Learning the Valuations of a k-demand Agent

Hanrui Zhang Vincent Conitzer

Duke University

this talk:

- optimal (up to lower order terms) algorithm for actively learning the valuations of a k-demand agent
- algorithm with polynomial time & sample complexity for passively learning the valuations of a k-demand agent

k-demand agents and demand sets

k-demand agent: demands a set of items of size <=k maximizing her utility, i.e., total value - total price

demand set: the set of items the agent demands

Unit-demand agents

value:	$10	$12	$8
price:	$6	$5	$5
surplus:	$4	$7	$3
agent buys:	✘	✔	✘

k-demand agents and demand sets

value:	\$5	\$6	\$4	\$3
price:	\$4	\$3	\$2	\$2

agent is <u>2-demand</u> — they want <u>no more than 2</u> items

k-demand agents and demand sets




value:	\$5	\$6	\$4	\$3
price:	\$4	\$3	\$2	\$2
surplus:	\$1	\$3	\$2	\$1
2-demand agent buys:	✘	✔	✔	✘

k-demand agents and demand sets




demand set

value:	\$5	\$6	\$4	\$3
price:	\$4	\$3	\$2	\$2
surplus:	\$1	\$3	\$2	\$1
2-demand agent buys:	✘	✔	✔	✘

Demand queries

demand query: given a vector of <u>prices</u>, returns a <u>demand set</u> (which may not be unique)




value:	v_1 = \$5	v_2 = \$6	v_3 = \$4	v_4 = \$3
price:	p_1 = \$4	p_2 = \$2	p_3 = \$2	p_4 = \$2
2-demand agent buys:	✘	✔	✔	✘

value:	$v_1 = \$5$	$v_2 = \$6$	$v_3 = \$4$	$v_4 = \$3$
price:	$p_1 = \$4$	$p_2 = \$2$	$p_3 = \$2$	$p_4 = \$2$
2-demand agent buys:	✘	✔	✔	✘
price:	$p_1 = \$2$	$p_2 = \$5$	$p_3 = \$3$	$p_4 = \$1.5$
2-demand agent buys:	✔	✘	✘	✔




value:	$v_1 = \$5$	$v_2 = \$6$	$v_3 = \$4$	$v_4 = \$3$
price:	$p_1 = \$4$	$p_2 = \$2$	$p_3 = \$2$	$p_4 = \$2$
2-demand agent buys:	✘	✔	✔	✘
price:	$p_1 = \$2$	$p_2 = \$5$	$p_3 = \$3$	$p_4 = \$1.5$
2-demand agent buys:	✔	✘	✘	✔
price:	$p_1 = \$7$	$p_2 = \$3.5$	$p_3 = \$5.5$	$p_4 = \$4$
2-demand agent buys:	✘	✔	✘	✘

Actively learning the valuations

- suppose there are <u>n items</u>, and the value $\underline{v_i}$ of each item is an <u>integer between 1 and W</u>

- how many demand queries suffice to learn the <u>full valuations (i.e., $(v_i)_i$)</u> of a k-demand agent?

- spoiler: optimal number of queries is

$$(n \log W) / (k \log (n / k)) + n / k \pm o(\ldots)$$

Sketch of lower bound

$(n \log W) / (k \log (n / k)) + n / k \pm o(\ldots)$

amount of information encoded in $(v_i)_i$

maximum amount of information per query

Sketch of lower bound

$(n \log W) / (k \log (n / k)) + n / k \pm o(\ldots)$

necessary in the following case:

- exactly one item is special, which has value 0
- all other items have value 1
- the special item is chosen uniformly at random

Sketch of upper bound

- warmup: $n = k = 1$
- need to learn: a single number v_1 in $\{1, 2, \ldots, W\}$
- query: given p, returns whether $p < v_1$
- optimal solution: binary search — $\log W$ queries

Sketch of upper bound

- slight generalization: $n = k$ ~~$(= 1)$~~
- need to learn: a vector $(v_i)_i$ of integers in $\{1, 2, \ldots, W\}$
- query: given $(p_i)_i$, returns, <u>for each item i</u>, whether $p_i < v_i$
- optimal solution: <u>simultaneous</u> binary search — log W queries

Sketch of upper bound

- general case: $n \geq k \geq 1$
- straightforward solution: (1) divide items into groups of size k, and (2) perform simultaneous binary search for each group sequentially
- $(n / k) \log W$ queries
- LB is $(n \log W) / (k \log (n / k))$ — can we do better?

