analysis

September 19, 2022

1 Hanabi game – Analysis of simulation results

1.1 Data import

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from itertools import product
```

[ ]:
```
path = "/home/nmontes/Documentos/hanabi-results/info-gain"

results = {}
results[True] = {}
results[False] = {}
player_configs = [2, 3, 4, 5]

for abd, n in product((True, False), player_configs):
    results_file = "{}/summary_{}_{}_players.csv".format(path, str(abd).
 ↪lower(), n)
    df = pd.read_csv(results_file, sep=';', index_col="seed")

    # add efficiency as ratio score : hint
    eff = df.apply(lambda x: x.score/x.hints if x.hints > 0 else 0, axis=1)
    df.insert(df.shape[1], "efficiency", eff)

    results[abd][n] = df
```

[ ]:
```
for abd, n in product((True, False), player_configs):
    print("{} players, abduction {}".format(n, abd))
    print(results[abd][n][["score", "efficiency"]].describe().loc[["mean",␣
 ↪"std"]])
    print()
```

```
2 players, abduction True
          score  efficiency
mean  18.606000    0.702015
std    5.918854    0.232094
```

```
3 players, abduction True
           score  efficiency
mean  17.972000    0.700878
std    1.938745    0.101515

4 players, abduction True
           score  efficiency
mean  16.502000    0.641741
std    1.608468    0.092351

5 players, abduction True
          score  efficiency
mean  14.42200    0.615392
std    1.36955    0.086177

2 players, abduction False
           score  efficiency
mean  14.568000    0.457975
std    2.930281    0.103422

3 players, abduction False
           score  efficiency
mean  12.524000    0.419426
std    1.557651    0.070456

4 players, abduction False
           score  efficiency
mean  11.232000    0.375438
std    1.357151    0.060059

5 players, abduction False
          score  efficiency
mean  9.232000    0.331239
std   1.296742    0.060487
```

1.2 Summary plots

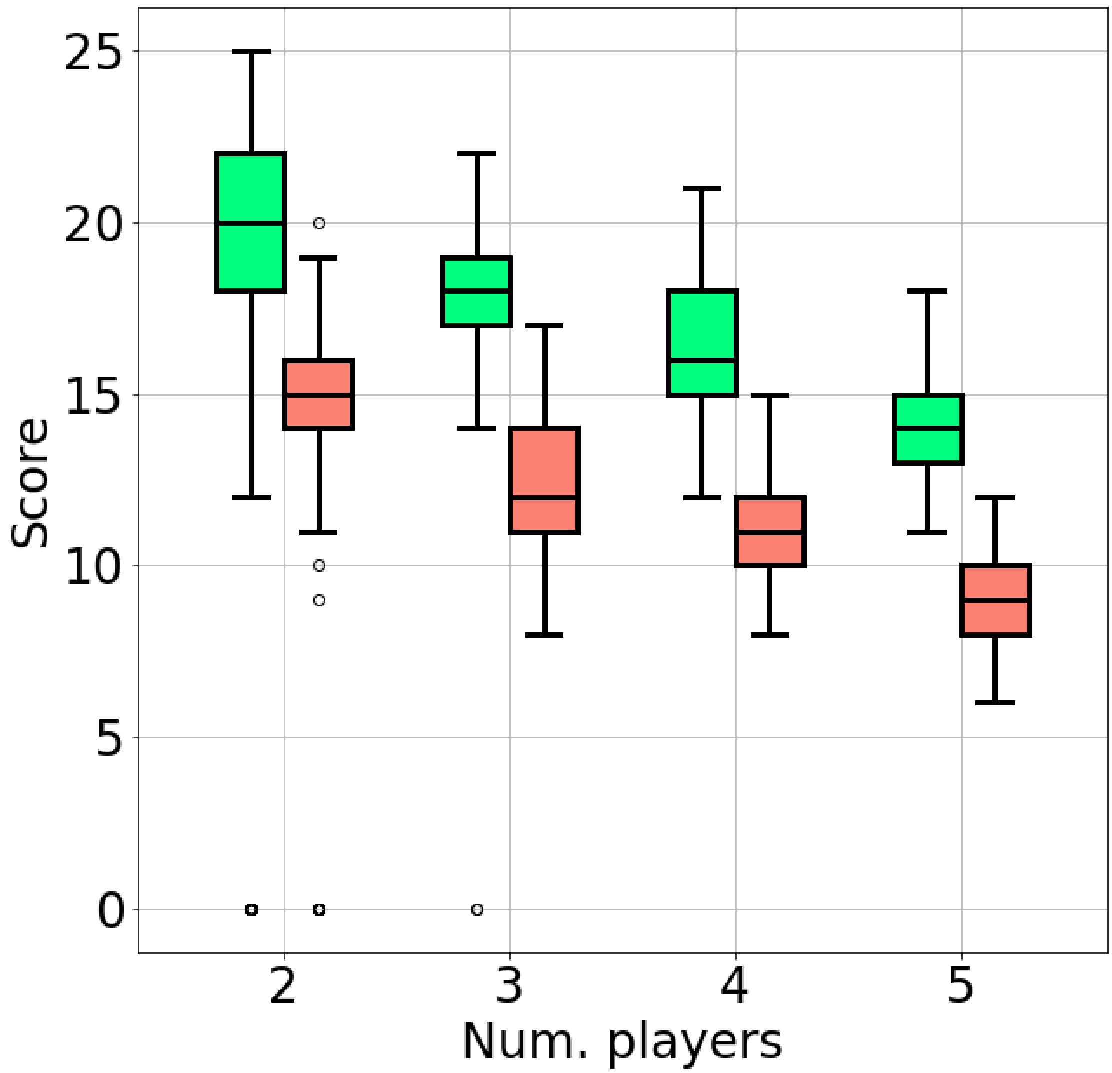
[ ]:
```
plt.rcParams.update({'font.size': 28})
fig, ax = plt.subplots(figsize=(10, 10), facecolor='white')

