Machine Learning for Optimal Matchmaking

Tom Minka, Ryan Cleven, Josh Menke

GDC 2020

Online Game Technology Summit

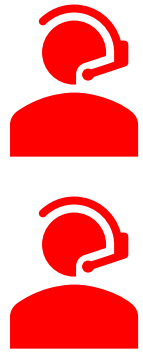
Microsoft Research

What is matchmaking?

Matchmaker

Name	Bob
Skill	1677
Ping	35
Party Size	2
Streak	1

Team 1

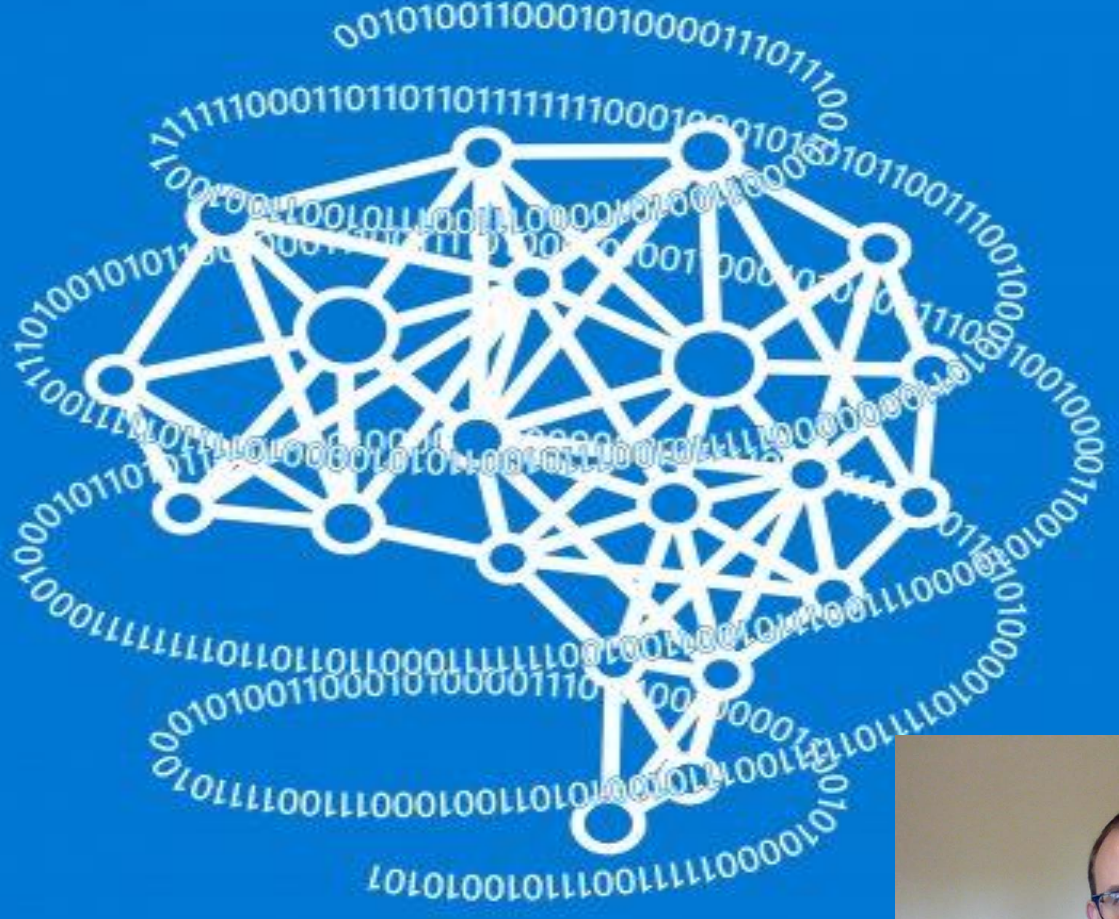


VS

Team 2

What is Machine Learning?

An algorithm that tunes itself using data rather than by hand.



Machine Learning and Matchmaking

Matchmaking algorithms are traditionally tuned by hand

Machine learning lets us tune them automatically from data

State of the Art Today

HOPE FOR THE BEST POSSIBLE MATCH

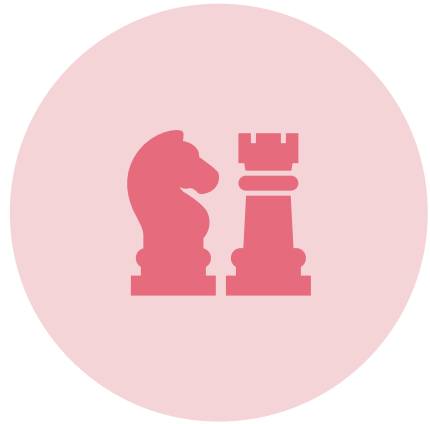
WAIT

SETTLE FOR SOMETHING WORSE

State of the Art Today: Skill

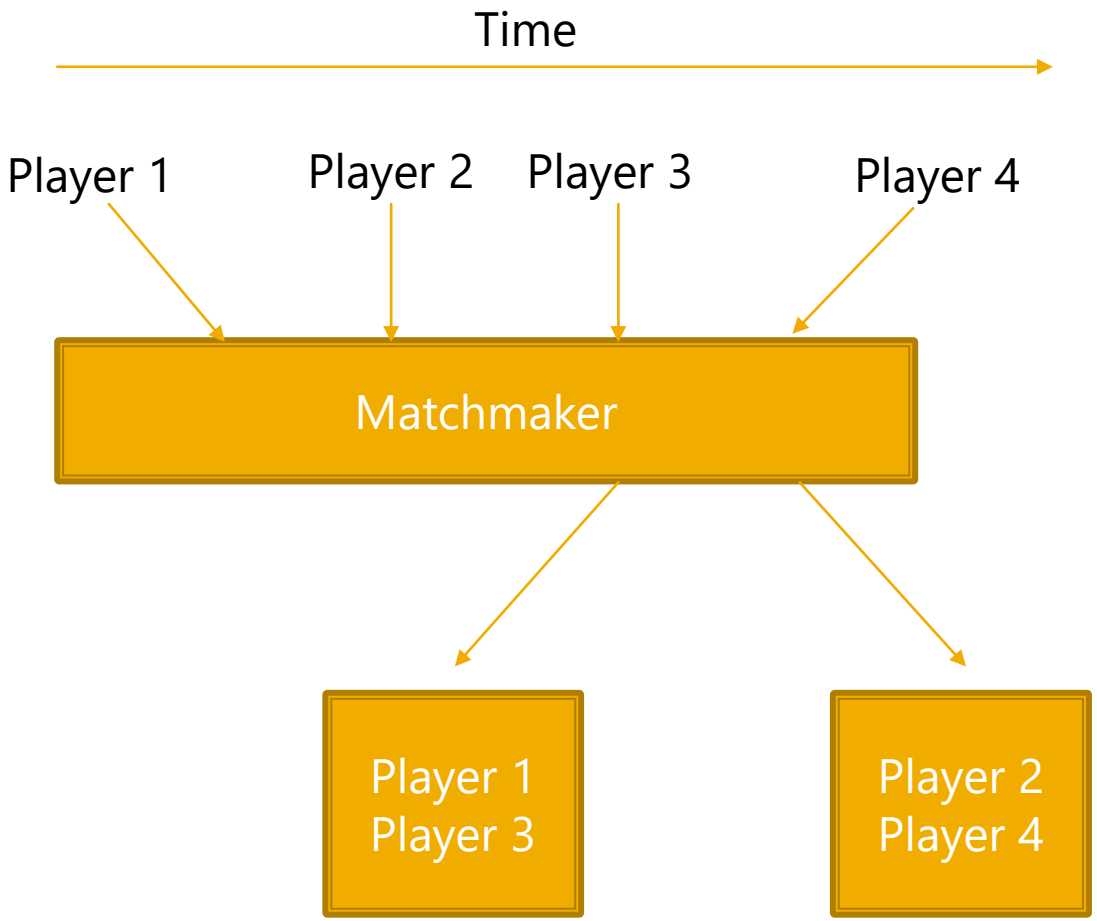
MATCHMAKER LOOKS FOR SAME-SKILLED PLAYERS

WAITS



SETTLES FOR IMBALANCE

Real-time matchmaking



If matchmakers knew what was coming, they could create optimal matches using combinatorial optimization.

So let's use machine learning to predict what's coming and be more optimal.

Conventional Matchmaker

First Configure a Set of Rules

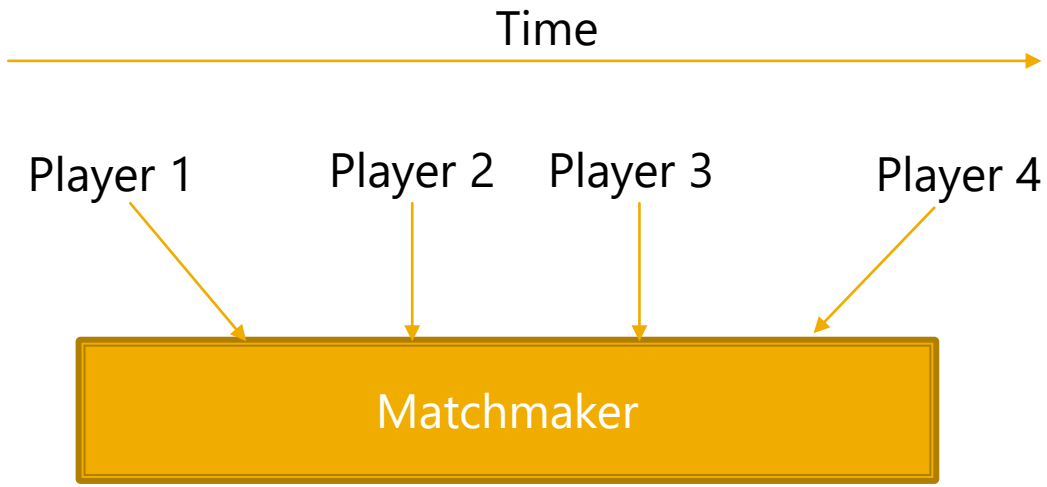
- Need min and max of 8 players: as 2 teams of 4
- Allowed Latency: grows 50ms to 200ms
- Allowed Skill gap: grows 1 to 10
- Build Versions must match
- Playlists must match