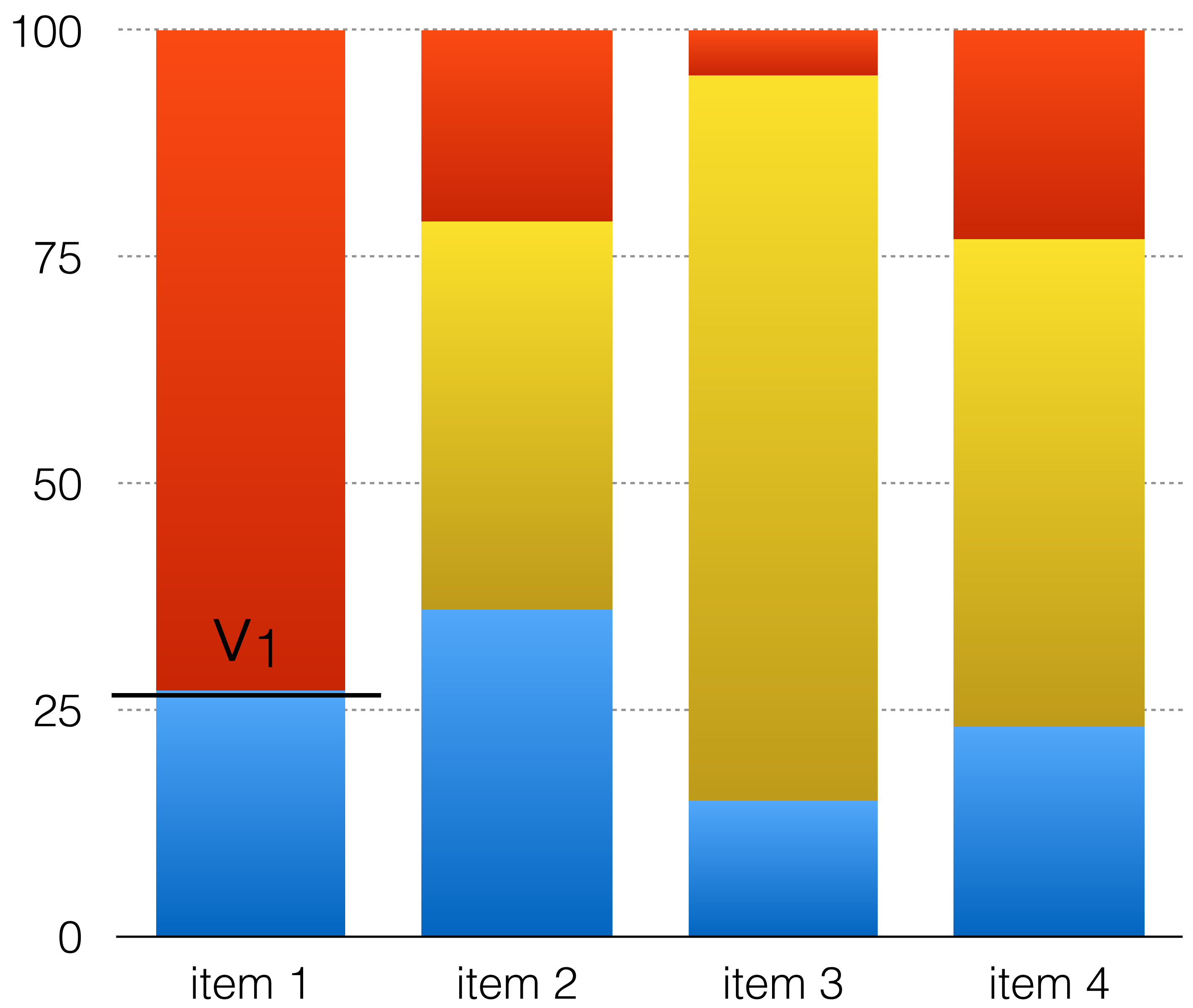
Sketch of upper bound

idea: biased binary search

- learn v_1 using log W queries, use item 1 as reference
- in each query, post $p_1 = v_1 - 0.5$, so item 1 is marginally attractive
- for all other items, post biased (rather than middle-of-possible-range) prices