epsilon = 0.15
lw = 3
colors = {True: 'springgreen', False: 'salmon'}

for n in player_configs:
```

```
    for abd in (True, False):
        if abd:
            x = n-epsilon
        else:
            x = n+epsilon
        data = results[abd][n]["score"].to_numpy()
        bp = ax.boxplot(
            data,
            positions=[x],
            widths=[epsilon*2],
            boxprops={'linewidth': lw},
            medianprops={'linewidth': lw, 'color': 'black'},
            capprops={'linewidth': lw},
            whiskerprops={'linewidth': lw},
            patch_artist=True
        )
        for patch in bp['boxes']:
            patch.set(facecolor=colors[abd])

ax.set_xticks(player_configs, labels=player_configs)
ax.grid()
ax.set_xlabel("Num. players")
ax.set_ylabel("Score")
# plt.savefig("score.png")
plt.show()
```

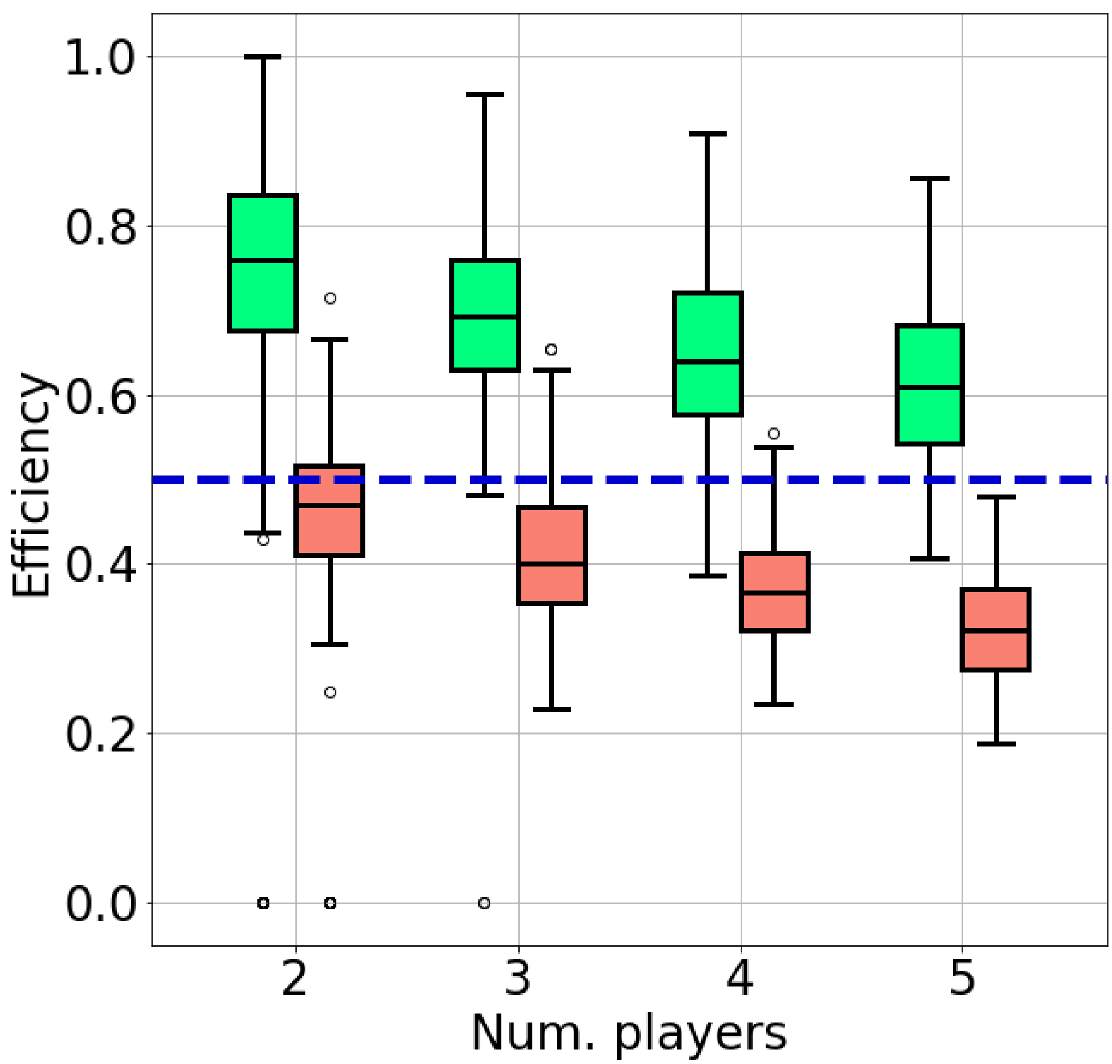



