Combining Multiple Heuristics Online

Matthew Streeter[1] Daniel Golovin[1] Stephen F. Smith[2]

Carnegie Mellon University

[1]Computer Science Department

[2]The Robotics Institute

AAAI

July 24, 2007

Why heuristics?

- The world is full of NP-hard problems
- Heuristics often work well in practice
  - SAT solvers handle formulae with 10^6 variables, used for hardware and software verification
- Much interest in developing better heuristics (annual SAT solver competition)

Heuristics have complementary strengths & weaknesses

- Running time of heuristics varies widely across instances

Instance	SatELiteGTI CPU (s)	MiniSat CPU (s)
liveness-unsat-2-01dlx_c_bp_u_f_liveness	33	15
vliw-sat-2-0/9dlx_vliw_at_b_iq6_bug4	376	≥ 120000
vliw-sat-2-0/9dlx_vliw_at_b_iq6_bug9	≥ 120000	131

- Can often reduce average-case run time by running several heuristics in parallel (on a single processor)

This talk

- **Goal:** interleave the execution of multiple heuristics in order to improve average-case running time
- Schedule for interleaving can be learned online while solving a sequence of problems

Related work

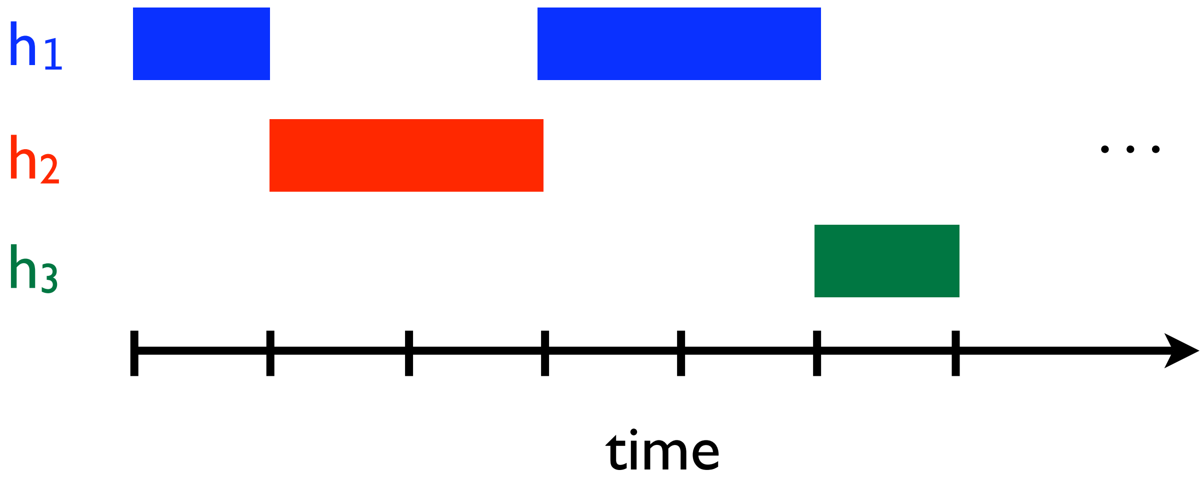
- *Algorithm portfolios*
  - Assign each heuristic a fixed proportion of CPU time, plus a fixed restart threshold (Huberman et al. 1997, Gomes et al. 2001)
  - Later work used instance features to predict which heuristic will finish first (e.g., Leyton-Brown et al. 2003, Xu et al. 2007)
- *Combining Multiple Heuristics* (Sayag et al. 2006)
  - considered *resource-sharing schedules* and *task-switching schedules*
  - gave offline algorithms + bounds for PAC learning; algorithms are exponential in #heuristics

Formal setup

- Given: k heuristics, n decision problems to solve
- Using j^{th} heuristic to solve i^{th} instance takes time τ_{ij}, where $\tau_{ij} \in \{1, 2, ..., B\} \cup \{\infty\}$
- For each i, assume $\min_j \tau_{ij} \leq B$
- Solve each problem by interleaving execution of heuristics according to a *task-switching schedule*

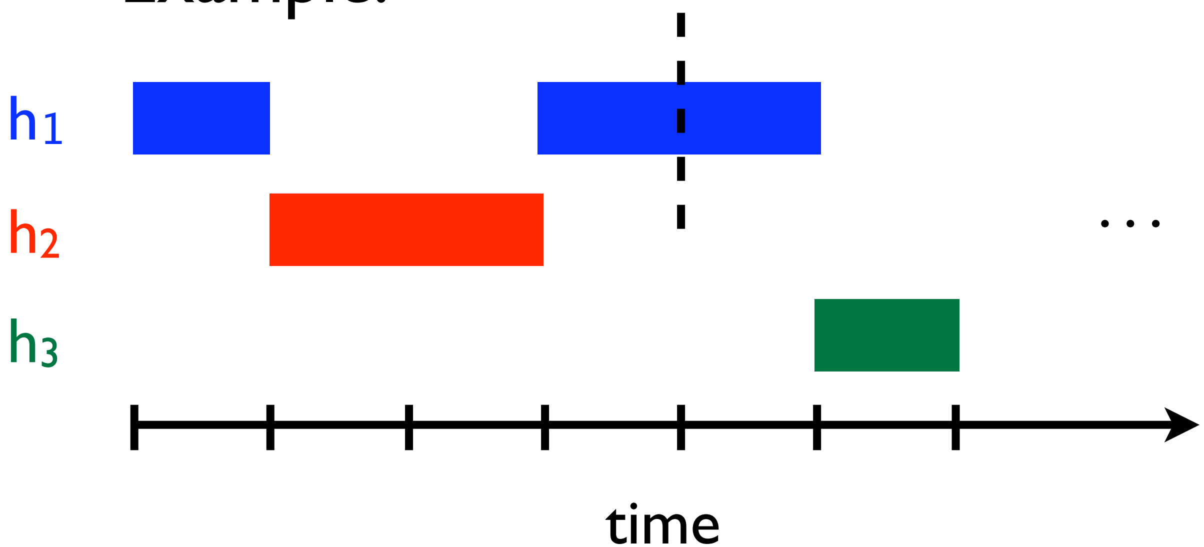
Task-switching schedules

- Mapping S from integer time slices to heuristics; $S(t)$ = heuristic to run from time t to time $t+1$
- Example:



Task-switching schedules

- Mapping S from integer time slices to heuristics; $S(t)$ = heuristic to run from time t to time $t+1$
- Example:

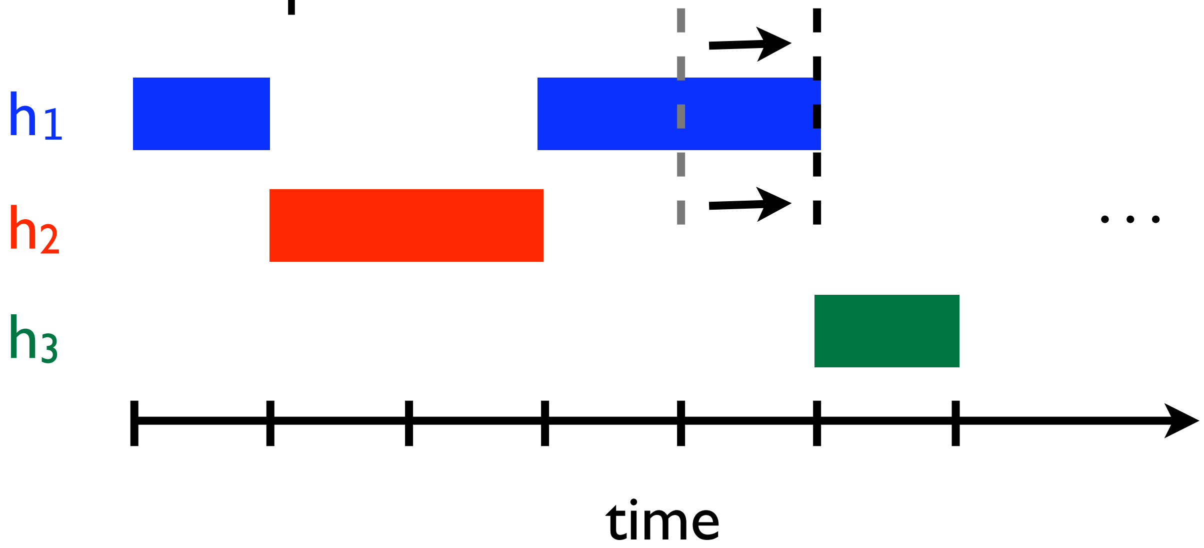


Completion times

	h_1	h_2	h_3
I_1	2	7	7

Task-switching schedules

- Mapping S from integer time slices to heuristics; $S(t)$ = heuristic to run from time t to time $t+1$
- Example:



Completion times

	h_1	h_2	h_3
I_1	2	7	7

- Can also handle model where we throw away all work when switching between heuristics

Three settings

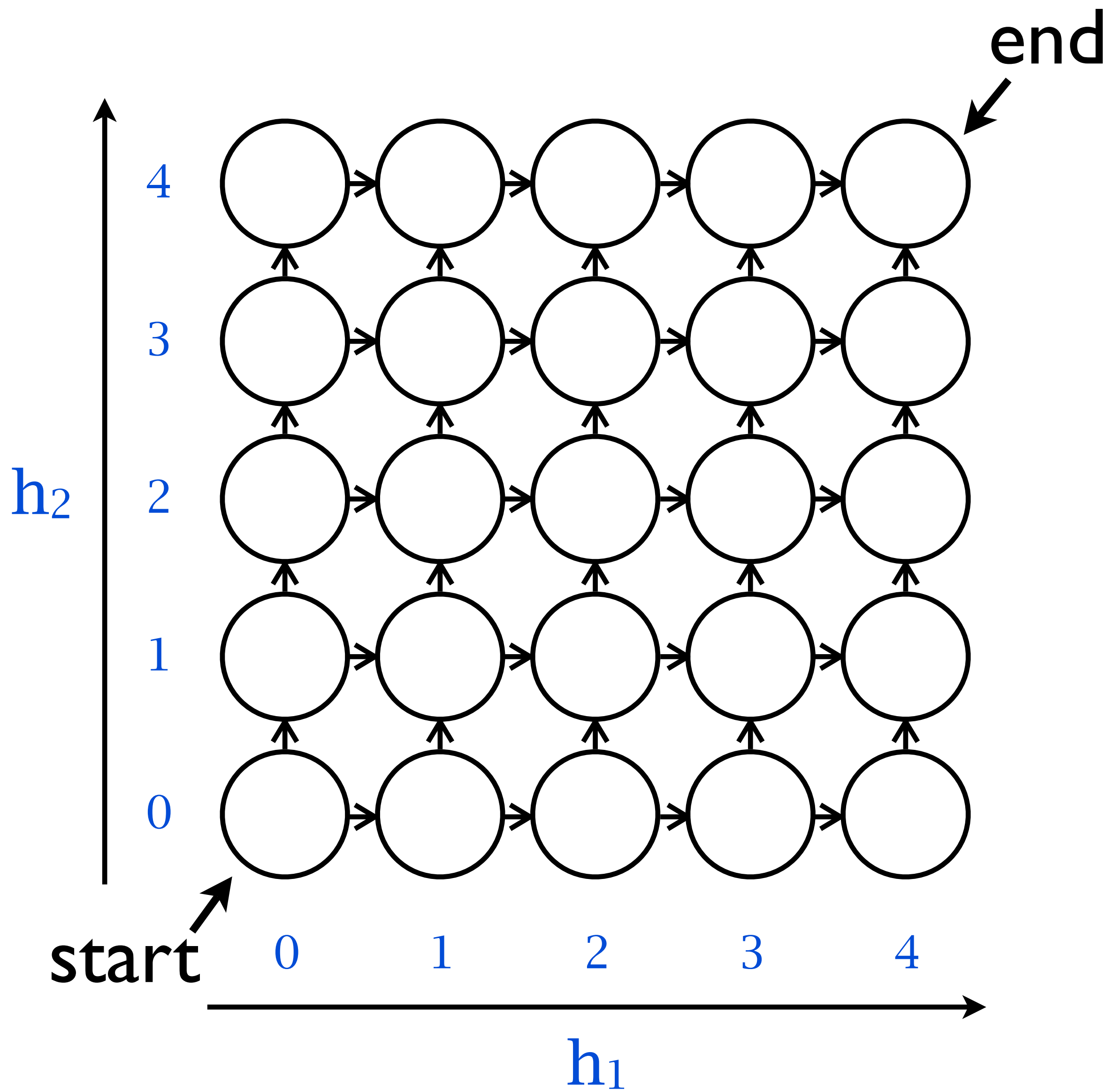
- We consider selecting task-switching schedules in three settings:
  - ☆ **Offline:** given table τ as input, compute an optimal schedule
  - **Learning-theoretic:** PAC-learn an optimal restart schedule from training instances
  - ☆ **Online:** you are fed an *arbitrary* sequence of instances one at a time, and must solve each instance before moving on to the next

The offline setting

- Offline problem: given table τ of completion times, compute task-switching schedule that minimizes sum of CPU time over all instances
- Think τ of as training data