Example Configuration

```
"MatchmakingQueue": {
    "Name": "Standard4v4TeamsQueue",
    "MinMatchSize": 8,
    "MaxMatchSize": 8,
    "ServerAllocationEnabled": false,
    "Teams": [
        {
            "Name": "Red",
            "MinTeamSize": 4,
            "MaxTeamSize": 4
        },
        {
            "Name": "Blue",
            "MinTeamSize": 4,
            "MaxTeamSize": 4
        }
    ],
```

```
"Rules": [
        {
            "Type": "TeamDifferenceRule",
            "Attribute": {
                "Path": "Skill",
                "Source": "User"
            },
            "Difference": 0.2,
            "DefaultAttributeValue": 0.5,
            "Expansion": {
                "Delta": 0.1,
                "Limit": 0.5,
                "Type": "Linear",
                "SecondsBetweenExpansions": 5
            },
            "Name": "TeamSkillRule",
            "SecondsUntilOptional": 30
        }
    ]
}
```

Matchmaker Receives Requests



Matchmaker Request

Creation time

Player id

Skill rating (0-40)

Latency table (ping to each datacentre)

(East US: 40)(West Europe: 100)(Brazil: 200)

Can be summarized as Region = East US

Playlist id

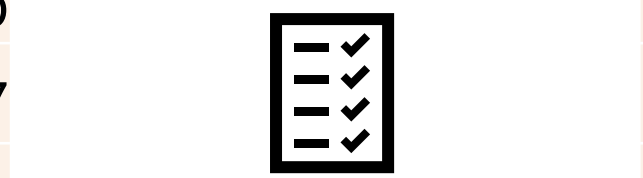
Build version

…

Compares Requests

Checking Each Rule

Name	Bob
Skill	27
Region	US
Creation	16:27:32
Build	15283
Playlist	Slayer



Skill gap < 10?

Name	Alice
Skill	32
Region	EU
Creation	16:26:23
Build	15283
Playlist	Slayer

Creates a Match

When all Rules pass

Name	Bob
Skill	27
Region	US
Creation	16:27:32
Build	15283
Playlist	Slayer

Skill gap < 10?

