AI::CBR
Case-Based Reasoning For Perl

Darko Obradovic

AI::CBR

Case-Based Reasoning for Perl

Darko Obradovic

Kaiserslautern.pm

Kaisers Lautern Perl mongers

http://kaiserslautern.pm.org
darko@kaiserslautern.pm.org

DFKI GmbH

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

http://www.dfki.uni-kl.de/~obradovic
darko.obradovic@dfki.de

Another Vietnam?

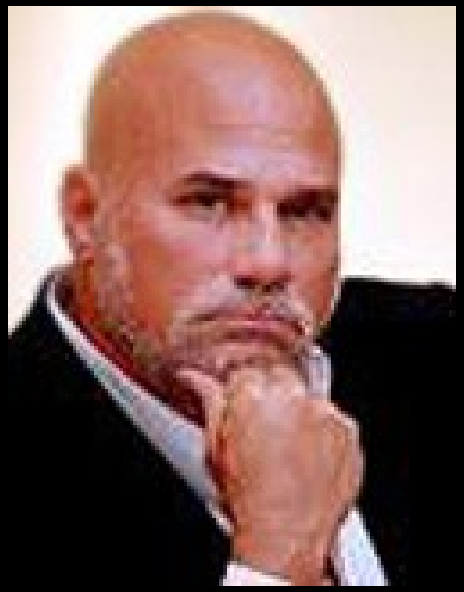
Ed Timms
Dallas Morning News
1995

Artificial Intelligence?

AI is the mimicking of human thought and cognitive processes to solve complex problems automatically.

(one definition amongst many)