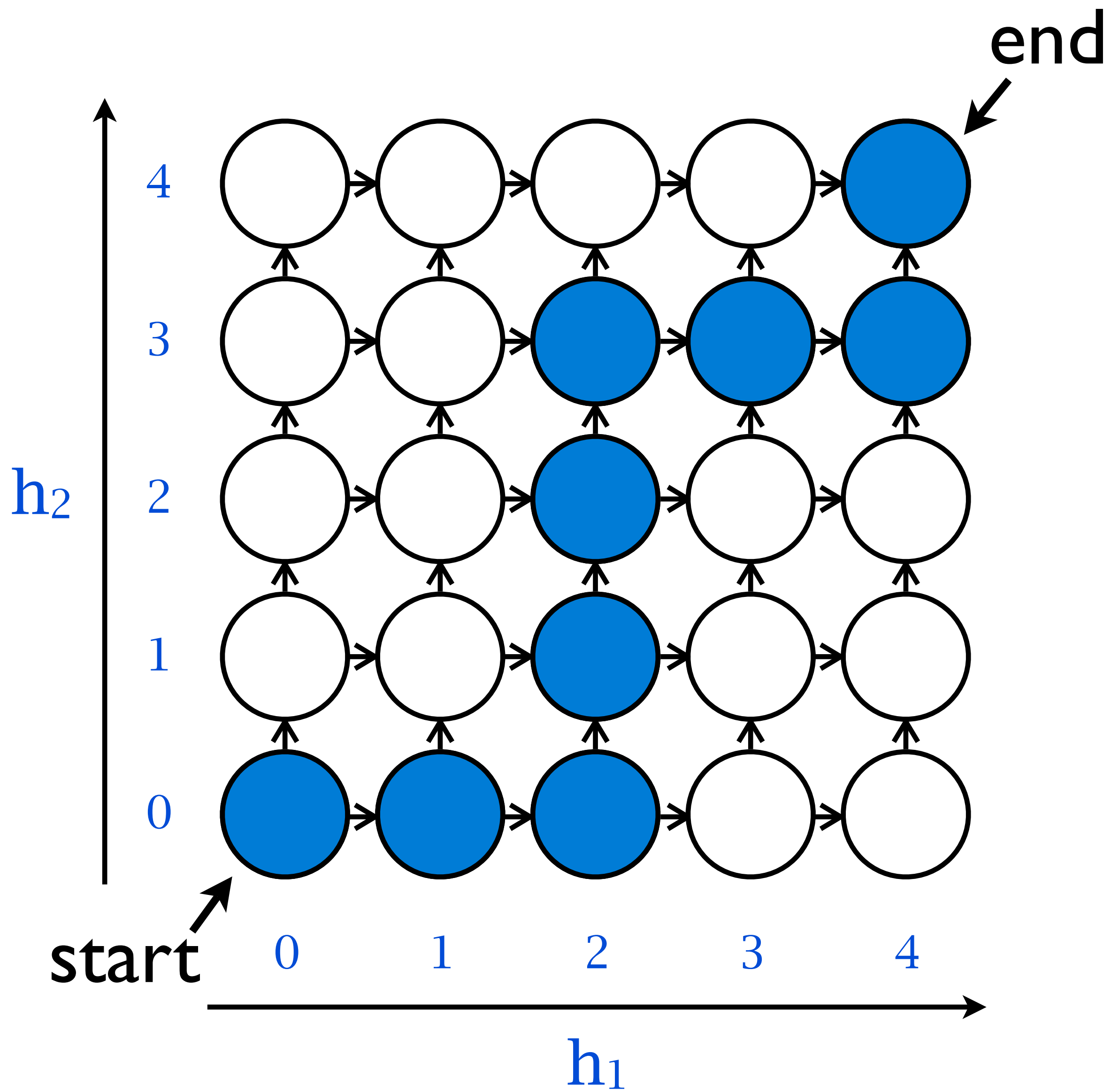
Offline shortest path algorithm

- Can think of a task-switching schedule as a path in a k-dimensional grid with sides of length B+1 (here B=4)



Offline shortest path algorithm

- Can think of a task-switching schedule as a path in a k-dimensional grid with sides of length B+1 (here B=4)
- E.g. "run h_1 for 2 seconds, then run h_2 for 3 seconds..."