Name	Alice
Skill	32
Region	EU
Creation	16:26:23
Build	15283
Playlist	Slayer


Rules Apply Globally
Same thresholds across regions


Rules Are Static
Don’t Change As Pop Waxes and Wanes
Inflexible

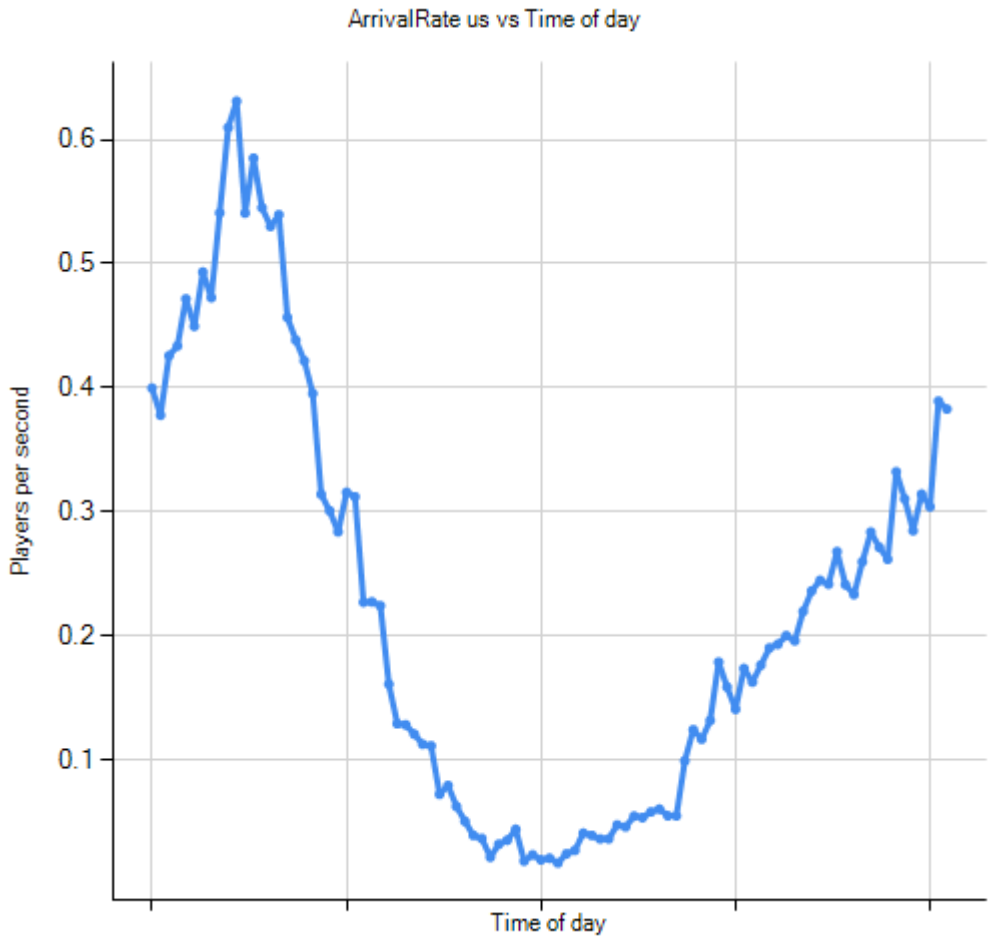

Conventional Consequences

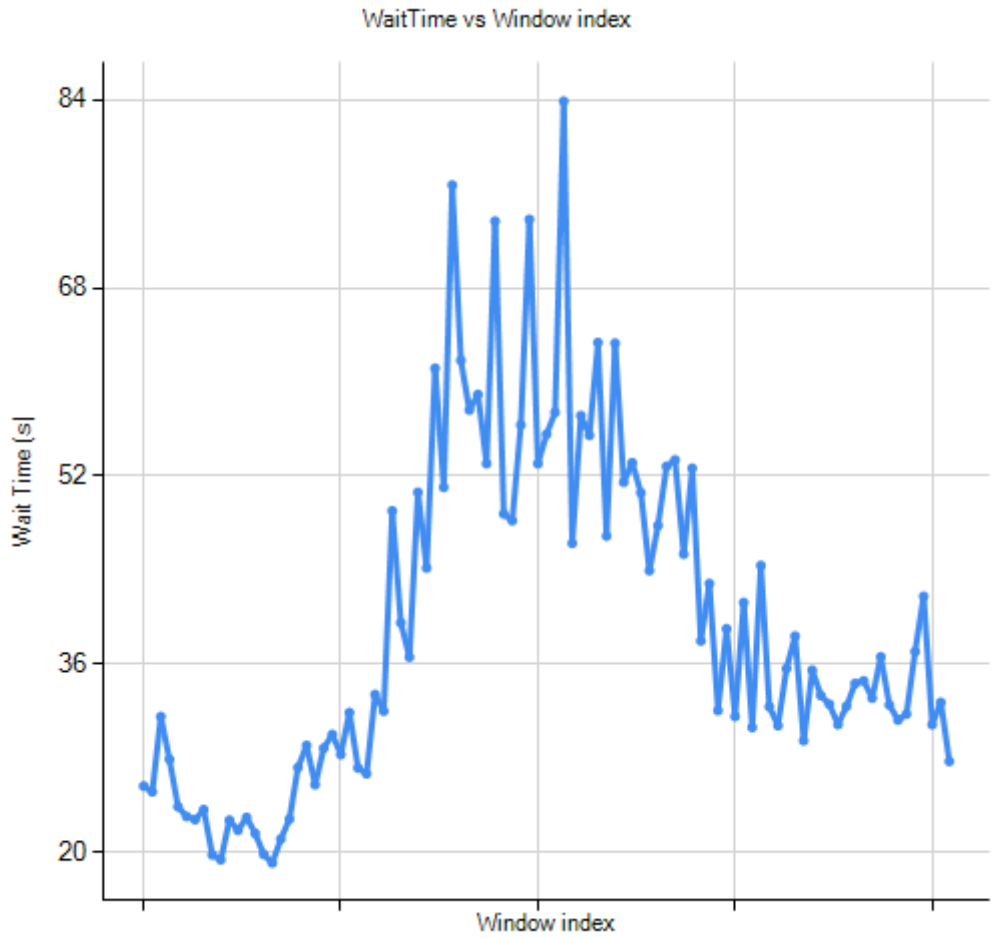
Predictability

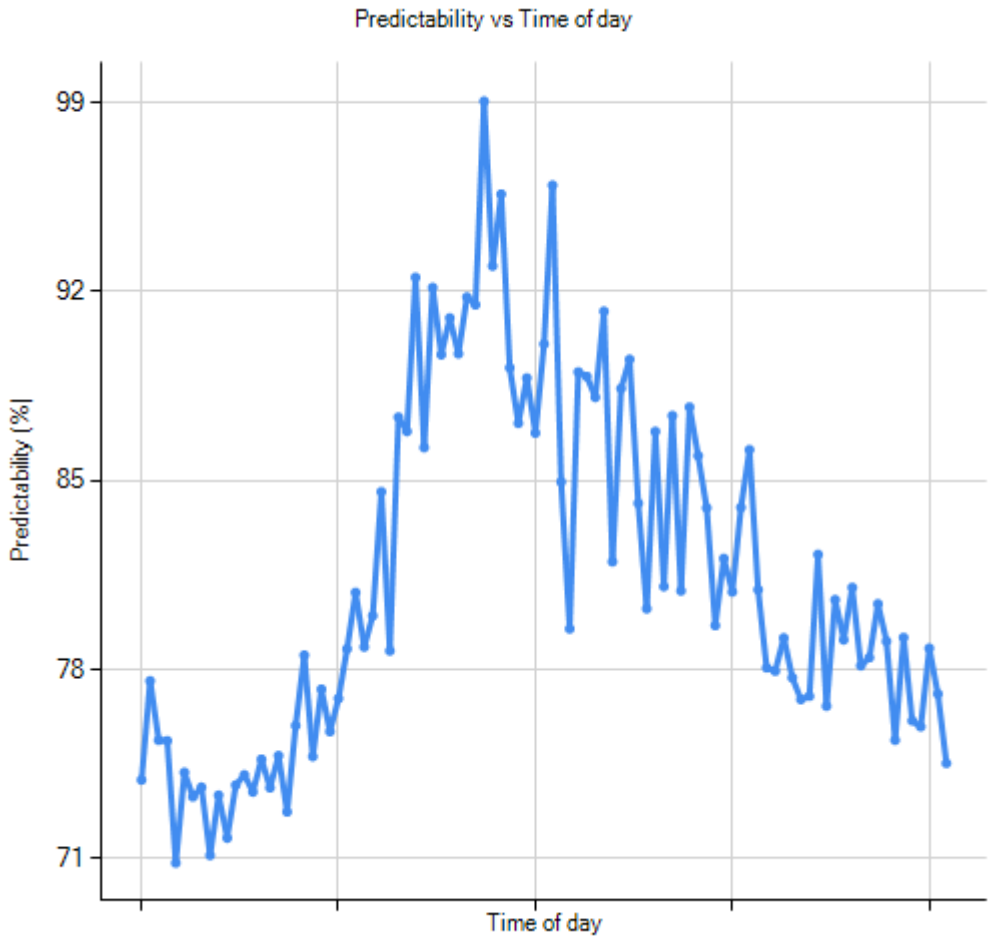
How often the better team wins

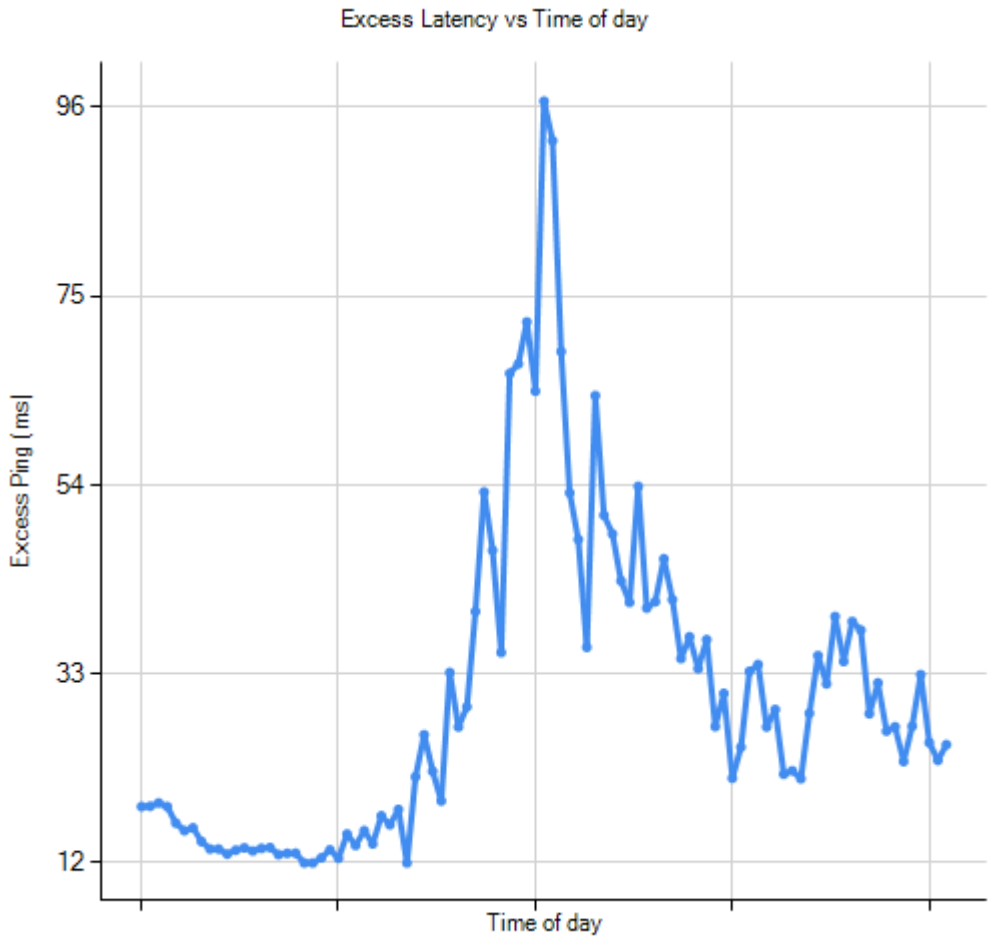
Primary measure of fairness

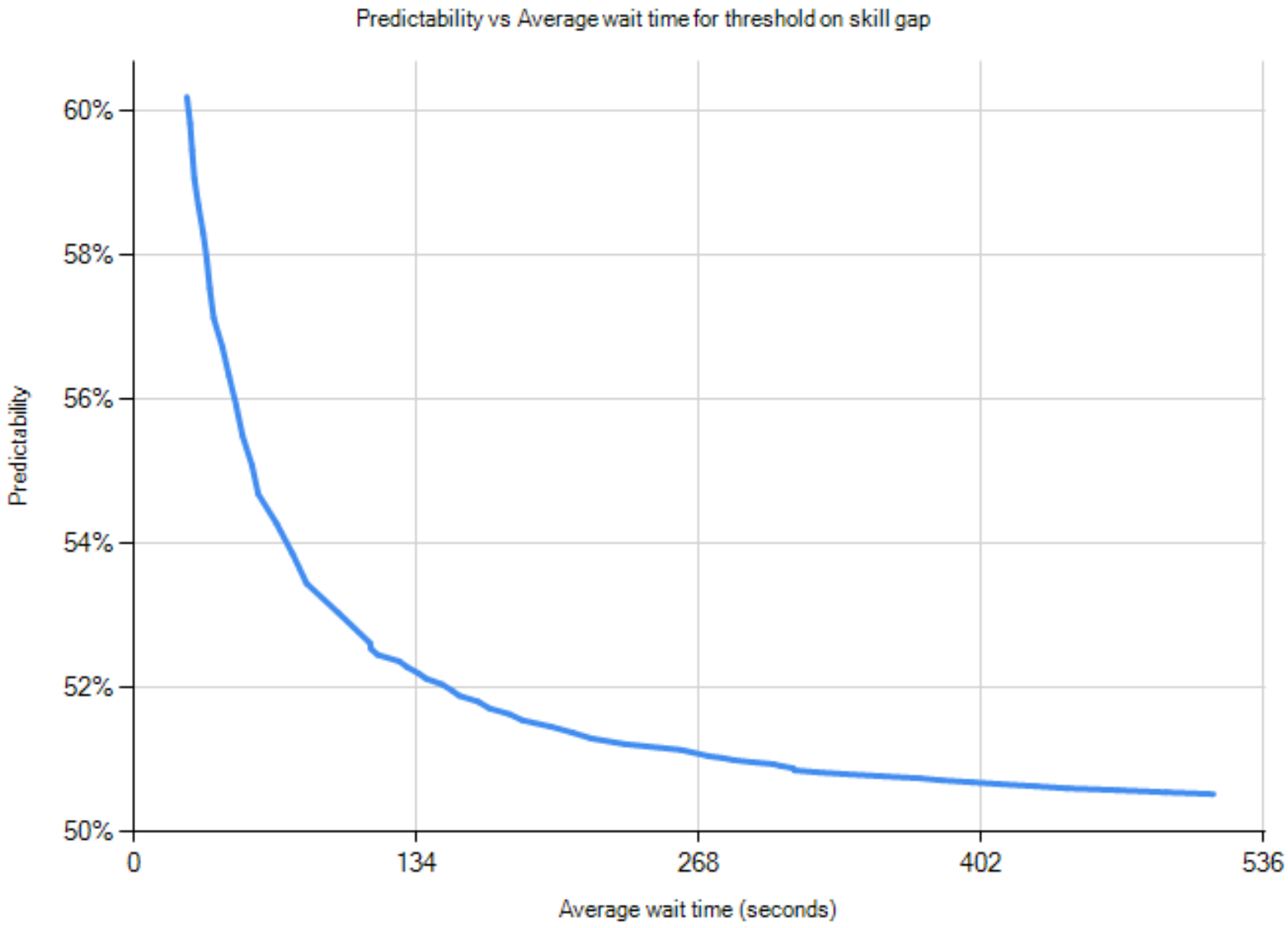
50% would be perfect fairness


ArrivalRate us vs Time of day
Players per second
0.6
0.5
0.4
0.3
0.2
0.1
Time of day


WaitTime vs Window index
Wait Time [s]
84
68
52
36
20
Window index


Predictability vs Time of day
99
92
85
78
71
Predictability [%]
Time of day


Excess Latency vs Time of day
96
75
54
33
12
Excess Ping [ms]
Time of day


Predictability vs Average wait time for threshold on skill gap
60%
58%
56%
54%
52%
50%
Predictability
0
134
268
402
536
Average wait time (seconds)

Conventional Takeaway

Conventional Matchmaking ignores

Realtime Incoming Request Rate

Realtime Skill distribution: Anyone good around?

Realtime region distribution Is EU playing now?

