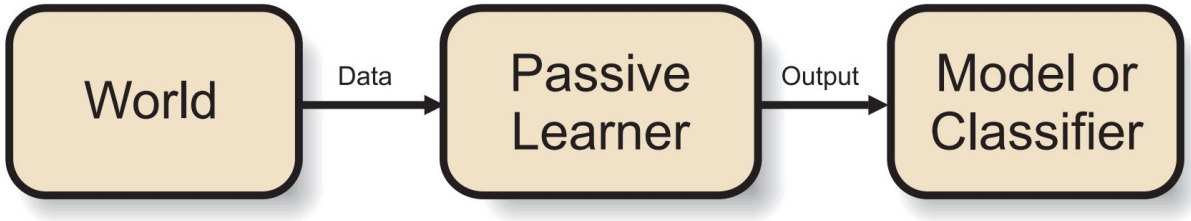
Near Optimal Bayesian Active Learning for Decision Making

Quick review: Active vs. Passive Learning

Passive learning: receive labeled training data, then train on supervised learning set.



Active learning: (selective, supervised) receive unlabeled training examples, choose which examples to get labeled and learn from

Types of active learning

Selective (supervised):

query= choice of unlabelled sample

response= corresponding label given from training data

output= classifier

Active learning for decision making:

query= experiment/test

response= observation

output= correct hypothesis, or decision corresponding to correct hypothesis

20 Questions

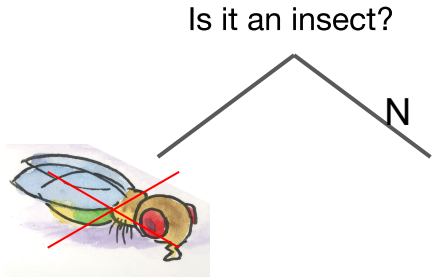
- Player 1 chooses an object/topic without revealing it
- Player 2 can ask up to 20 yes/no questions
- Goal is for player 2 to decide what object/topic player 1 chose after 20 questions (or fewer) have been asked

simplified setting: think of one of the following six animals



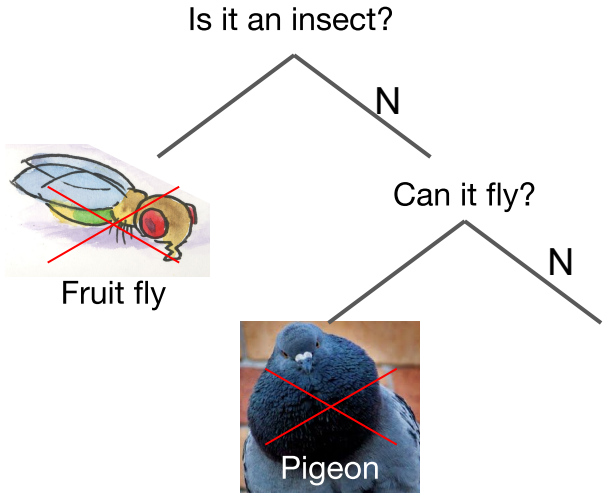



Bad strategy



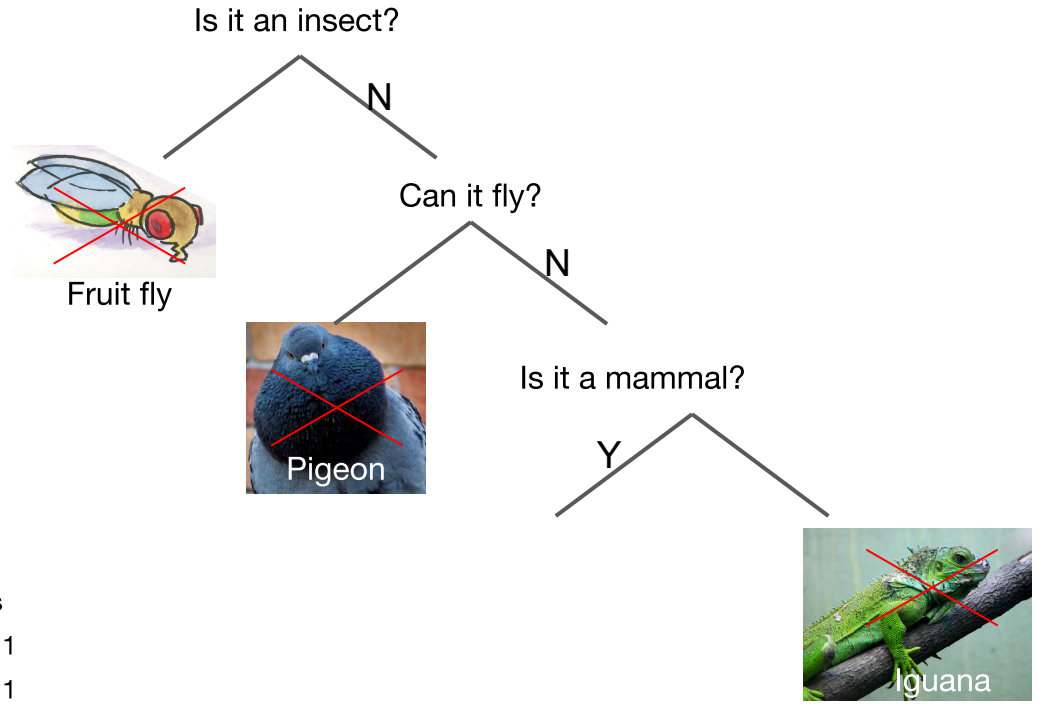
	Mammal	Aquatic	Flies	Insect	Lays eggs
Fruit fly	0	0	1	1	1
Iguana	0	0	0	0	1
Pigeon	0	0	1	0	1
Platypus	1	1	0	0	1
Squirrel	1	0	0	0	0
Whale	1	1	0	0	0

Bad strategy



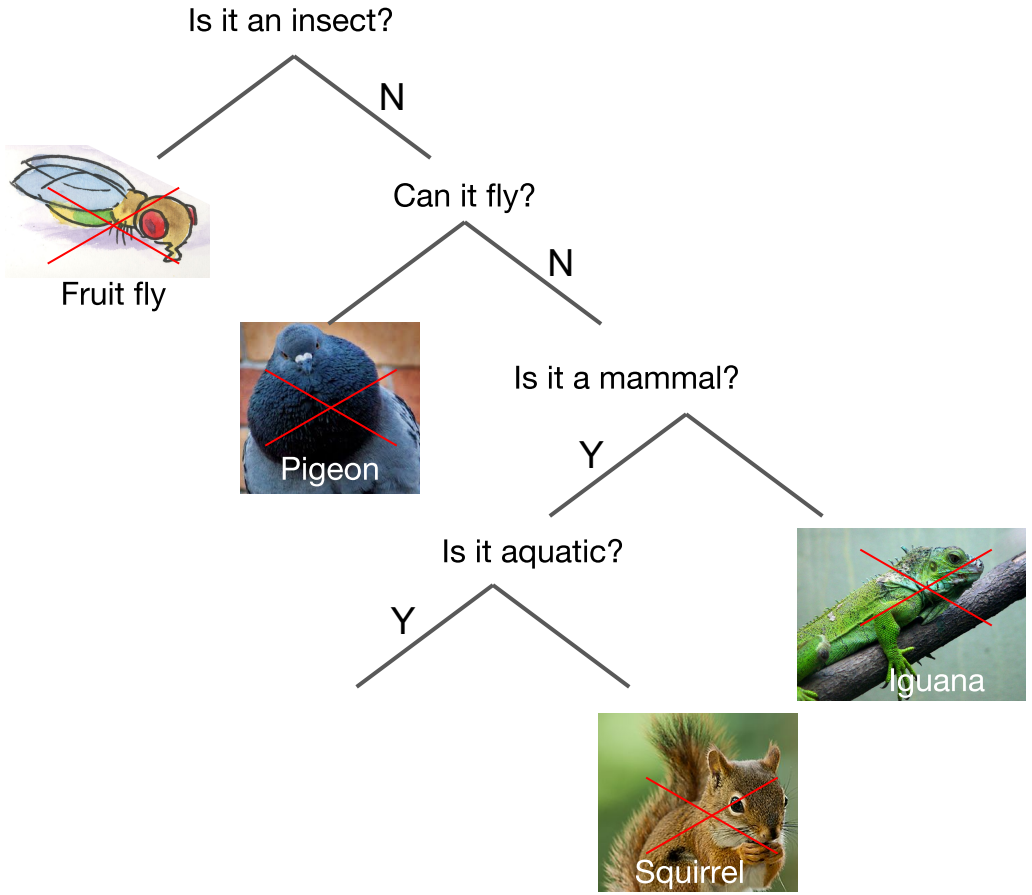
	Mammal	Aquatic	Flies	Lays eggs
Iguana	0	0	0	1
Pigeon	0	0	1	1
Platypus	1	1	0	1
Squirrel	1	0	0	0
Whale	1	1	0	0

Bad strategy



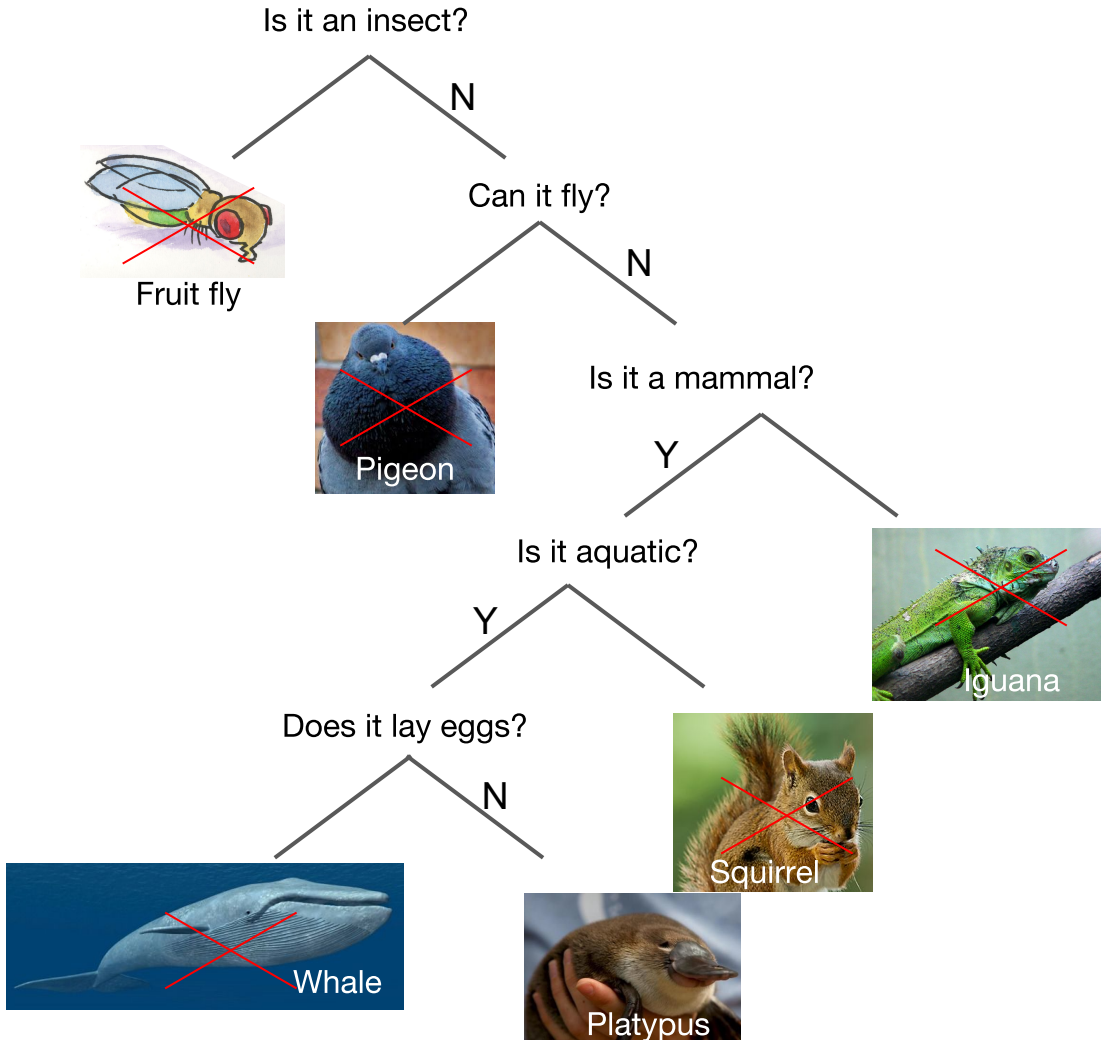
	Mammal	Aquatic	Lays eggs
Iguana	0	0	1
Platypus	1	1	1
Squirrel	1	0	0
Whale	1	1	0

Bad strategy



	Aquatic	Lays eggs
Platypus	1	1
Squirrel	0	0
Whale	1	0

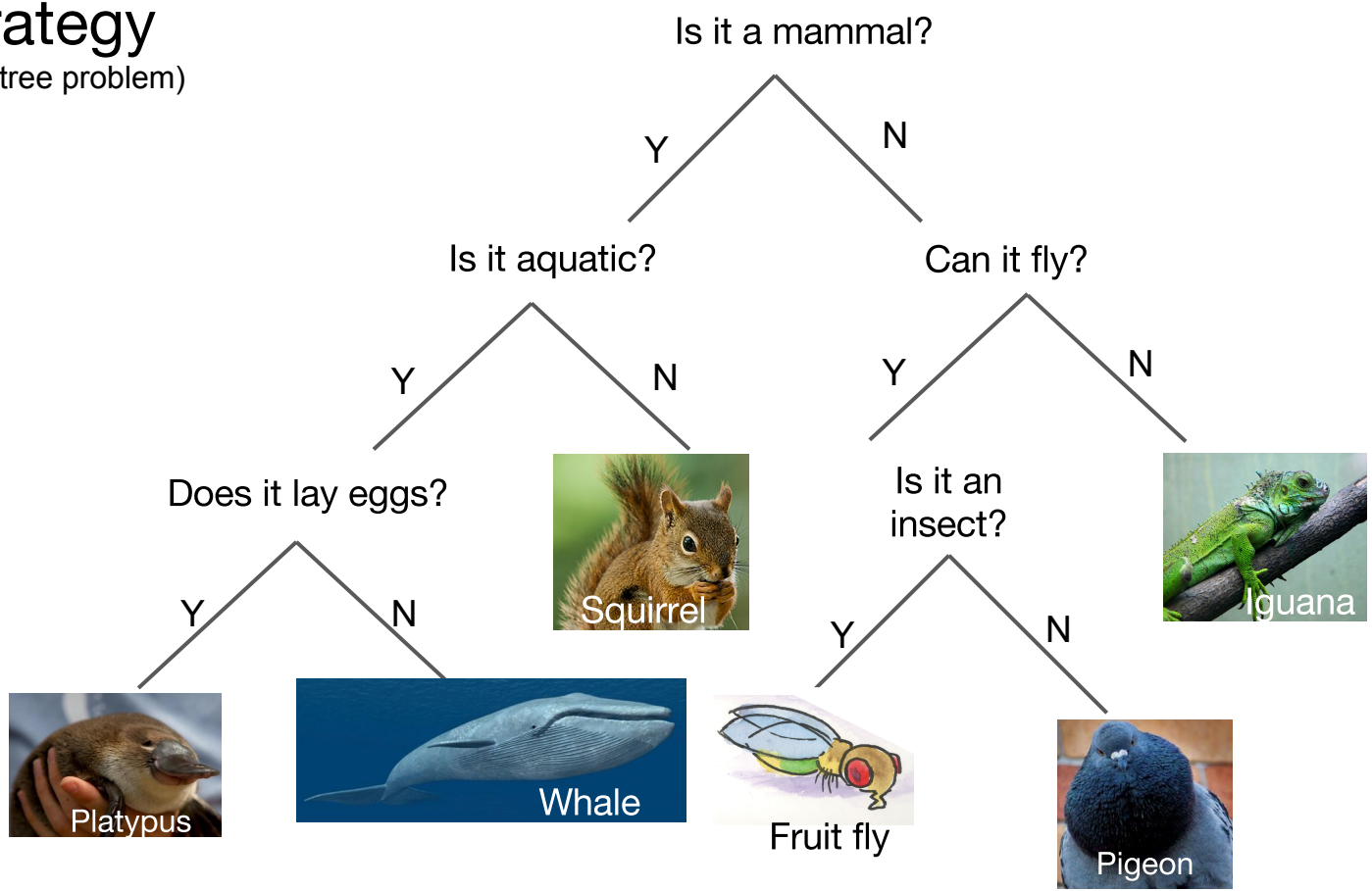
Bad strategy



	Lays eggs
Platypus	1
Whale	0

Good strategy

(optimal decision tree problem)