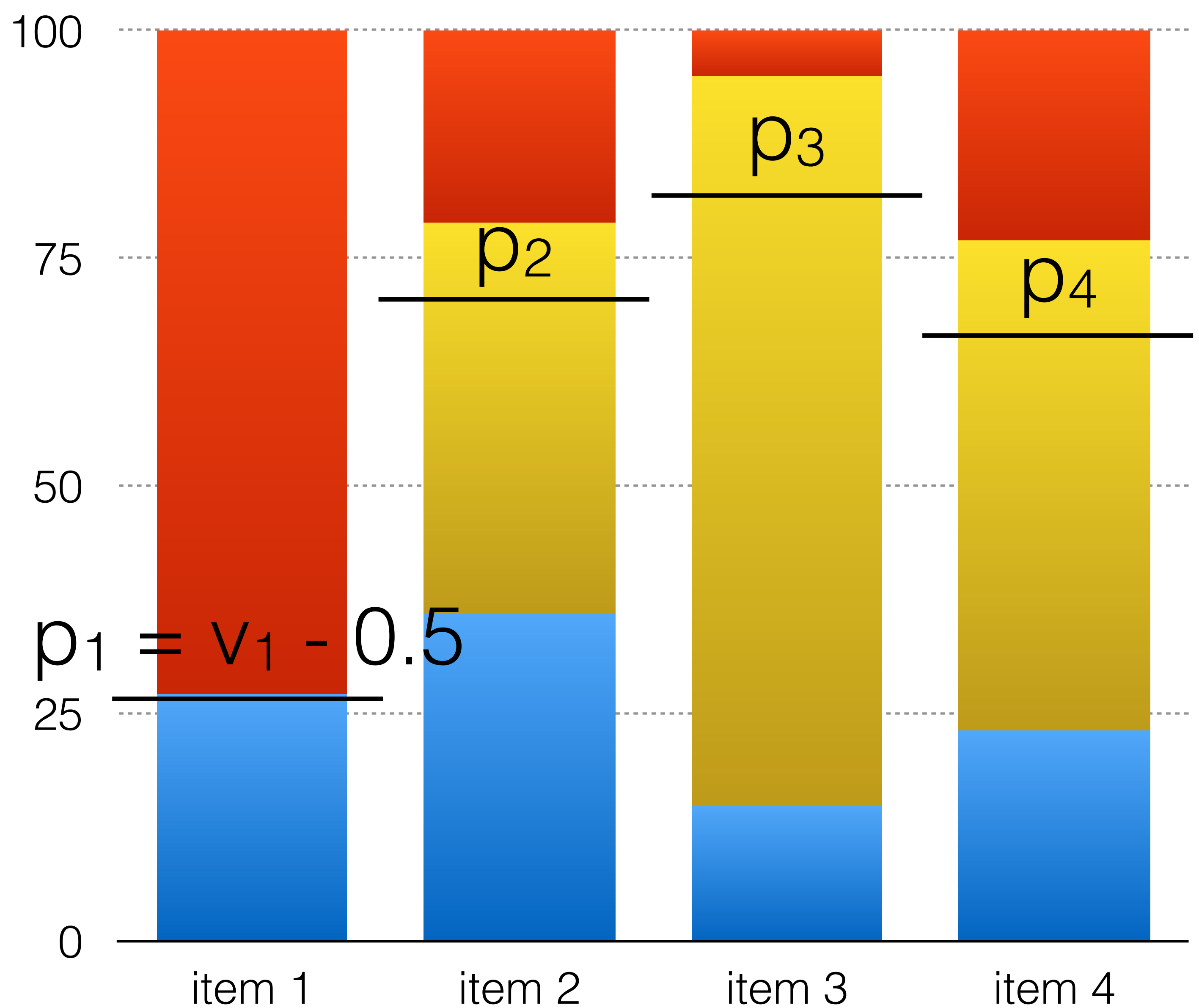
Sketch of upper bound

n = 4
k = 1



Sketch of upper bound

n = 4
k = 1



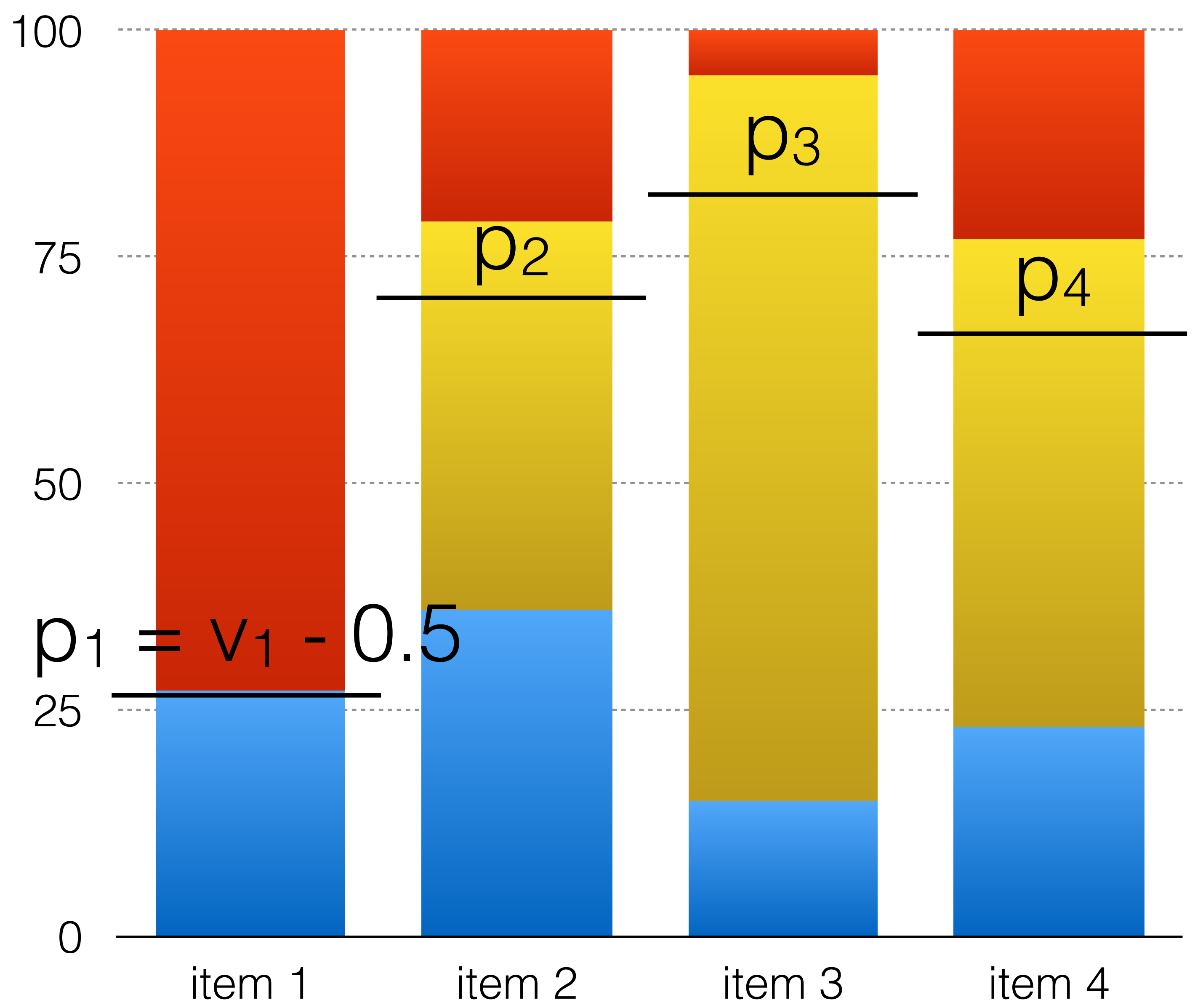
prices biased toward higher end of possible ranges