Result

Long wait times

or

Matches we know are Bad

(Large skill gaps)

More Optimal Approach

Utility Function
    Put a number on optimal

Real-time statistics
    (arrival rate and skill distribution)

Machine learning

Optimizer

Defining Optimal

Original image credit: Pixelflare ~Regret

Designers put weights on:

- Wait time
- Skill gap
- Equal win rates (by skill)
- Latency

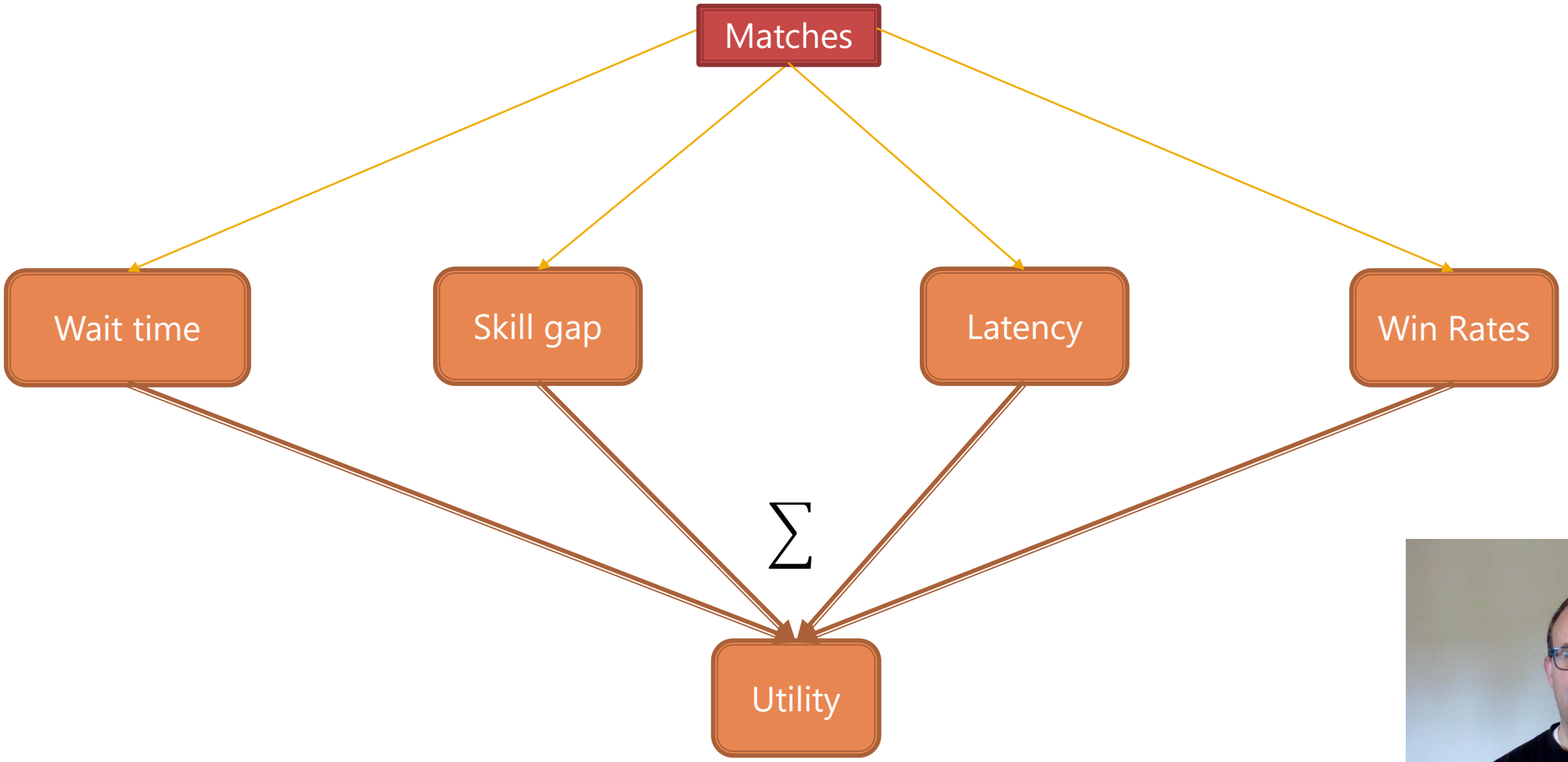
TrueMatch adapts in real time to optimize metrics and matches

High-level Comparison

	Conventional Matchmaking	TrueMatch
Tuning	Manual	Automatic
Tuning rate	Monthly (typical)	By minute
Granularity of tuning	Worldwide	By region and skill
Predictions	No	By region and skill
Monitoring, alerting	No	Yes

- Gears of War 5 launched with TrueMatch
- Halo 5 switched over to TrueMatch
- Potential to make it a standard service

Unified objective function



As an Equation

Always 1

$$Utility = w_1 wait + w_2 predictability + w_3 latency + w_4 winrate$$

How many seconds would you wait for 1ms bette

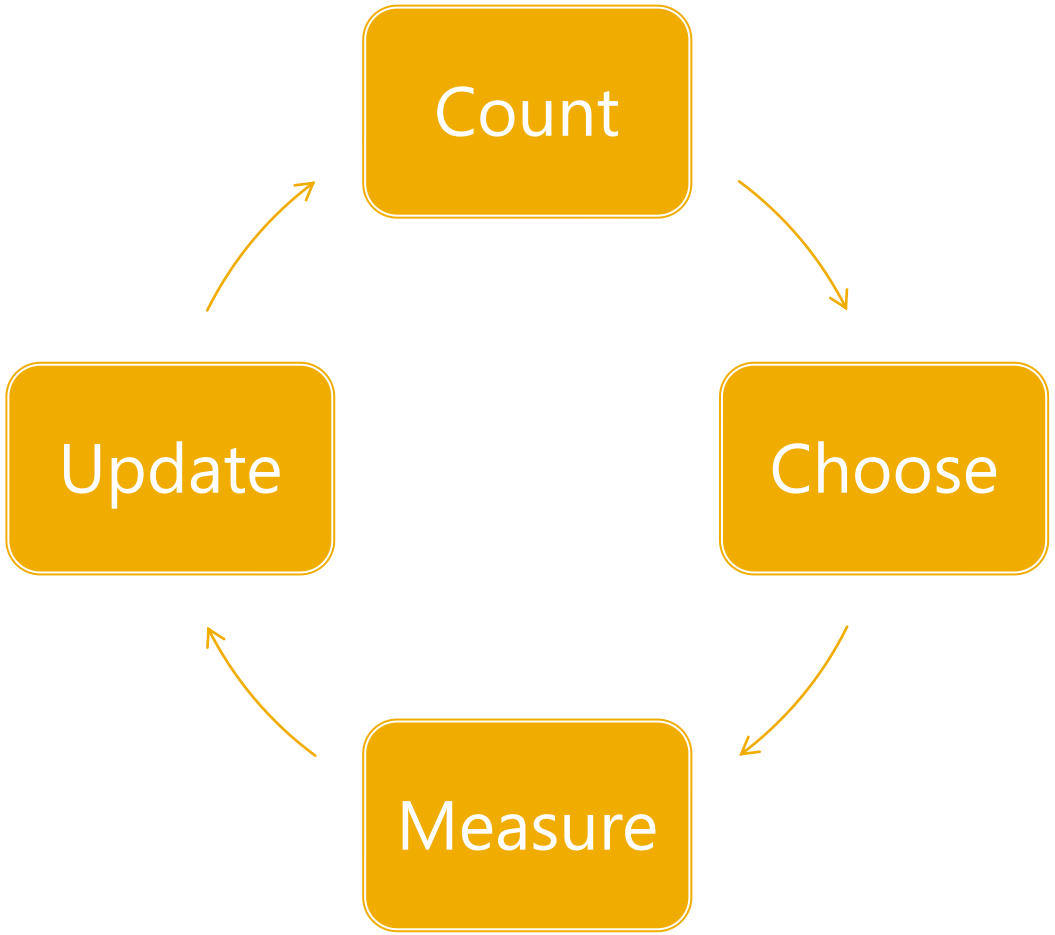
Utility Function Use

Each title (or even each *player*) can weigh metrics differently

Can evaluate any change to matchmaking

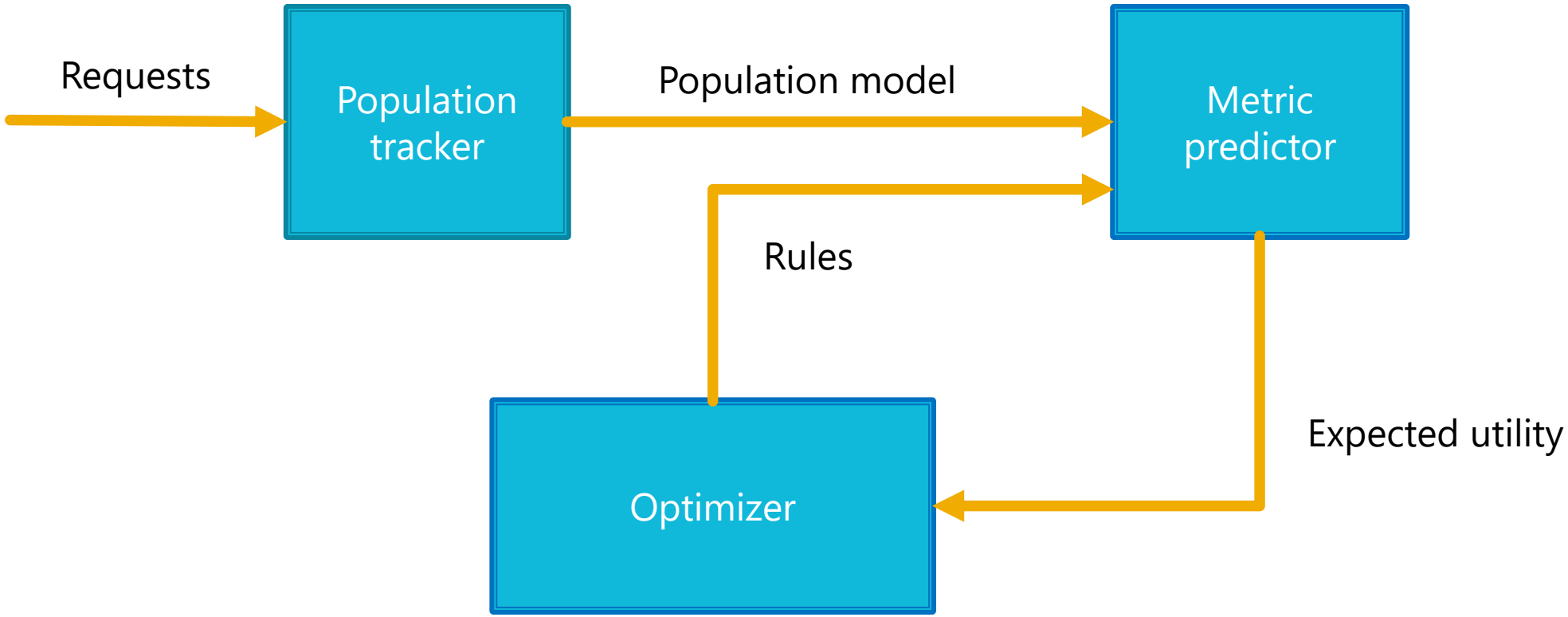
Can be reverse-engineered from existing threshold values