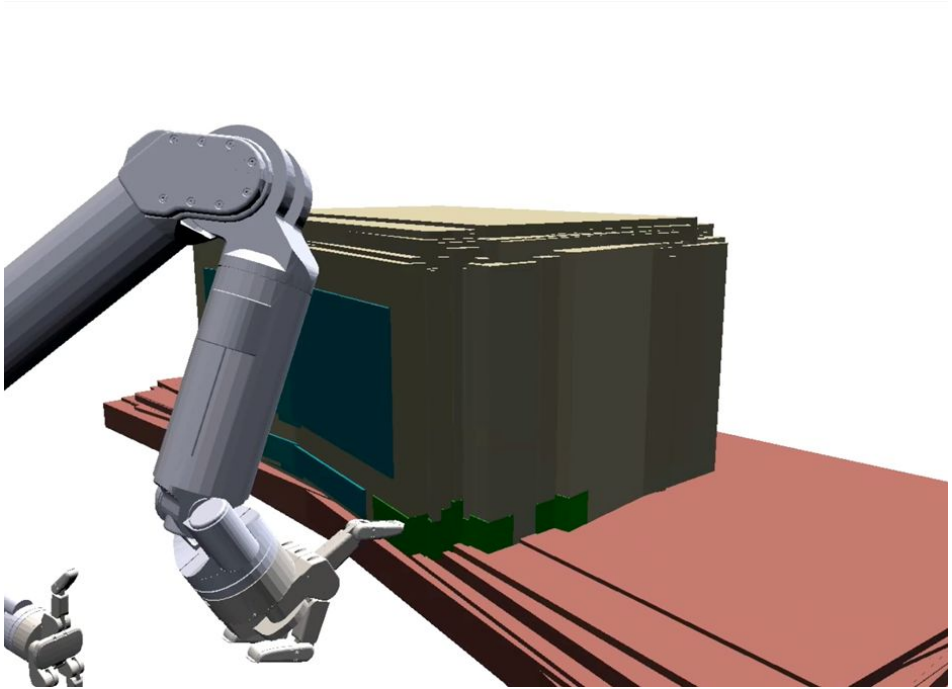


Robotics

Query: localization trajectory

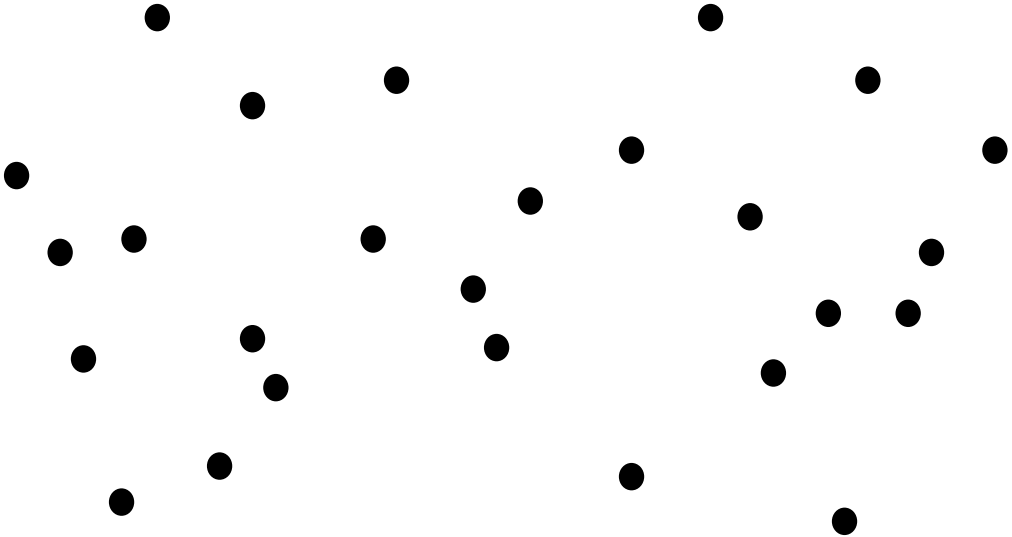
Response: distance until contact

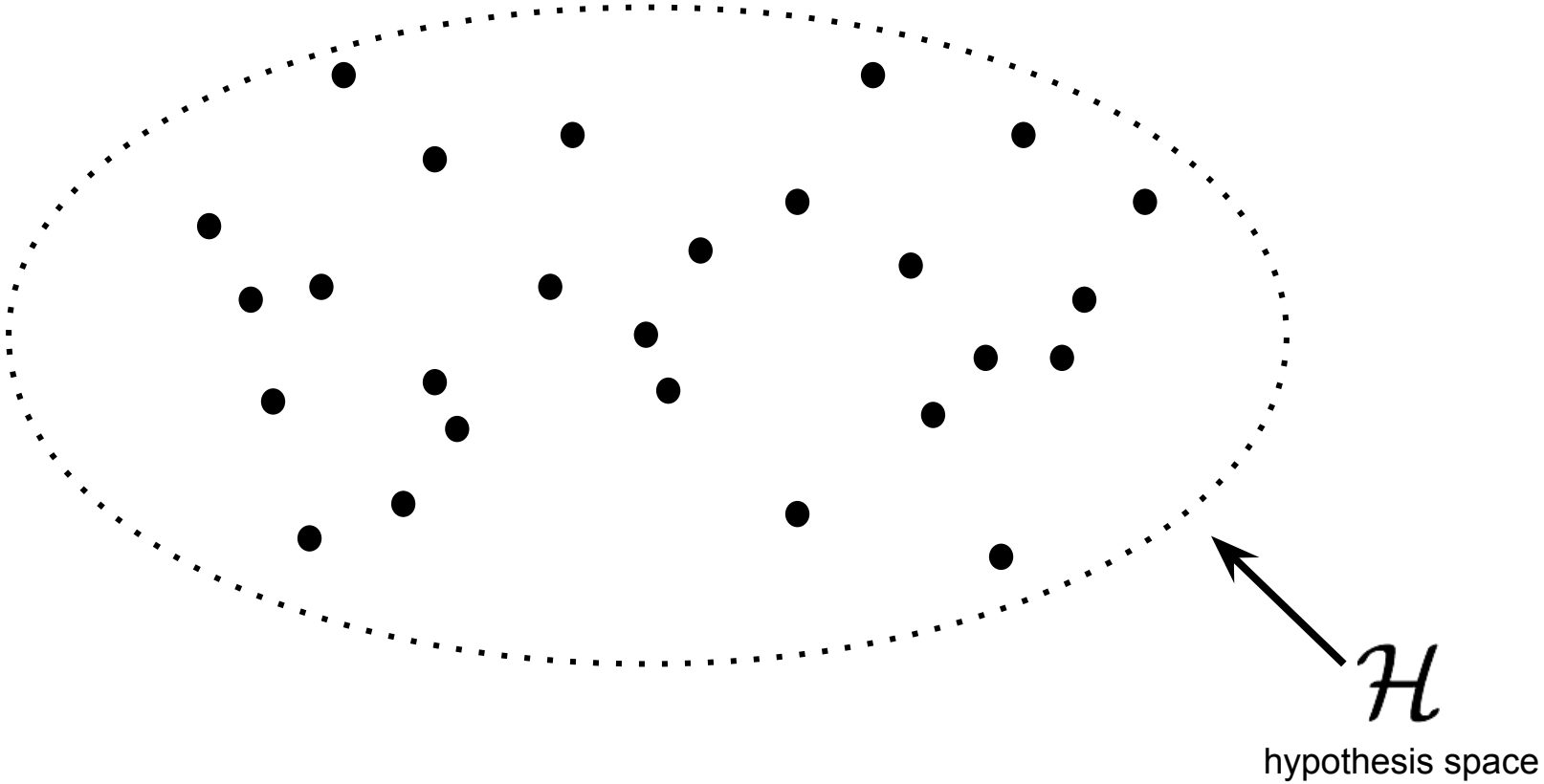
Output: button-push trajectory

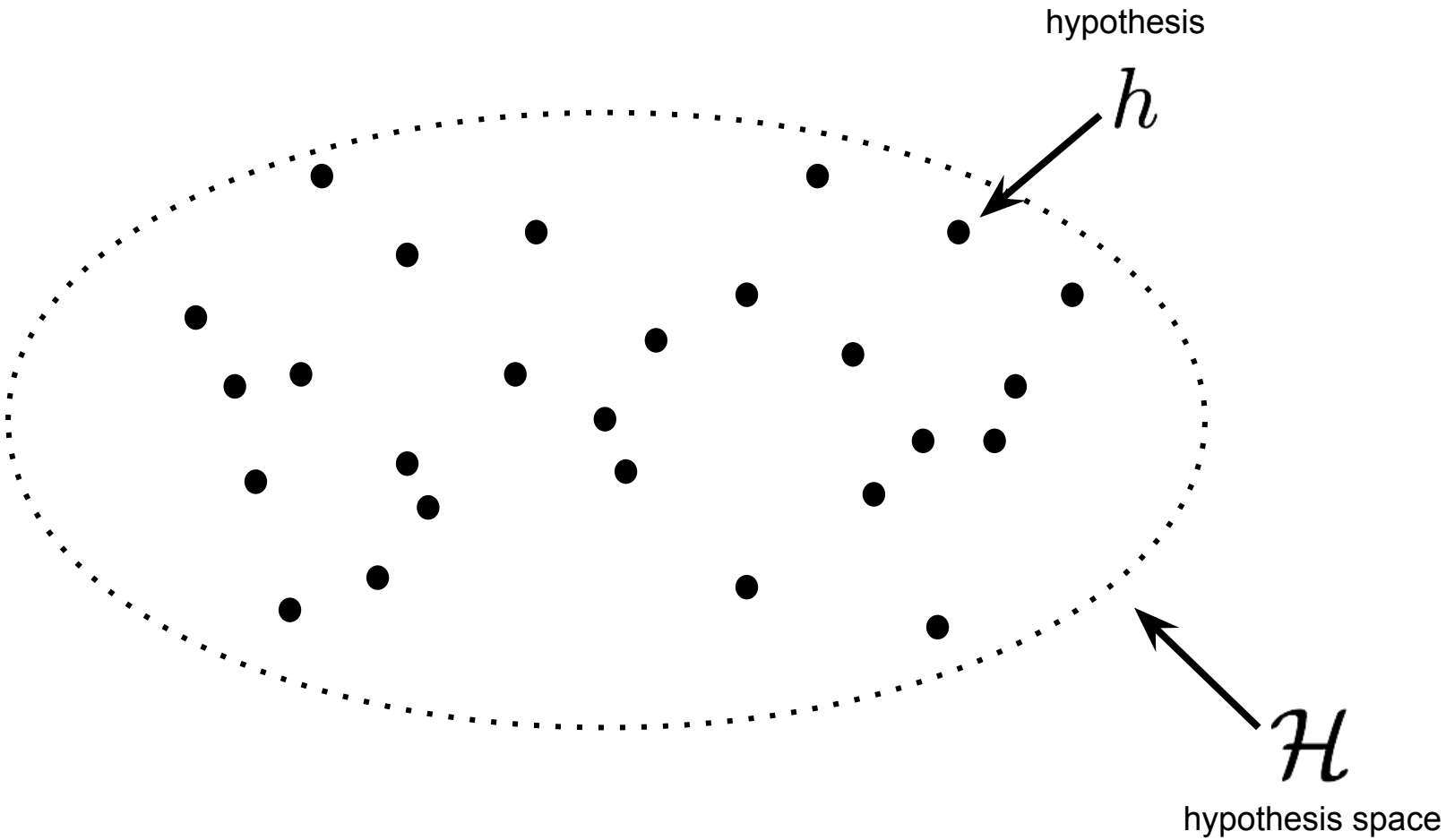


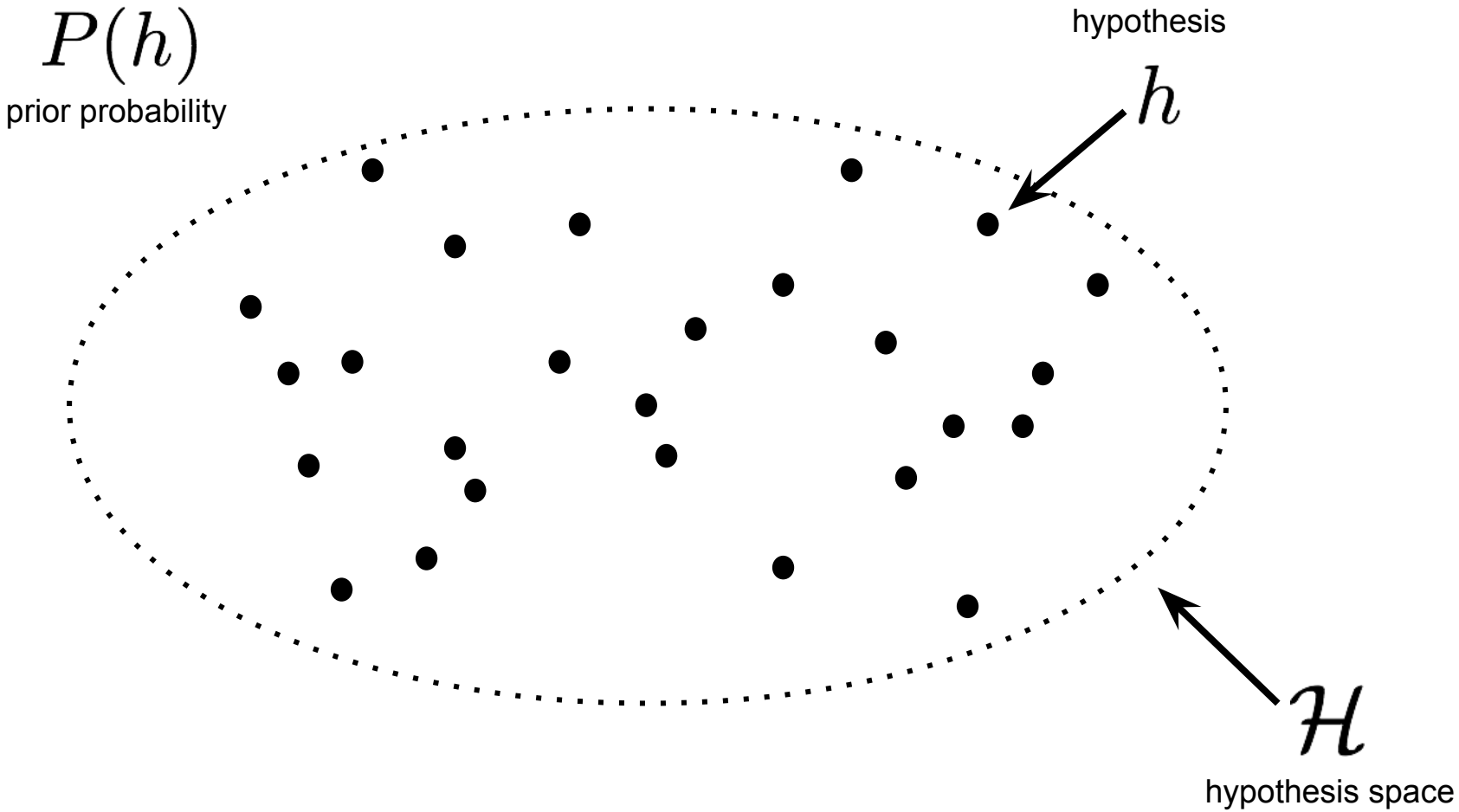
Javdani, Shervin. "Gathering Information For Decision Making In Touch Based Localization". Presentation.

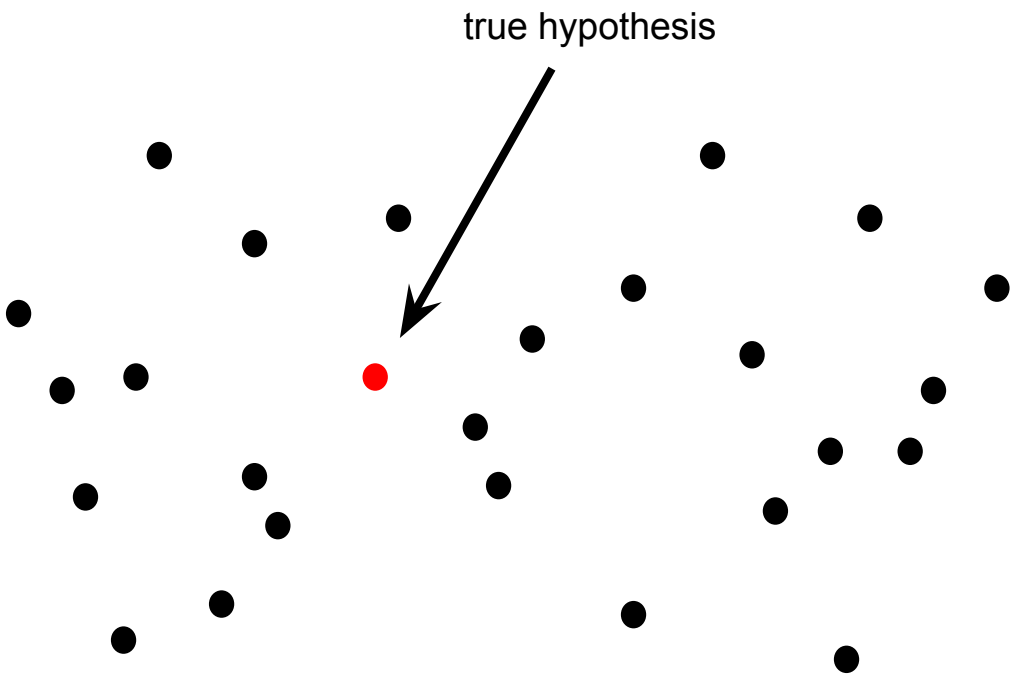
Setting




$\mathcal{H}$
hypothesis space


hypothesis
h
$\mathcal{H}$
hypothesis space


$P(h)$
prior probability
hypothesis
h
$\mathcal{H}$
hypothesis space


true hypothesis

test

$$t \in \mathcal{T}$$

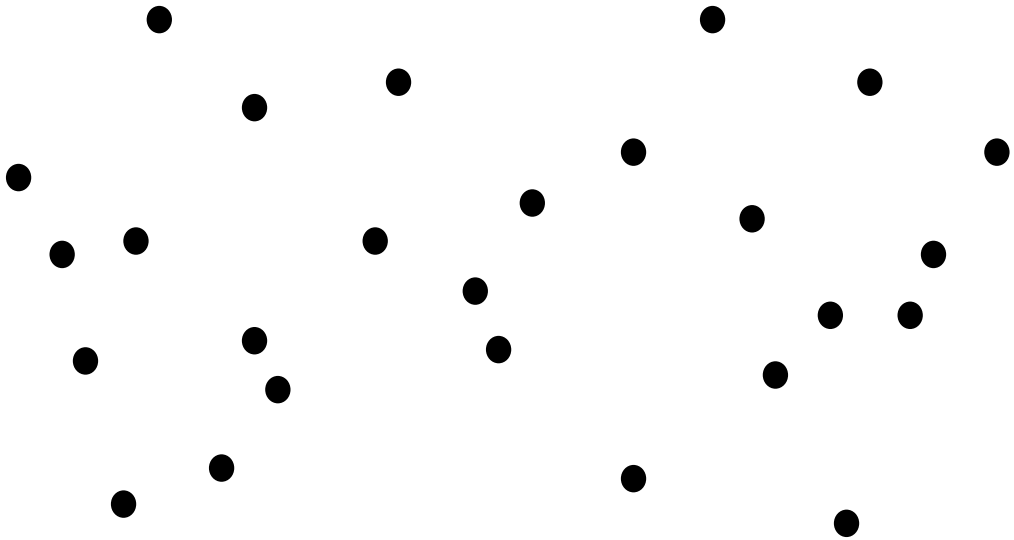
outcome

$$o \in \mathcal{O}$$

hypotheses map tests to outcomes

$$h : \mathcal{T} \to \mathcal{O}$$

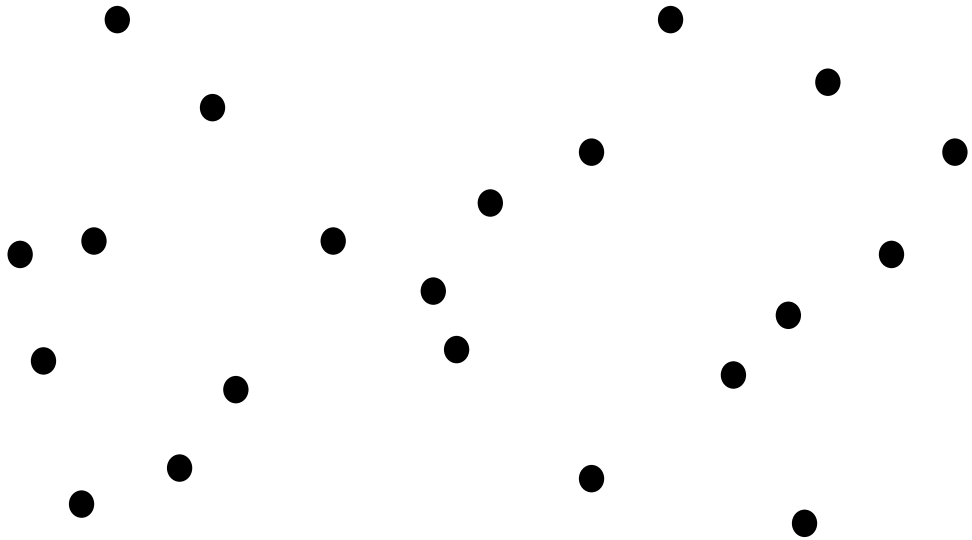
$$h(t) = o$$



run test t_1

observe outcome o_1

rule out inconsistent hypotheses



run test t_2

observe outcome o_2

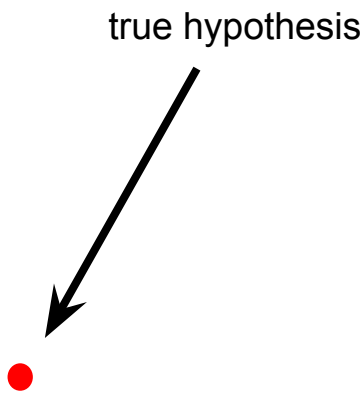
rule out inconsistent hypotheses

continue running tests...

run test t_m

observe outcome o_m

rule out inconsistent hypotheses


true hypothesis

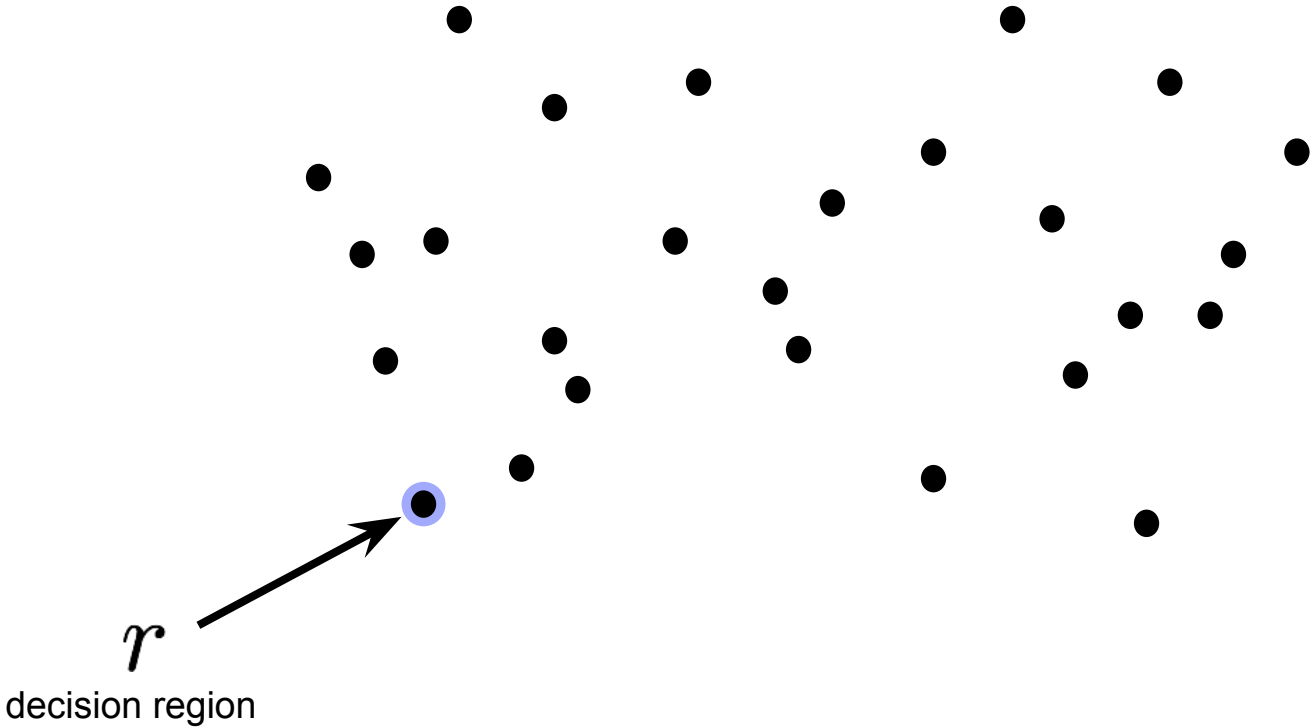
evidence

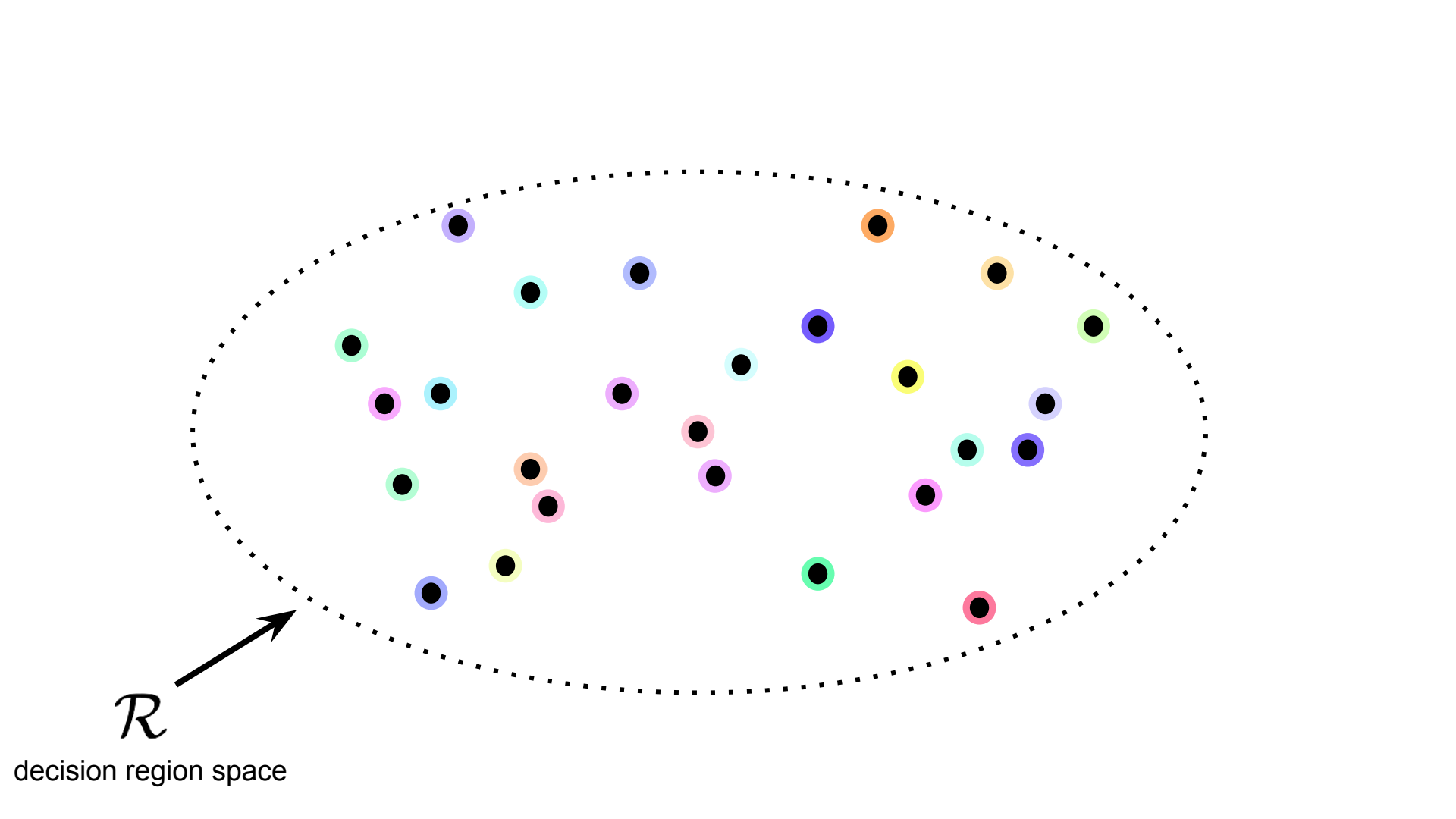
$$\mathcal{S} = \{(t_1, o_1), \ldots (t_m, o_m)\}$$

consistent hypotheses

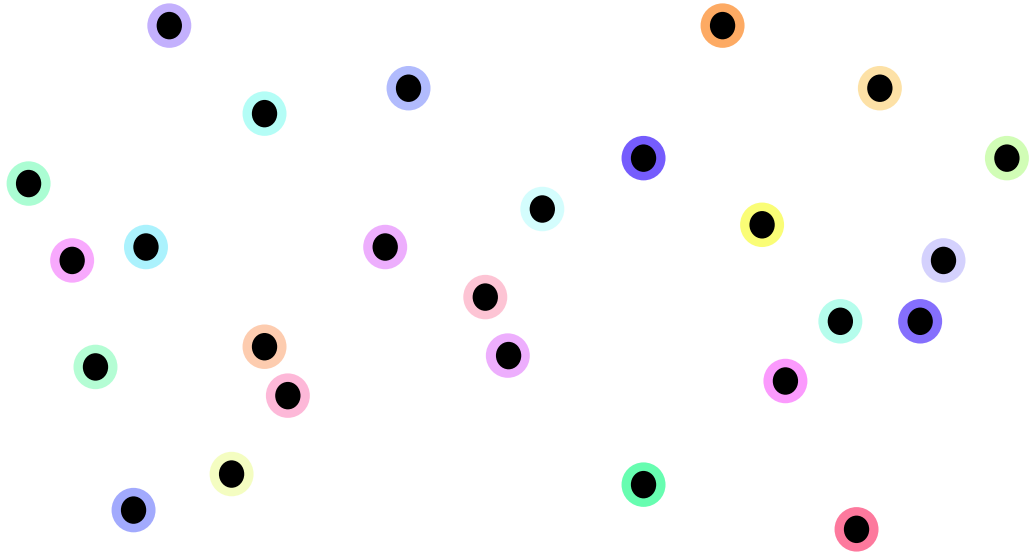
$$\mathcal{V}(\mathcal{S}) = \{h \in \mathcal{H} : \forall (t, o) \in \mathcal{S}, h(t) = o\}$$

choose tests in a systematic way in order to maximally reduce uncertainty in the **hypothesis**


r
decision region

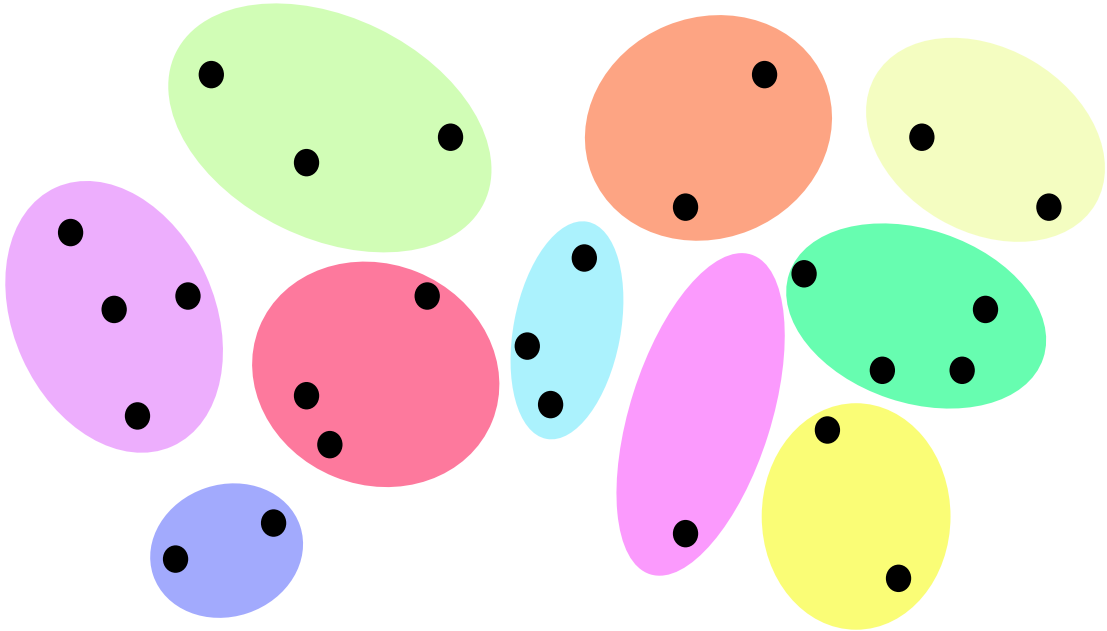

$\mathcal{R}$
decision region space

single hypothesis per decision region, disjoint



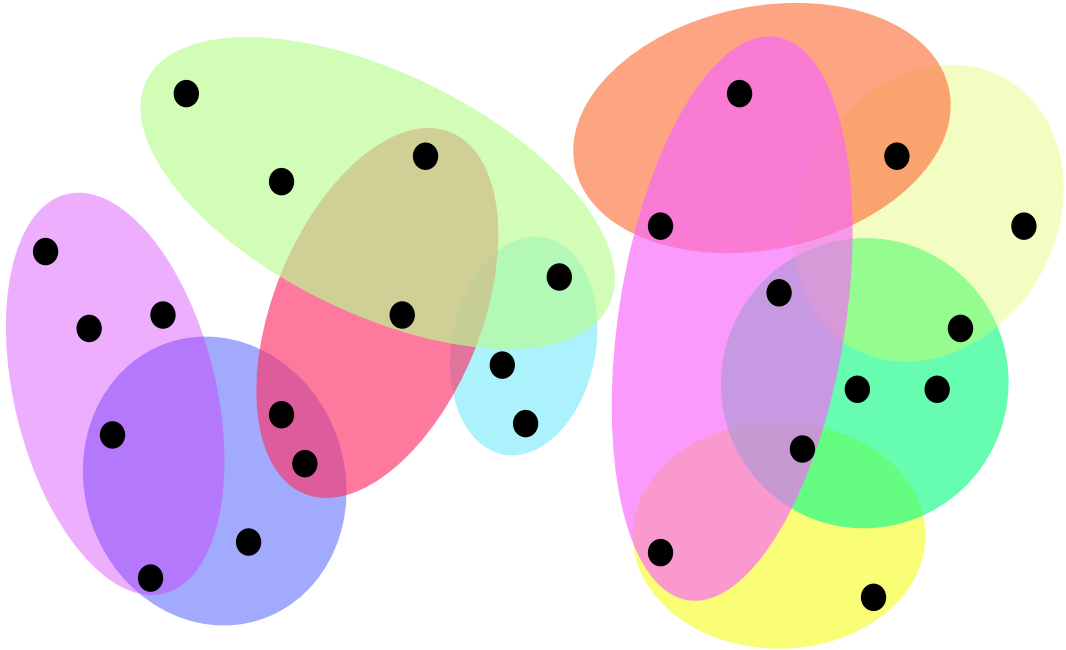
optimal decision tree (ODT) problem → greedy binary search