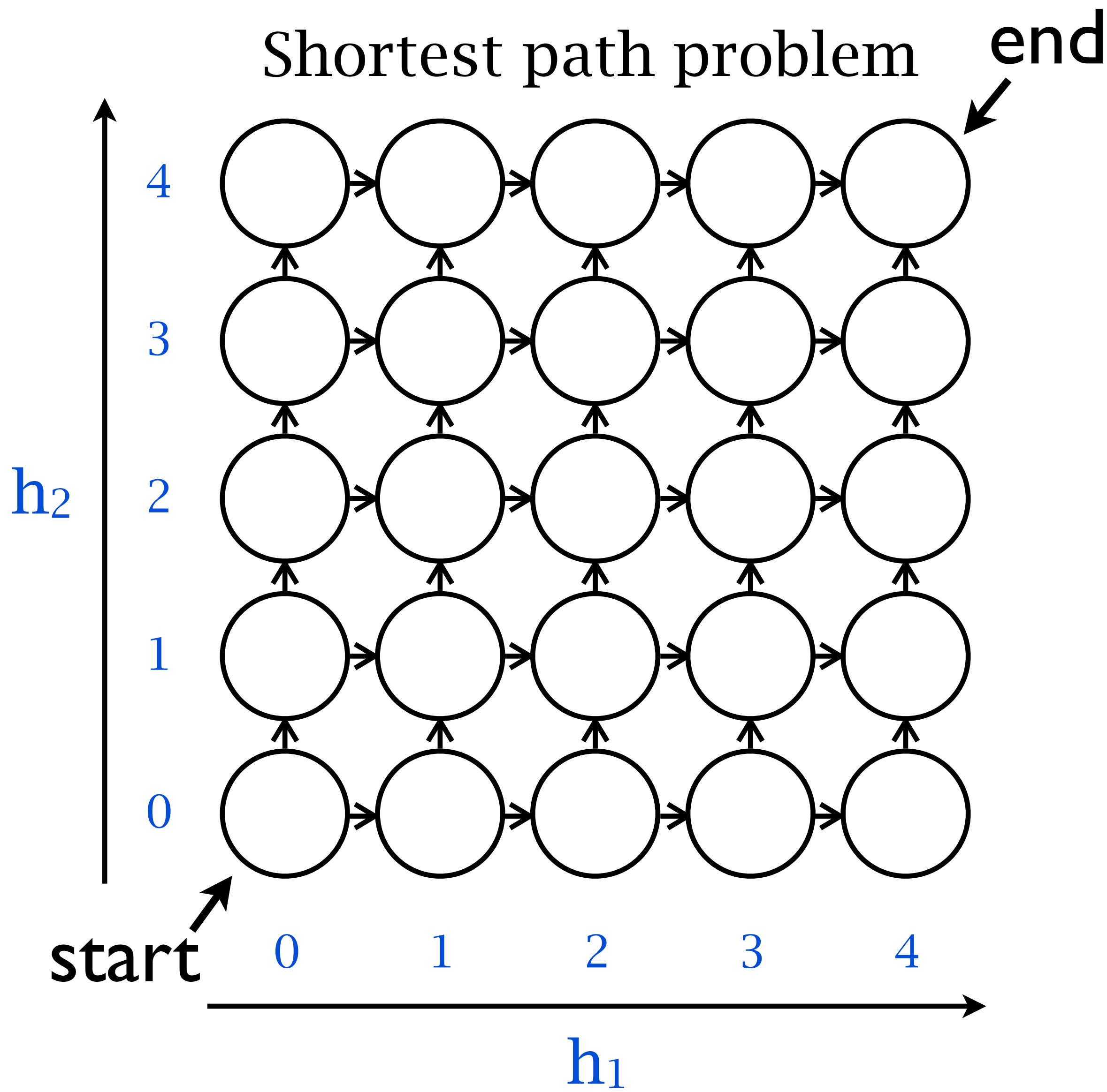


Offline shortest path algorithm

Completion times

	h_1	h_2
I_1		
I_2		
I_3		
I_4		

Shortest path problem

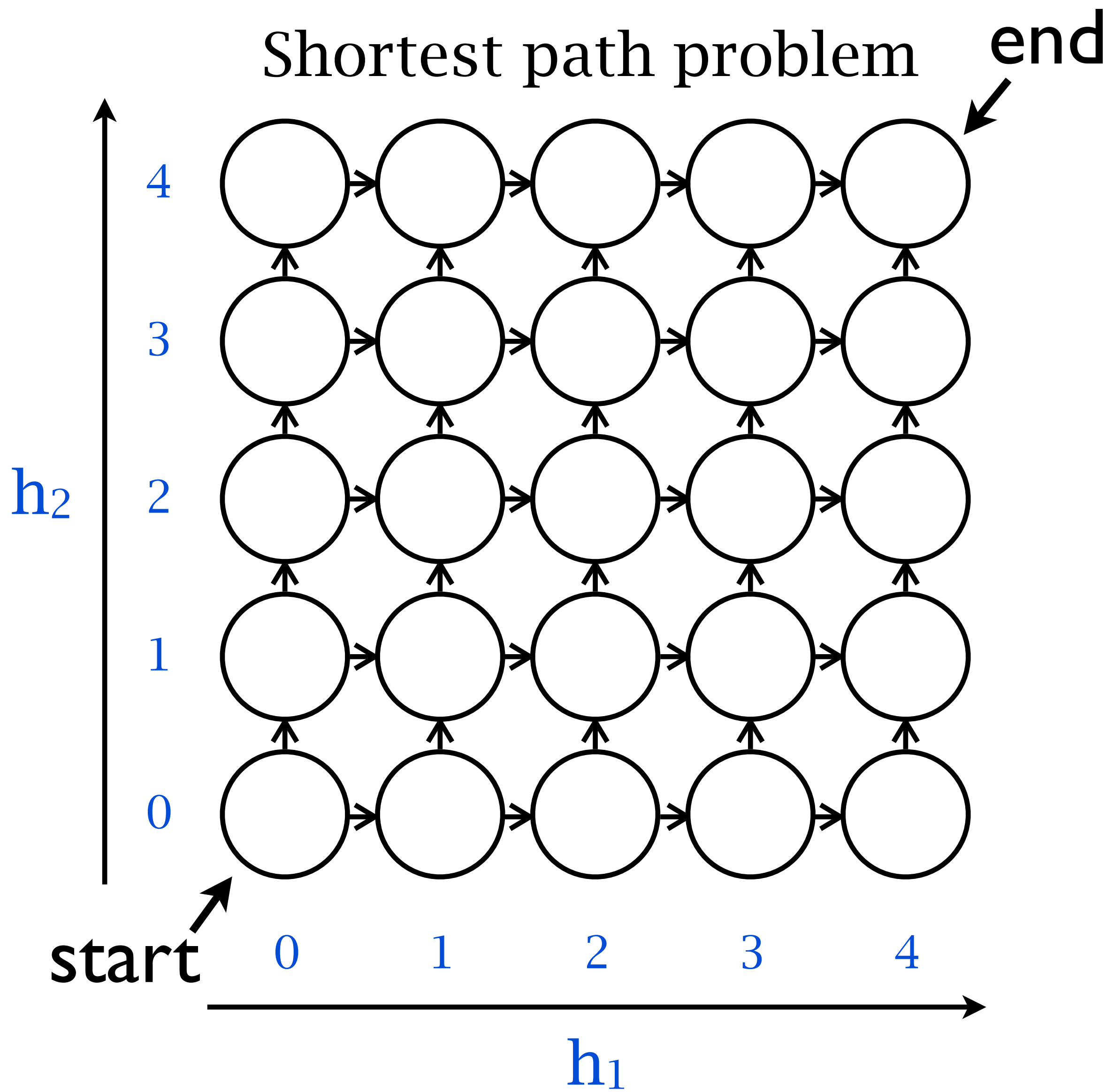


Offline shortest path algorithm

Completion times

	h_1	h_2
I_1		
I_2		
I_3		
I_4		

Shortest path problem

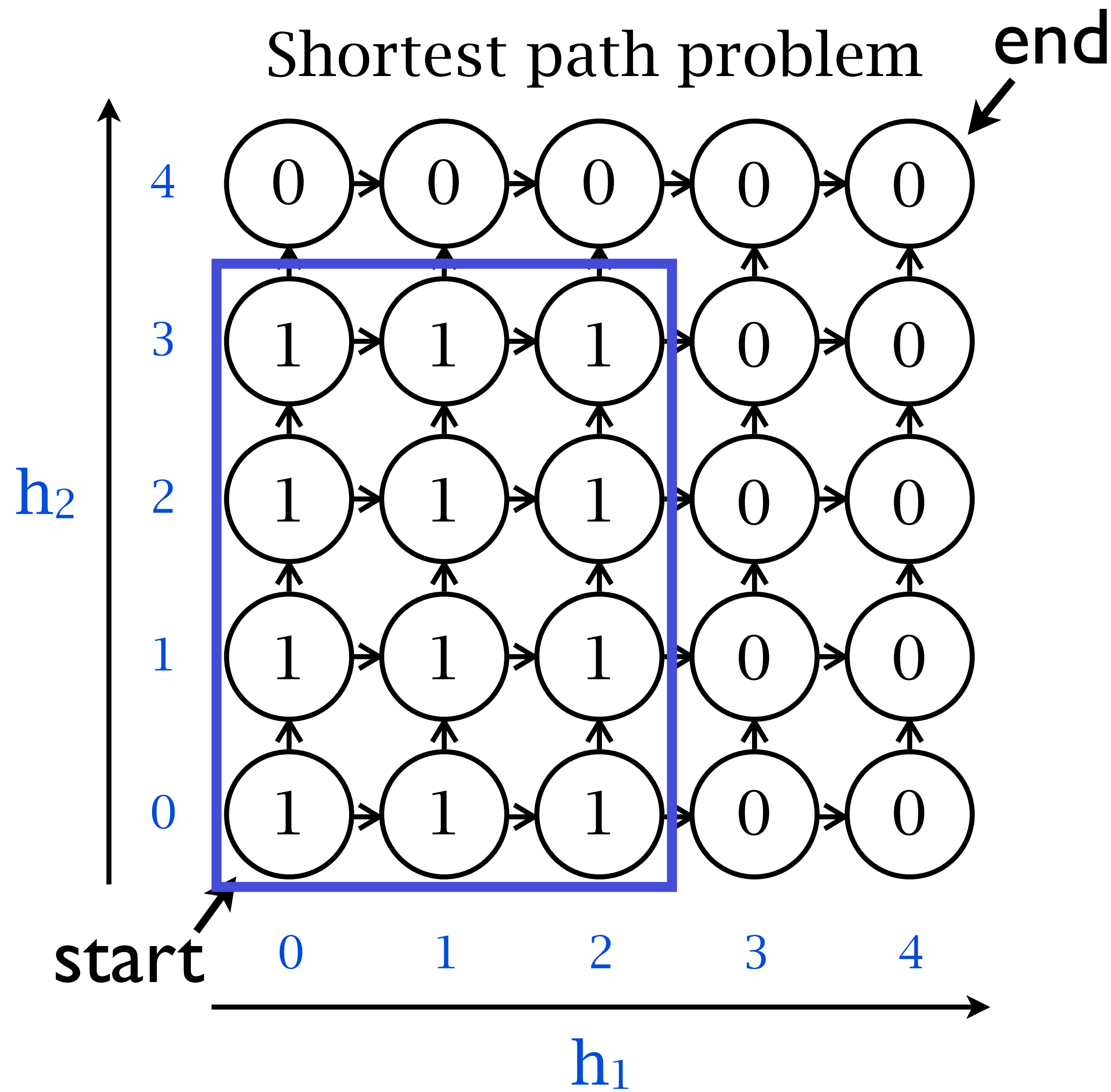