Sketch of upper bound

- in each query, post $p_1 = v_1 - 0.5$, so item 1 is marginally attractive
- for all other items, post biased (rather than middle-of-possible range) prices
- if item 1 in demand set: many items are overpriced; shrink their possible ranges by a little
- if item 1 not in demand set: a few items are underpriced; shrink their possible ranges by a lot

Sketch of upper bound

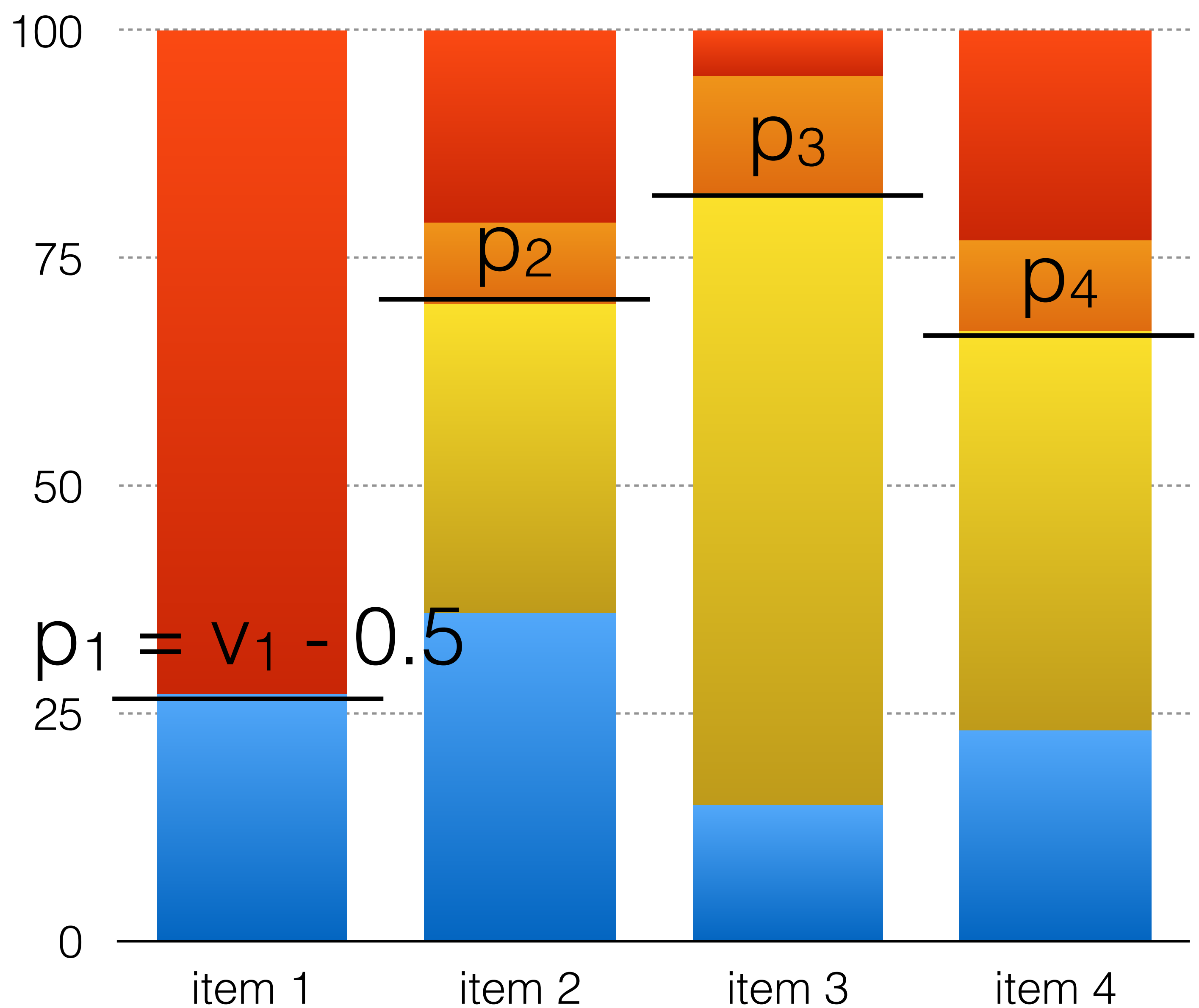
$n = 4$
$k = 1$



if item 1 in demand set: <u>many</u> items are overpriced; shrink their possible ranges by <u>a little</u>

Sketch of upper bound

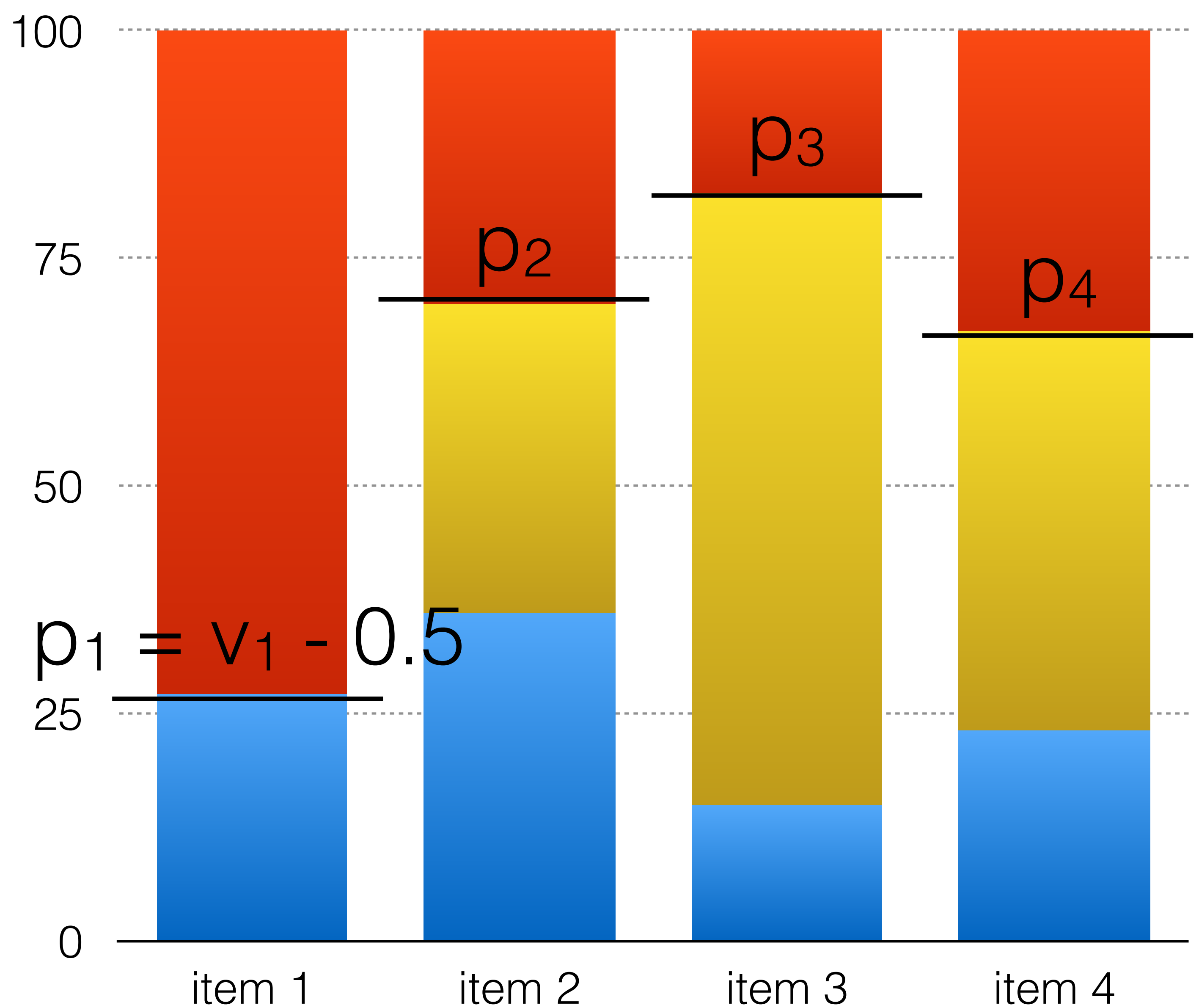
$n = 4$
$k = 1$



if item 1 in demand set: <u>many</u> items are overpriced; shrink their possible ranges by <u>a little</u>