multiple hypotheses per decision region, disjoint



equivalence class determination (ECD) problem

multiple hypotheses per decision region, non-disjoint

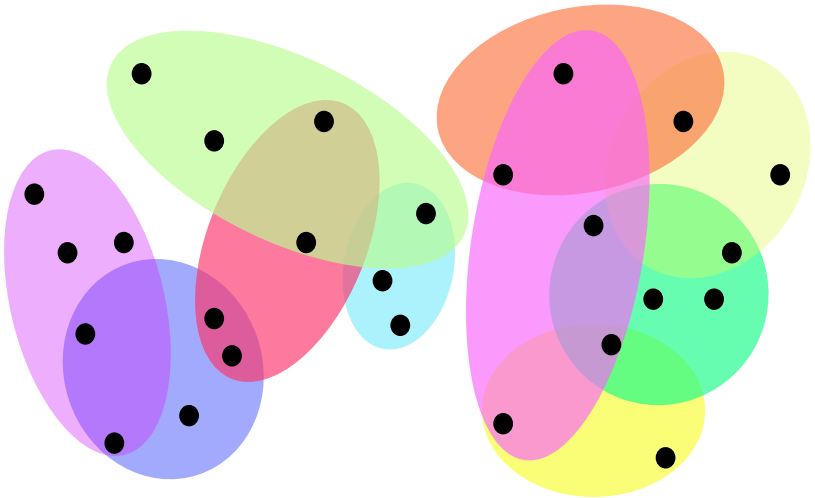


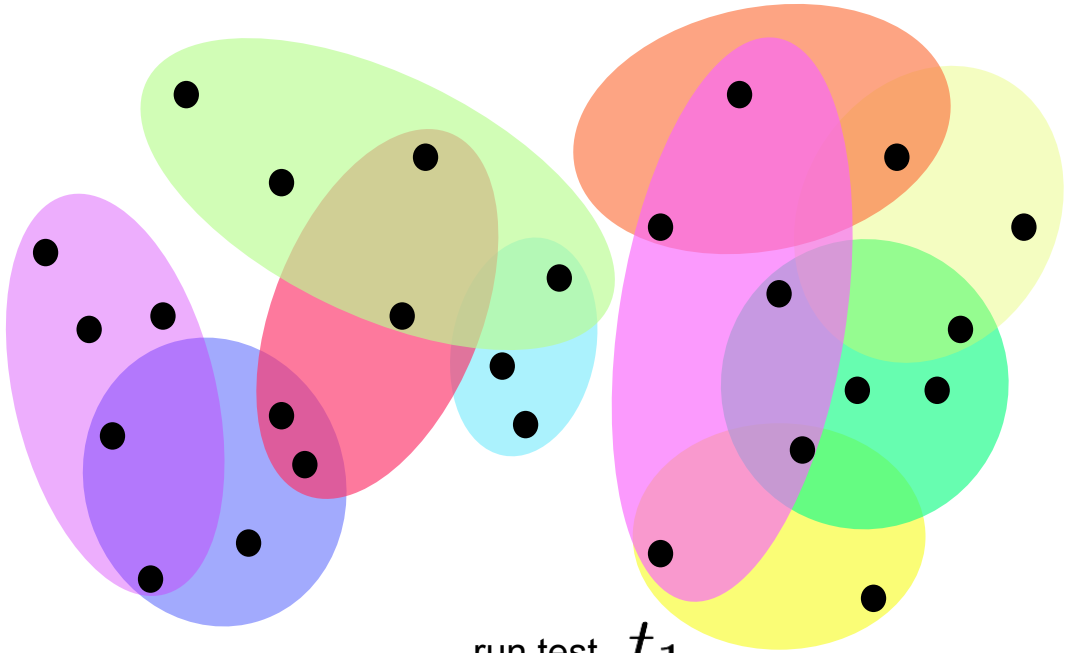
decision region determination (DRD) problem

Given:

- Hypotheses
- Tests
- Decision Regions
- Prior probability distributions over set of hypotheses

All decisions are acceptable, one decision is not preferred over the other

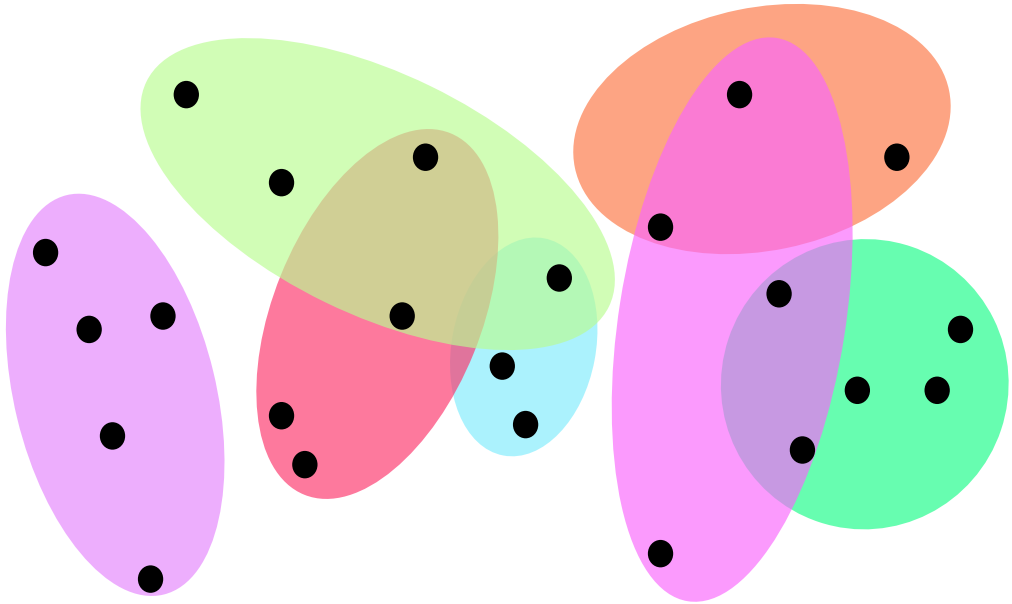




run test t_1

observe outcome o_1

rule out inconsistent hypotheses

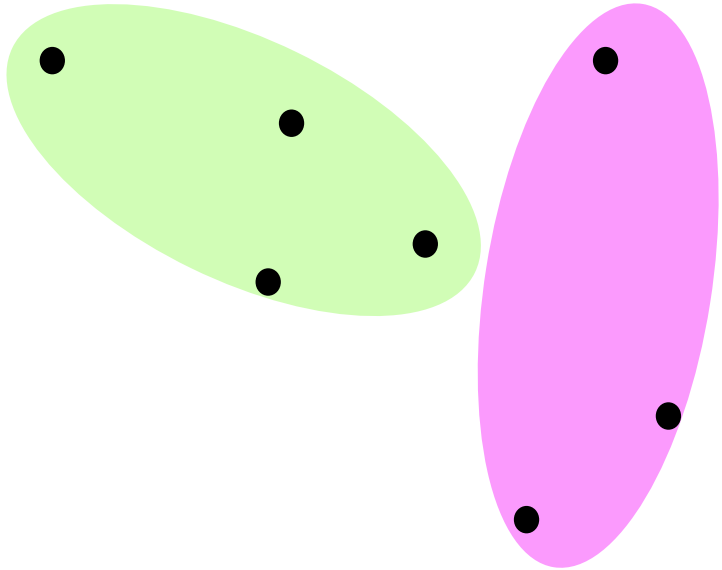


run test t_2

observe outcome o_2

rule out inconsistent hypotheses

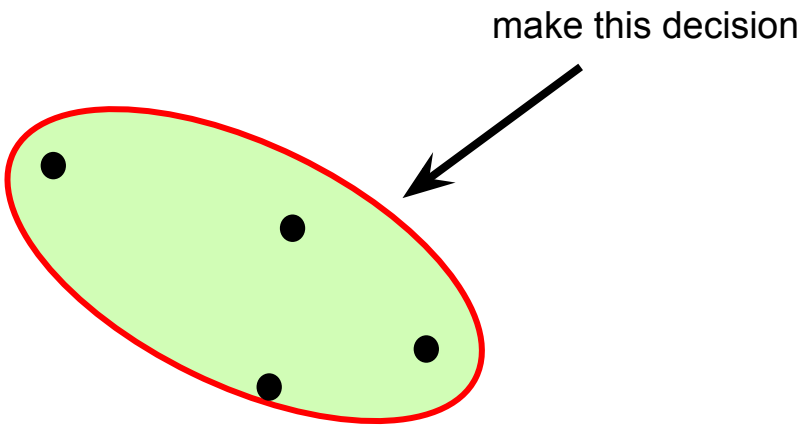
continue running tests...



run test t_m

observe outcome o_m

rule out inconsistent hypotheses


make this decision

choose tests in a systematic way in order to drive uncertainty in the hypothesis toward a **single decision**

use policy π to choose which tests to run

a policy maps the current evidence $\mathcal{S}$ to the next test to run (or to stop running tests)

a *feasible* policy π eventually finds the decision region containing the correct hypothesis; it drives all remaining uncertainty in the hypotheses toward a single decision region

$$\longrightarrow \mathcal{V}(\mathcal{S}) \subseteq r$$

expected cost of feasible policy π

$$\mathcal{C}(\pi) = \sum_{h \in \mathcal{H}} P(h) |\mathcal{T}(\pi, h)|$$

$\mathcal{T}(\pi, h)$ is the set of tests chosen by policy π when h is the correct hypothesis

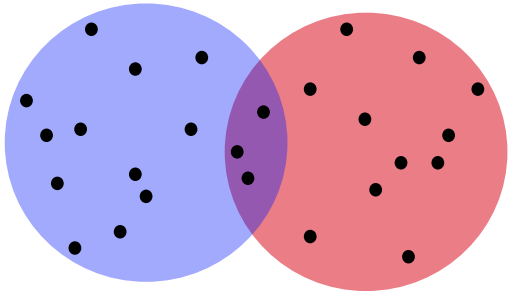
optimal policy π^*

$$\pi^* = \text{argmin}_\pi \ \mathcal{C}(\pi) \qquad \text{s.t. } \forall h, \exists r : \mathcal{V}(\mathcal{T}(\pi, h)) \subseteq r$$

(the policy satisfies feasibility)

Define a hypergraph $\mathbf{G} = (X, E)$ where X is a set of **nodes** and E is a collection of sets of X called **hyperedges**

a **region hypergraph** is defined as $\mathbf{G}^r = (\mathcal{H}, \mathcal{R})$



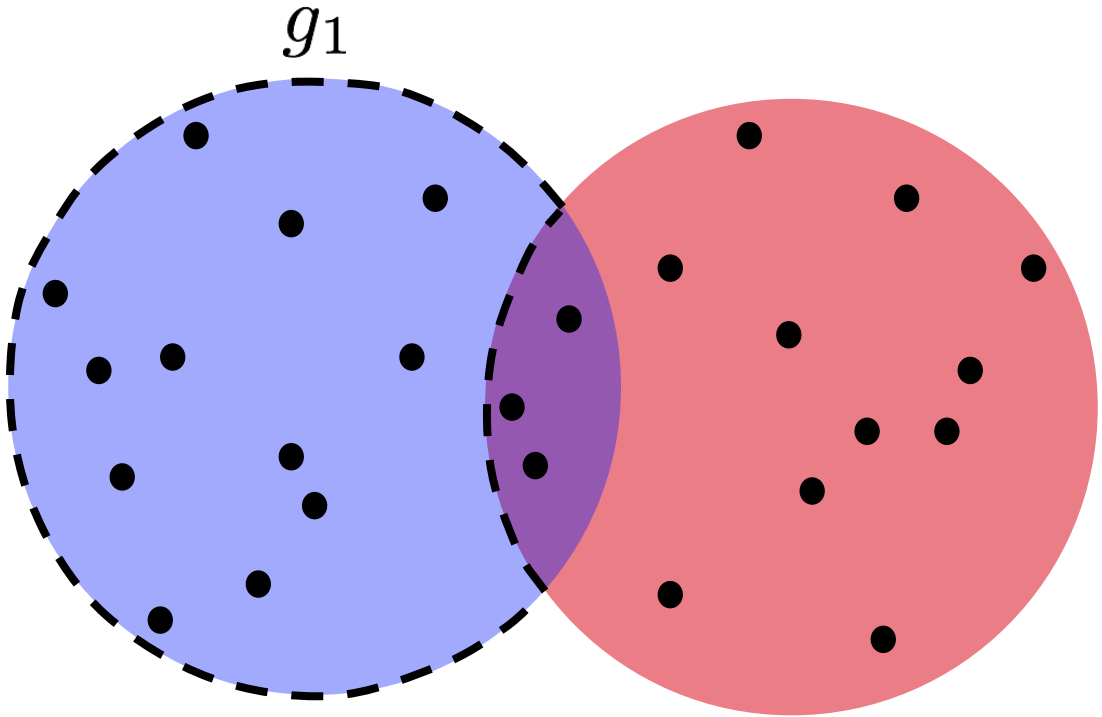
i.e. each hypothesis is a node, and each decision region is a hyperedge containing a set of hypotheses

A **splitting hypergraph** $\mathbf{G}^s = (\mathcal{G}, \mathcal{E})$ can be constructed

Each subregion $g \in \mathcal{G}$ is a node

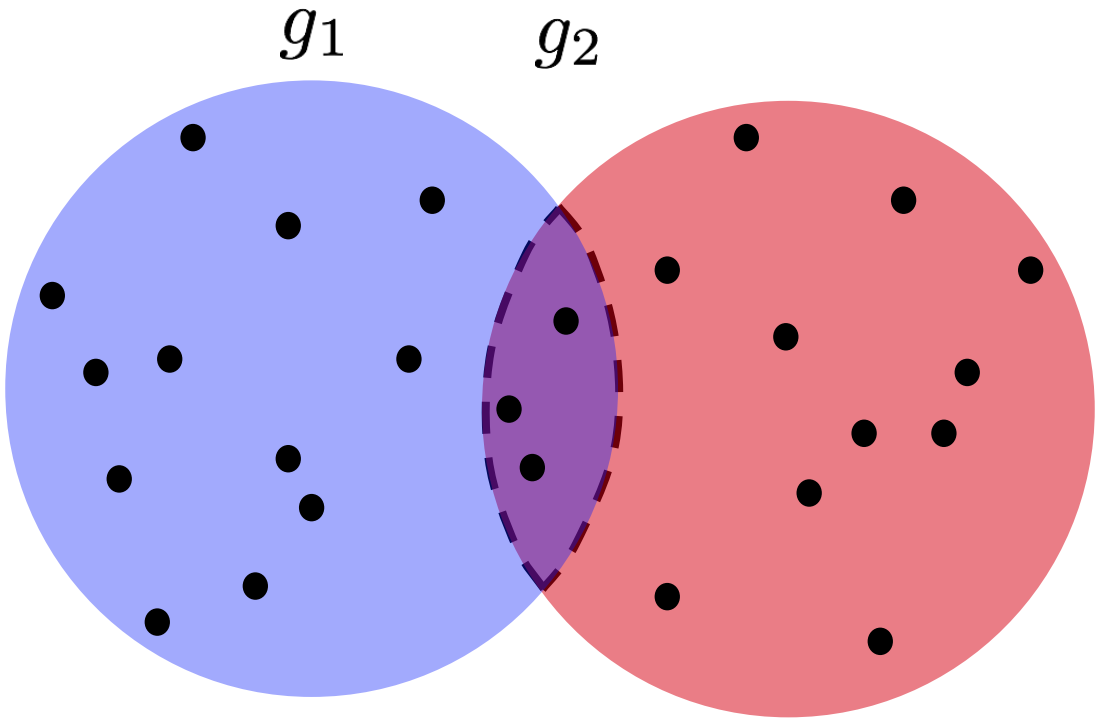
A **splitting hypergraph** $\mathbf{G}^s = (\mathcal{G}, \mathcal{E})$ can be constructed

Each subregion $g \in \mathcal{G}$ is a node



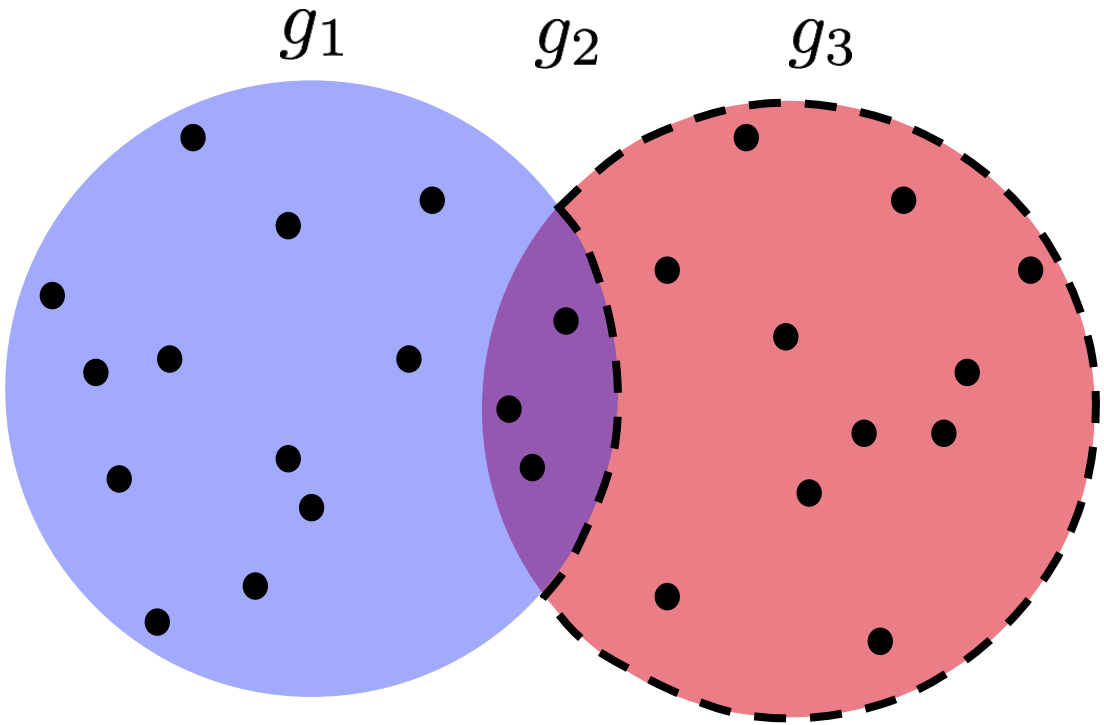
A **splitting hypergraph** $\mathbf{G}^s = (\mathcal{G}, \mathcal{E})$ can be constructed

Each subregion $g \in \mathcal{G}$ is a node



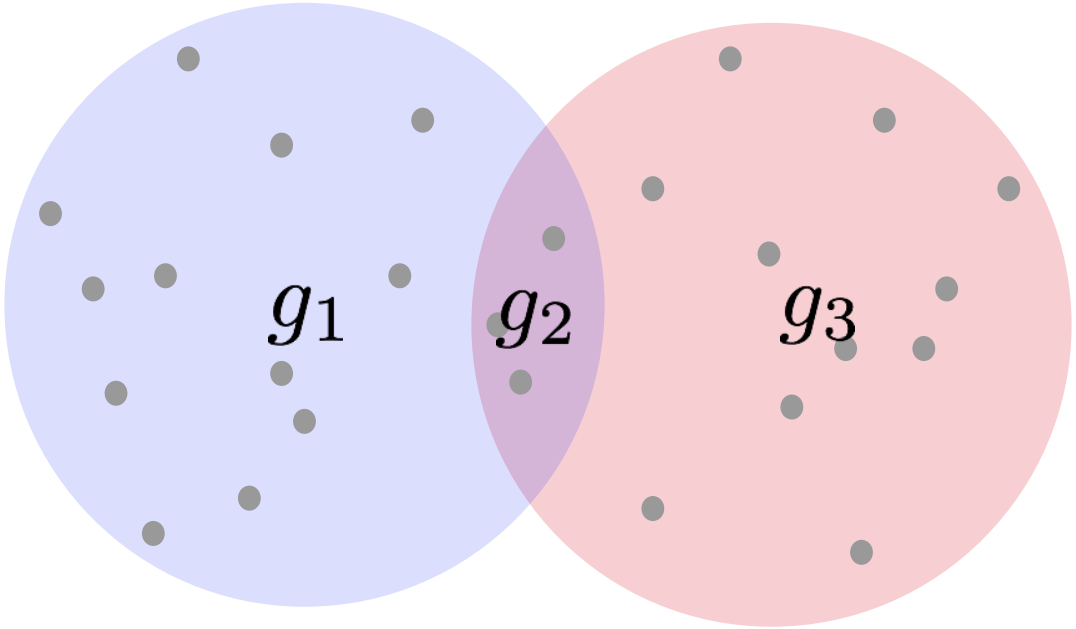
A **splitting hypergraph** $\mathbf{G}^s = (\mathcal{G}, \mathcal{E})$ can be constructed

Each subregion $g \in \mathcal{G}$ is a node

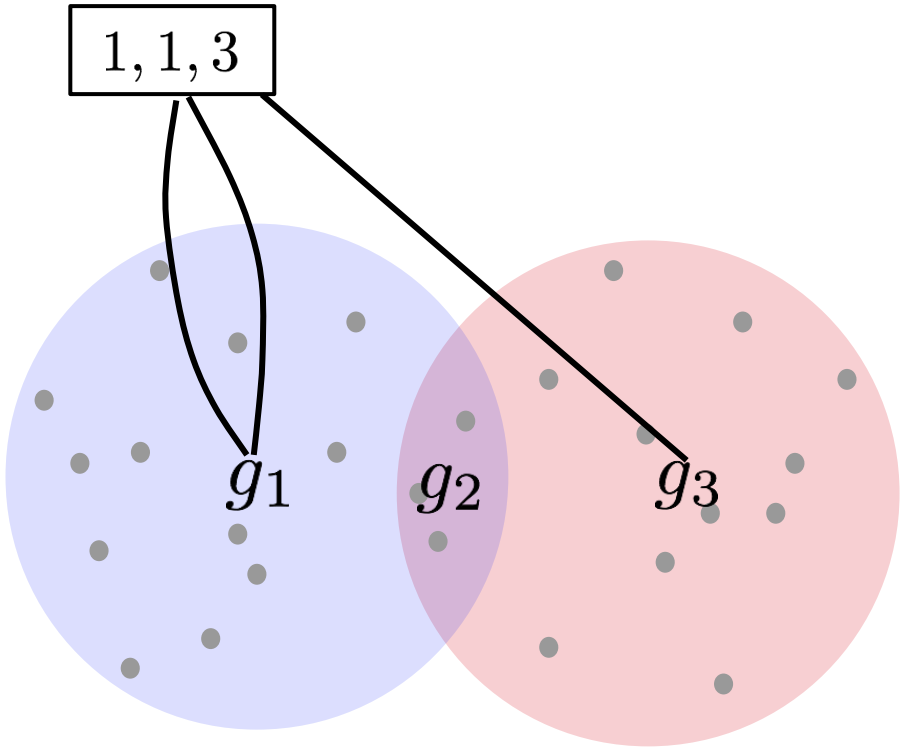


A **splitting hypergraph** $\mathbf{G}^s = (\mathcal{G}, \mathcal{E})$ can be constructed