Offline shortest path algorithm

Completion times

	h_1	h_2
I_1	3	4
I_2		
I_3		
I_4		

Shortest path problem

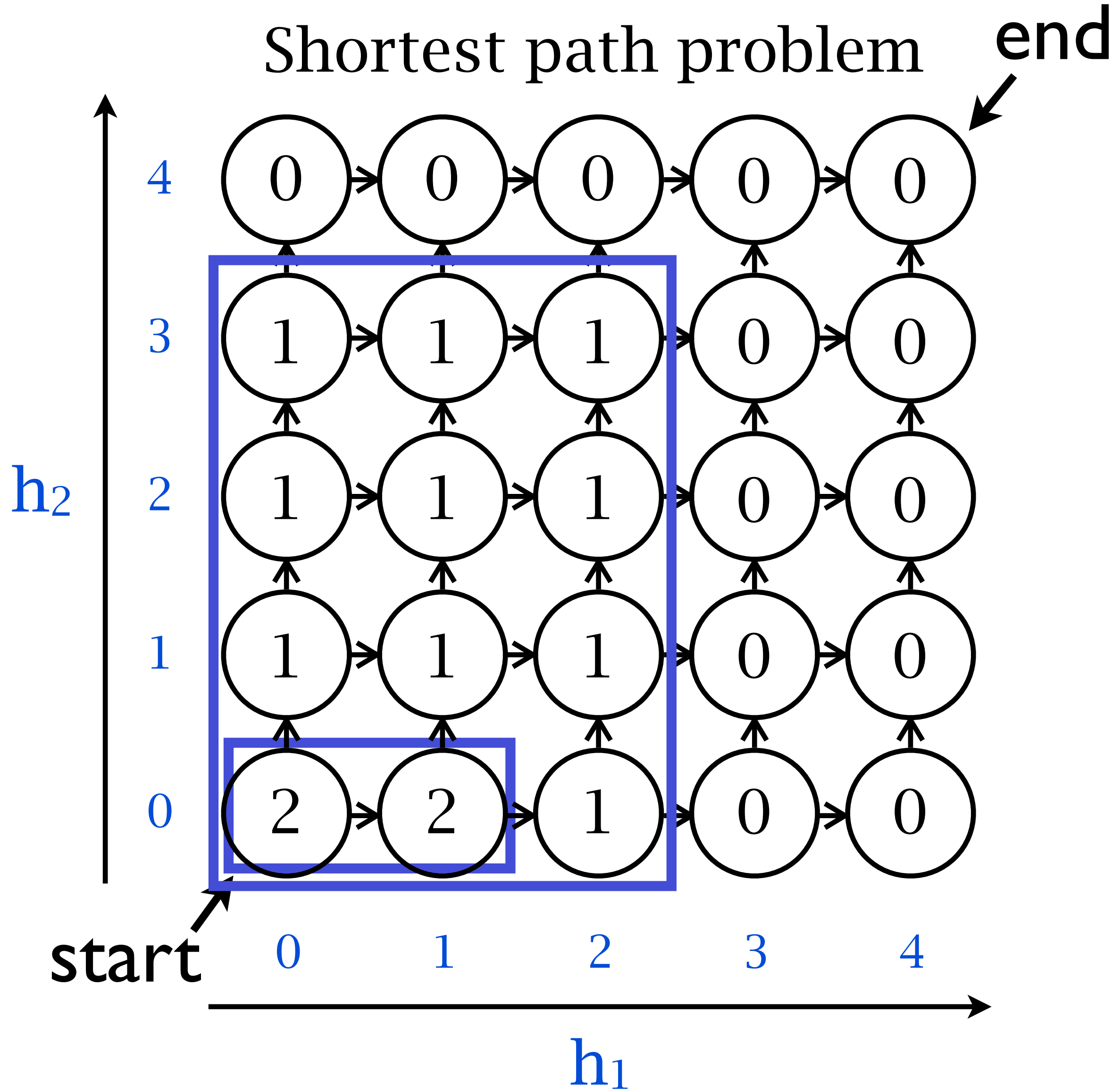


Offline shortest path algorithm

Completion times

	h_1	h_2
I_1	3	4
I_2	2	1
I_3		
I_4		

Shortest path problem

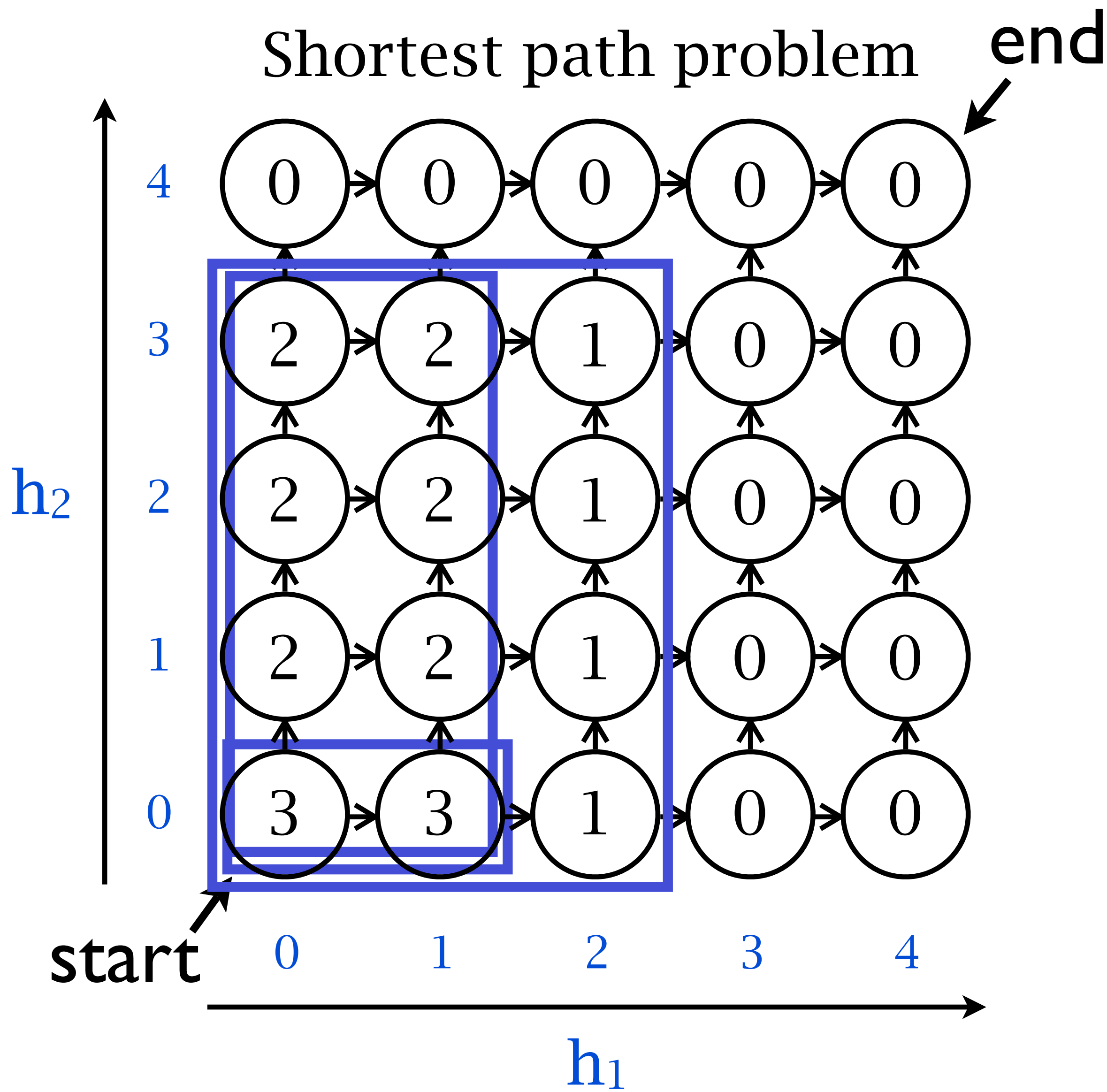


Offline shortest path algorithm

Completion times