TrueMatch algorithm



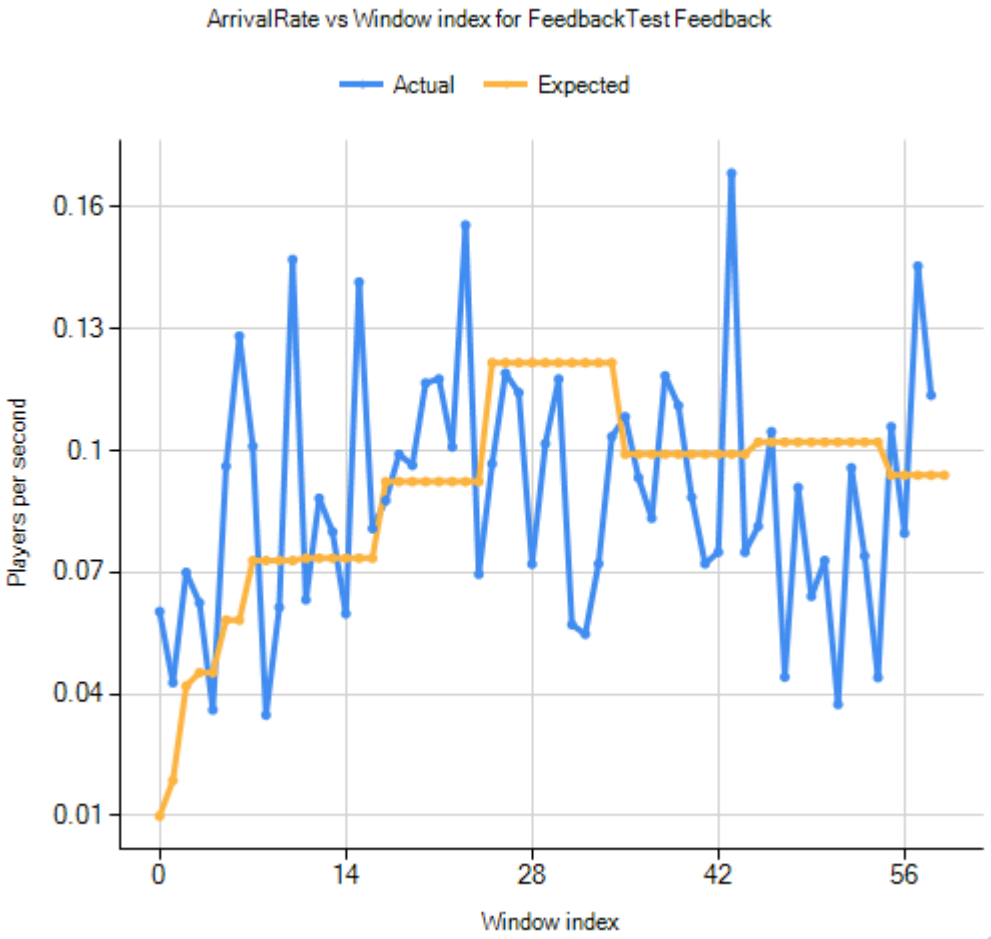
- **Count** requests by type
- **Choose** the best rules
- **Measure** results
- **Update** the model

TrueMatch components



Population tracker

- stream of requests
  -> population model
- Assumes slowly changing rate
- Simple: circular buffer
- Compact: e.g. histogram



Population model

Stores request rates

Given a request type, returns the rate of that type

(0.3 < Skill < 0.4 and Region=Brazil) => 0.01 requests per second

Metric Predictor

- Has free parameters tuned from feedback
- After rules are chosen, actual metrics are measured for 15 minutes and fed back to predictor

Metric Predictor

Has a formula to predict each metric.

Wait time, Latency, Predictability, etc.

Formulas have learned parameters that adapt over time

Simplest wait time formula

- matchable(t) = requests that can match with t

$$wait = \frac{\sum_t \left[rate(t) \frac{1}{2(rate(matchable(t)))} \right]}{\sum_t rate(t)}$$

Parameterized wait time formula

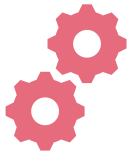
$$wait = C \frac{\sum_t rate(t) \frac{0.5}{rate(matchable(t))}}{\sum_t rate(t)} + B$$

Accounts for contention, cancellations, >2 players per match
(Expect C=N-1 for N-player match)

Buffering time

Optimizer: How does it make Rules?

WE pick FORM of the rules



Optimizer picks the values – tunes the parameters

In other words: Optimizer searches over parameter vectors

Algorithm options:

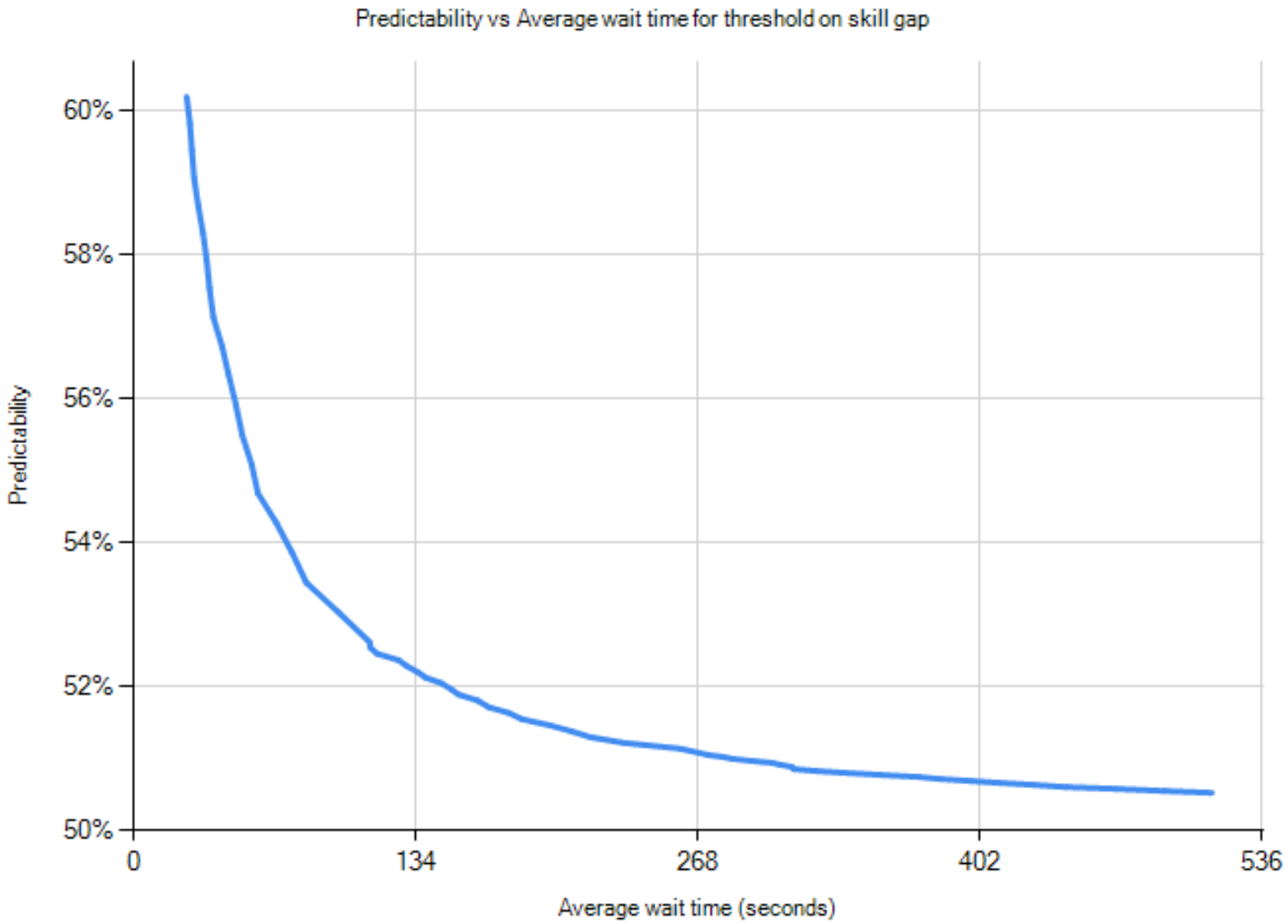
Gradient ascent

Branch and bound

Start with Current Rules

Skill Gap

| skill1 – skill2 | < (parameter)


Predictability vs Average wait time for threshold on skill gap
60%
58%
56%
54%
52%
50%
Predictability
0
134
268
402
536
Average wait time (seconds)

Find Optimal Transform

- Need to rewrite to search all curves
- Proved an equivalent, searchable, form is:
  $| f(skill1) - f(skill2) | < 1$
- E.g. $| skill1 - skill2 | < 0.5$ is $f(skill) = 2*skill$
- Nice improvement! Scale skill instead of gap

We found it!

Searched all fs (curves)

Found the best one!

Found one almost as good, but super simple to use!