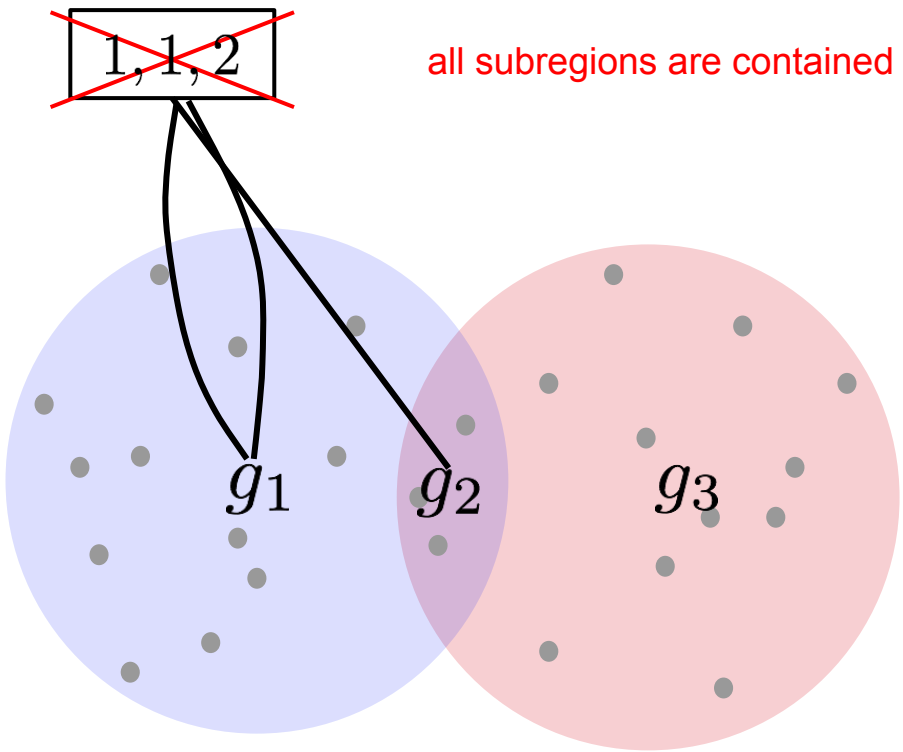
Each subregion $g \in \mathcal{G}$ is a node



Hyperedges $e \in \mathcal{E}$ are all multisets of k subregions such that one decision region does not contain them all

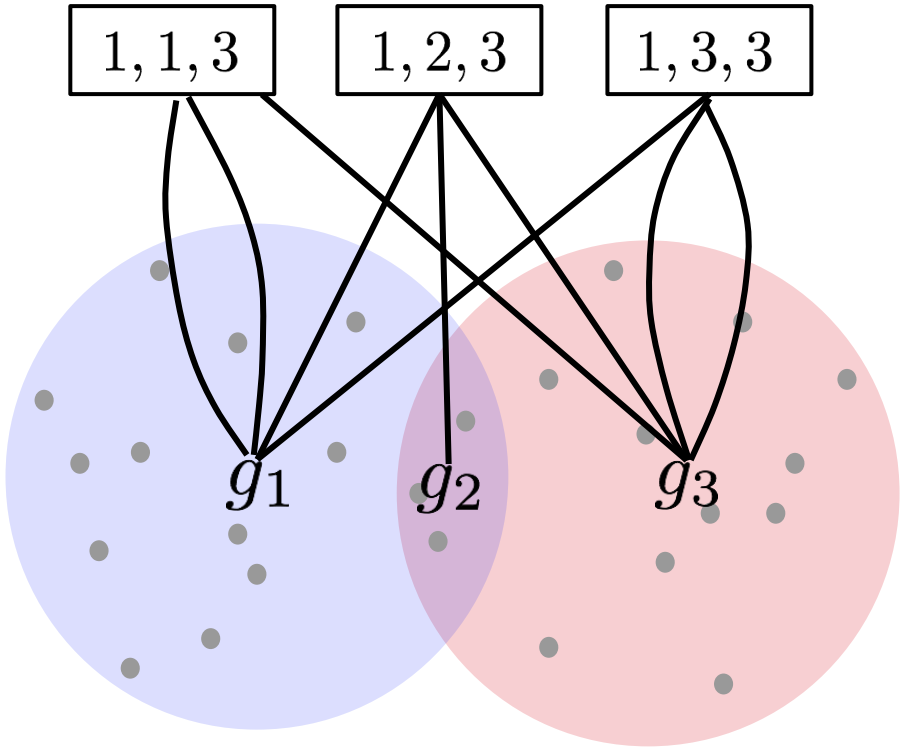


Hyperedges $e \in \mathcal{E}$ are all multisets of k subregions such that one decision region does not contain them all



all subregions are contained in the blue decision region

$$\mathcal{G} = \{g_1, g_2, g_3\}$$
$$\mathcal{E} = \{(1,1,3), (1,2,3), (1,3,3)\}$$

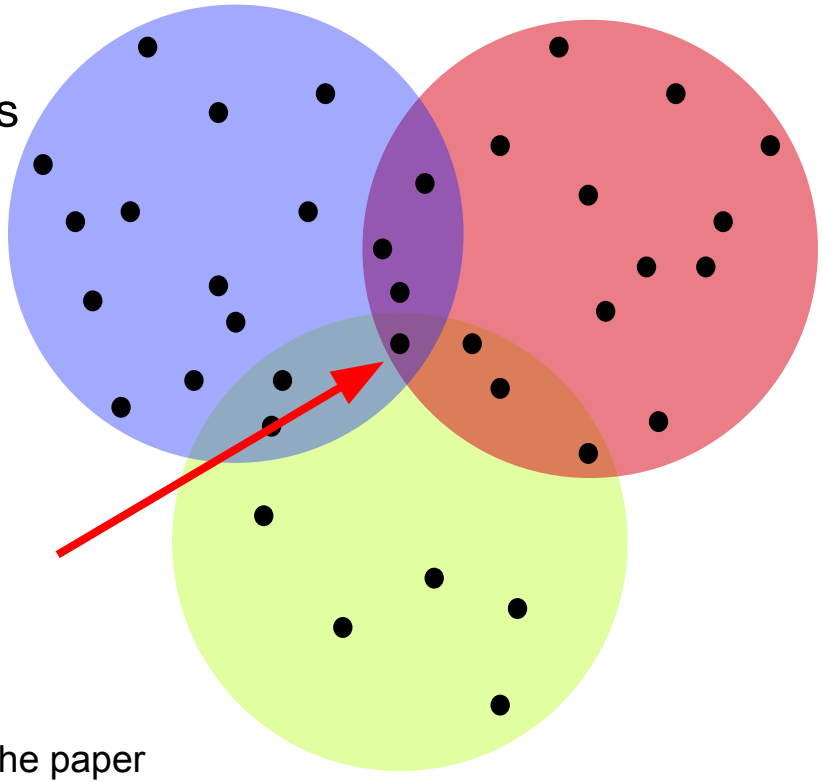


Cardinality

$$k = \min\left(\max_{h \in \mathcal{H}} |\{r : h \in r\}|, \max_{r \in \mathcal{R}} |\{g : g \in r\}|\right) + 1$$

maximum number of regions any hypothesis lies in

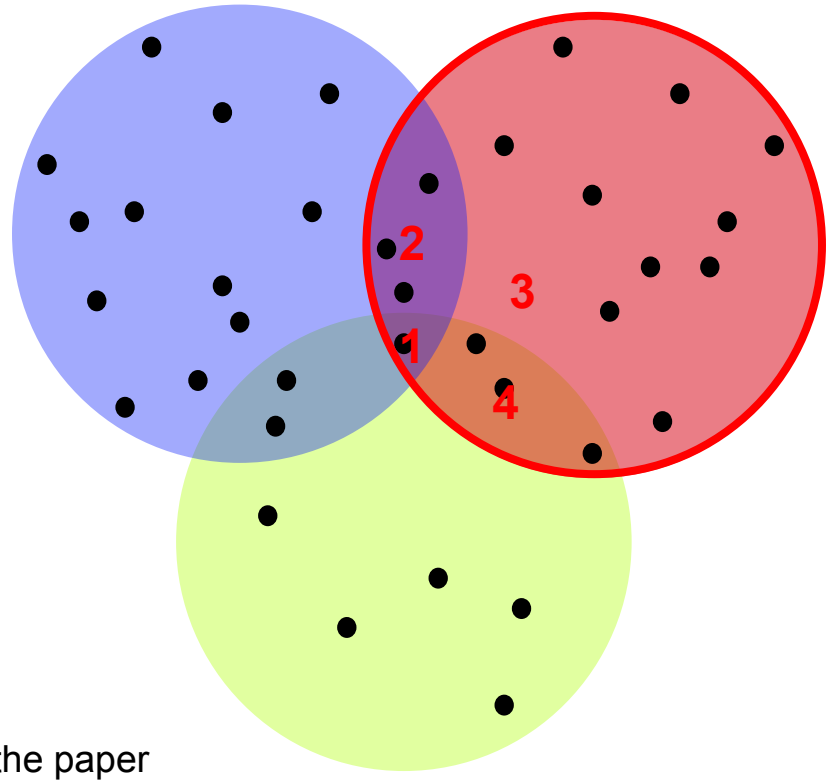
→ 3



For proof, see Section 7.1 from the paper

Cardinality

$$k = \min\left(\max_{h \in \mathcal{H}} |\{r : h \in r\}|, \max_{r \in \mathcal{R}} |\{g : g \in r\}|\right) + 1$$



maximum number of subregions in any region

→ 4

For proof, see Section 7.1 from the paper

The Hyperedge Cutting (HEC) Algorithm

set of consistent hyperedges, given evidence $\mathcal{S}$

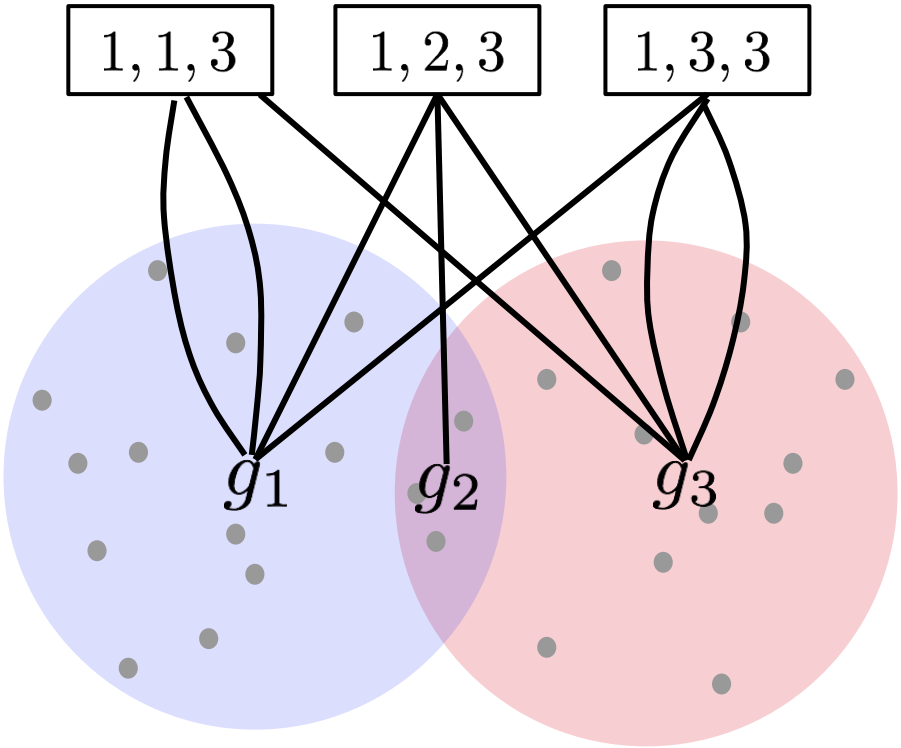
$$\mathcal{E}(\mathcal{S}) = \{e \in \mathcal{E} : \forall (t, o) \in \mathcal{S}, \forall h \in e, h(t) = o\}$$

Theorem 1

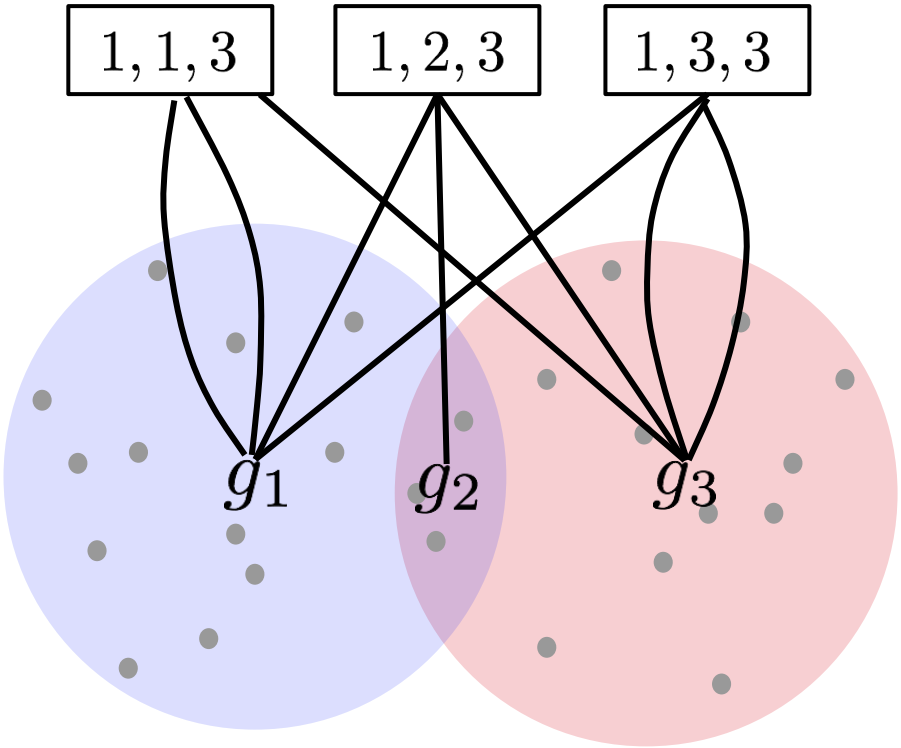
All consistent hypotheses lie in some decision region if and only if all hyperedges are cut, i.e.,

$$\boxed{\mathcal{E}(\mathcal{S}) = \emptyset \Leftrightarrow \exists r : \mathcal{V}(\mathcal{S}) \subseteq r}$$

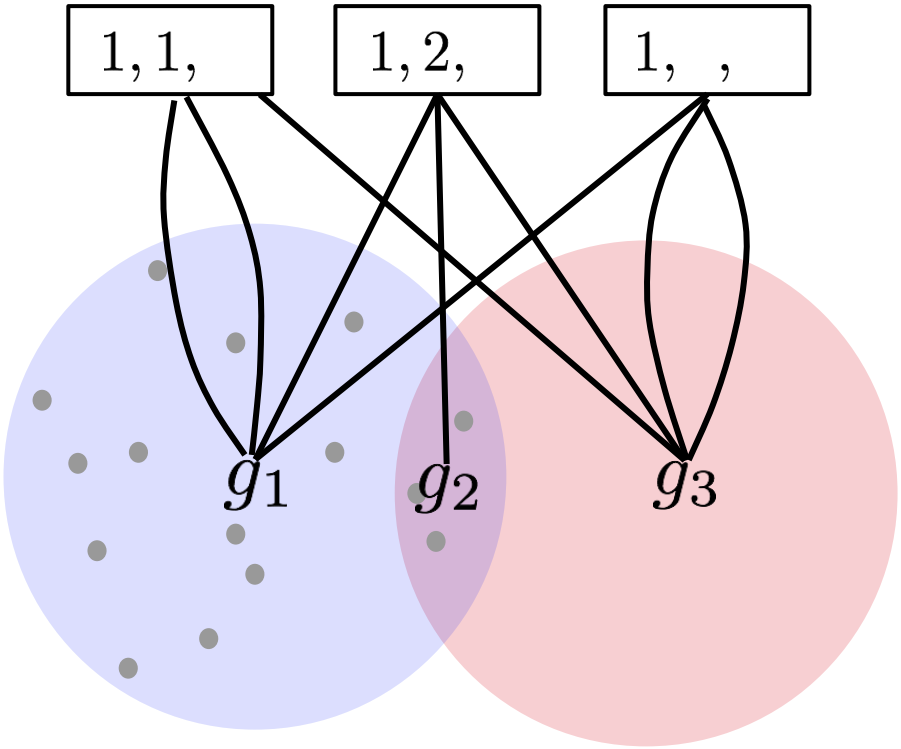
⟶ to make a decision, we need to cut all hyperedges in the splitting hypergraph


1, 1, 3
1, 2, 3
1, 3, 3
g_1
g_2
g_3

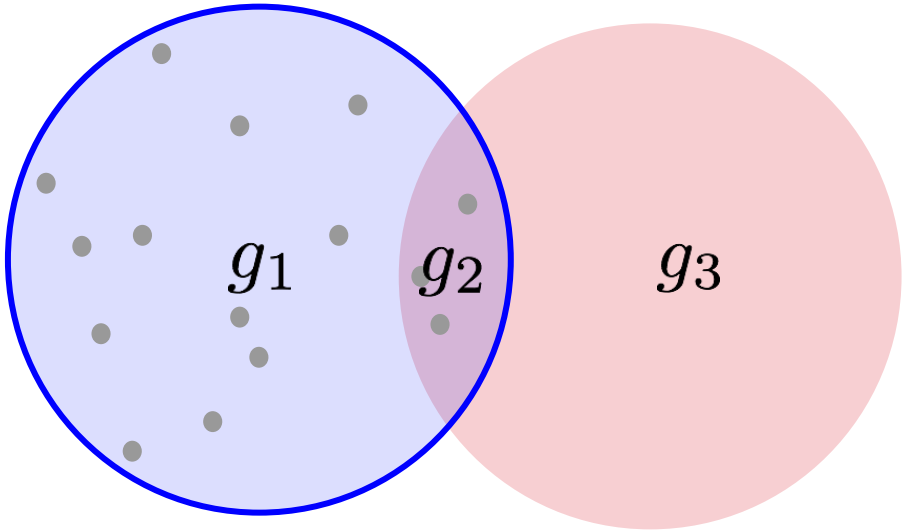
run tests → rule out all hypotheses in g_3



run tests → rule out all hypotheses in g_3



run tests ⟶ rule out all hypotheses in g_3



Goal:

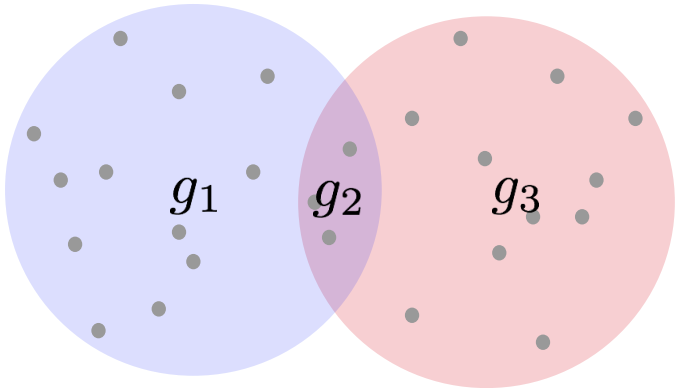
find the policy that cuts all hyperedges with minimum cost
(by running the minimum number of tests)

choose the test that cuts the most hyperedges in expectation

weight of a subregion

$$P(g) = \sum_{h \in g} P(h) \qquad \text{s.t. } \forall (t, o) \in \mathcal{S}, h(t) = o$$

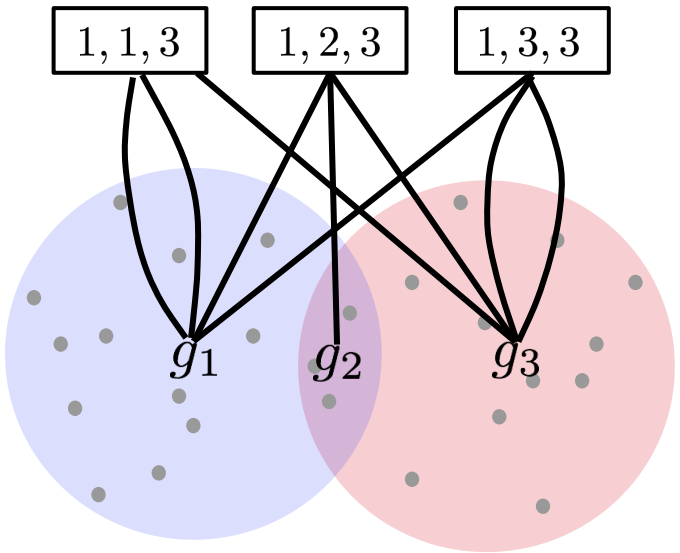
(consistent hypothesis)



for a subregion to be consistent, at least one of its hypotheses must be consistent

weight of a hyperedge $e = \{g_1, \ldots, g_k\}$

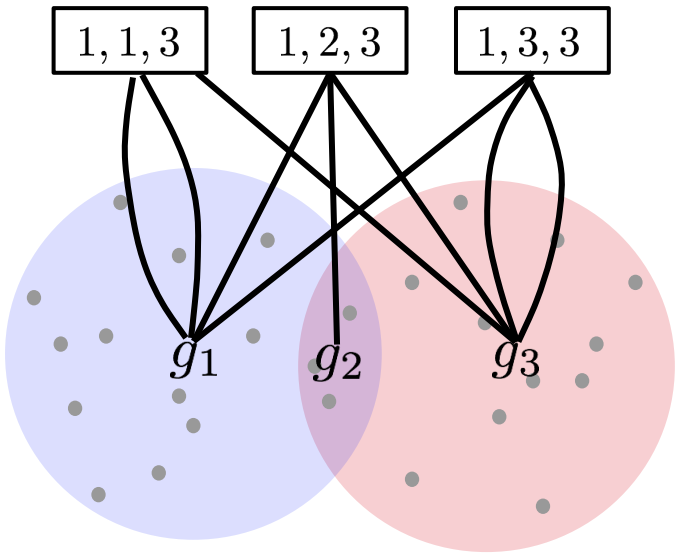
$$w(e) = \prod_{i=1}^{k} P(g_i)$$



for a hyperedge to be consistent, all of its subregions must be consistent

weight of a collection of hyperedges

$$w(\{e_1, \ldots, e_n\}) = \sum_{l=1}^{n} w(e_l)$$



total weight of a collection of hyperedges

weight of a subregion

$$P(g) = \sum_{h \in g} P(h) \qquad \text{s.t. } \forall (t, o) \in \mathcal{S}, h(t) = o$$

(consistent hypothesis)

weight of a hyperedge $e = \{g_1, \ldots, g_k\}$

$$w(e) = \prod_{i=1}^{k} P(g_i)$$

weight of a collection of hyperedges

$$w(\{e_1, \ldots, e_n\}) = \sum_{l=1}^{n} w(e_l)$$

total weight of all hyperedges $w(\mathcal{E})$

total weight of all consistent hyperedges $w(\mathcal{E}(\mathcal{S}))$

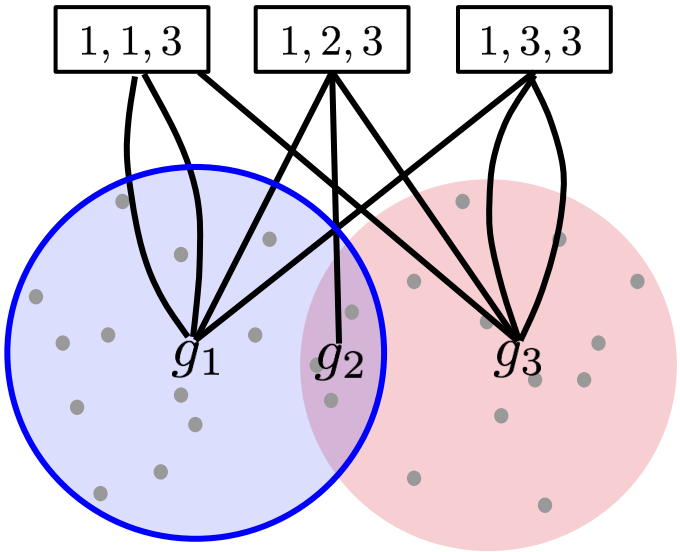
utility of evidence $\mathcal{S}$

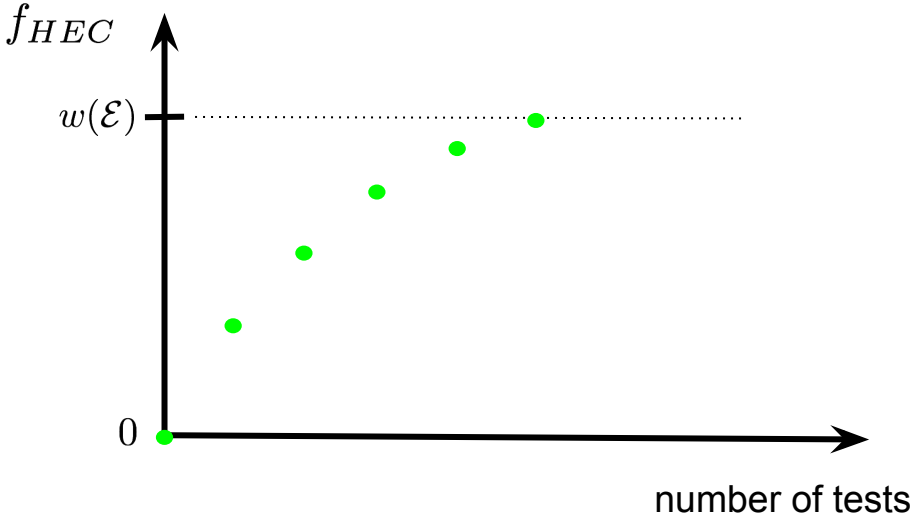
$$f_{HEC}(\mathcal{S}) = w(\mathcal{E}) - w(\mathcal{E}(\mathcal{S}))$$

total weight of all edges cut by observing evidence $\mathcal{S}$

utility of evidence $\mathcal{S}$

$$f_{HEC}(\mathcal{S}) = w(\mathcal{E}) - w(\mathcal{E}(\mathcal{S}))$$





greedy approach: iteratively choose the test t^* that maximally increases utility

$$\boxed{t^* = \text{argmax}_t \ \Delta(t|\mathcal{S})}$$

where $\Delta(t|\mathcal{S})$ is the **expected marginal utility gain** of test t

$$\Delta(t|\mathcal{S}) = \sum_h P(h|\mathcal{S}) \left(\underbrace{f_{HEC}\left(\mathcal{S} \cup \{(t, h(t))\}\right)}_{\text{utility with } t} - \underbrace{f_{HEC}(\mathcal{S})}_{\text{utility without } t} \right)$$

note that $\Delta(t|\mathcal{S}) = 0$ for all tests when all hyperedges have been cut

Hyperedge Cutting (HEC) Algorithm

initialize $\mathcal{S} = \emptyset$

while $\mathcal{E}(\mathcal{S}) \neq \emptyset$

run test $t^* = \text{argmax}_t \ \Delta(t|\mathcal{S})$ **greedy**

observe outcome $h(t^*)$

add $(t^*, h(t^*))$ to $\mathcal{S}$

return $r^* = r : \mathcal{V}(\mathcal{S}) \subseteq r$

How does the greedy policy compare with the optimal policy?