Case-Based Reasoning

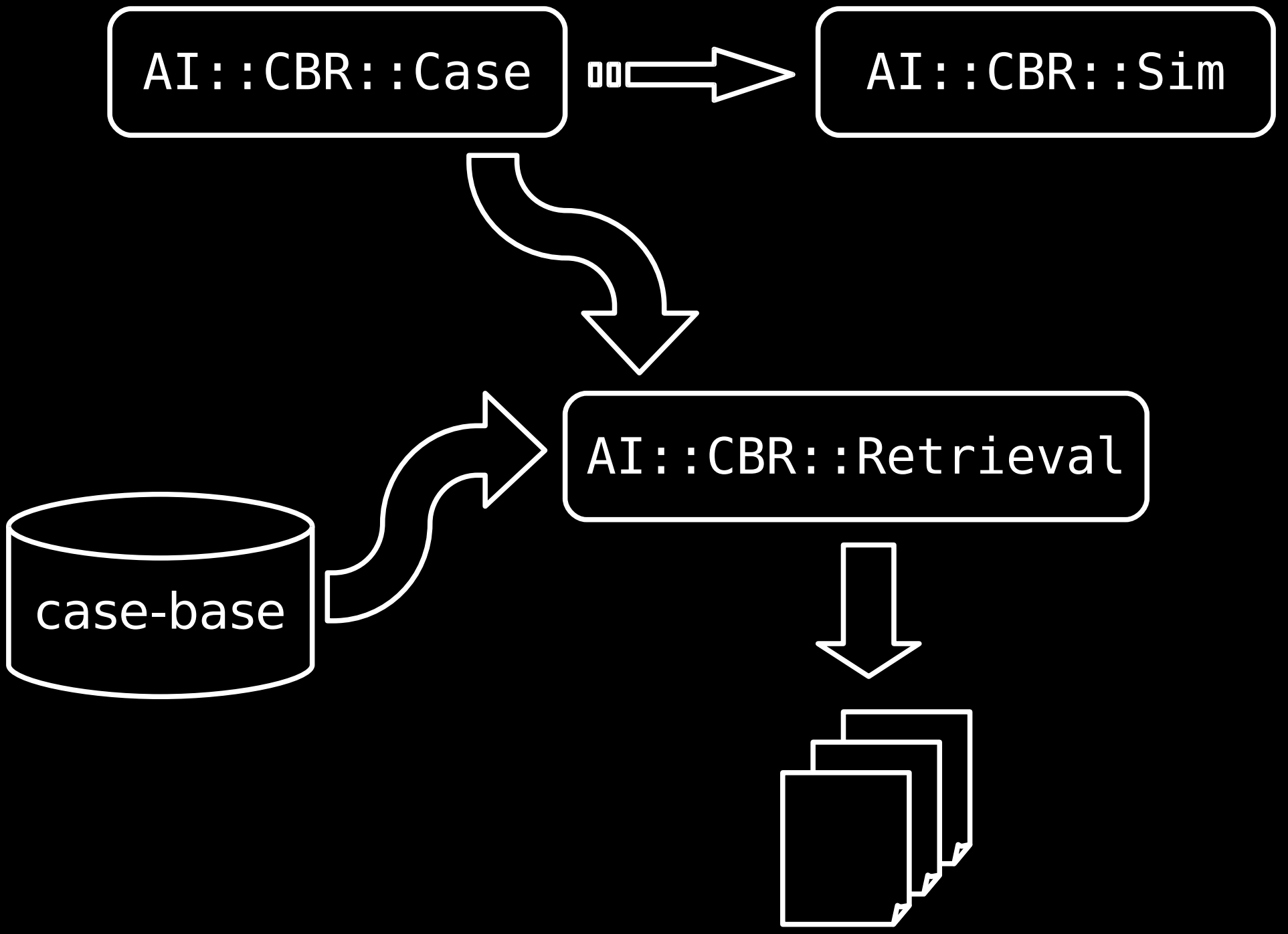


Foundations:
Roger Schank 1977/1983

1. new problem = new case
2. experience = case-base
3. remembering = retrieval
4. apply to new problem = reuse
5. observe & learn = revise
6. memorise = retain

1. new problem = new case
2. experience = case-base
3. remembering = retrieval
4. apply to new problem = reuse
5. observe & learn = revise
6. memorise = retain

AI::CBR


AI::CBR::Case
AI::CBR::Sim
AI::CBR::Retrieval
case-base

An example!

```
# the case
{
   age       => 40,
   gender    => 'male',
   job       => 'programmer',
   symptoms => ['headache',
                'cough'],
}
```

$diagnosis = ?

It's all about similarity!

compare two objects A and B
on attribute level
with similarity functions:

```
$sim_1 = sim($att_A_1, $att_B_1)
$sim_2 = sim($att_A_2, $att_B_2)
...
$sim_n = sim($att_A_n, $att_B_n)
```

compare two objects A and B
on attribute level
with similarity functions:

```
$sim_1 = sim($att_A_1, $att_B_1)
$sim_2 = sim($att_A_2, $att_B_2)
...
$sim_n = sim($att_A_n, $att_B_n)
```

similarity values are between 0 and 1
- 0.0 is no similarity at all
- 1.0 is equality

the overall similarity
between two objects A and B
is the mean value
of their attribute's similarities:

```
$sim_A_B = (
  $sim_1 + $sim2 + … + $sim_n
) / n;
```

the overall similarity
between two objects A and B
is the mean value
of their attribute's similarities:

```
$sim_A_B = (
  $sim_1 + $sim2 + … + $sim_n
) / n;
```

similarity values are between 0 and 1
- 0.0 is no similarity at all
- 1.0 is equality

```
use CBR::AI::Sim qw(sim_eq sim_frac sim_set);

# for symbolic values
sim_eq('programmer', 'programmer'); # 1.0
sim_eq('programmer', 'manager');    # 0.0
```

```
use CBR::AI::Sim qw(sim_eq sim_frac sim_set);

# for symbolic values
sim_eq('programmer', 'programmer'); # 1.0
sim_eq('programmer', 'manager');    # 0.0

# for numbers
# fraction of the smaller wrt the greater
sim_frac(2, 4);   # 0.5
sim_frac(40, 30); # 0.75
```

```
use CBR::AI::Sim qw(sim_eq sim_frac sim_set);

# for symbolic values
sim_eq('programmer', 'programmer'); # 1.0
sim_eq('programmer', 'manager');    # 0.0

# for numbers
# fraction of the smaller wrt the greater
sim_frac(2, 4);   # 0.5
sim_frac(40, 30); # 0.75

# for sets/lists of symbolic values
# |intersection elements| / |union elements|
sim_set([qw(a b c d)], ['a']);     # 0.25
sim_set([qw(a b c)], [qw(b c d)]); # 0.5
```

```
# the case
{
   age       => 40,
   gender    => 'male',
   job       => 'programmer',
   symptoms => ['headache',
                'cough'],
}
```

$diagnosis = ?