Map the skills so Wait is Constant

- Instead of doing:
  | skill1 – skill2 | < (parameter)

- Map skills first:
  f(skill) = (parameter)*SkillPercentile

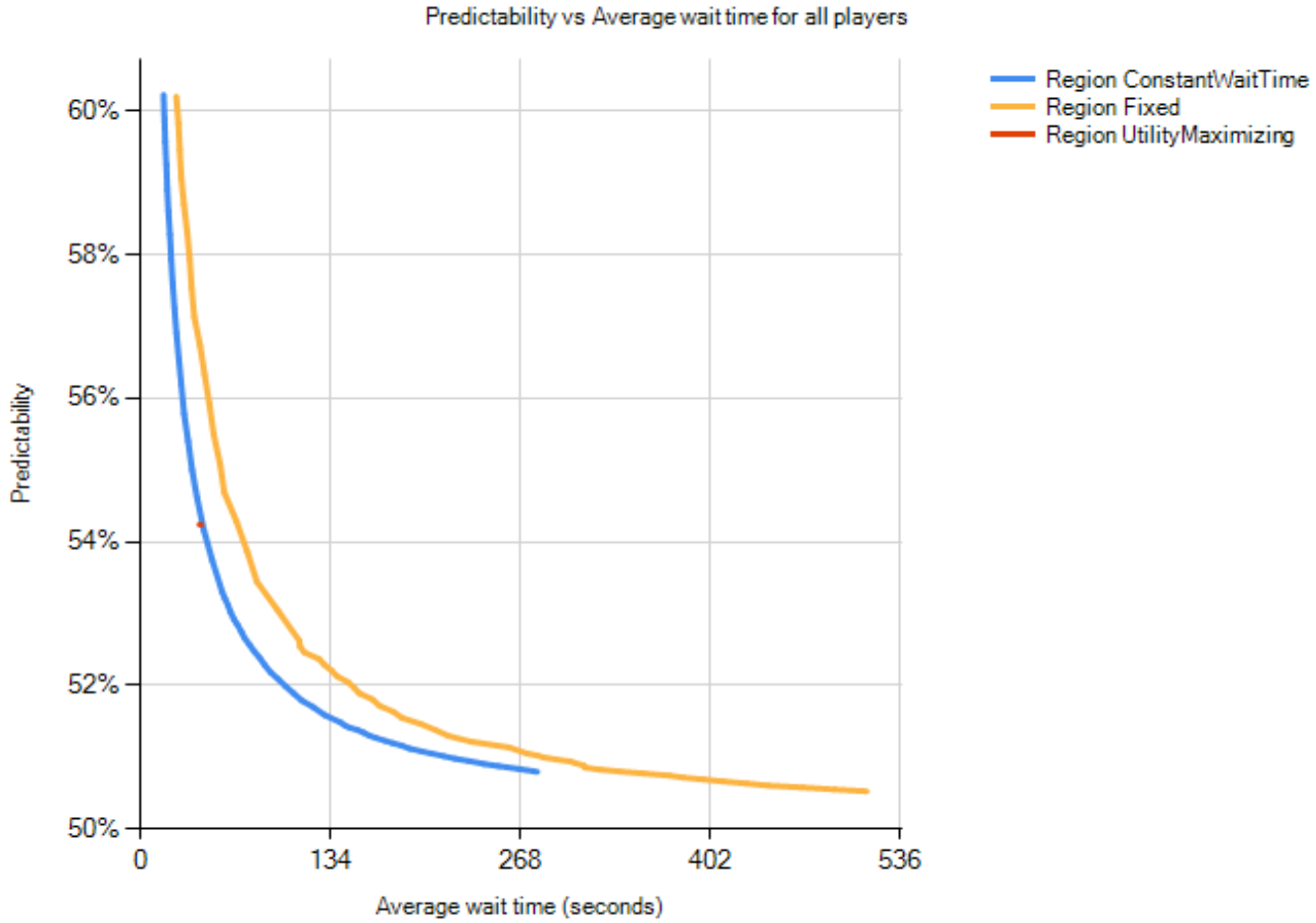
- And then have the rule enforce:
  | f(skill1) – f(skill2) | < 1

TrueMatch Rule Example

- Scale = 10, remember gap of 1 is OK

Player	Skill	Percentile	Scaled Skill
A	0.50	0.69	6.9
B	1.00	0.84	8.4
C	1.50	0.93	9.3

Mapped vs. Conventional



1v1 matches, simulated

TrueMatch Rules

Compare skill percentiles instead of skills

Everyone sees same size pool (x%)

Scale the percentiles as pop changes

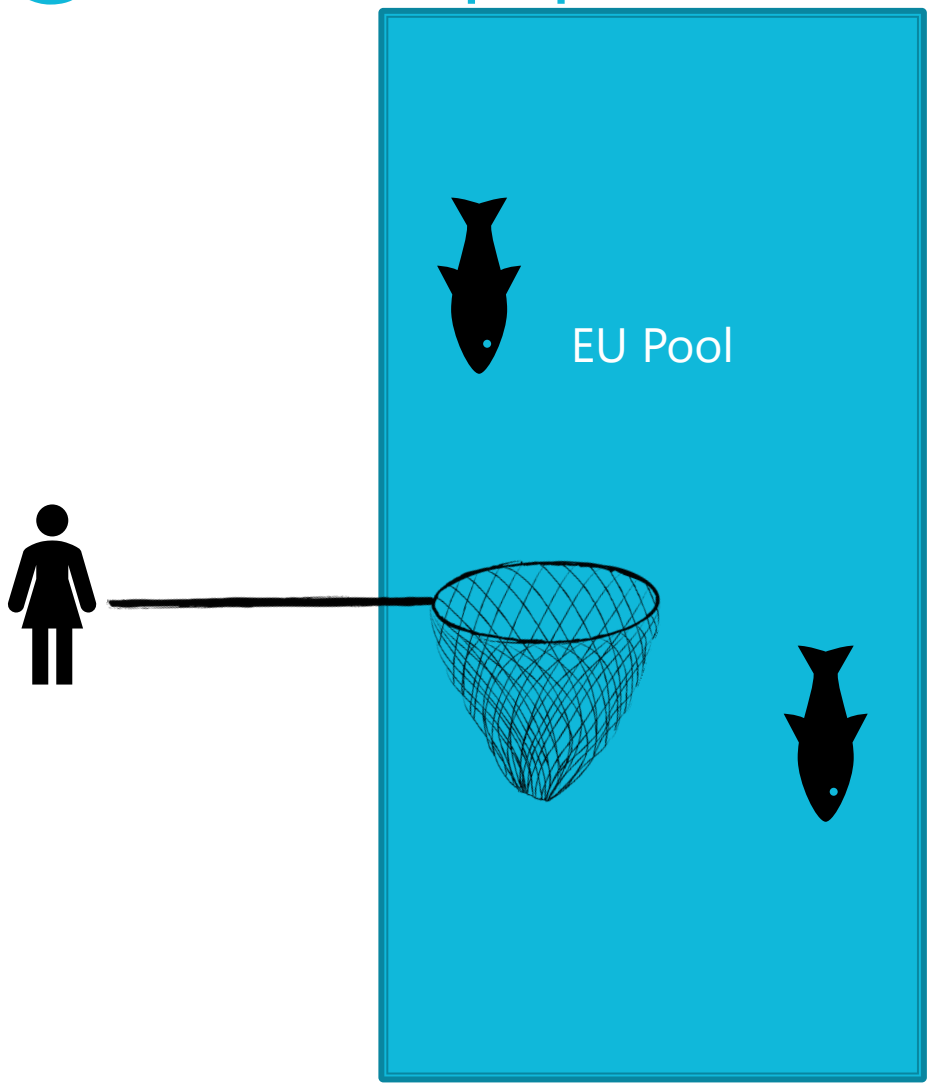
Doubly optimal! Optimal curve and Optimal Point

What about Regions?

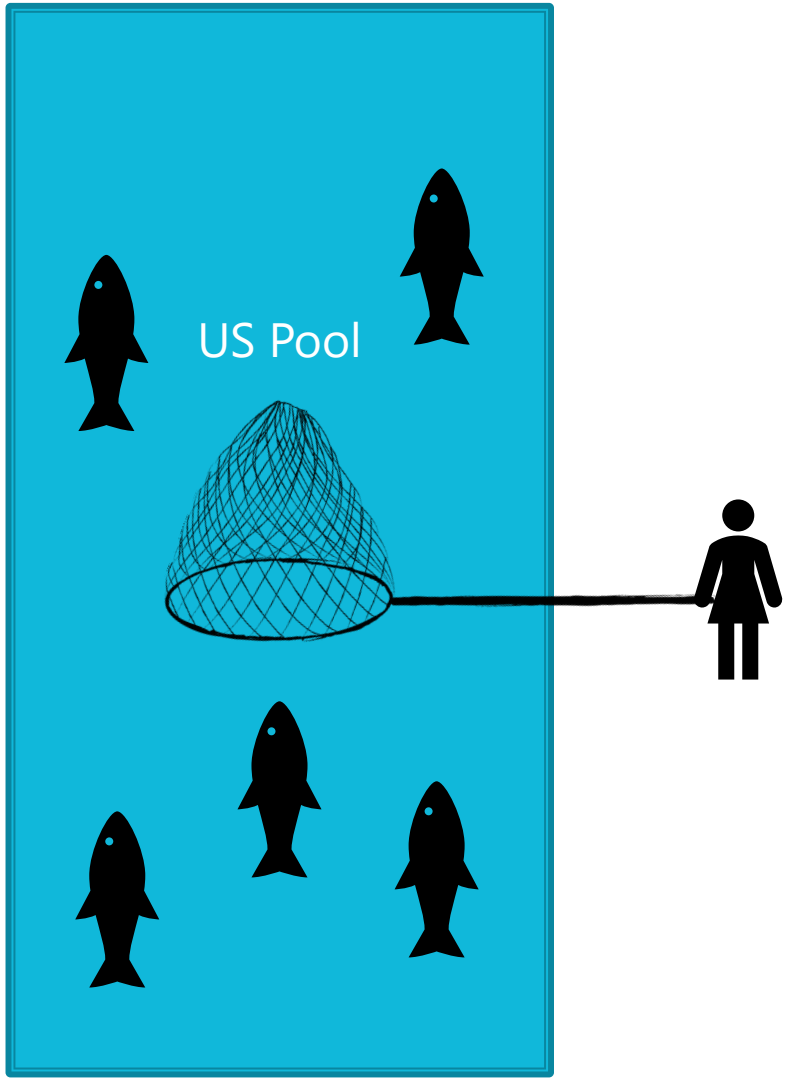
- Every pair has an optimized rule
- Let’s walk through what that does!