$\Delta(t|\mathcal{S})$ satisfies **adaptive monotonicity** and **adaptive submodularity**

$\Delta(t|\mathcal{S})$ satisfies **adaptive monotonicity** and **adaptive submodularity**

adaptive monotonicity

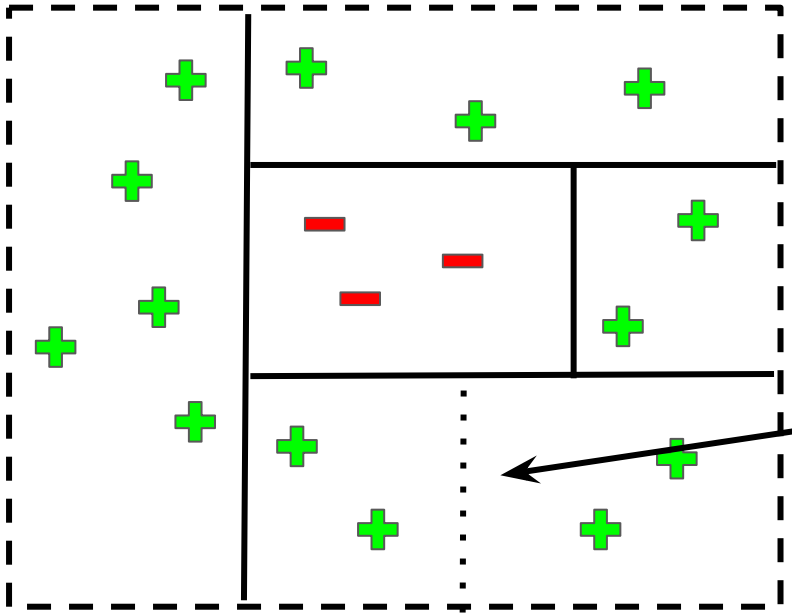
$$\Delta(t|\mathcal{S}) \geq 0 \qquad \forall t \in \mathcal{T}, \mathcal{S} \subseteq \mathcal{T} \times \mathcal{O}$$

running a test can never introduce hyperedges, only remove them

For proof, see Lemma 2 in Section 7.3 of the paper.

$\Delta(t|\mathcal{S})$ satisfies **adaptive monotonicity** and **adaptive submodularity**

adaptive monotonicity (in binary decision tree)



asking a question (i.e. going deeper in the tree) cannot hurt you

worst case: gain no new information

can always recover previous classification, no matter what boundary we draw

$\Delta(t|\mathcal{S})$ satisfies **adaptive monotonicity** and **adaptive submodularity**

adaptive submodularity

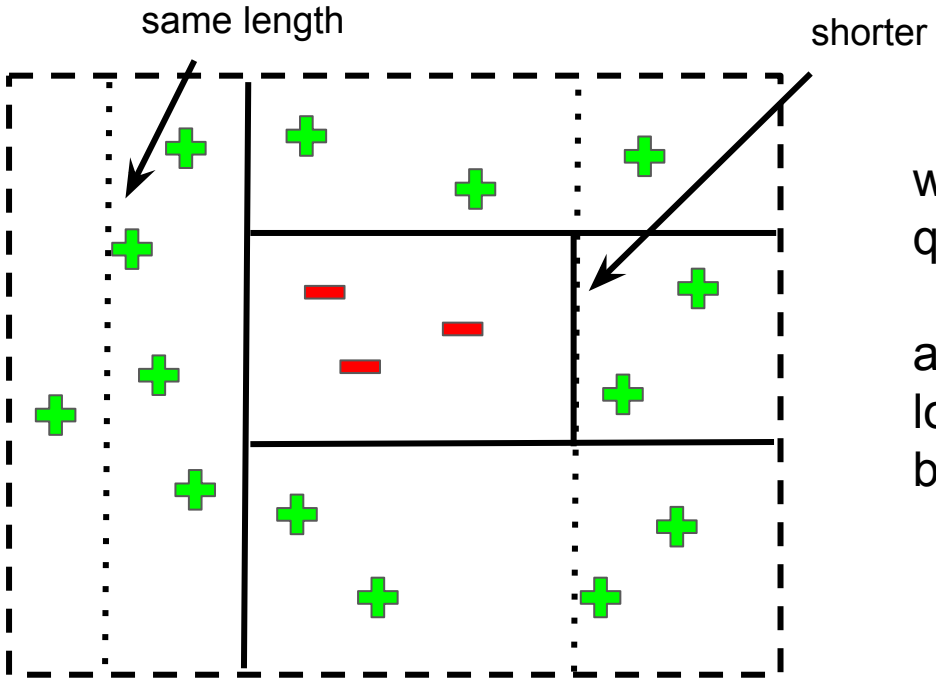
$$\Delta(t|\mathcal{S}) \geq \Delta(t|\mathcal{S}') \qquad \forall t \in \mathcal{T}, \mathcal{S} \subseteq \mathcal{S}' \subseteq \mathcal{T} \times \mathcal{O}$$

for a fixed test t , the marginal utility gain cannot increase as we gain additional evidence

For proof, see Lemma 3 in Section 7.3 of the paper.

$\Delta(t|\mathcal{S})$ satisfies **adaptive monotonicity** and **adaptive submodularity**

adaptive submodularity (in binary decision tree)



we cannot gain more information from a question by moving it deeper in the tree

a classification boundary cannot get longer by adding other classification boundaries first

$\Delta(t|\mathcal{S})$ satisfies **adaptive monotonicity** and **adaptive submodularity**

Golovin & Krause (2011) showed that for an objective satisfying both of these conditions, **a greedy policy π_{HEC} will be competitive with the optimal policy π^***, with cost bounded by

$$\mathcal{C}(\pi_{HEC}) \leq (k \ln(1/p_{\min}) + 1)\,\mathcal{C}(\pi^*)$$

where $$p_{\min} = \min_{h \in \mathcal{H}} P(h)$$

Golovin, Daniel, and Andreas Krause. "Adaptive submodularity: Theory and applications in active learning and stochastic optimization." *Journal of Artificial Intelligence Research* (2011): 427-486.

Demo

Requirements

To run HEC you will need to define:

- A discrete set of hypotheses.
- Prior probabilities for the hypotheses.
- A set of decision regions.
- A discrete set of deterministic tests.

“Naive” Implementation

$$\Delta(t|\mathcal{S}) = \sum_{h} P(h|\mathcal{S}) \left(f_{HEC}\left(\mathcal{S} \cup \{(t, h(t))\}\right) - f_{HEC}(\mathcal{S}) \right)$$

“Naive” Implementation

This can be computed once per iteration.

$$\Delta(t|\mathcal{S}) = \sum_h P(h|\mathcal{S}) \left(f_{HEC}\left(\mathcal{S} \cup \{(t, h(t))\}\right) - f_{HEC}(\mathcal{S}) \right)$$

“Naive” Implementation

This can be computed once per iteration.

$$\Delta(t|\mathcal{S}) = \sum_h P(h|\mathcal{S}) \left(f_{HEC}\left(\mathcal{S} \cup \{(t, h(t))\}\right) - f_{HEC}(\mathcal{S})\right)$$

To compute this, you have to construct the splitting hypergraph for each test, hypothesis pair

“Naive” Implementation

$$\Delta(t|\mathcal{S}) = \sum_h P(h|\mathcal{S}) \left(f_{HEC}\left(\mathcal{S} \cup \{(t, h(t))\}\right) - f_{HEC}(\mathcal{S})\right)$$

To compute this, you have to construct the splitting hypergraph for each test, hypothesis pair

```
for t in tests:
  for h in hypotheses:
    nodes, hyperedges = build_splitting_hypergraph(...)
    weight = compute_weight_of_hyperedges(...)
```

“Naive” Implementation

This requires enumerating every multiset of order k and checking if any decision region contains all the subregions

$$O(|\mathcal{G}|^k)$$

```
for t in tests:
  for h in hypotheses:
    nodes, hyperedges = build_splitting_hypergraph(...)
    weight = compute_weight_of_hyperedges(...)
```

“Efficient” implementation

Intuition:

Compute the weight of ***all*** multisets (i.e. hyperedges) and iteratively subtract off the weight of invalid multisets (i.e. hyperedges whose subregions are contained within one decision region).

“Efficient” implementation

There is an algebraic structure that we can take advantage of when computing the weight of a collection of hyperedges.

Computing a sum of multisets, where a multiset corresponds to a product, is equivalent to computing a complete homogeneous symmetric polynomial (CHP).

“Efficient” implementation

$$G \subseteq \mathcal{G}$$ ← Subregions (nodes of our splitting hypergraph)

$$\mathcal{G}_{\hat{k}}(G) = \{\{g_1, \ldots, g_{\hat{k}}\} \subseteq \mathcal{G}\}$$

All multisets over group G of cardinality $\hat{k}$

“Efficient” implementation

$$\mathcal{G}_{\hat{k}}(G) = \{\{g_1, \ldots, g_{\hat{k}}\} \subseteq \mathcal{G}\}$$

$$w(\mathcal{G}_{\hat{k}}(G)) = \sum_{\mathcal{G}_{\hat{k}}(G)} \prod_{g} P(g)$$

“Efficient” implementation

$$\mathcal{G}_{\hat{k}}(G) = \{\{g_1, \ldots, g_{\hat{k}}\} \subseteq \mathcal{G}\}$$

$$w(\mathcal{G}_{\hat{k}}(G)) = \sum_{\mathcal{G}_{\hat{k}}(G)} \prod_{g} P(g)$$

Computing a sum of multisets, where a multiset corresponds to a product, is equivalent to computing a complete homogeneous symmetric polynomial (CHP).

“Efficient” implementation

$$\mathcal{G}_{\hat{k}}(G) = \{\{g_1, \ldots, g_{\hat{k}}\} \subseteq \mathcal{G}\}$$

$$w(\mathcal{G}_{\hat{k}}(G)) = \sum_{\mathcal{G}_{\hat{k}}(G)} \prod_{g} P(g)$$

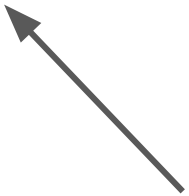
$$w(\mathcal{G}_{\hat{k}}(G)) = CHP_{\hat{k}}(G)$$

Efficient Implementation

$$CHP_{\hat{k}}(G) = CHP_{\hat{k}}(\{g_1, \ldots, g_n\})$$

$$= \frac{1}{\hat{k}} \sum_{j=1}^{\hat{k}} CHP_{\hat{k}-j}(\{g_1, \ldots, g_n\}) PS_j(\{g_1, \ldots, g_n\})$$

$$PS_j(\{g_1, \ldots, g_n\}) = \sum_{i=1}^{n} P(g_i)^i$$



$O(\hat{k}|G|)$ □□

“Efficient” implementation

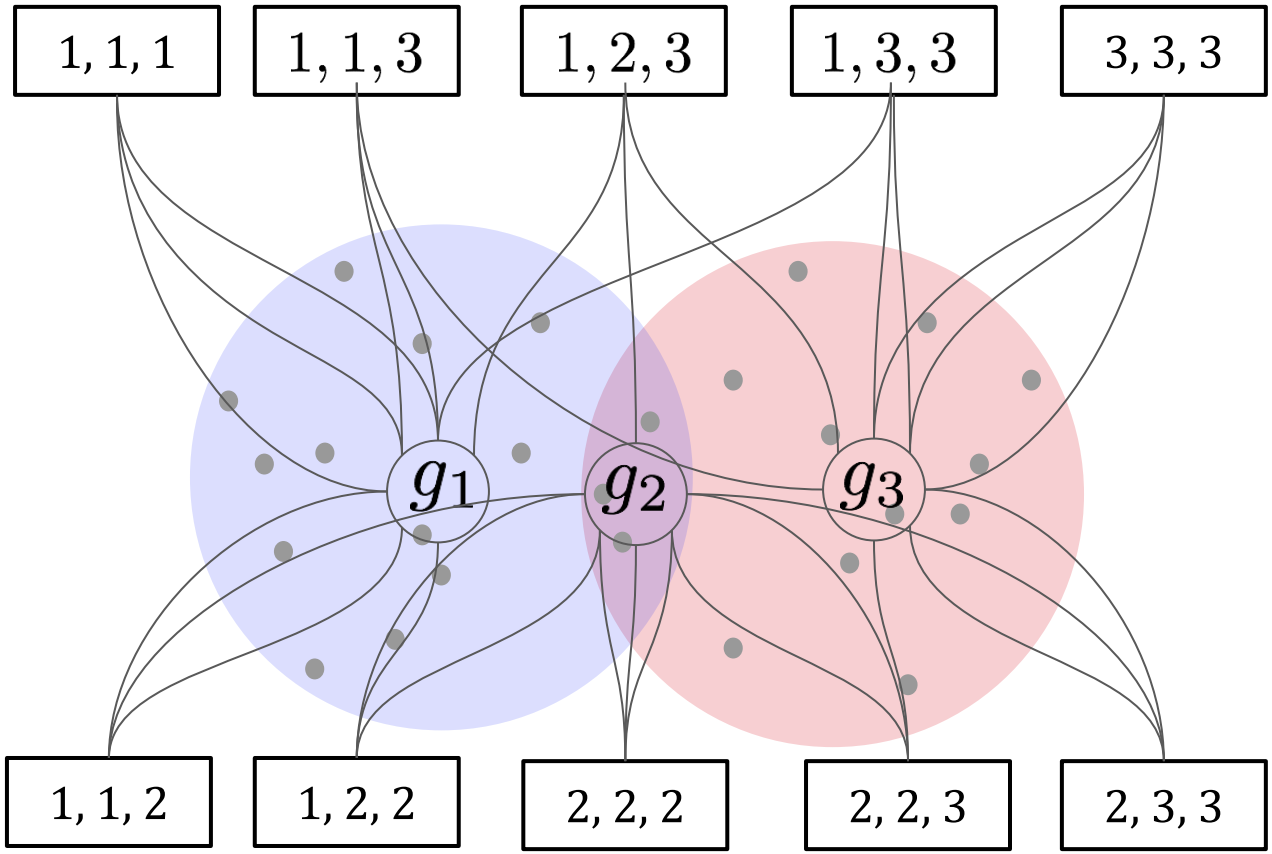
But we are not done yet. We just calculated the collective weight of *all* hyperedges. We need to remove the weight due to hyperedges whose subregions are all contained in one decision region.

“Efficient” implementation

But we are not done yet. We just calculated the collective weight of *all* hyperedges. We need to remove the weight due to hyperedges whose subregions are all contained in one decision region.

Intuition:
Compute the weight of all multisets. Iteratively subtract the weight of increasing multiset cardinality while pruning away multisets that whose subsets contain valid hyperedges.

Requirement:
Use hash tables to keep things speedy! We can precompute which multisets are invalid.



W =

1, 1, 1

✚ 1, 1, 2

✚ 1, 1, 3

✚ 1, 2, 2

✚ 1, 2, 3

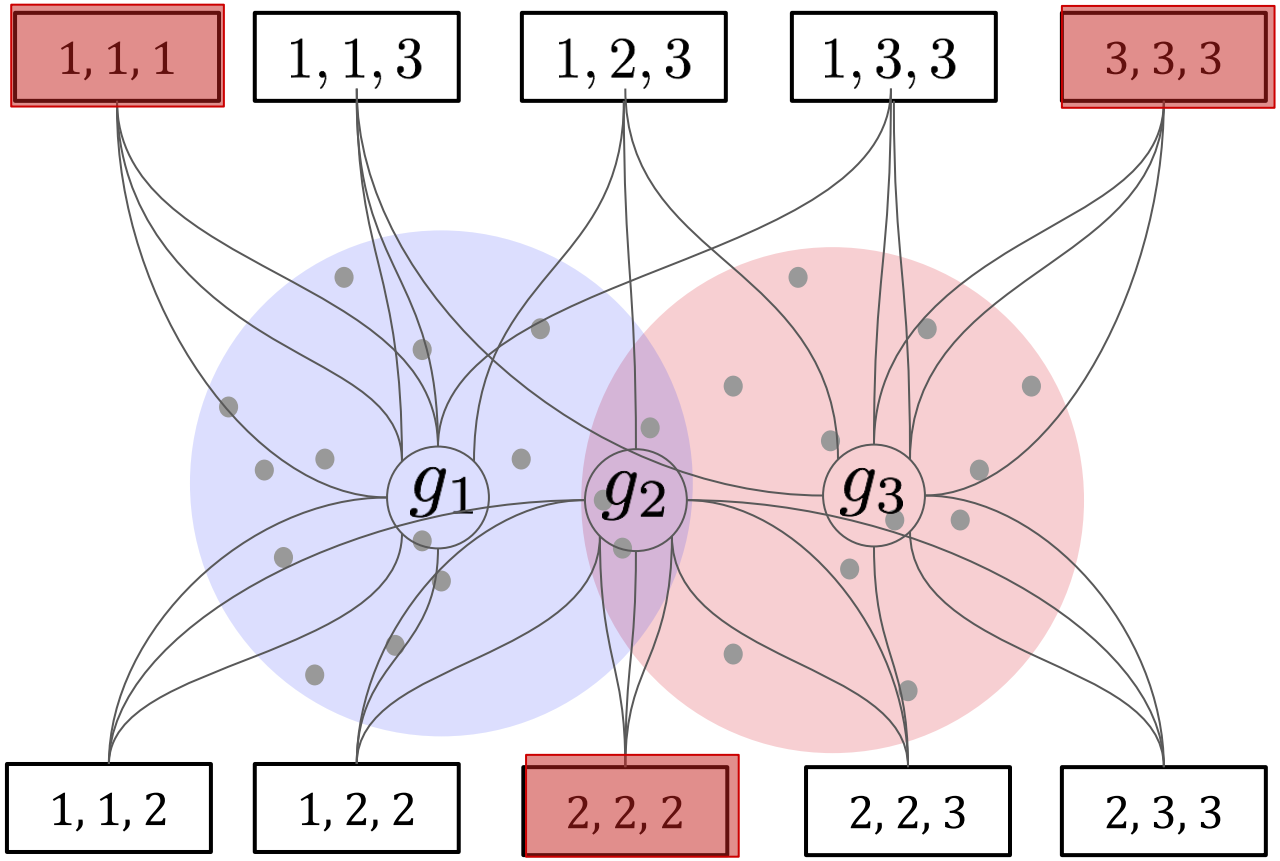
✚ 1, 3, 3

✚ 2, 2, 2

✚ 2, 2, 3

✚ 2, 3, 3

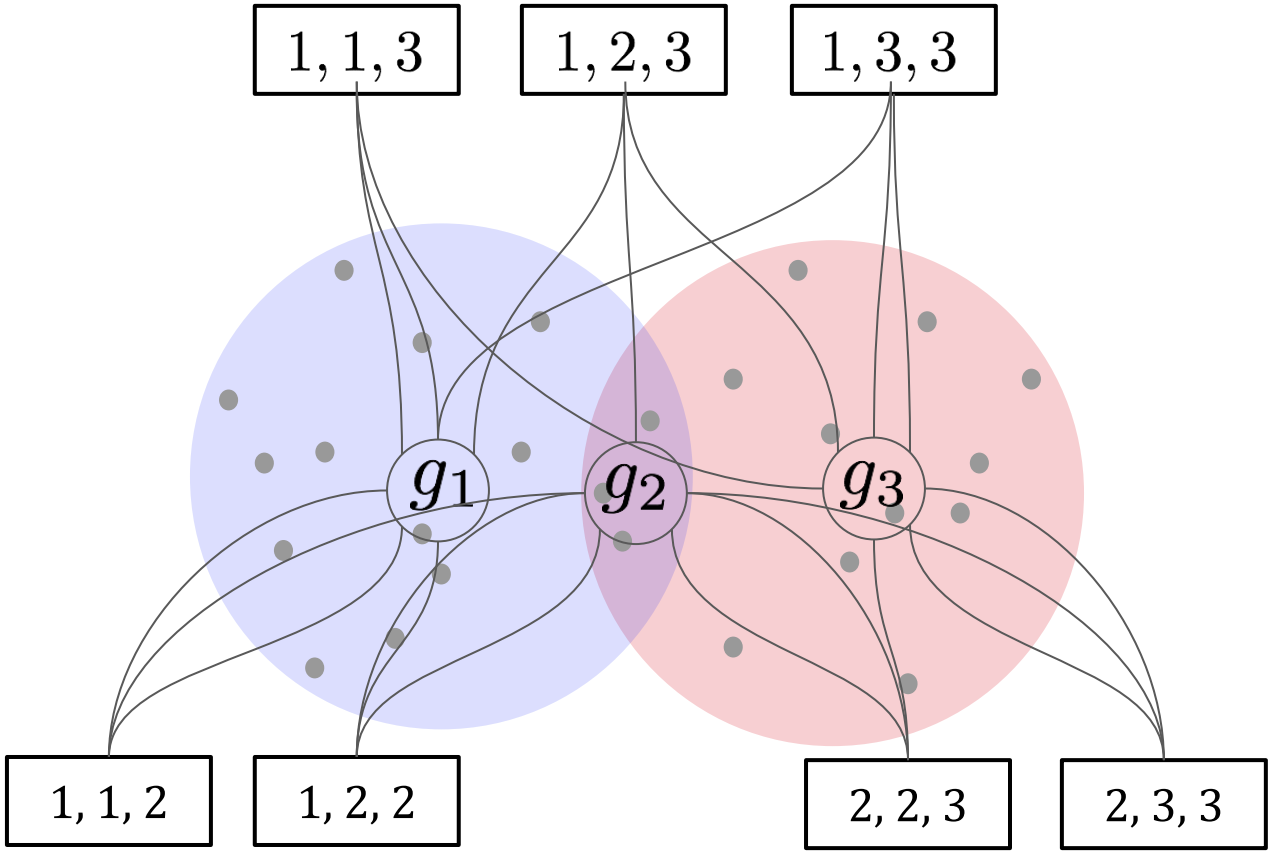
✚ 3, 3, 3



W -= 1, 1, 1

W -= 2, 2, 2

W -= 3, 3, 3



W = 1, 1, 2

✚ 1, 1, 3

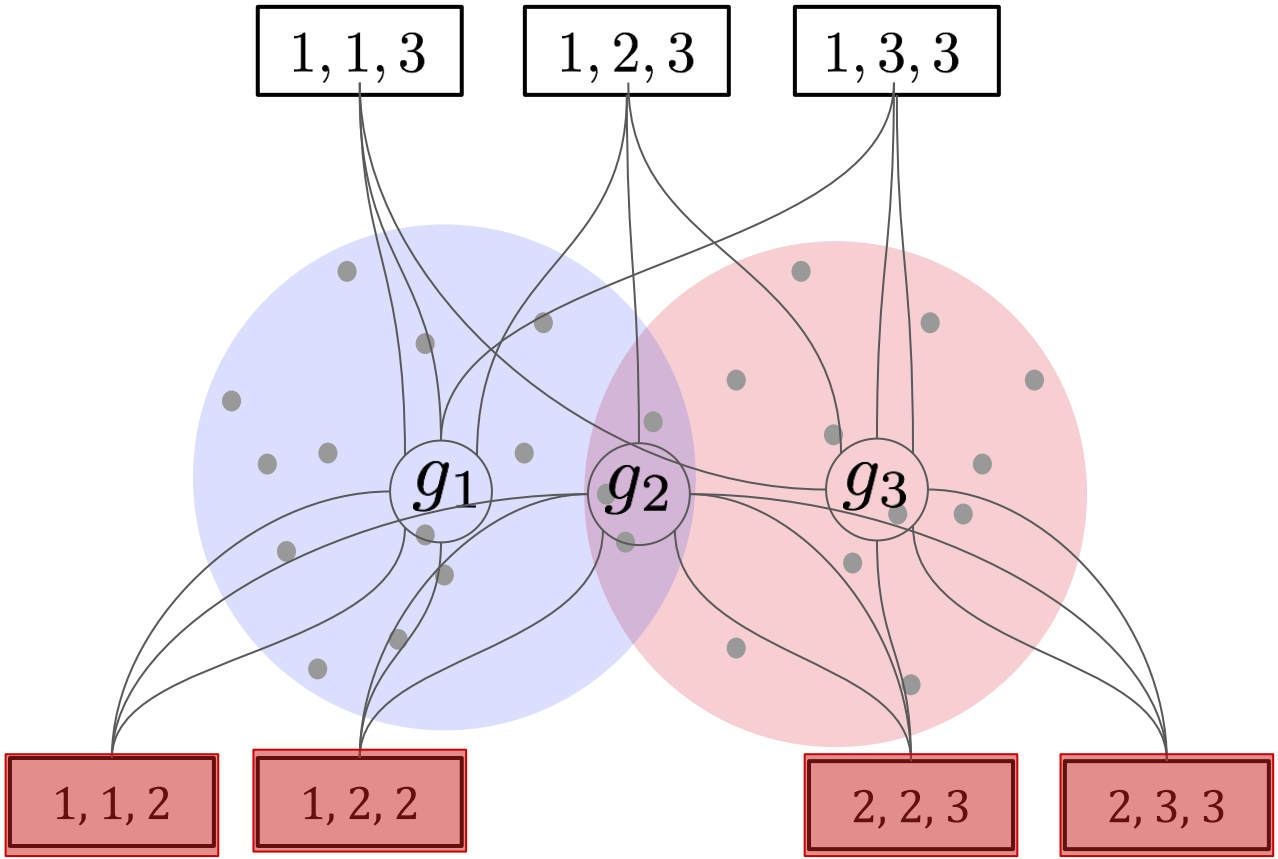
✚ 1, 2, 2

✚ 1, 2, 3

✚ 1, 3, 3

✚ 2, 2, 3

✚ 2, 3, 3

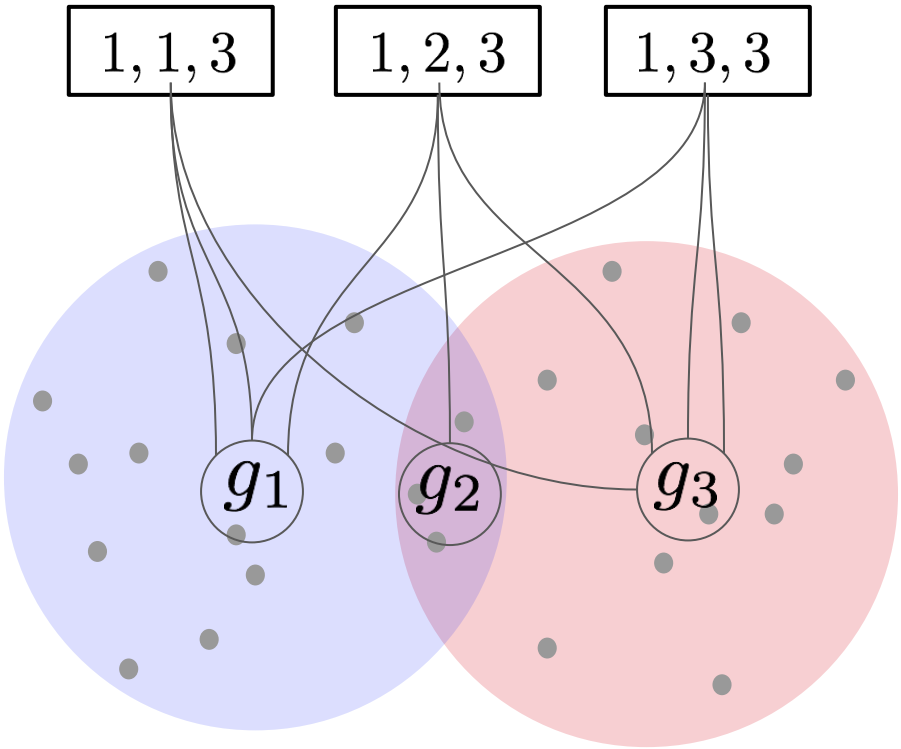


W -= 1, 1, 2

W -= 1, 2, 2

W -= 2, 2, 3

W -= 2, 3, 3



W =

1, 1, 3

✚ 1, 2, 3

✚ 1, 3, 3

“Efficient” implementation

In the worst case, this algorithm is still $O(|\mathcal{G}|^k)$ □

To see the benefit of this algorithm, we will need the number of regions to be much larger than the cardinality of the multisets, k.

