various conferences hold solver competitions with the following format:
  - each submitted heuristic is run on a sequence of instances (subject to time limit)
  - awards for heuristics that solve the most instances in various instance categories
- Downloaded tables of completion times, computed (approximately) optimal task-switching schedules, and compared them to best individual solver

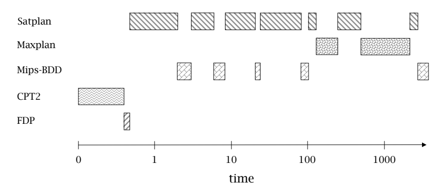
Results for 2006 A.I. Planning Competition

- A.I. planning involves finding a minimum-length sequence of actions that lead from a start state to a goal state
- Six "optimal" planners were submitted to 2006 A.I. planning competition
  - each run on 240 instances with 30 minute time limit per instance
  - 110 instances were solved by at least one of the six

Results for 2006 A.I. Planning Competition

Solver	Avg. CPU time (s)	Num. solved
Greedy schedule	358	98
Satplan	507	83
Maxplan	641	88
Mips-BDD	946	54
CPT2	969	53
FDP	1079	46
Parallel schedule	1244	89
IPPLAN-1SC	1437	23

Greedy schedule:



Summary of results

Domain	**Venue**	**Speedup factor** (range across categories)
satisfiability	SAT 2005	1.1x-2x
planning	ICAPS 2006	1.4x
constraint solving	CP 2006	1x-1.5x
theorem proving	CADE 2006	1x-7.7x
satisfiability modulo theories	CAV 2006	1x-165x

Proposed work

- Experimental results for online setting (good preliminary results in *distributional* online setting)
- Incorporate two forms of "expert advice":
  - Expert looks at features of instance, predicts which heuristic will work best
  - Given run-so-far, expert predicts how much longer heuristic will take

Generalization: restart schedules

- If H contains randomized heuristics, it may help to periodically restart them with a fresh random seed
- A restart schedule $\pi:\mathbb{Z}\mapsto H\times\{0,1\}$ is a task-switching schedule augmented with a flag that says whether to restart (if $|H|=1$, can represent as a sequence of thresholds $t_1, t_2, t_3, \ldots$)
- In online problem, adversary picks run length distributions rather than completion times

Generalization: restart schedules

- **Motivation:** solvers based on chronological backtracking often exhibit heavy-tailed run length distributions, and restarting can reduce mean run length by orders of magnitude (Gomes et al. 1998)

Generalization: restart schedules

- Our results:
  - Similar shortest path formulation; to get an α^2-approximation you need $B^{(\log_\alpha \log_\alpha B)k}$ vertices; online shortest paths gives no-regret strategy
  - Greedy algorithm gives 4-approximation
- Proposed work: experimental evaluation, engineering of strategies that work well in practice

Related work

- *Algorithm portfolios* (Huberman et al. 1997, Gomes et al. 2001, ...)
  - assigns each heuristic a fixed proportion of CPU time, plus a fixed restart threshold
  - previous work considered *offline* setting, and assumed each heuristic has same run length distribution on each instance

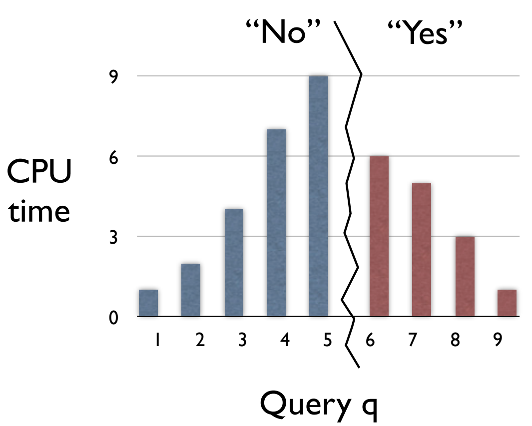
Proposed work: online reduction from optimization to decision

Setup

- Fed a sequence of minimization problems (e.g., find a plan of minimum length), each with minimum in {1, 2, ..., B}
- Given: oracle that answers queries of the form "Does there exist a solution with cost at most q?"
- CPU time required by oracle depends on q (and on instance)
- Goal: find a (provably) optimal solution to each instance, minimizing CPU time consumed by oracle calls

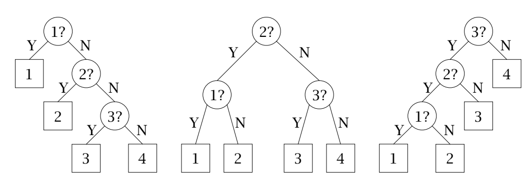
Example

(single function to minimize)