Conventional Region Approach

Try EU first. Times out after 5 wasted minutes.

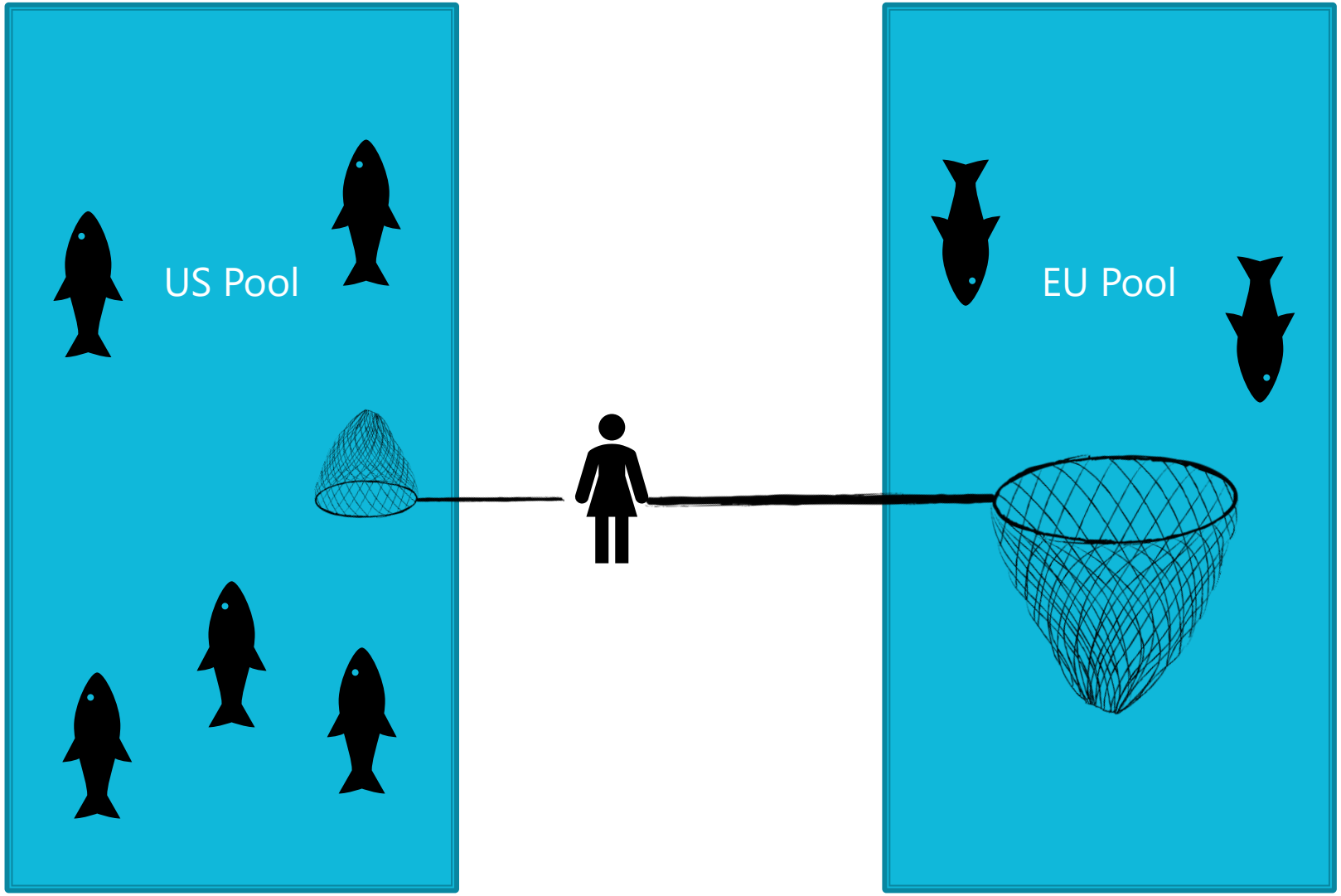


Conventional Region Approach



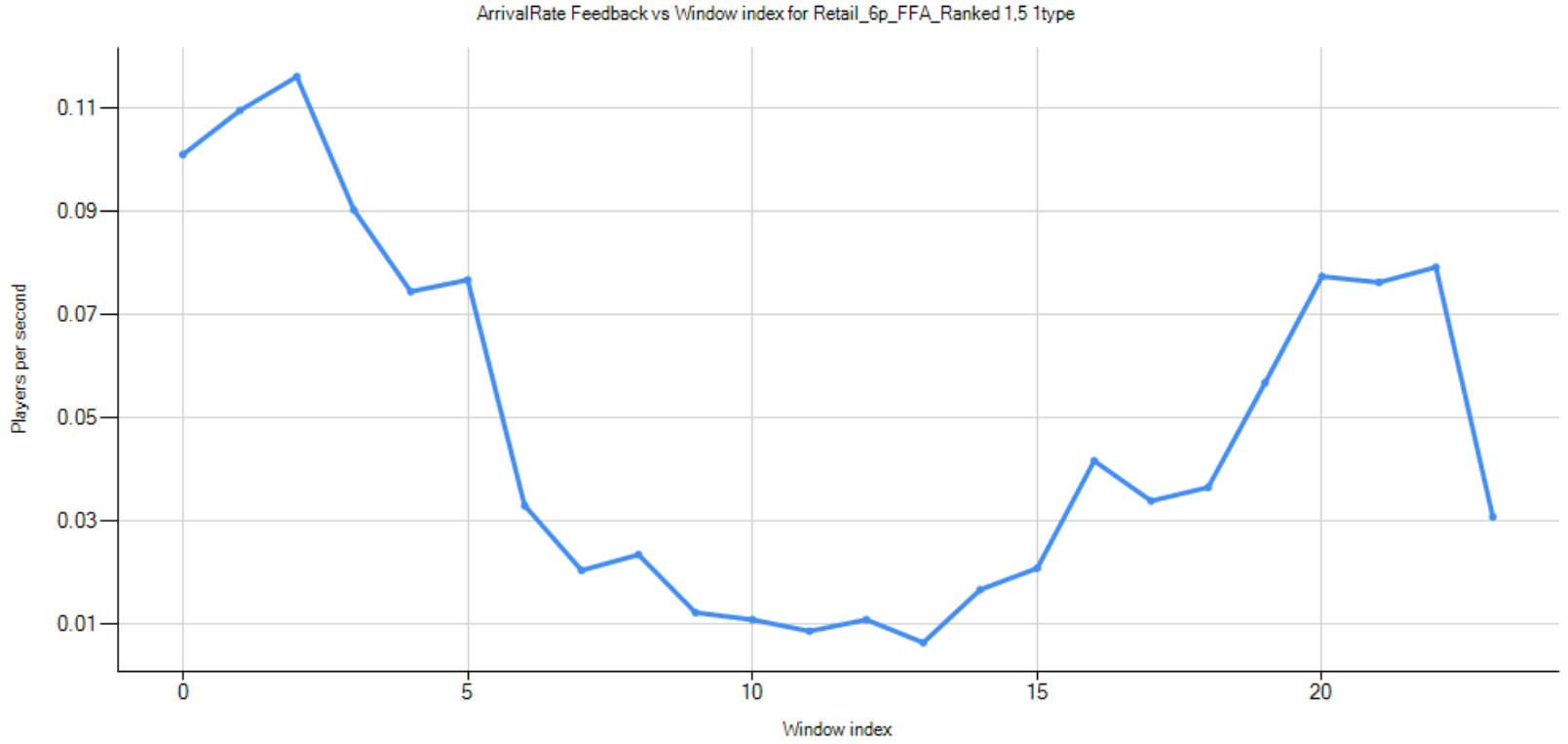
Settle for US.