Computational Complexity

This algorithm is good at minimizing the number of tests, but at a high computational cost. The computational bottleneck for HEC lies in the construction of the hypergraph:

> The computation is **exponential** in hyperedge cardinality k.

If your tests are expensive, then perhaps you are fine with waiting.

If your tests are cheap, then using a simpler algorithm that conducts more tests could run much faster than HEC.

Computational Complexity

There is a follow up paper[1] where the authors attempt to mitigate this computational cost:

“... our algorithm is exponentially faster than HEC in theory, significantly faster (often by orders of magnitude) in practice, while offering similar empirical performance.”

[1]Chen, Yuxin, et al. "Submodular Surrogates for Value of Information." *AAAI*. 2015.

Robotic Decision Making Application

Decision Making Task

Javdani, Shervin. "Gathering Information For Decision Making In Touch Based Localization". Presentation.

Decision Making Setting

Hypotheses

Prior probability distribution over the set of hypotheses

Tests

Decision regions

Decision Making Setting

Hypotheses: object locations

Prior probability distribution over the set of hypotheses

Tests

Decision regions

Decision Making Setting

Hypotheses: object locations

- Prior probability distribution over the set of hypotheses

Tests: guarded moves

- Test outcomes: distance until contact

Decisions regions

Decision Making Setting

Hypotheses: object locations