case definition

```
use AI::CBR::Case;
use AI::CBR::Sim qw(
  sim_eq sim_frac sim_set
);

my $new_case = AI::CBR::Case->new(
  age      => { sim => \&sim_frac },
  gender   => { sim => \&sim_eq   },
  job      => { sim => \&sim_eq   },
  symptoms => { sim => \&sim_set  },
);
```

set case values

```
$new_case->set_values(
  age       => 40,
  gender    => 'male',
  job       => 'programmer',
  symptoms => ['headache','cough'],
);

# case definition and values
# could have been done in one step
```

fetch case-base

```
use DBIx::Simple;
my $db = DBIx::Simple->new(...);

my @case_base = $db->query(
  SELECT age, gender, job, symptoms,
         diagnosis
  FROM patients
)->hashes();

foreach @case_base {
  $_->{symptoms} = [split ',', $_->{symptoms}]
}
```

retrieve most similar case

```
use AI::CBR::Retrieval;

my $r = AI::CBR::Retrieval->new(
  $new_case, [@case_base]
);
$r->compute_sims();

my $solution = $r->most_similar_candidate();
print $solution->{diagnosis};
```


```
# in case-base
{
  age       => 28,
  gender    => 'male',
  job       => 'programmer',
  symptoms  => ['headache'],
  diagnosis => 'hangover',
}
```

```
# for age:
sim_frac(40, 28) # 0.7
```

```
# for age:
sim_frac(40, 28) # 0.7
# for gender:
sim_eq('male', 'male') # 1.0
```

```
# for age:
sim_frac(40, 28) # 0.7
# for gender:
sim_eq('male', 'male') # 1.0
# for job:
sim_eq('programmer', 'programmer') # 1.0
```

```
# for age:
sim_frac(40, 28) # 0.7
# for gender:
sim_eq('male', 'male') # 1.0
# for job:
sim_eq('programmer', 'programmer') # 1.0
# for symptoms:
sim_set(['headache','cough'], ['headache'])
# 0.5
```

```
# for age:
sim_frac(40, 28) # 0.7
# for gender:
sim_eq('male', 'male') # 1.0
# for job:
sim_eq('programmer', 'programmer') # 1.0
# for symptoms:
sim_set(['headache','cough'], ['headache'])
# 0.5

$sim = 3.2 / 4; # 0.8
```

```
# for age:
sim_frac(40, 28) # 0.7
# for gender:
sim_eq('male', 'male') # 1.0
# for job:
sim_eq('programmer', 'programmer') # 1.0
# for symptoms:
sim_set(['headache','cough'], ['headache'])
# 0.5

$sim = 3.2 / 4; # 0.8
```

$diagnosis => 'hangover'!

What else can you do?

Similarity Functions

- define your own ones!
  - there are millions of different use-cases
  - flexibility with optional parameter

Similarity Functions

- define your own ones!
  - there are millions of different use-cases
  - flexibility with optional parameter
- code them in C
  - most efficient tuning

Retrieval

- multiple methods:
  - `most_similar_candidate()`
  - `n_most_similar_candidates($n)`
  - `first_confirmed_candidate('diagnosis')`

Retrieval

- multiple methods:
  - `most_similar_candidate()`
  - `n_most_similar_candidates($n)`
  - `first_confirmed_candidate('diagnosis')`
    1. Hangover
    2. Stress
    3. Whiplash Injury
    4. Stress
    5. …

Retrieval

- multiple methods:
  - `most_similar_candidate()`
  - `n_most_similar_candidates($n)`
  - `first_confirmed_candidate('diagnosis')`
    1. Hangover
    2. Stress
    3. Whiplash Injury
    4. Stress
    5. …

Retrieval

- multiple methods:
  - `most_similar_candidate()`
  - `n_most_similar_candidates($n)`
  - `first_confirmed_candidate('diagnosis')`
    1. Hangover
    2. Stress
    3. Whiplash Injury
    4. Stress
    5. …
- subclass & extend!

Intelligent Product Retrieval

Year

All
1995
1994
1993
1992

Looking for older or classic cars?
Use our 1982 and older search.