TrueMatch Approach



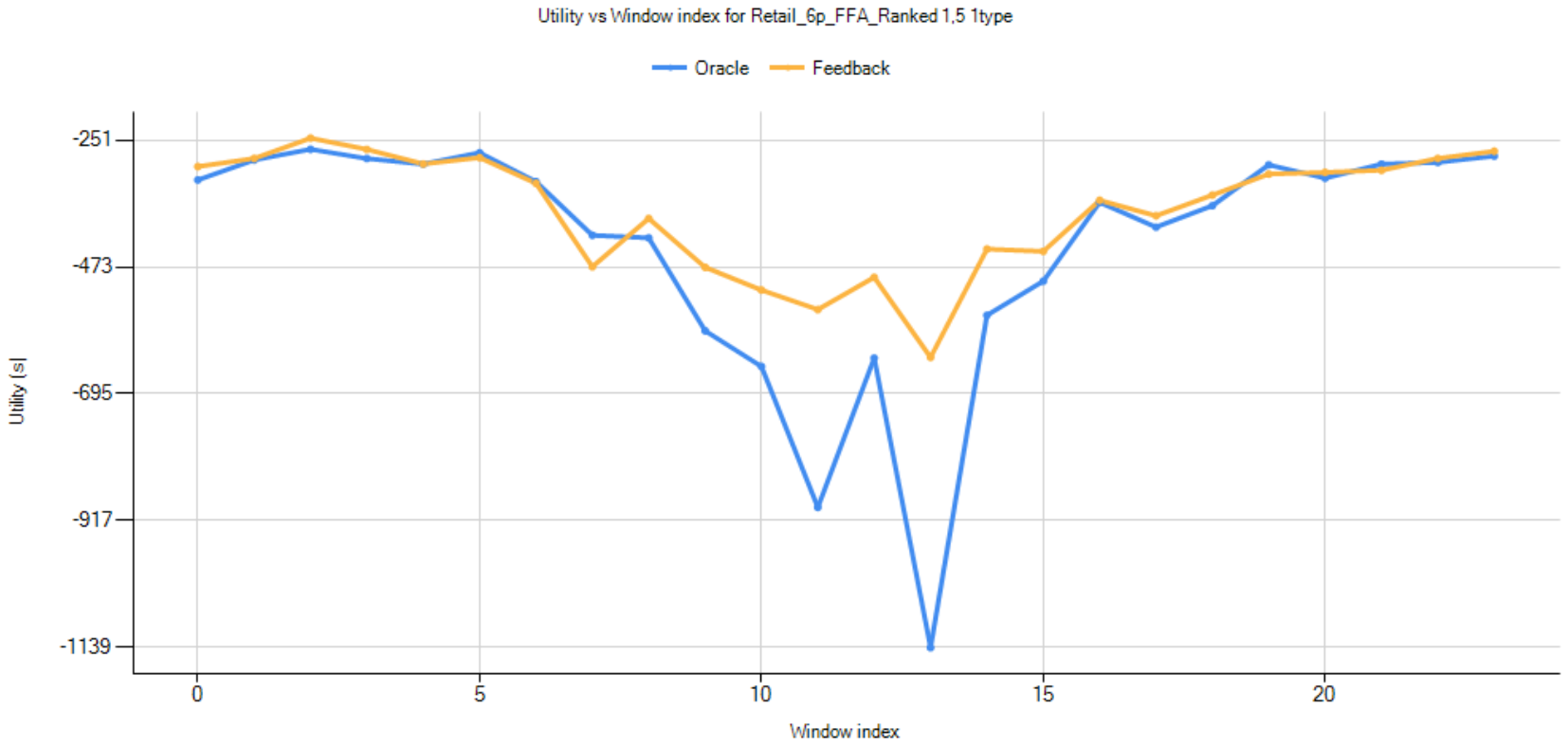
FFA example

- 6-player Free-for-all in Halo 5
- Request rate varies by 10x over a day

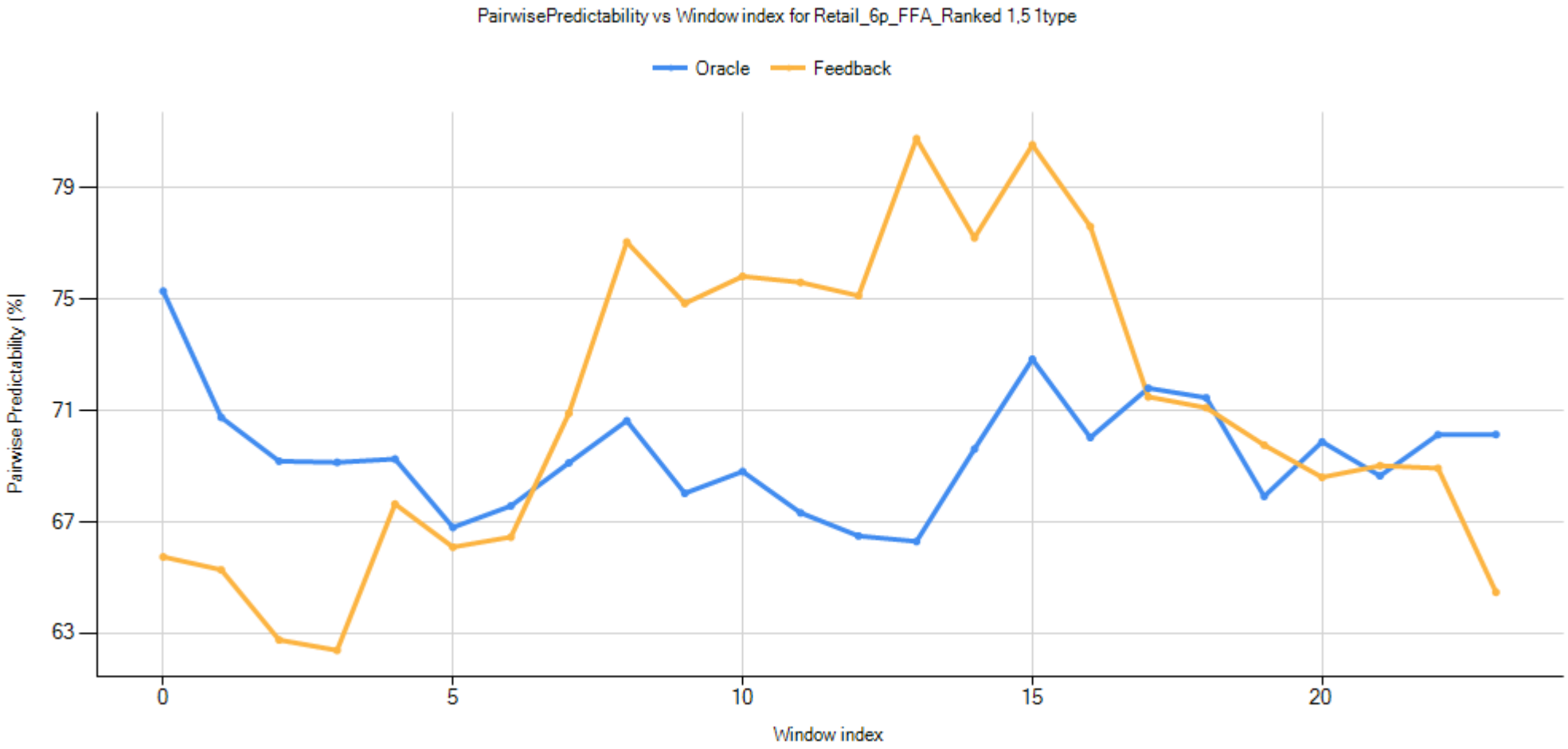


FFA example

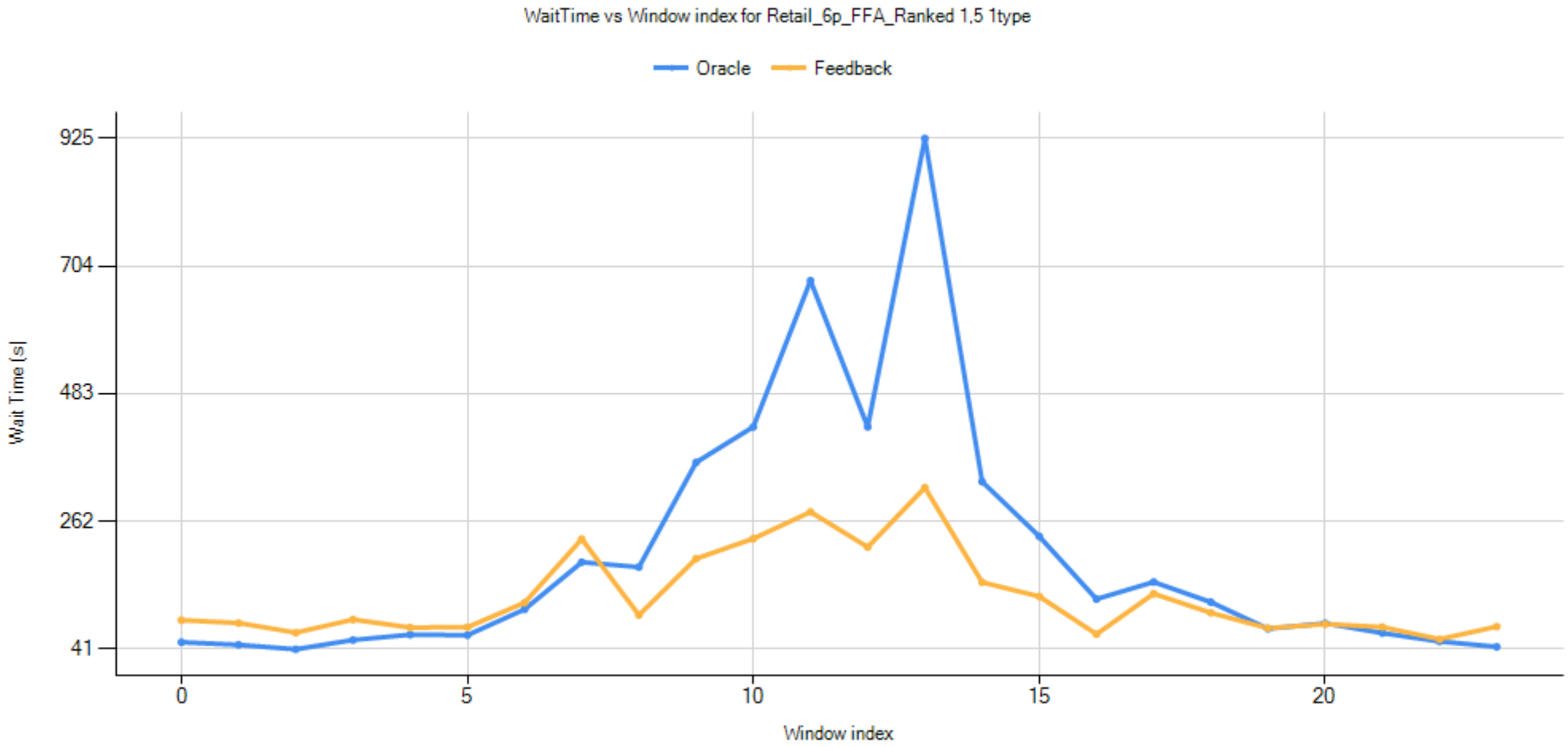
- **Oracle** = Best static rules in hindsight
- **Feedback** = Rules tuned online



Skill gap variation



Wait time variation



TrueMatch FFA Take-away

- Same Utility as Oracle during normal population hours
- Cuts negative utility in half during times with less players
- Drops wait time by 72% (10 min) during lower population hours
- Trades off 13% predictability (fairness) for 600 seconds of wait time (seems fine!)
- Matches design intent

TrueMatch Takeaway

- Utility Function defines wait trade-offs
- Metric Predictor can customize predicted wait times!
- Optimizer uses real-time statistics and feedback to create optimal rules.
- Results in real-time optimized matches

Simple Improvements

Matchmake on scaled Skill Percentiles

3-4 times per hour:

- Update Population Statistics
- Update percentile mappings (skill to %)
- Update Percentile Scale

Thank you! Questions?

Twitter:

@joshua_menke

Discord:

Zaedyn#4987