- Prior probability distribution over the set of hypotheses

Tests: guarded moves

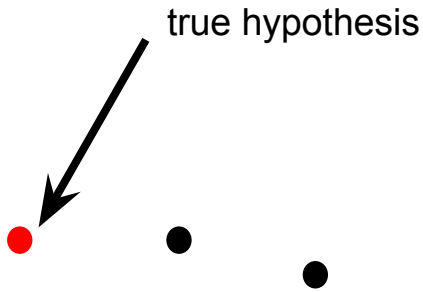
- Test outcomes: distance until contact

Decisions: button-push moves

- **Decision regions**: sets of hypotheses on which decisions will succeed

Given: Hypotheses

Given: Hypotheses



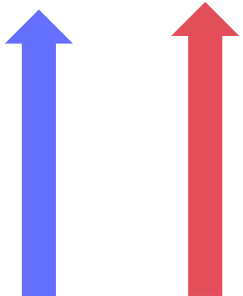
Given: Prior Probabilities

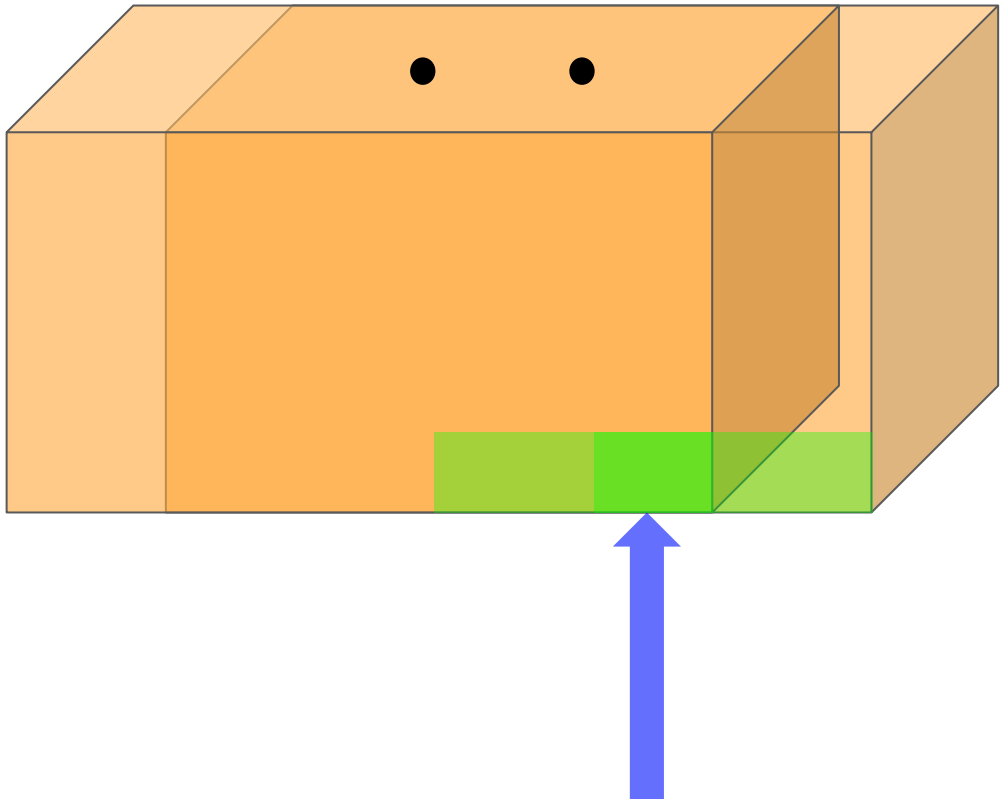
$$p(h_1) = \frac{1}{3}$$

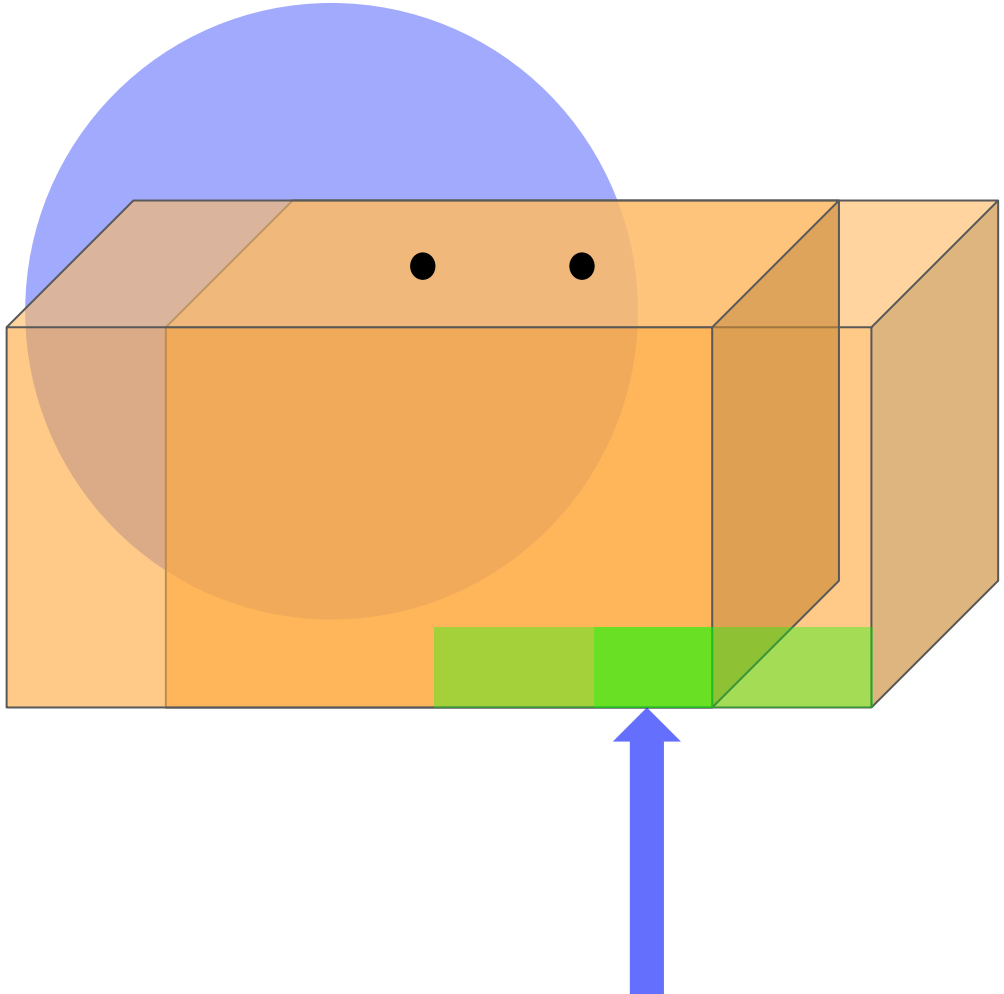
$$p(h_2) = \frac{1}{3}$$

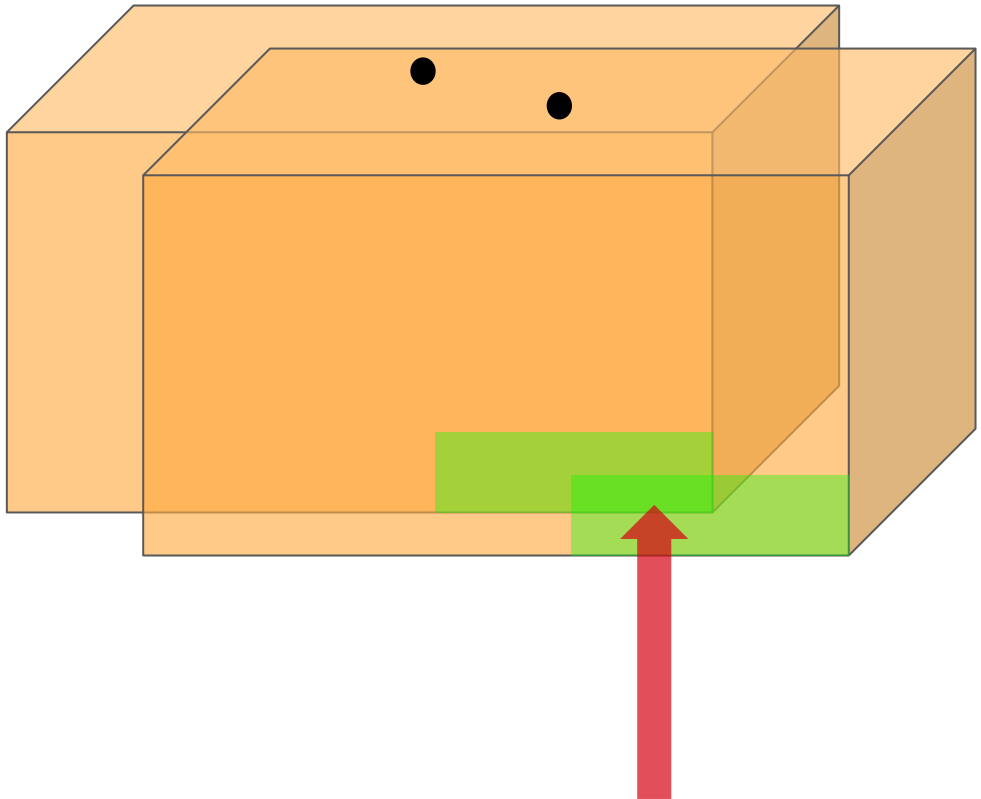
$$p(h_3) = \frac{1}{3}$$

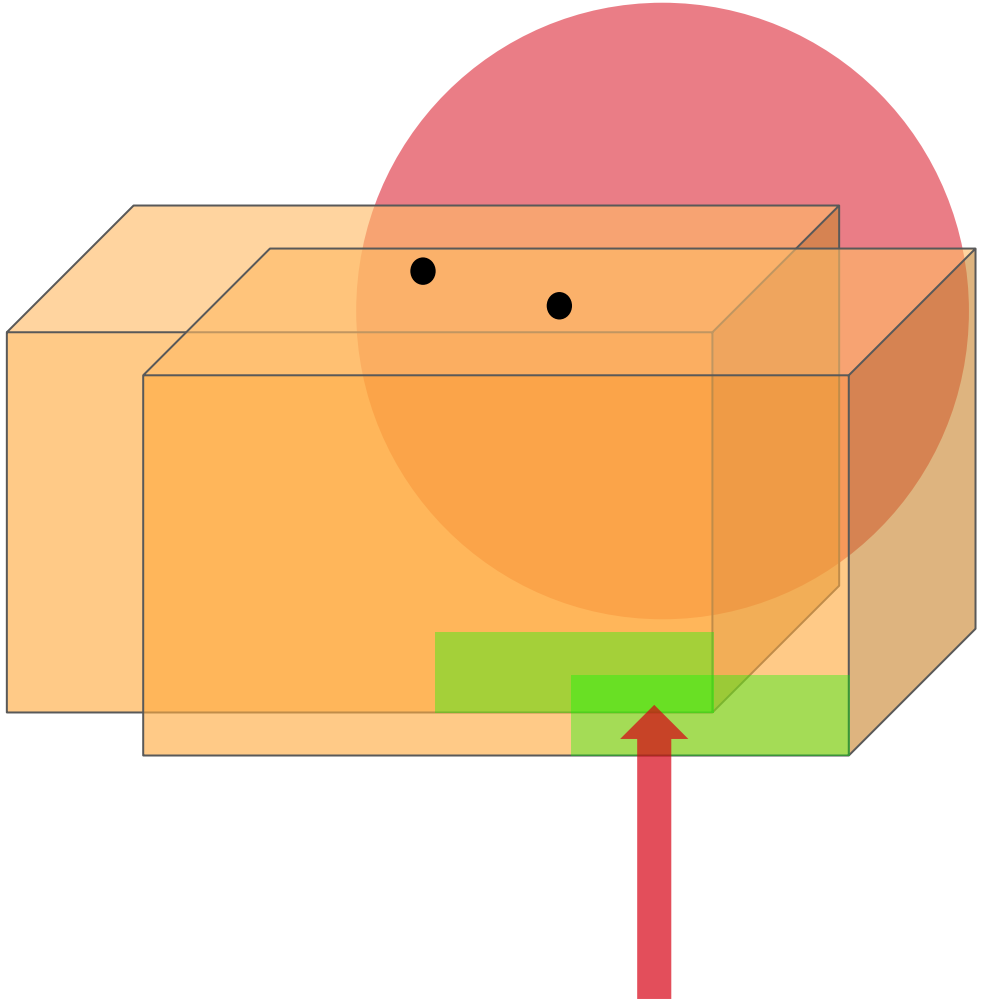
Given: Decisions

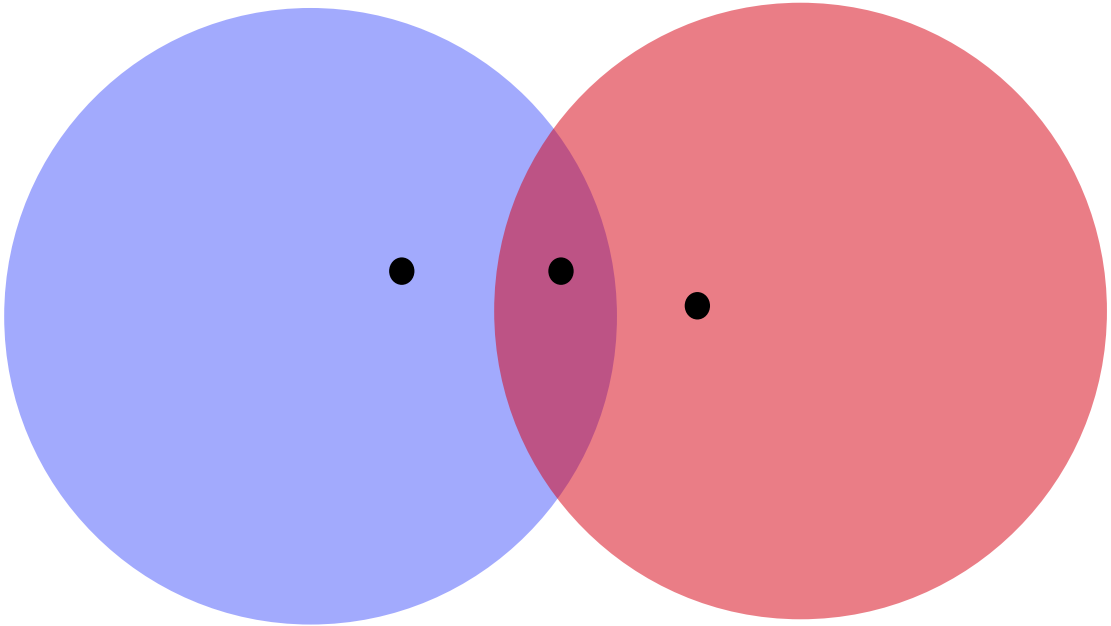




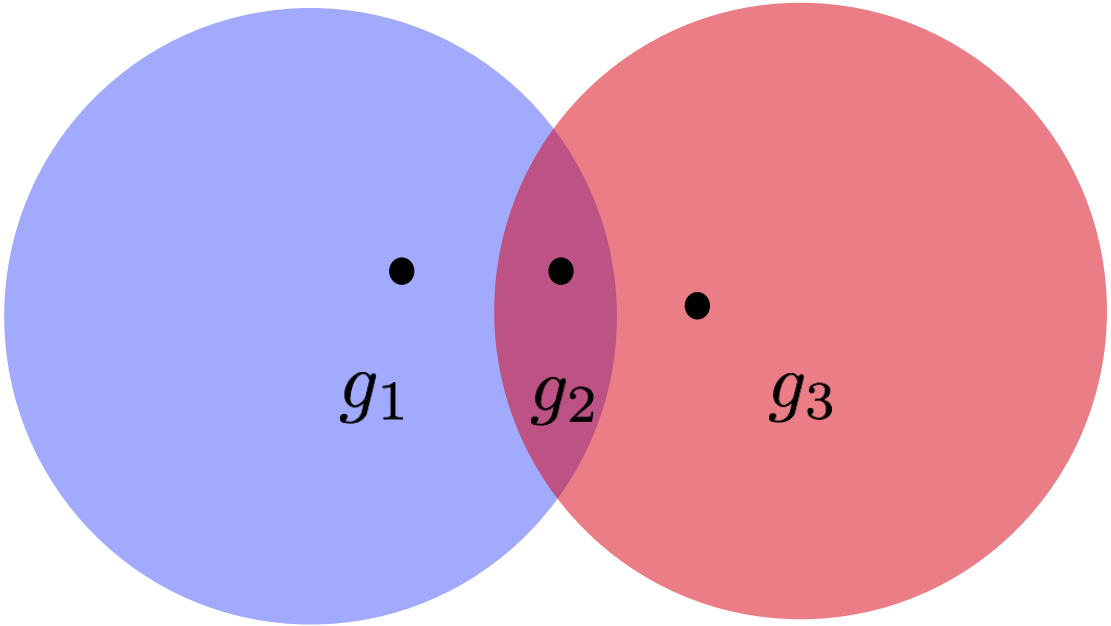




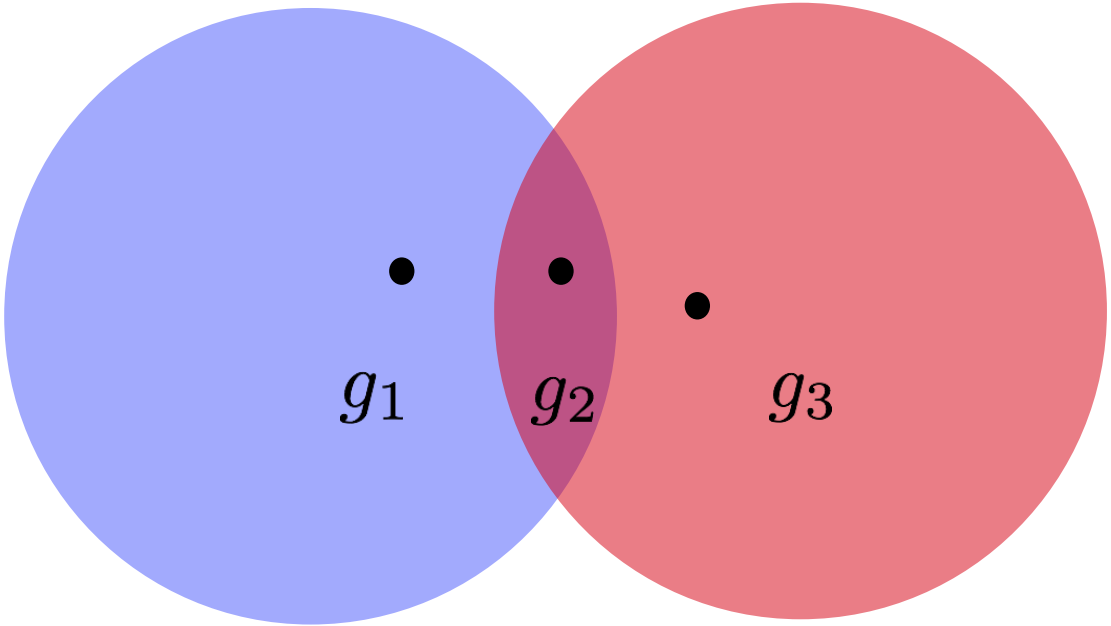




Given: Decision Regions

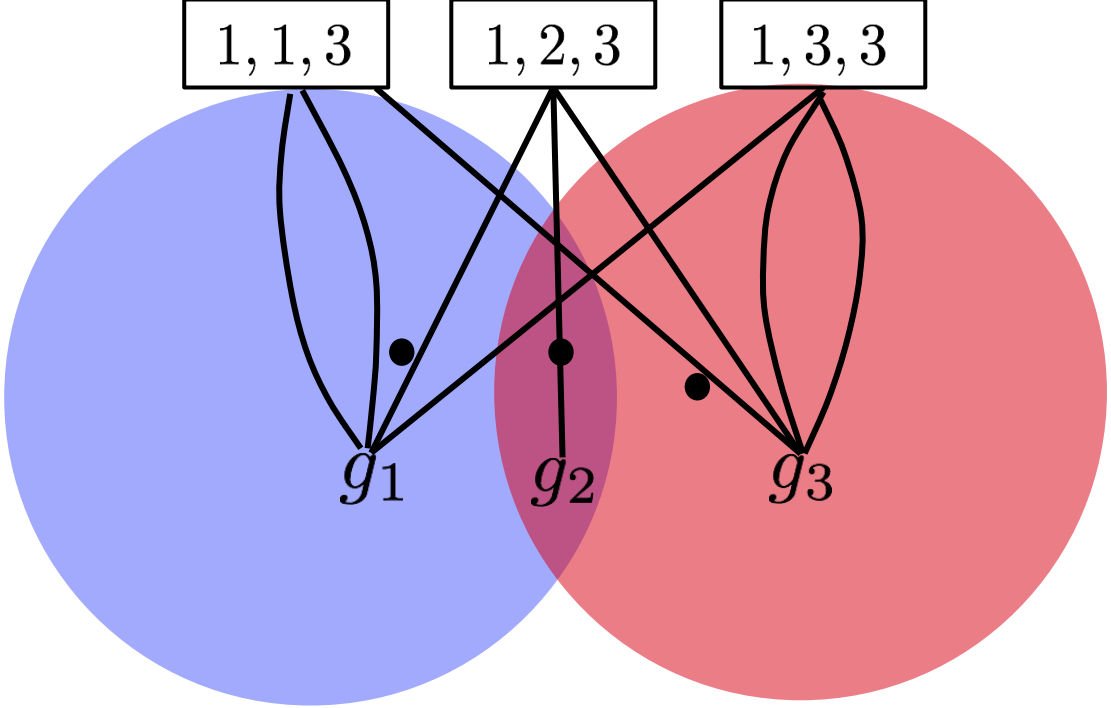


Hypergraph: Nodes

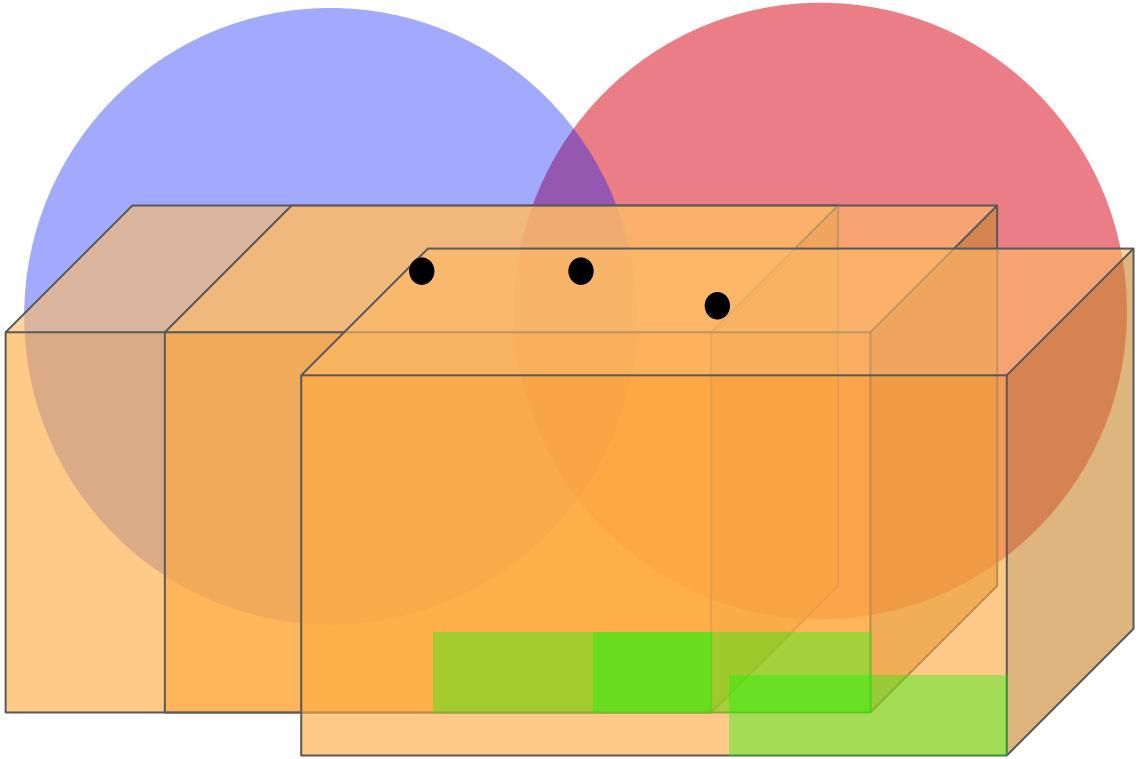


$$k = \min\left(\max_{h \in \mathcal{H}} |\{r : h \in r\}|, \max_{r \in \mathcal{R}} |\{g : g \in r\}|\right) + 1 \quad = 3$$

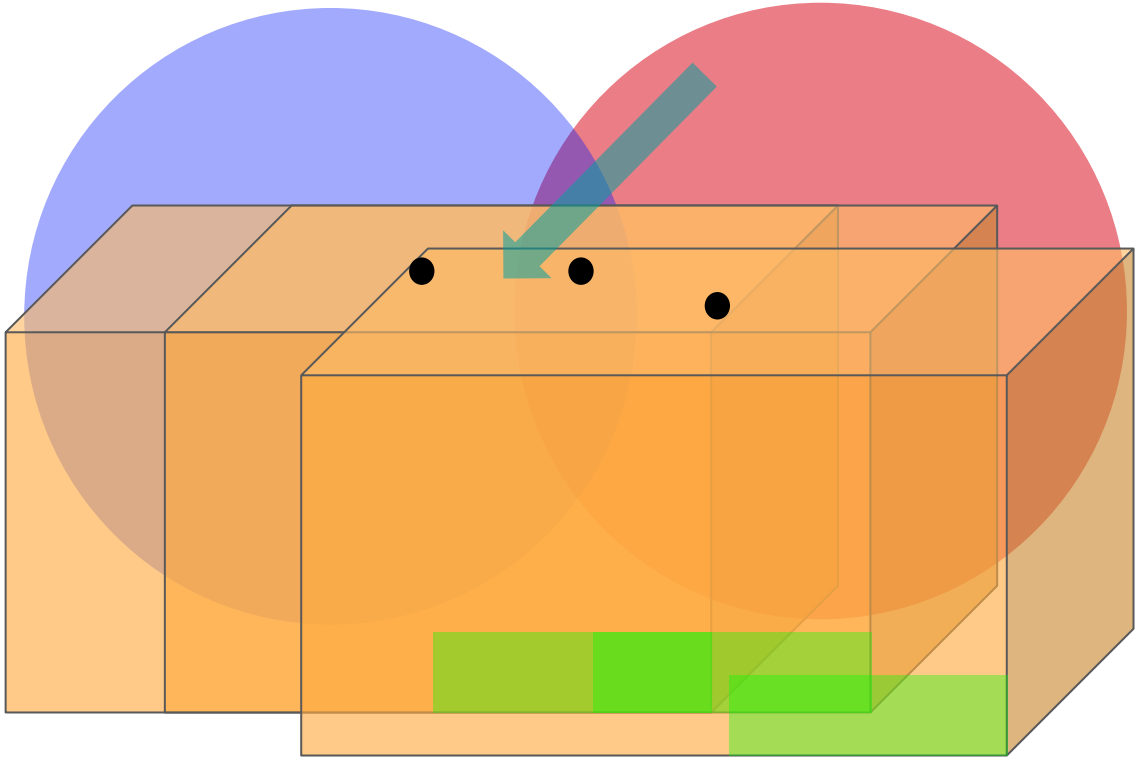
Hypergraph: Cardinality

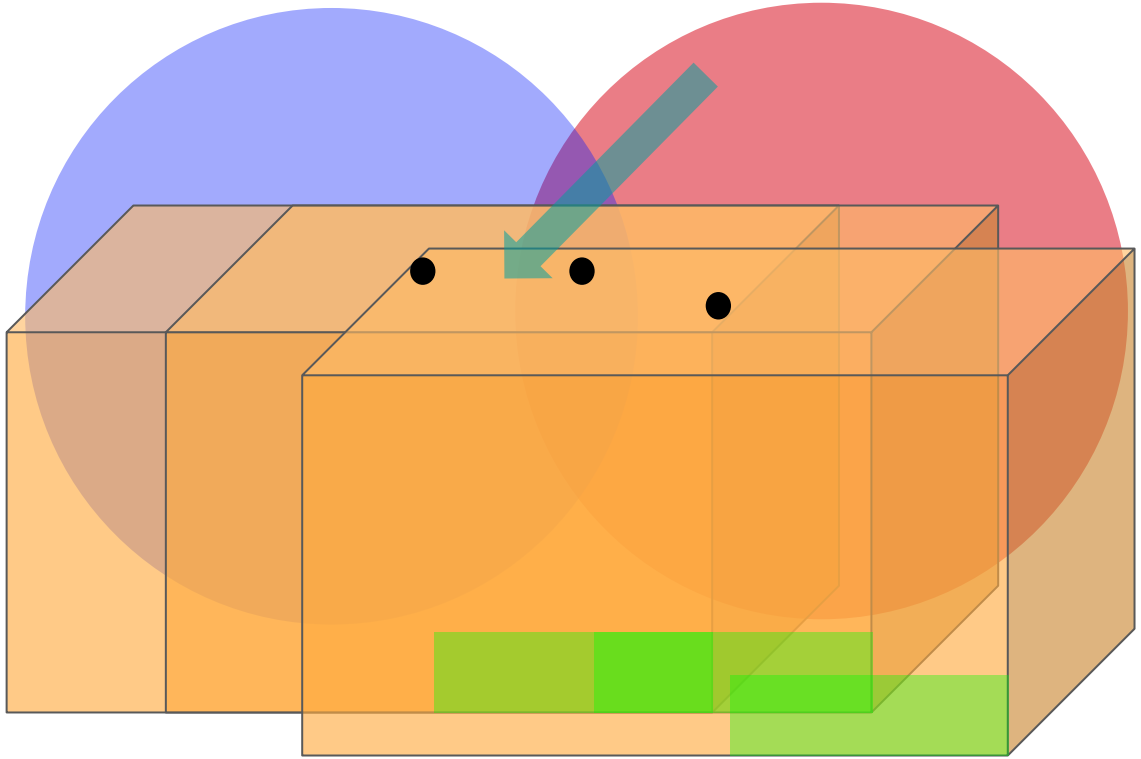


Hypergraph: Hyperedges



DRD: Tests

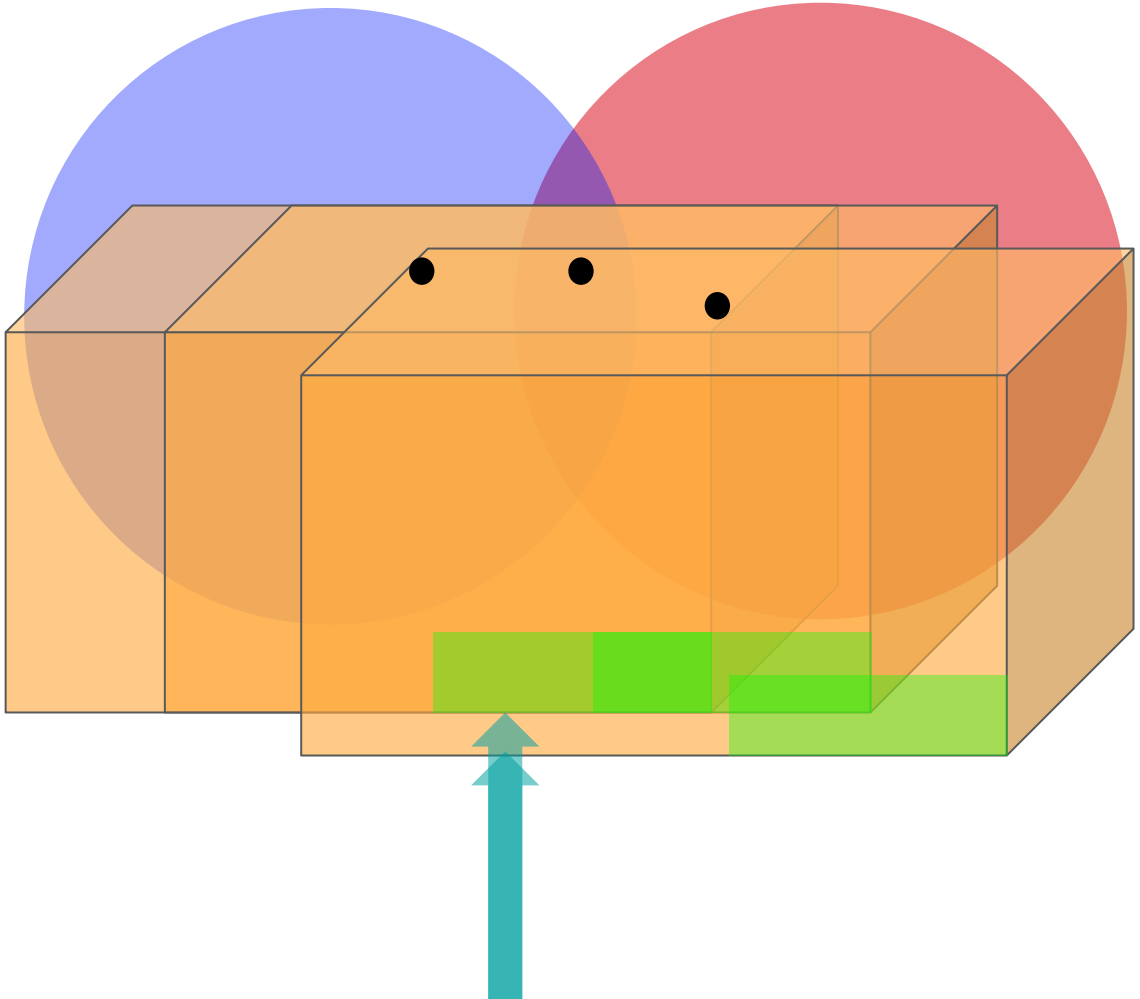


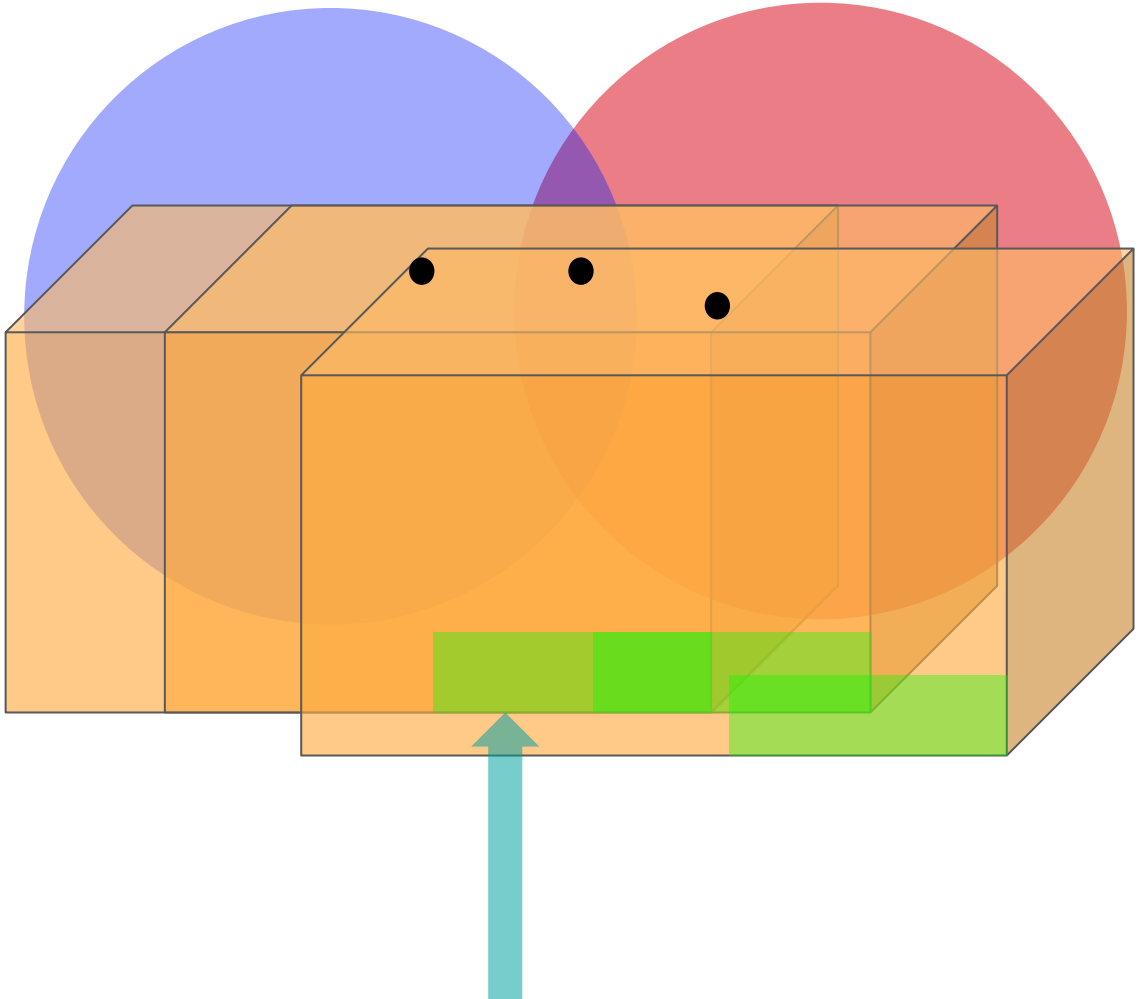


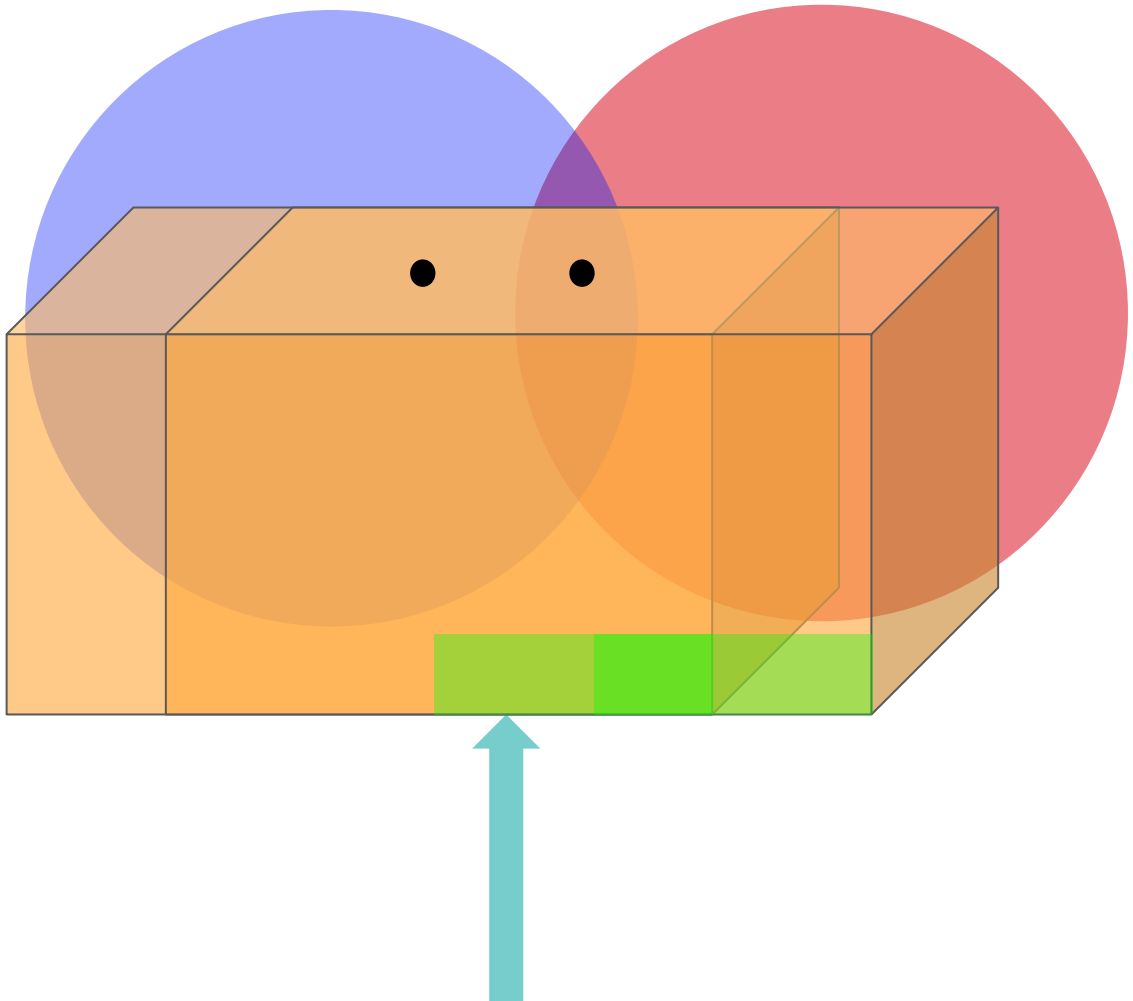
$$f_{HEC}(\mathcal{S}) = w(\mathcal{E}) - w(\mathcal{E}(\mathcal{S}))$$

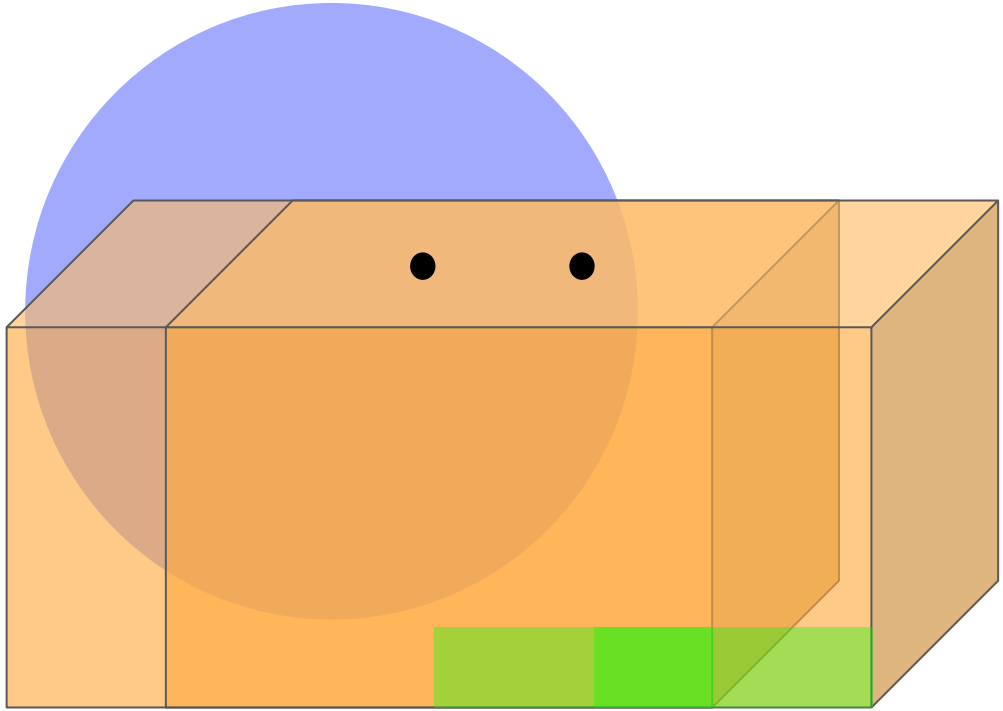
$$\Delta(t|\mathcal{S}) = \sum_{h} P(h|\mathcal{S}) \left(f_{HEC}\left(\mathcal{S} \cup \{(t, h(t))\}\right) - f_{HEC}(\mathcal{S}) \right)$$