Sketch of upper bound

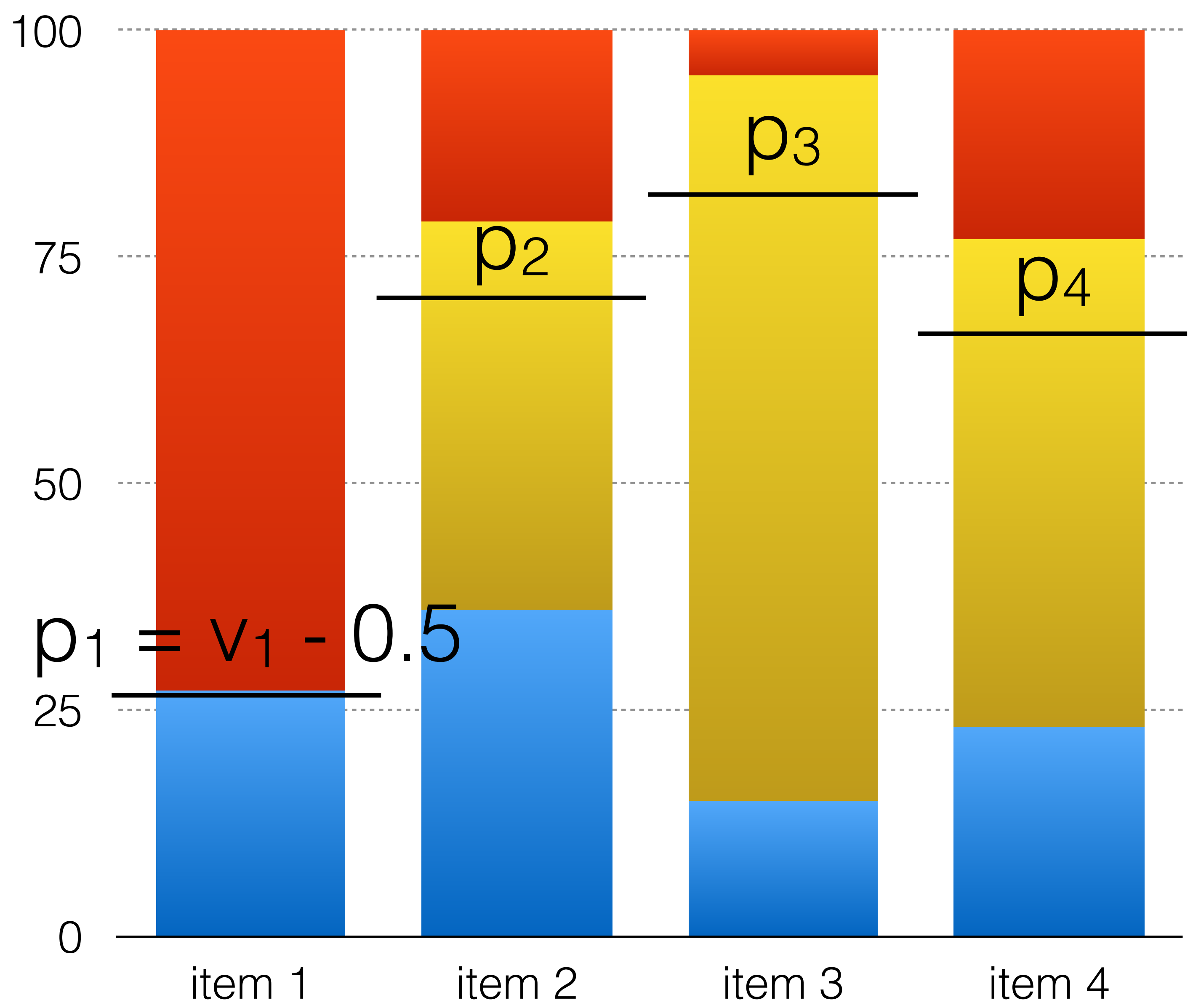
n = 4
k = 1



if item 1 in demand set: <u>many</u> items are overpriced; shrink their possible ranges by <u>a little</u>