Price Range $0 to No Maxim

Mileage Range 0 to No Maxim

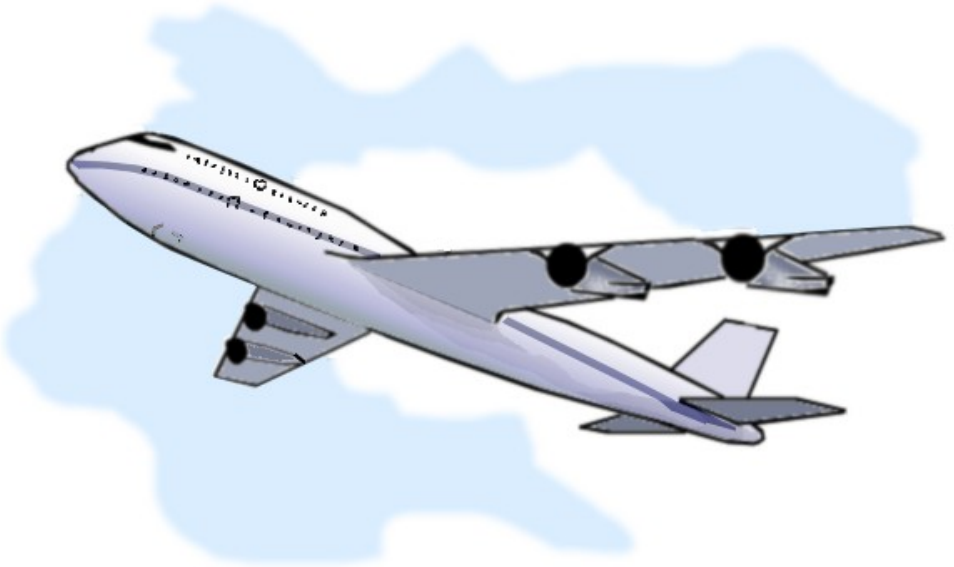
Search within 30 miles of ZIP Code

— OR —

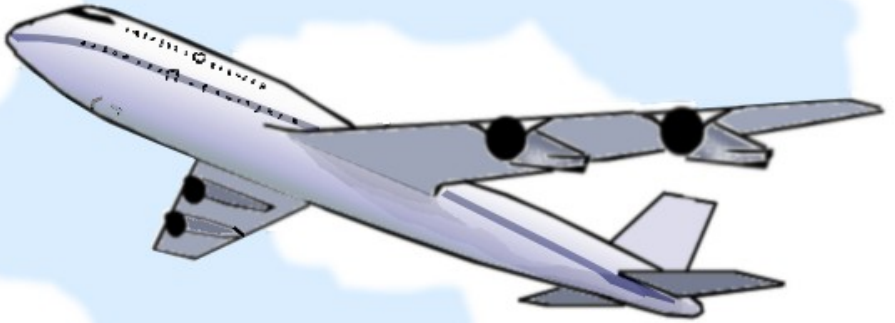
Choose a City

1. query = new case
2. products = case-base