```
[ ]: fig, ax = plt.subplots(figsize=(10, 10), facecolor='white')

     for n in player_configs:
         for abd in (True, False):
             if abd:
                 x = n-epsilon
             else:
                 x = n+epsilon
             data = results[abd][n]["efficiency"].to_numpy()
             bp = ax.boxplot(
                 data,
                 positions=[x],
                 widths=[epsilon*2],
                 boxprops={'linewidth': lw},
```

```
            medianprops={'linewidth': lw, 'color': 'black'},
            capprops={'linewidth': lw},
            whiskerprops={'linewidth': lw},
            patch_artist=True
        )
        for patch in bp['boxes']:
            patch.set(facecolor=colors[abd])

ax.set_xticks(player_configs, labels=player_configs)
ax.grid()
ax.set_xlabel("Num. players")
ax.set_ylabel("Efficiency")
ax.axhline(0.5, linewidth=5, color='mediumblue', linestyle='--')
# plt.savefig("efficiency.png")
plt.show()
```



1.3 Average information per hint

[ ]:
```python
for abd, n in product((True, False), player_configs):
    total_explicit_info = {}
    avg_explicit_info = {}
    if abd:
        total_implicit_info = {}
        avg_implicit_info = {}
        avg_explicit_distance = {}
        avg_implicit_distance = {}

    for seed in range(500):
        results_file = "{}/info_gain_{}_{}_{}.csv".format(path, str(abd).
↪lower(), n, seed)
        df = pd.read_csv(results_file, sep=';')

        # compute information gain (relative to the pre-action distribution)
        # per move, aka the sum across all slots
        explicit_info_all_slots = df.groupby(["move"])["explicit_info"].sum().
↪replace(0, np.NaN)
        total_explicit_info[seed] = explicit_info_all_slots.sum()
        avg_explicit_info[seed] = explicit_info_all_slots.mean()
        if abd:
            implicit_info_all_slots = df.groupby(["move"])["implicit_info"].
↪sum().replace(0, np.NaN)
            total_implicit_info[seed] = implicit_info_all_slots.sum()
            avg_implicit_info[seed] = implicit_info_all_slots.mean()

    results[abd][n]["total_explicit_info"] = pd.Series(total_explicit_info)
    results[abd][n]["avg_explicit_info"] = pd.Series(avg_explicit_info)
    if abd:
        results[abd][n]["total_implicit_info"] = pd.Series(total_implicit_info)
        results[abd][n]["avg_implicit_info"] = pd.Series(avg_implicit_info)
```

1.4 Score rate that can be assigned to explicit and implicit information

[ ]:
```python
for n in player_configs:
    score = {}
    explicit_info = {}

    score_rate = {}

    results[False][n]["score_rate"] = results[False][n]["score"] /␣
↪results[False][n]["total_explicit_info"]
    score_rate['explicit'] = results[False][n]["score_rate"].mean()
```

```
    results[True][n]["score_by_expl_info"] =␣
↪results[True][n]["total_explicit_info"]*score_rate['explicit']
    results[True][n]["residual_score"] = results[True][n]["score"] -␣
↪results[True][n]["score_by_expl_info"]

    results[True][n]["impl_score_rate"] = results[True][n]["residual_score"] /␣
↪results[True][n]["total_implicit_info"]
    score_rate['implicit'] = results[True][n]["impl_score_rate"].replace([np.
↪inf, -np.inf], np.nan).mean()

    print("{} players:".format(n))
    print("Explicit score rate: {:.2f}".format(score_rate['explicit']))
    print("Implicit score rate: {:.2f}".format(score_rate['implicit']))
    print("Ratio of implicit to explicit score rate: {:.1f}\n".
↪format(score_rate['implicit']/score_rate['explicit']))
```