Sketch of upper bound

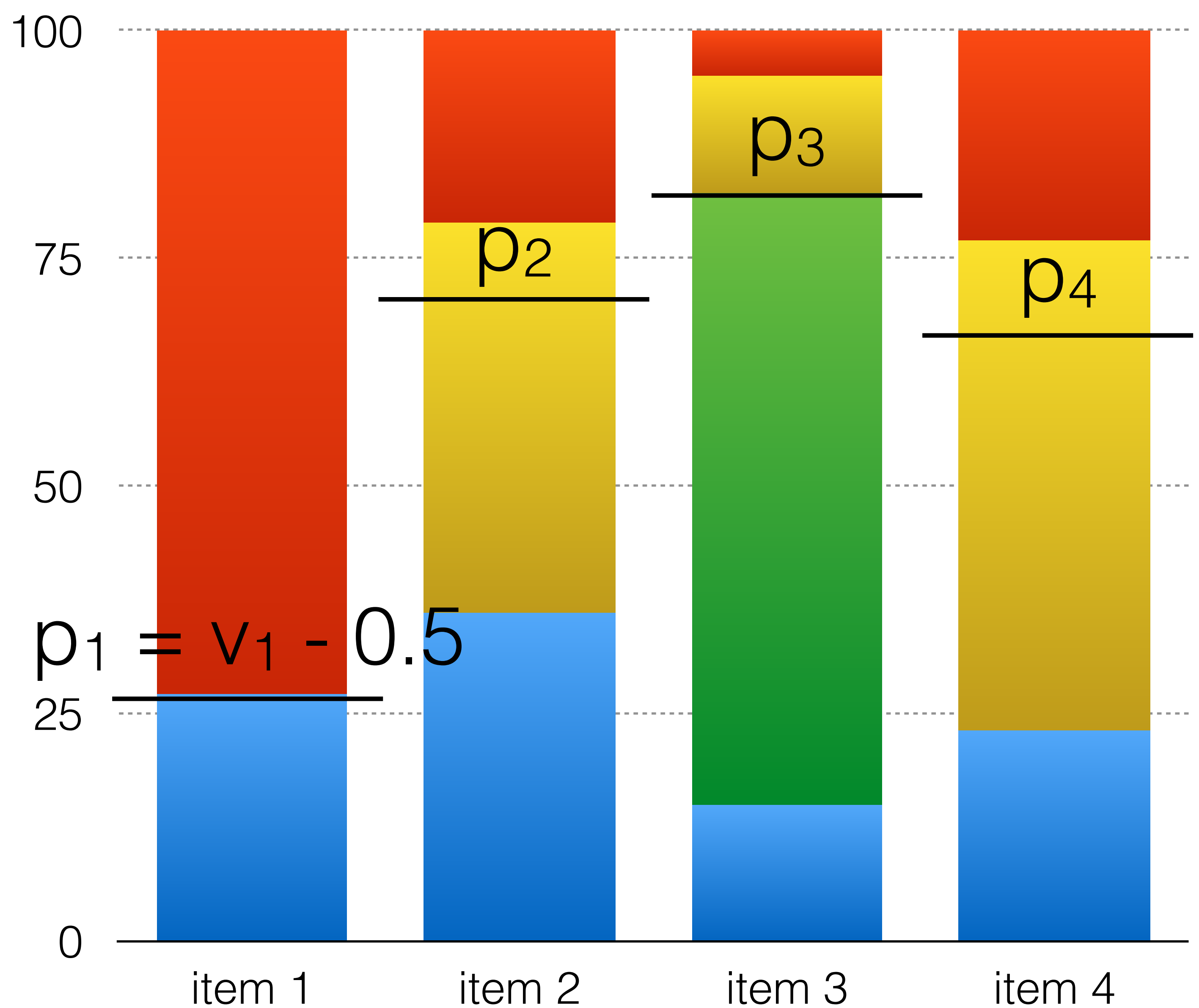
$n = 4$
$k = 1$



if item 1 not in demand set: <u>a few</u> items are underpriced; shrink their possible ranges by <u>a lot</u>