3. search = retrieval

Compound Cases



Trip

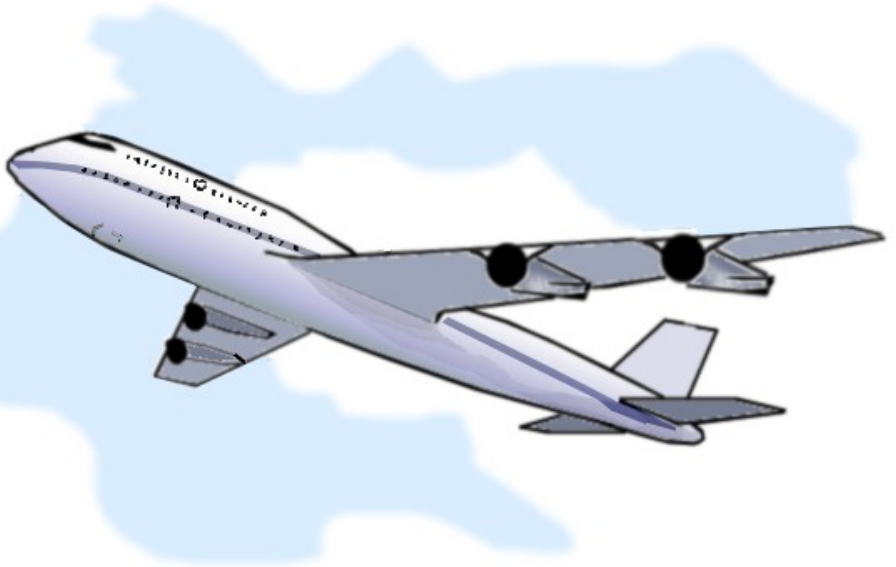


0.9

Trip

0.9

0.9



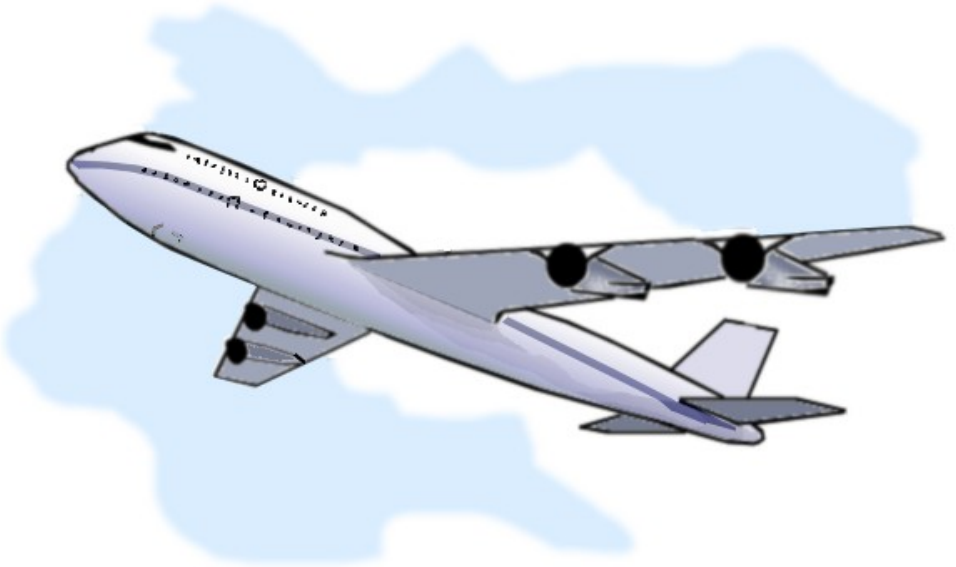
0.9

0.9

Trip

0.9

0.9

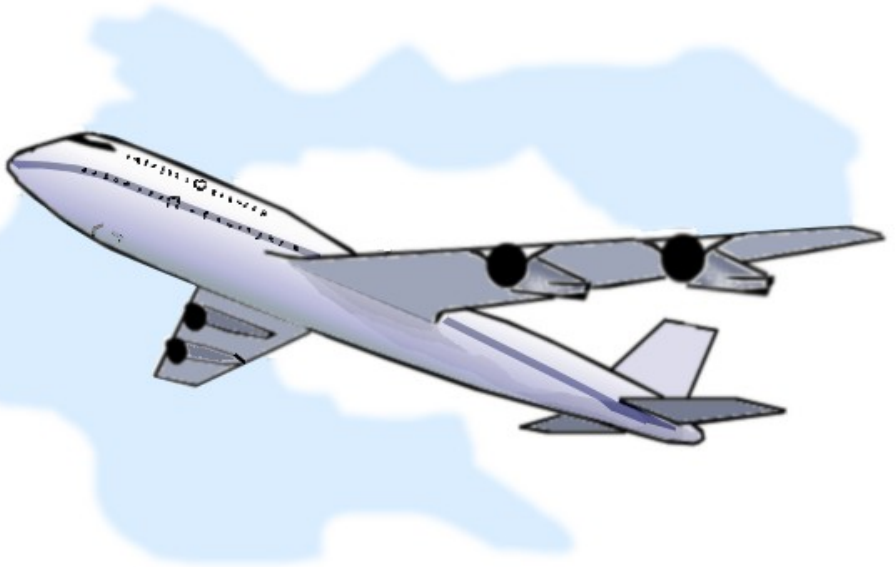


0.9

Trip

0.9

0.1



0.9

0.63 ?