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

Shortest path problem

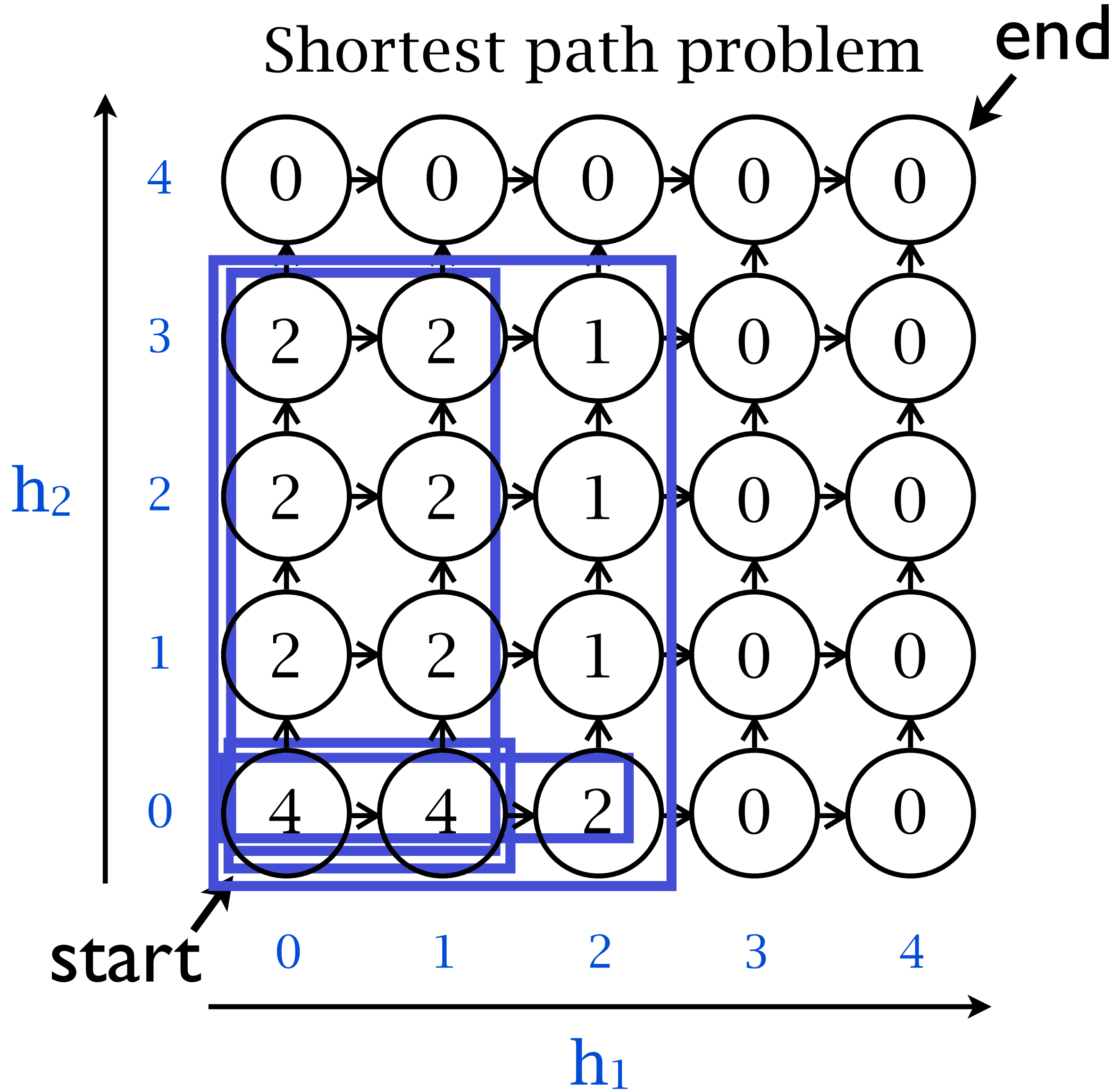


Offline shortest path algorithm

Completion times

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

Shortest path problem

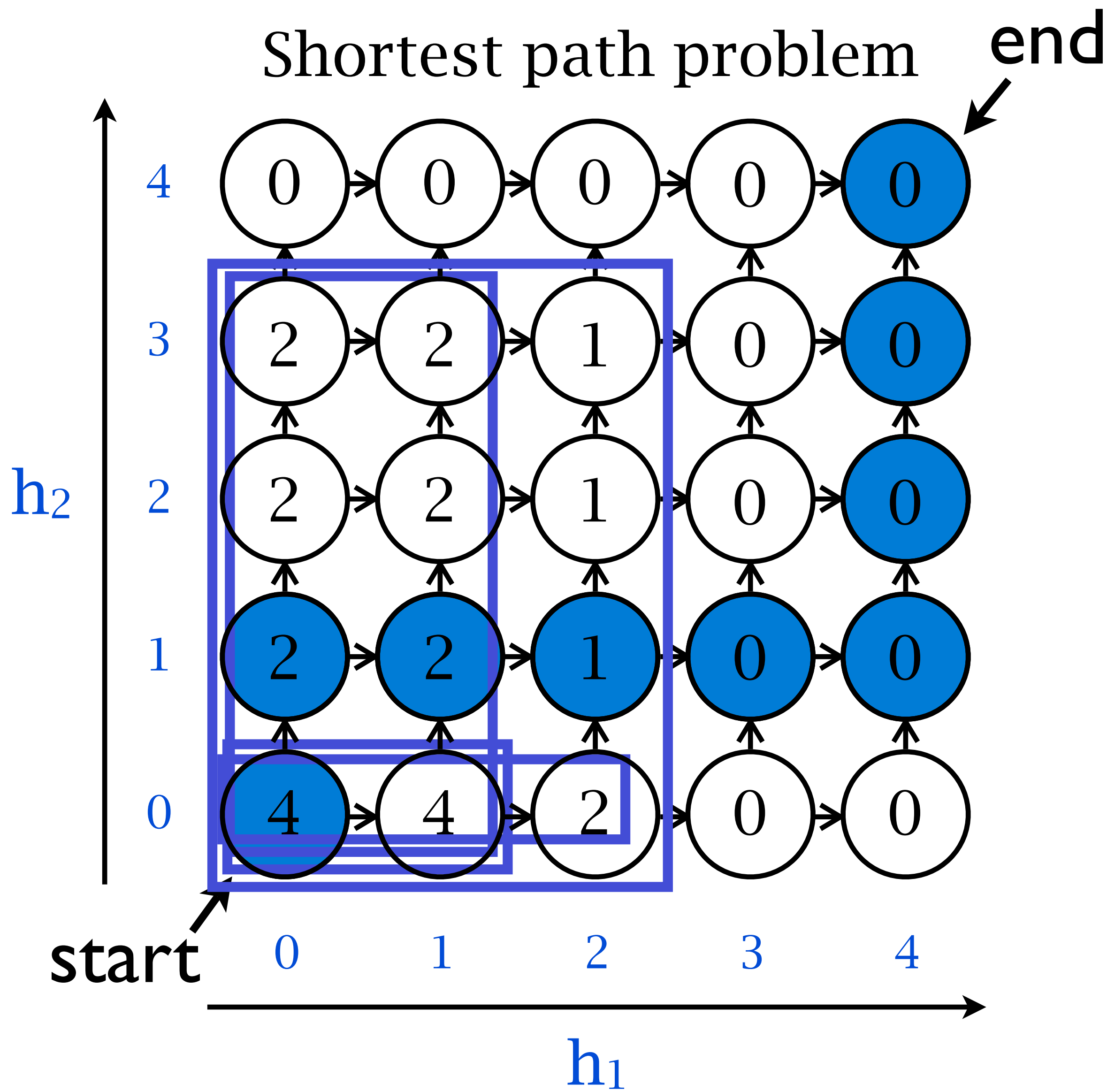


Offline shortest path algorithm

Completion times