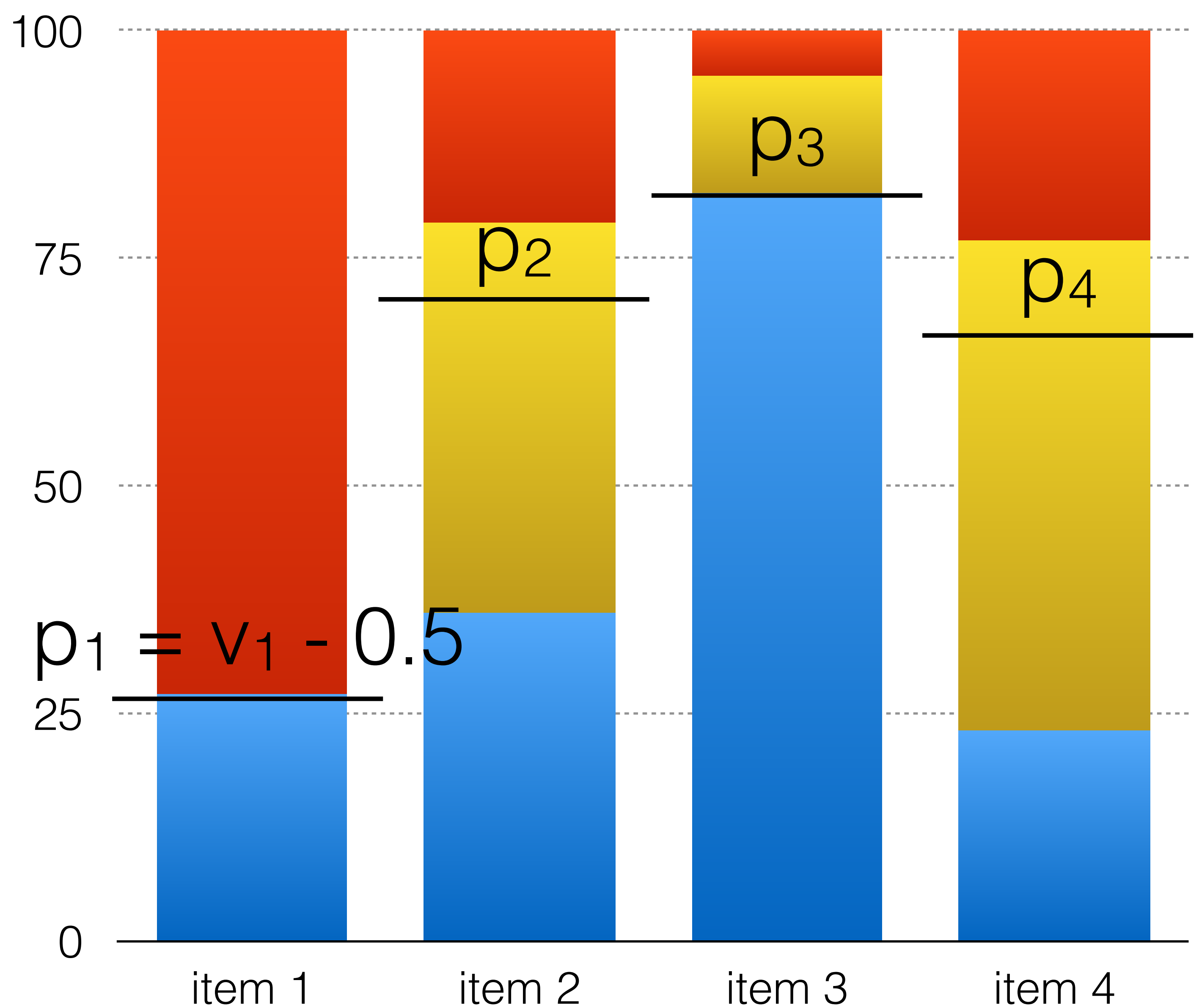
Sketch of upper bound

$n = 4$
$k = 1$



if item 1 not in demand set: <u>a few</u> items are underpriced; shrink their possible ranges by <u>a lot</u>

Sketch of upper bound



$n = 4$
$k = 1$

if item 1 not in demand set: <u>a few</u> items are underpriced; shrink their possible ranges by <u>a lot</u>

Sketch of upper bound

- if item 1 in demand set: <u>many</u> items are overpriced; shrink their possible ranges by <u>a little</u>
- if item 1 not in demand set: a <u>few</u> items are underpriced; shrink their possible ranges by <u>a lot</u>
- adjust bias to equalize information gain
- larger information gain (~ k log (n / k)) in both cases!

- so far: tight UB & LB for active learning
- next: (very brief discussion of) computation & sample efficient algorithm for passive learning

Passively learning valuations

- prices are distributed according to a distribution $\mathscr{D}$
- true valuations v: a vector of real numbers
- algorithm observes m iid sample price vectors p^j together with demand set S^j under p^j
- given $\{(S^j, p^j)\}$, algorithm outputs a hypothesis vector h which recovers v in a PAC sense — algorithm succeeds with probability 1 - δ, in which case with probability 1 - ε, demand set under (v, p) = demand set under (h, p)

Passively learning valuations

- idea: empirical risk minimization
- tool: multiclass ERM principle & Natarajan dimension
- treat problem as multiclass classification with < n^k labels
- hypothesis class has <u>Natarajan dimension n</u>
- sample complexity is poly(n, k, log(1 / δ), 1 / ε)
- solving ERM = finding a feasible solution to an LP

Future directions

- more general valuations, e.g., matroid-demand
- tighter sample complexity bounds for passive learning

Thanks for your attention!

Questions?

Related research

- in economic theory: learning utility functions from revealed preferences (Samuelson, 1938; Afriat, 1967; Beigman & Vohra, 2006; …)

- in CS: preference elicitation (Blum et al., 2004; Lahaie & Parkes, 2004; Sandholm & Boutilier, 2006; …)