```
2 players:
Explicit score rate: 0.70
Implicit score rate: 1.18
Ratio of implicit to explicit score rate: 1.7

3 players:
Explicit score rate: 0.60
Implicit score rate: 1.50
Ratio of implicit to explicit score rate: 2.5

4 players:
Explicit score rate: 0.61
Implicit score rate: 1.32
Ratio of implicit to explicit score rate: 2.2

5 players:
Explicit score rate: 0.52
Implicit score rate: 1.42
Ratio of implicit to explicit score rate: 2.7
```

1.5 Statistical tests

Perform statistical tests to compare the performance metrics (score and efficiency) between the case when abduction is *off* and *on*.

[ ]:
```
plt.rcParams.update({'font.size': 24})

def plot_histogram(n, variable):
    fig, ax = plt.subplots(figsize=(8, 7.5), facecolor="white")
```

```python
    if variable == "score":
        bins = range(0, 26)
    elif variable == "efficiency":
        bins = np.linspace(0, 1, 25)
    for abd in (True, False):
        x = results[abd][n][variable]
        ax.hist(x, bins, fc=colors[abd], ec="black")
    locs, labels = plt.yticks()
    labs = ["{:.2f}".format(i/500) for i in locs]
    ax.set_yticks(locs, labels=labs)
    if variable == "score":
        ax.set_xticks([0, 5, 10, 15, 20, 25])
    elif variable == "efficiency":
        ax.set_xticks([0., 0.25, 0.5, 0.75, 1.])
    ax.set_axisbelow(True)
    ax.grid()
    ax.set_xlabel(variable.capitalize())
    ax.set_ylabel("Frequency")
    # plt.savefig("hist_{}_{}.png".format(n, variable))
    plt.show()
```

[ ]:
```python
from scipy import stats

# Step 1: Check that the variables are normally distributed
variables = ["score", "efficiency"]
for n, abd, var in product(player_configs, (True, False), variables):
    x = results[abd][n][var]
    _, pvalue = stats.shapiro(x)
    print("{} players, abduction {}, {}: {:.2e}".format(n, abd, var, pvalue))
```

```
2 players, abduction True, score: 1.16e-31
2 players, abduction True, efficiency: 1.62e-28
2 players, abduction False, score: 5.68e-29
2 players, abduction False, efficiency: 7.97e-23
3 players, abduction True, score: 7.00e-25
3 players, abduction True, efficiency: 5.36e-15
3 players, abduction False, score: 4.60e-10
3 players, abduction False, efficiency: 2.84e-06
4 players, abduction True, score: 5.66e-10
4 players, abduction True, efficiency: 1.38e-02
4 players, abduction False, score: 6.47e-12
4 players, abduction False, efficiency: 1.18e-07
5 players, abduction True, score: 1.41e-11
5 players, abduction True, efficiency: 1.50e-04
5 players, abduction False, score: 3.37e-13
5 players, abduction False, efficiency: 2.36e-10
```

```
[ ]: # Step 2: Pair-wise t-test
for n, var in product(player_configs, variables):
    x1 = results[True][n][var]
    x2 = results[False][n][var]
    _, pvalue = stats.ttest_rel(x1, x2)
    print("{} players, {}: {:.2e}".format(n, var, pvalue))
```

```
2 players, score: 2.62e-35
2 players, efficiency: 9.75e-72
3 players, score: 1.97e-194
3 players, efficiency: 6.26e-201
4 players, score: 1.09e-223
4 players, efficiency: 2.56e-217
5 players, score: 4.45e-232
5 players, efficiency: 3.80e-230
```

1.6 Compare distances to the *true* values

For every hint, find the *focus* slot (if there is one). Then, compare the distance of the probability distribution at the focus slot between the **post-action** distribution and the **post-explanation**

```
[ ]: distance_red = {p: [] for p in player_configs}

for n, seed in product(player_configs, range(500)):
    results_file = "{}/info_gain_true_{}_{}.csv".format(path, n, seed)
    df = pd.read_csv(results_file, sep=';')
    df["delta"] = df["distance_explicit"] - df["distance_implicit"]
    df_focus = df.loc[df["delta"] > 0.]
    percentage_reduction = (df_focus["distance_explicit"] -␣
 ↪df_focus["distance_implicit"]) / df_focus["distance_explicit"] * 100
    distance_red[n].append(percentage_reduction.mean())
```

```
[ ]: print("Percentage gain in distance to the ground truth:")
for p in player_configs:
    print("{} players: {:.2f}%".format(p, np.nanmean(distance_red[p])))
```

```
Percentage gain in distance to the ground truth:
2 players: 85.33%
3 players: 88.29%
4 players: 89.43%
5 players: 91.49%
```