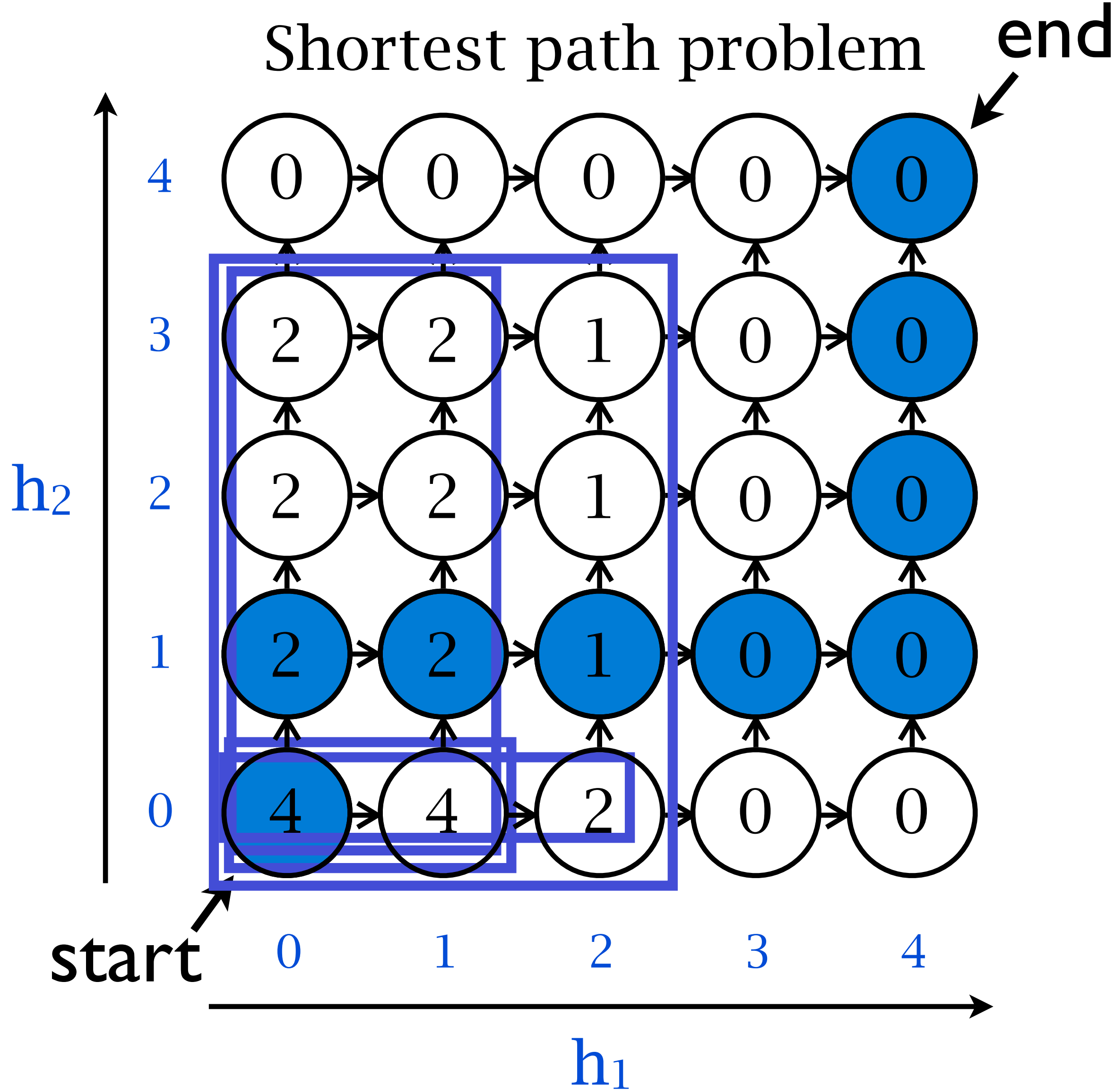
	h_1	h_2
I_1	3	4
I_2	2	1
I_3	2	4
I_4	3	1

Shortest path problem



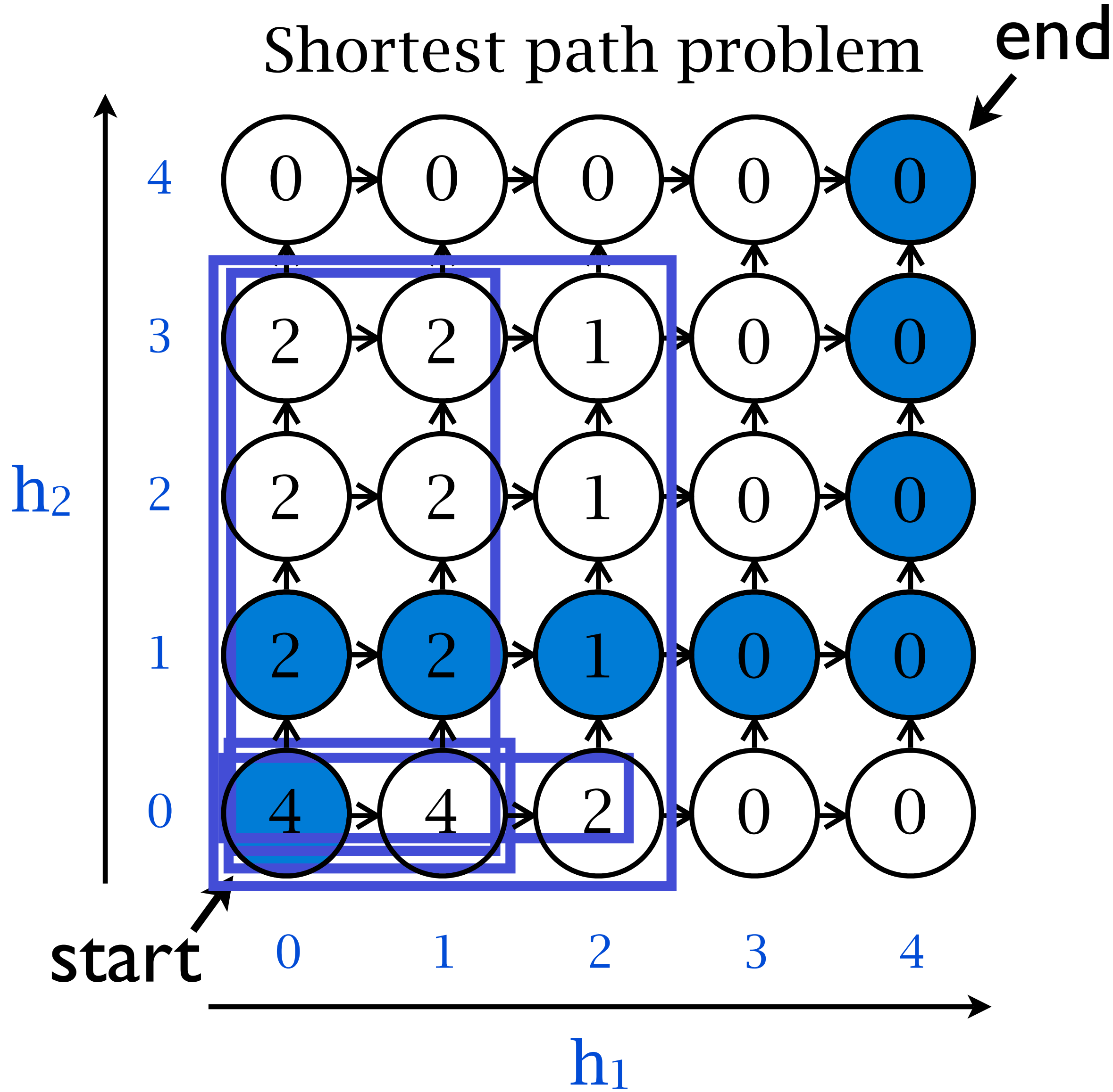
Offline shortest path algorithm

- Time complexity is $O(nk(B+1)^k)$



Offline shortest path algorithm

- Time complexity is $O(nk(B+1)^k)$
- Can get α-approximation in time $O(nk(1+\log_\alpha B)^k)$