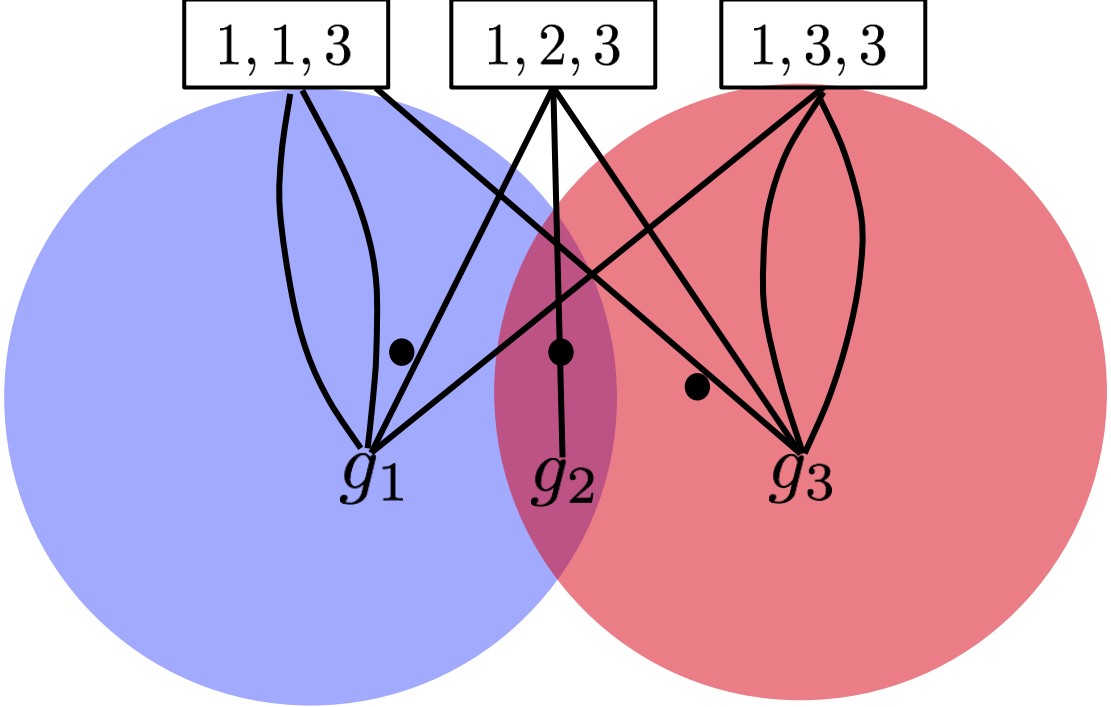
$$\Delta(t|\mathcal{S}) = 0$$

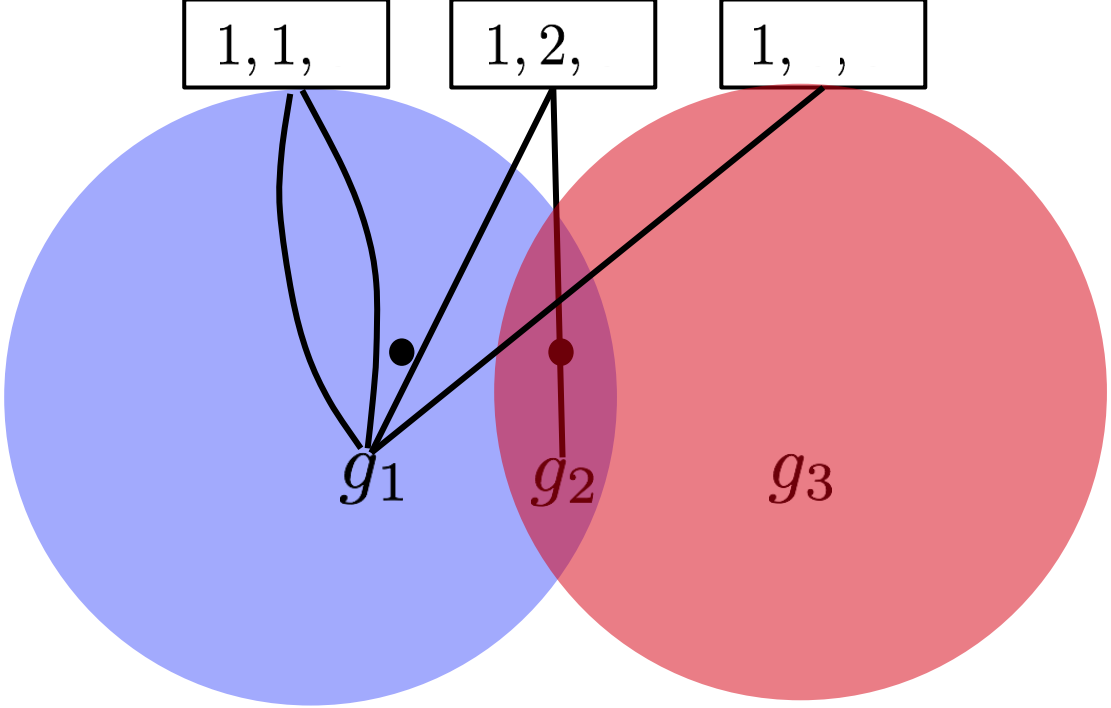


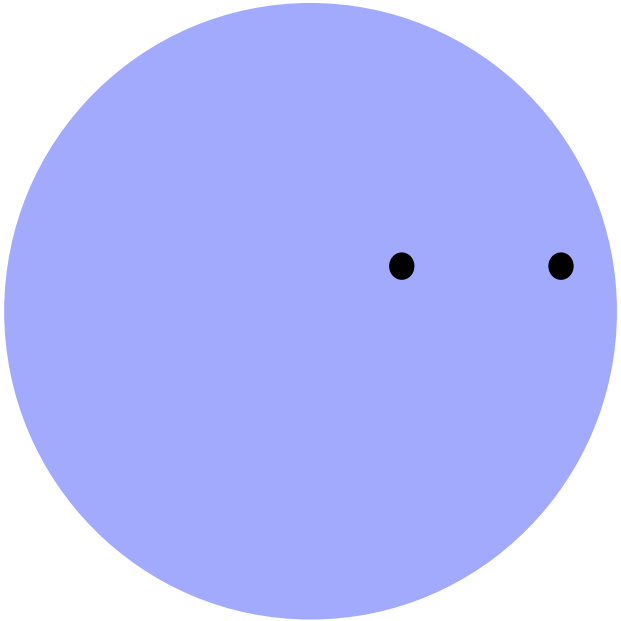


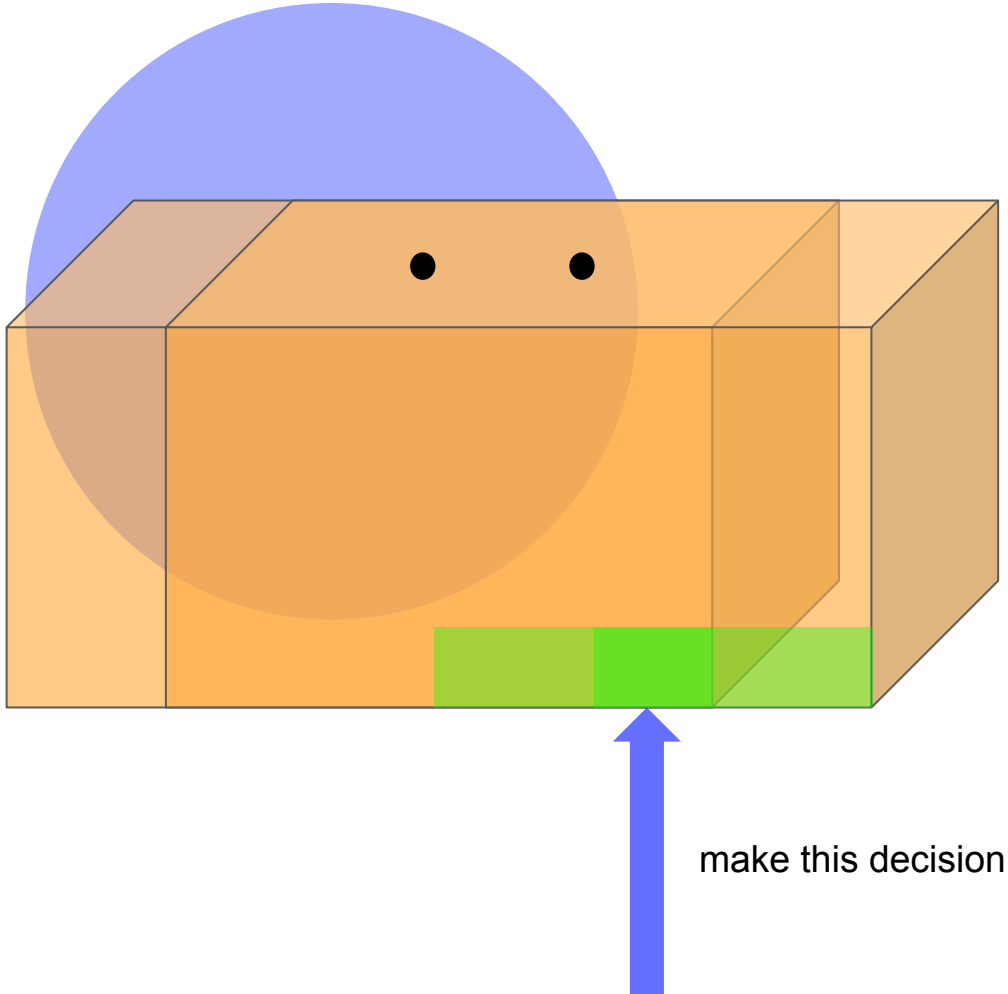





1, 1, 3
1, 2, 3
1, 3, 3
g_1
g_2
g_3


1, 1,
1, 2,
1, ,
g_1
g_2
g_3

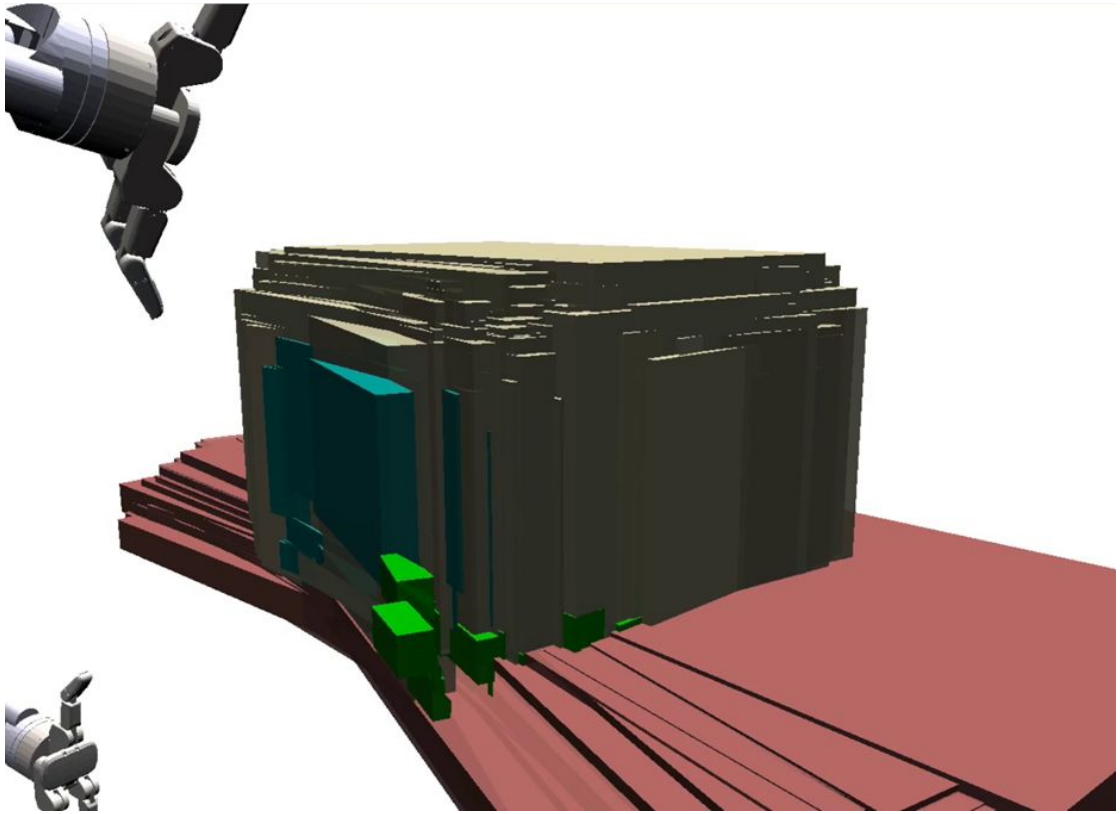



make this decision

Hypotheses

Sample random
object locations (2000)

Uncertainty due to noise from sensors, inaccurate models, calibration error



Javdani, Shervin. "Gathering Information For Decision Making In Touch Based Localization". Presentation.

Tests

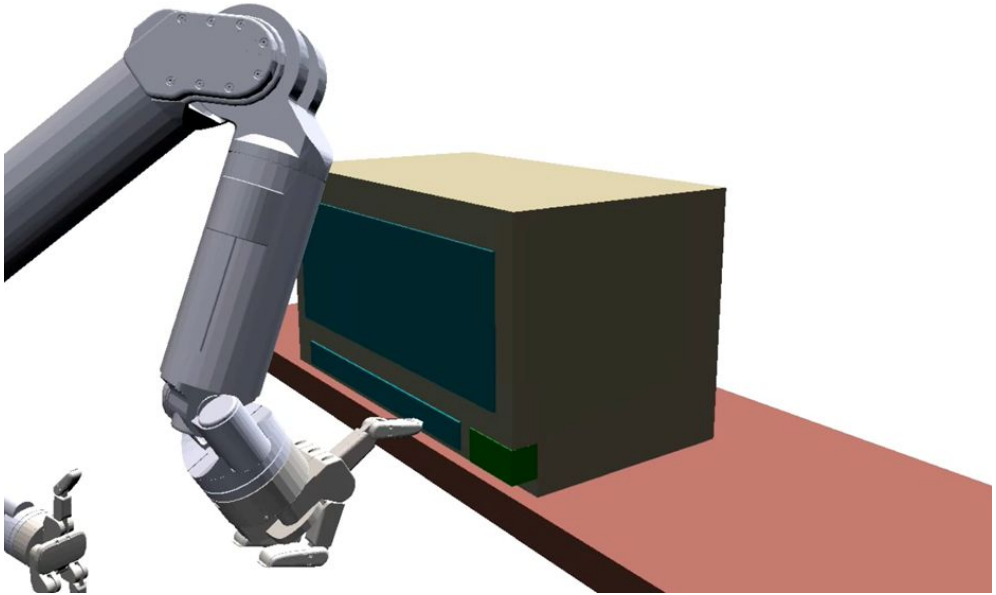
Sample random start locations and orientations (150)

Move along path until contact is sensed

h(t) = *o* = distance travelled

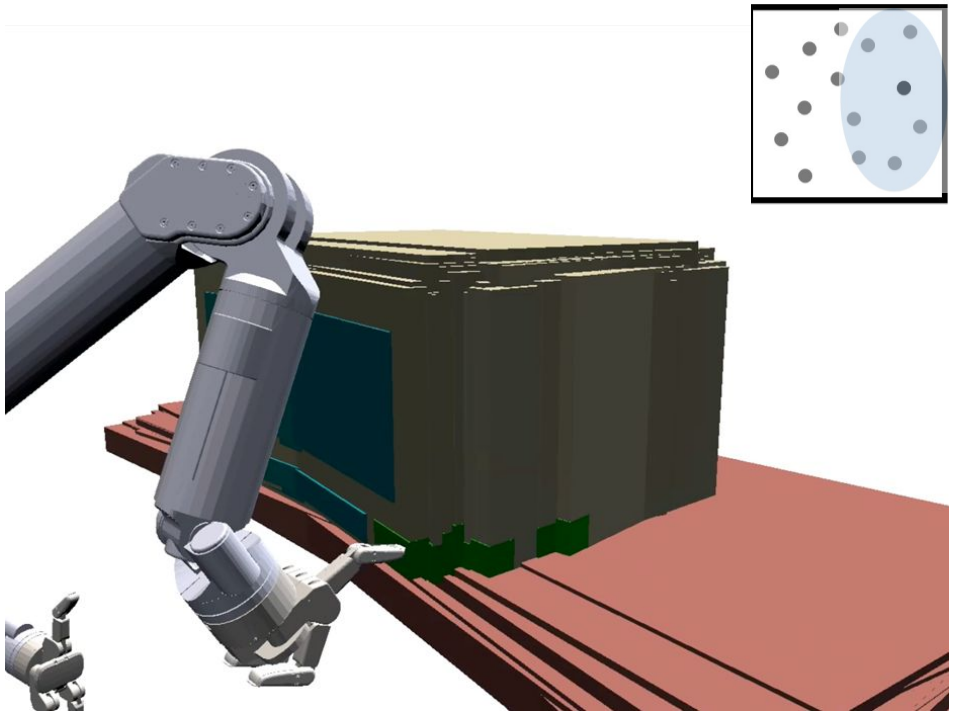
Decisions

Set of start locations (50)

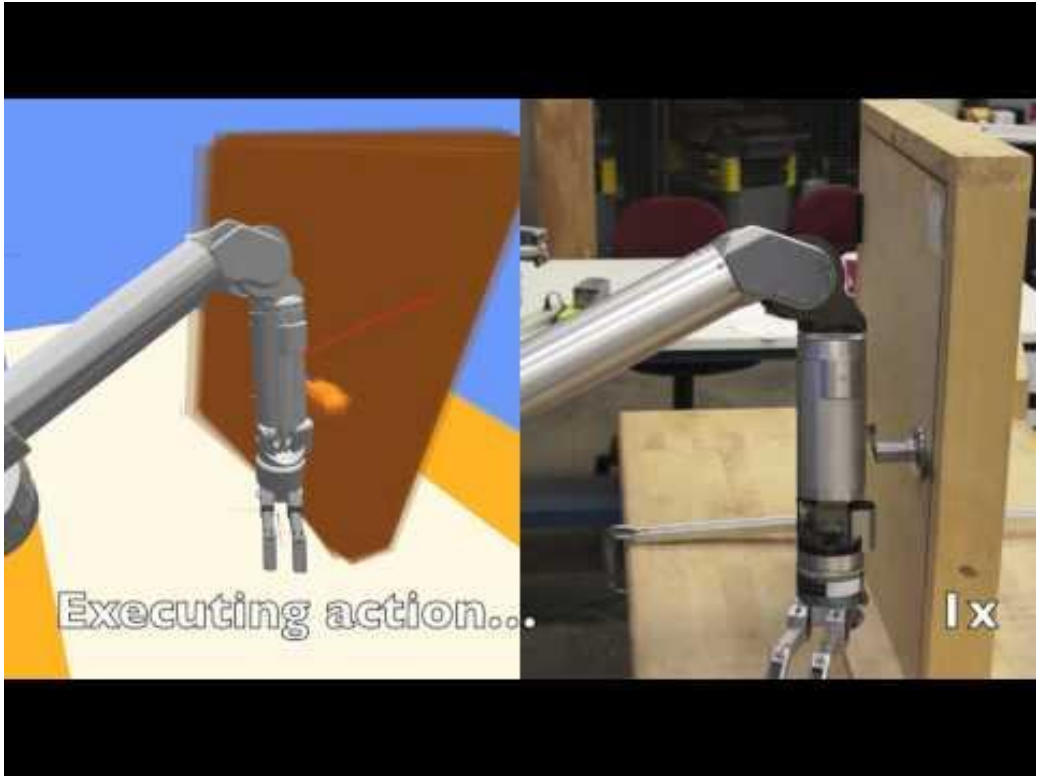


Javdani, Shervin. "Gathering Information For Decision Making In Touch Based Localization". Presentation.

Decision Regions



Javdani, Shervin. "Gathering Information For Decision Making In Touch Based Localization". Presentation.



Javdani, Saeed, Matthew Klingensmith, J. Andrew Bagnell, Nancy S. Pollard, and Siddhartha S. Srinivasa. "Efficient touch based localization through submodularity." In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 1828-1835. IEEE, 2013.

End

Simon, Tong. *Active Learning: Theory and Applications.* Stanford, 2001.
http://www.robotics.stanford.edu/~stong/papers/tong_thesis.pdf

Images:
http://www.smh.com.au/content/dam/images/1/m/m/n/r/w/image.related.articleLeadwide.620x349.1mmcg3.png/1429172171452.jpg

http://s7d2.scene7.com/is/image/woodstream/hh-animals-squirrel-4?$ProductPgLarge2$

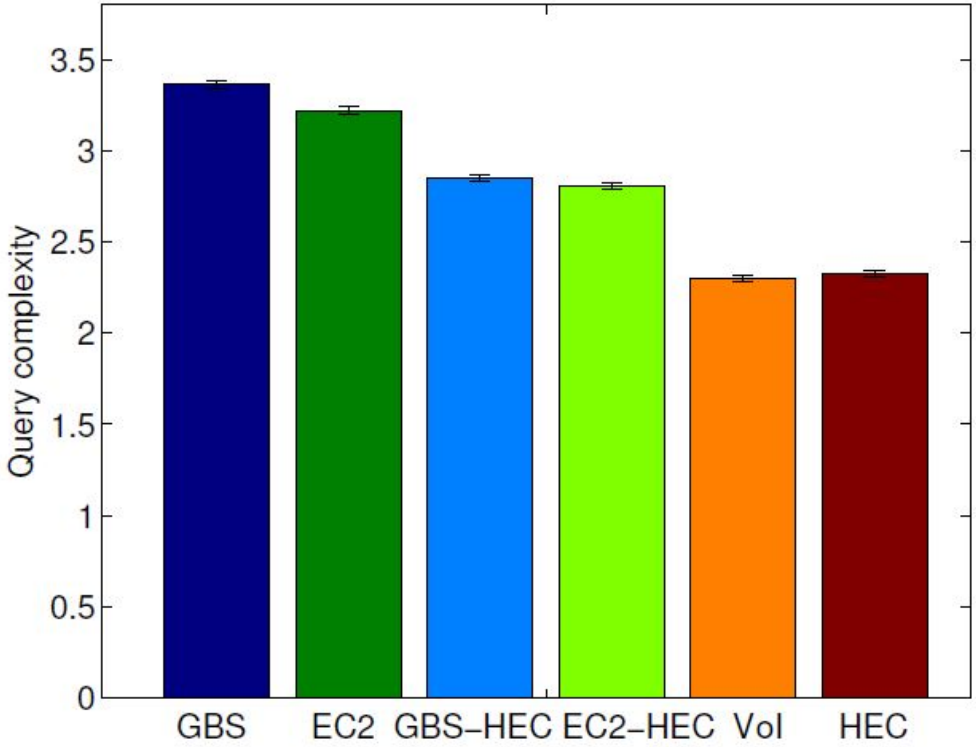
http://platypus.fi/img/1413727023_99P8kd6.jpg

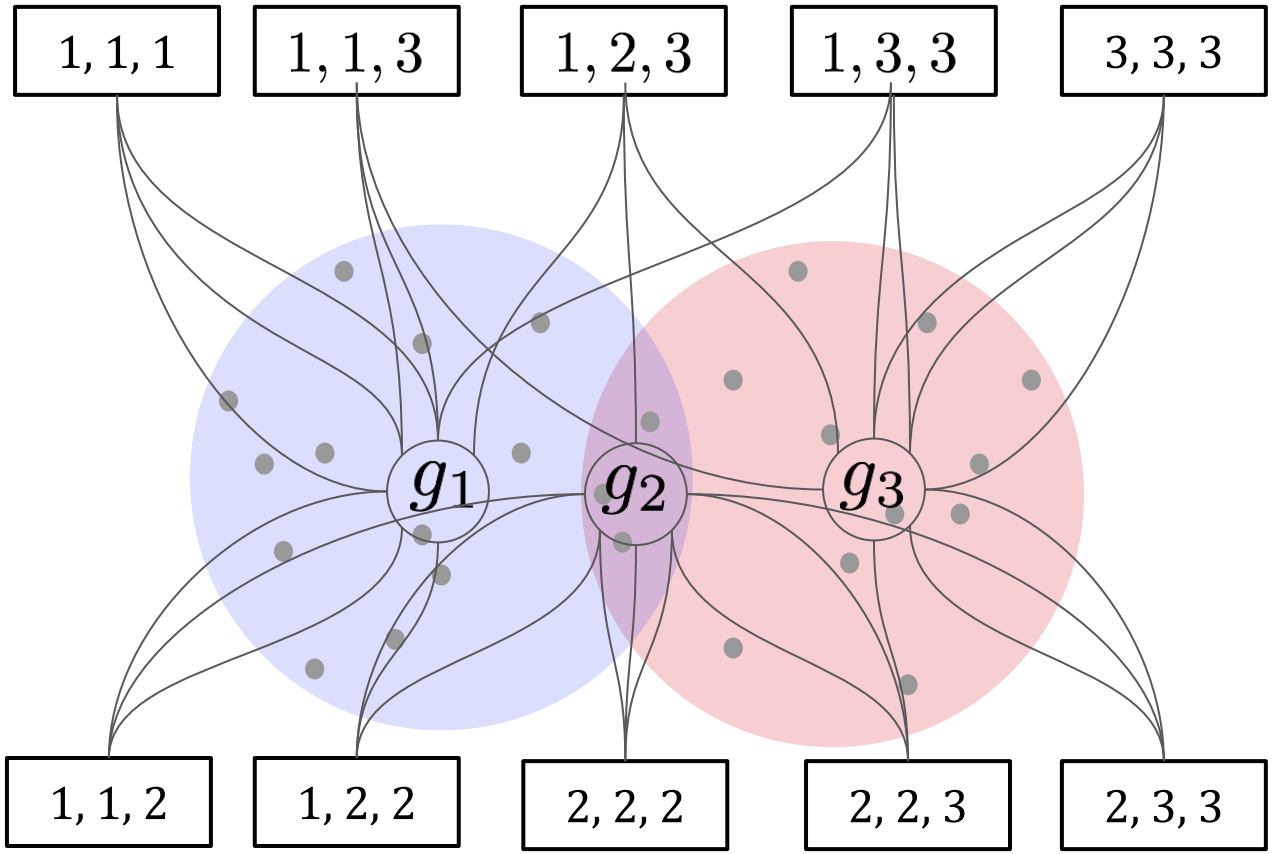
http://a-z-animals.com/media/animals/images/original/iguana8.jpg

https://i.ytimg.com/vi/fSR3x2i-ncM/hqdefault.jpg

https://aggietranscript.wordpress.com/2014/03/12/drosophila-are-cute/

Results


Query complexity
3.5
3
2.5
2
1.5
1
0.5
0
GBS
EC2
GBS–HEC
EC2–HEC
Vol
HEC



1, 1, 1

1, 1, 2

1, 1, 3

1, 2, 2

1, 2, 3

1, 3, 3

2, 2, 2

2, 2, 3

2, 3, 3

3, 3, 3

$$w(\{e_1,\ldots,e_n\}) = \sum_{l=1}\prod_{i=1} P(g_i)$$

$$w(\{e_1,\ldots,e_n\}) = CHP_k(\boldsymbol{x}) = \frac{1}{k}\sum_{j=1}^{k} CHP_{k-j}(\boldsymbol{x})PS_j(\boldsymbol{x})$$

$$w(\{e_1,\ldots,e_n\}) = CHP_k(\boldsymbol{x})$$

$$CHP_k(\boldsymbol{x}) = \frac{1}{k}\sum_{j=1}^{k} CHP_{k-j}(\boldsymbol{x})PS_j(\boldsymbol{x})$$

$$PS_i(\boldsymbol{x}) = \sum_{\boldsymbol{x}\subseteq\boldsymbol{x}} x^i$$