Reduction policies

- A reduction policy is a binary search tree with key set {1,2, ..., B}, e.g. for B=4:



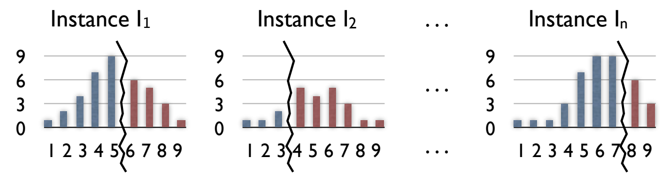
(A) ramp-up (B) binary search (C) ramp-down

Motivating example

- At 2006 A.I. Planning Competition, first place for optimal planners went to SATPLAN and MAXPLAN
- Both construct a sequence of boolean formulae $\sigma_1, \sigma_2, \ldots$ where σ_q is satisfiable iff. there exists a plan of length $\leq q$
  - both use SAT solver as oracle
  - SATPLAN uses ramp-up; MAXPLAN uses ramp-down

Solving the offline problem

- Offline problem: given CPU time for each instance I_j and query q, compute reduction policy that minimizes total CPU time



- Can solve in $O(B^3)$ time using dynamic programming

Solving the online problem

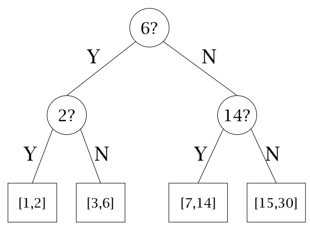
- Adversary determines CPU time for each query and instance; on each round you select a reduction policy
  - cost c_j = total CPU time used on round j
  - feedback f_j = list of oracle CPU times for each query you made on round j

Solving the online problem

- Use the *multiplicative weight update method:* on round j, select tree T with prob. proportional to $\alpha^{\text{(hypothetical cost of using T on first j-1 rounds)}}$, for some $\alpha<1$.
- Two difficulties:
  - limited feedback — handle using standard technique
  - there are exponentially many trees — use an $O(B^3)$ dynamic programming algorithm to implement efficiently
- Regret is at most $B\sqrt{\frac{2B \ln B}{n}}$ (assuming max. oracle CPU time = 1)

Finding approximately optimal solutions

- Just need to work with pruned trees
- E.g., 2-approximation for B=30:



- Can get additive or multiplicative approximations

Proposed work: online tuning of branch and bound algorithms

Setup

- Fed a sequence of minimization problems of the form

$$\min g(x_1, x_2, \ldots, x_d)$$

s.t.

$$x_i \in D_i \quad \forall i, 1 \le i \le d$$

$$\langle x_1, x_2, \ldots, x_d \rangle \text{ satisfies constraints}$$

- Will solve each problem using branch and bound
- Goal: given set R={r_1,r_2, ..., r_k} of available relaxations, select which relaxation(s) to run at each depth of the search tree so as to minimize CPU time

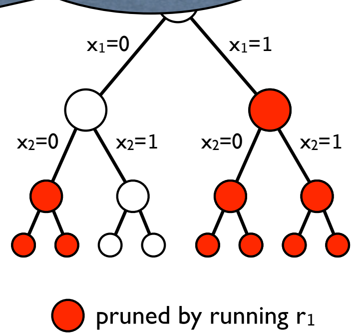
Branch and bound

- Branch and bound is a divide and conquer algorithm, widely used for solving NP-hard minimization problems
  - recursively breaks problem into subproblems by picking some variable and considering all values that can be assigned to it (branching)
  - solves relaxed versions of subproblems to quickly identify subproblems that can be safely ignored (bounding)

Branch and

Strong assumption:
relaxation(s) do not interact with variable or value ordering heuristics

- Variable selection and value ordering heuristics define a tree of subproblems
- At each tree node, can run one or more relaxation(s), which may cause the subtree rooted at that node to be pruned (ignored)
- A *relaxation policy* π:$\mathbb{Z} \mapsto 2^R$ lists the relaxation(s) to run at each depth
- Want to select relaxation policy that minimizes CPU time (sum of time to run relaxations plus time to process non-pruned nodes)



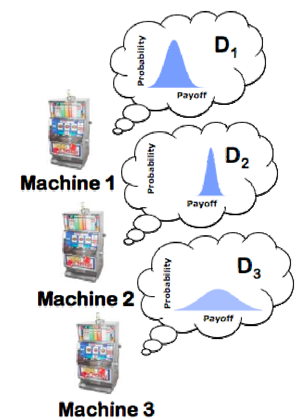
Outline of results

- Key fact: suppose |R|=1. The CPU time policy π spends on some particular node at depth i depends on:
  - largest depth < i where π ran relaxation
  - whether π runs relaxation at depth i
- Can compute optimal relaxation policy finding shortest path in graph with d^2 vertices, where d = max. tree depth (in general need d^{2k} vertices)
- Online shortest paths gives no-regret strategy, but using this idea naïvely shouldn't work well in practice
- Results on relaxations carry over to *constraint propagators* so long as key fact holds