Greedy approximation algorithm

Greedy approximation algorithm

- How hard is it to compute optimal task-switching schedule?
  - Special case $B=1$ is MIN-SUM SET COVER. NP-hard to get $4-\epsilon$ approx for any $\epsilon>0$ (Feige, Lovász, & Tetali 2002)
- Greedy alg. for MIN-SUM SET COVER gives 4-approx. We generalize to get 4-approx for task-switching schedules

Greedy approximation algorithm

- How hard is it to compute optimal task-switching schedule?
  - Special case $B=1$ is MIN-SUM SET COVER. NP-hard to get $4-\epsilon$ approx for any $\epsilon>0$ (Feige, Lovász, & Tetali 2002)
- Greedy alg. for MIN-SUM SET COVER gives 4-approx. We generalize to get 4-approx for task-switching schedules
- Algorithm: greedily pick pair (h,t) such that running heuristic h for t (additional) time steps maximizes #(new instances solved)/t (append (h,t) to schedule and repeat)

The online setting

- World secretly selects sequence of n instances
- For i from 1 to n
  - You select schedule S_i to use to solve i^{th} instance
  - As feedback you observe how much time S_i takes
- regret = **E**[your total time] - $\min_{(\text{schedules } S)}$ (S's total time)
- Want worst-case regret that is $o(n)$

Online algorithms

Online algorithms

- We give a strategy whose worst-case regret is $O(Bkn^{2/3}(Bk \log k)^{1/3}) = o(n)$
  - combines online shortest path algorithm of György et al. (2006) with technique from Cesa-Bianchi et al. (2005)
  - total decision-making time is comparable to running offline shortest path alg.

Online algorithms

- We give a strategy whose worst-case regret is $O(Bkn^{2/3}(Bk \log k)^{1/3}) = o(n)$
  - combines online shortest path algorithm of György et al. (2006) with technique from Cesa-Bianchi et al. (2005)
  - total decision-making time is comparable to running offline shortest path alg.
- **Ongoing work:** online version of greedy approximation algorithm

Experimental Evaluation

- Various conferences hold annual solver competitions
  - each submitted solver is run on a sequence of instances (subject to time limit)
  - awards for solvers that solve the most instances in various instance categories
- We downloaded tables of completion times, used them to run our offline algorithms

2006 A.I. Planning Competition

Solver	Avg. CPU (s)	#Solved
SATPLAN	507	83
MaxPlan	641	88
MIPS-BDD	946	54
CPT2	969	53
FDP	1079	46
IPPLAN-1SC	1437	23

2006 A.I. Planning Competition

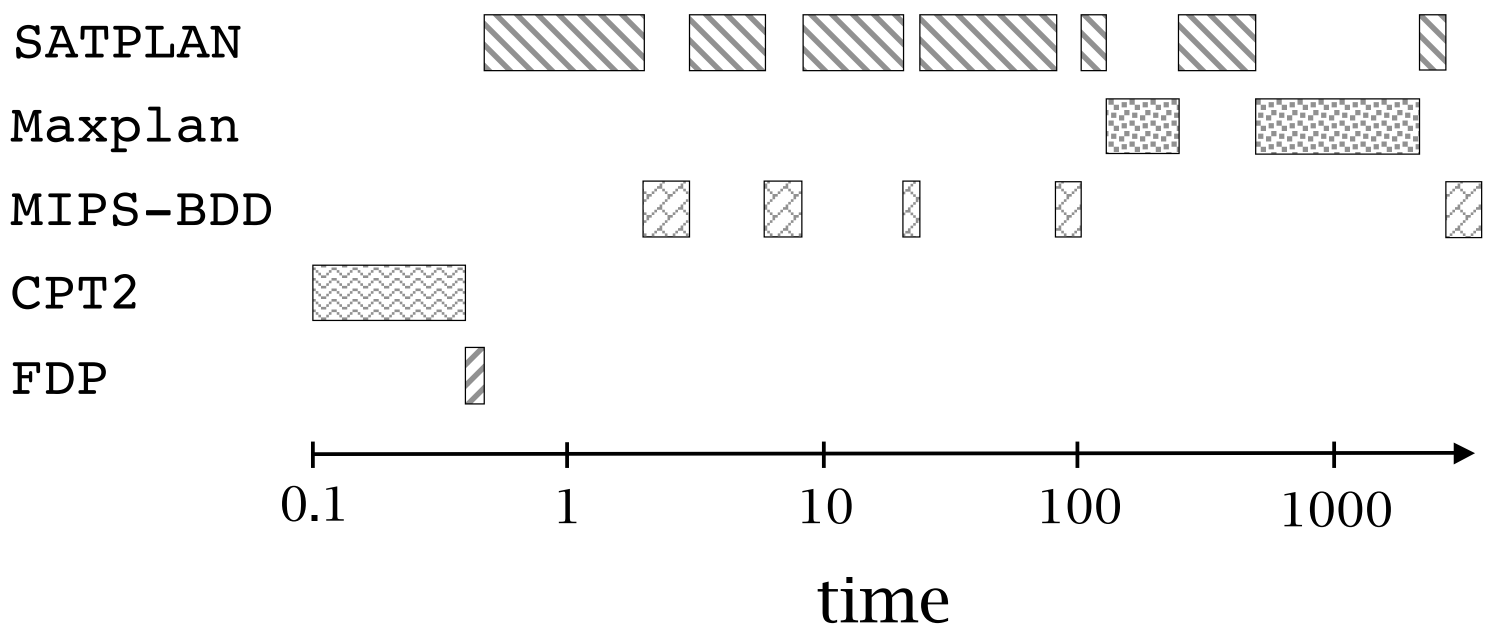
Solver	**Avg. CPU (s)**	**#Solved**
Greedy schedule (optimistic)	358	98
Greedy schedule (pessimistic)	476	96
SATPLAN	507	83
MaxPlan	641	88
MIPS-BDD	946	54
CPT2	969	53
FDP	1079	46
Parallel schedule	1244	89
IPPLAN-1SC	1437	23

2006 A.I. Planning Competition

Solver	**Avg. CPU (s)**	**#Solved**
Greedy schedule (optimistic)	358 (*407*)	98 (*97*)
Greedy schedule (pessimistic)	476 (*586*)	96 (*95*)
SATPLAN	507	83
MaxPlan	641	88
MIPS-BDD	946	54
CPT2	969	53
FDP	1079	46
Parallel schedule	1244	89
IPPLAN-1SC	1437	23

2006 A.I. Planning Competition

Greedy schedule



Summary of results

Solver competition	**Domain**	**Speedup factor** (range across categories)
SAT 2005	satisfiability	1.2 — 2.0
ICAPS 2006	planning	1.4
CP 2006	constraint satisfaction	1.0 — 1.5
IJCAR 2006	theorem proving	1.0 — 7.7

Future work

- Generalization to randomized algorithms (next talk)
- Online version of greedy algorithm (in progress)
- Exploiting features of instances/runs