Trip

0.9

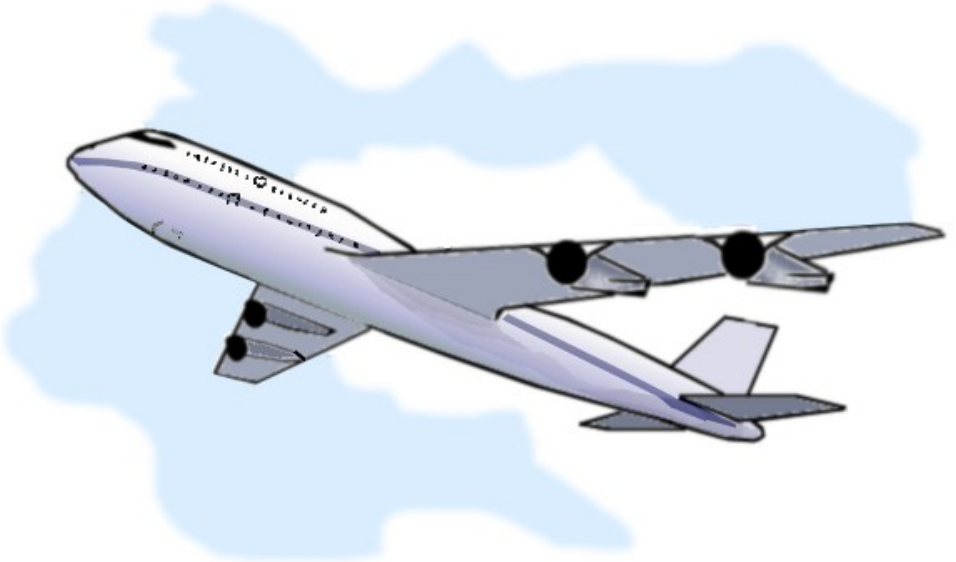
0.1

Compound Case

```
use AI::CBR::Case::Compound;

my $case = AI::CBR::Case::Compound->new([
  $flight_spec,
  $hotel_spec,
  $destination_spec,
]);

$sim = nroot(3,   $sim_flight
                * $sim_hotel
                * $sim_destination);
```



0.9

0.43

Trip

0.9

0.1

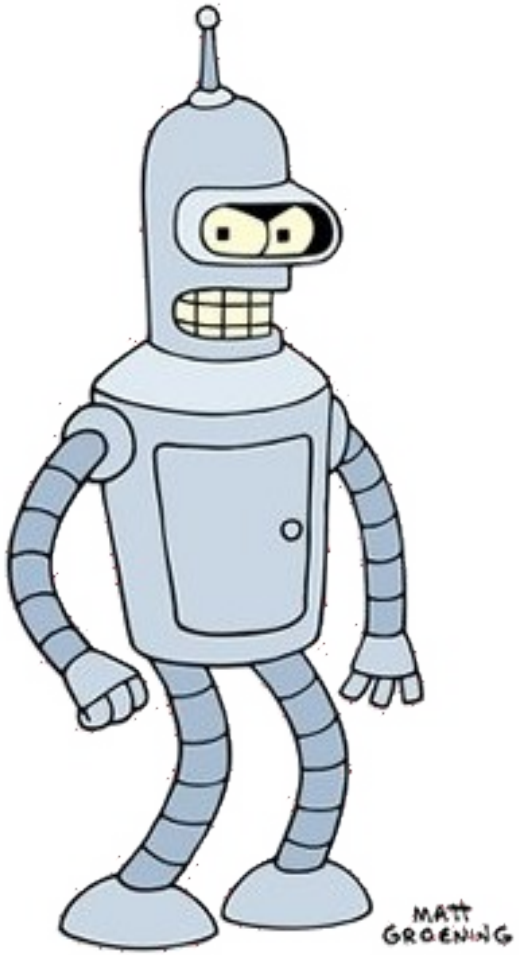
What did you do with it?

CYBORG

AI for a strategy game

observation

situations
+
decisions



reuse

situations
+
decisions

Conclusions

- observed 15-30 games per player
- thousands of comarisons per turn
  - ~400 turns per game

→ system performance ok

→ AI performance good

- for details, see homepage

The End

Thank you!

Questions?

matrix similarity function

```
my $os_matrix = {
  linux => { mac   => 0.8, win => 0.2 },
  mac   => { linux => 0.8, win => 0.5 },
  win   => { linux => 0.2, mac => 0.5 },
};

$new_case = AI::CBR::Case->new(
  os => { sim   => \&sim_matrix,
          param => $os_matrix },
  ...
);
```

matrix similarity function

```
sub sim_matrix {
  my ($a, $b, $matrix) = @_;
  return 1 if $a eq $b;
  return $matrix->{$a}->{$b};
}
```