The max k-armed bandit problem

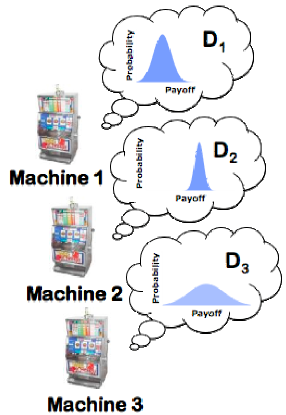
The classical k-armed bandit

- You are in a room with k slot machines
- Pulling arm of i^{th} machine returns payoff drawn from unknown distribution D_i
- Goal: maximize sum of payoffs given n pulls
- > 50 years of papers



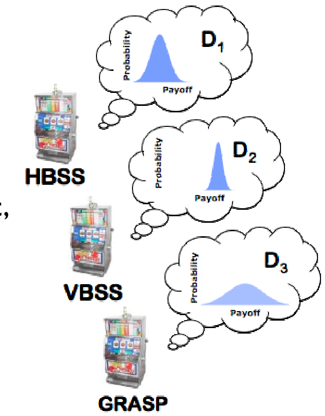
The max k-armed bandit

- You are in a room with k slot machines
- Pulling arm of i^{th} machine returns payoff drawn from unknown distribution D_i
- Goal: maximize highest payoff given n pulls
- Introduced by Cicirello & Smith (2003)



Application: selecting among heuristics

- Given: a *single* optimization problem, k randomized heuristics
- Each time you run a heuristic, get a solution with a certain quality
- Goal: given n runs, maximize quality of best solution



Previous work

- Fact: the *generalized extreme value* (GEV) distribution is to maxima what the Gaussian is to sums
- Cicirello & Smith (AAAI 2005 best paper) assumed payoffs drawn from Gumbel distributions (special case of GEV)
  - good experimental results for selecting among randomized greedy heuristics for the RCPSP/max
  - no rigorous performance guarantees

Our results: GEV payoffs

(Streeter & Smith, AAAI 2006)

- We give a no-regret strategy for the case where each payoff distribution is a GEV.
  - regret = $\max_{\text{arms } i} M_{i,n} - S_n$
  - $M_{i,n}$ = expected max. payoff from pulling i^{th} arm n times
  - S_n = expected max. payoff from using our strategy for n pulls
- Note: not possible to get a no-regret strategy for arbitrary payoff distributions (e.g., suppose payoffs are always 0 except for one "mystery" arm that gives payoff 1 with prob. 1/n)

Our results: Threshold Ascent

(Streeter & Smith, CP 2006)

- Threshold Ascent: use classical k-armed bandit solver to maximize #payoffs that exceed some threshold, ramp up threshold over time
- Outperforms strategy of Cicirello & Smith (2005) on RCPSP/max experiments

Strategy	Σ **Regret**	$\mathbb{P}[\text{Regret} = 0]$
Threshold Ascent	188	0.722
Round-robin sampling	345	0.556
LPF	355	0.675
MTS	402	0.657
QD-BEACON	609	0.538
RSM	2130	0.166
LST	3199	0.095
MST	4509	0.107

Proposed work

- Investigate online version in which we are fed a sequence of max k-armed bandit instances; perhaps can learn appropriate distributional assumptions

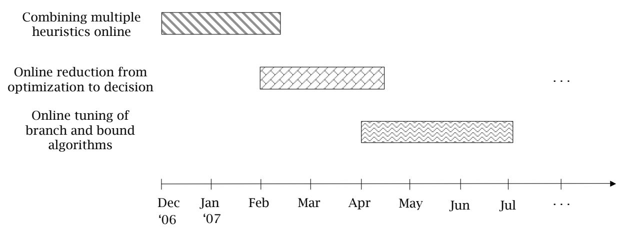
Summary & timeline

Summary

Problem	Results so far	Proposed work
Combining multiple heuristics online	no-regret strategy, greedy approximation, offline experiments	experimental evaluations, online version of greedy, expert advice
Online reduction from optimization to decision	no-regret strategy	experimental evaluation
Online tuning of branch and bound algorithms	no-regret strategy	more practical no-regret strategies, experimental evaluation
The max k-armed bandit	Threshold Ascent, no-regret strategy for GEV payoffs	online (multiple-instance) setting

Timeline

- Aim to finish between Dec. 2007 and May 2008
- Rough timeline:



Thanks!