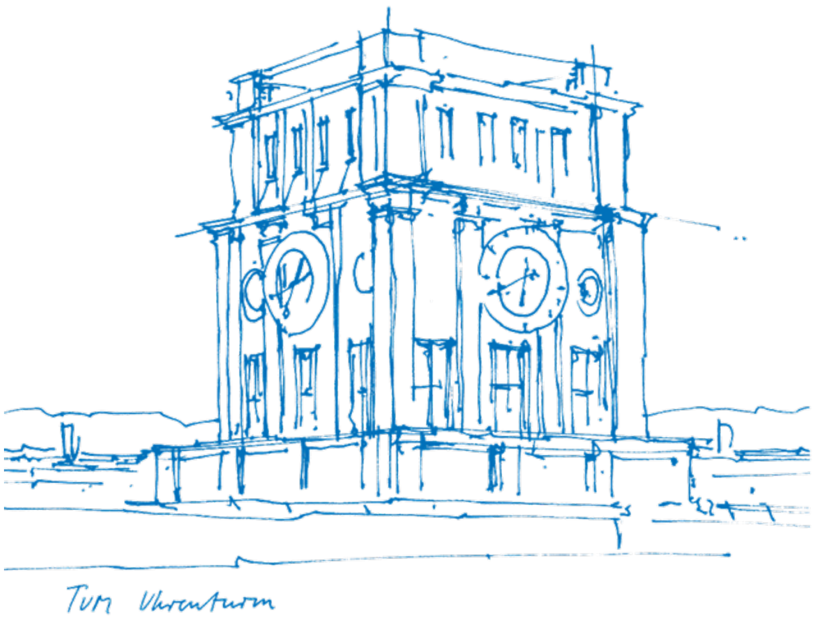
@Chair of Algorithms and Complexity
@Department of Informatics
Technical University of Munich

Development and Analysis of Strategies for the Card Game “The Game”

Bachelor-Kolloquium

Felix Dietrich

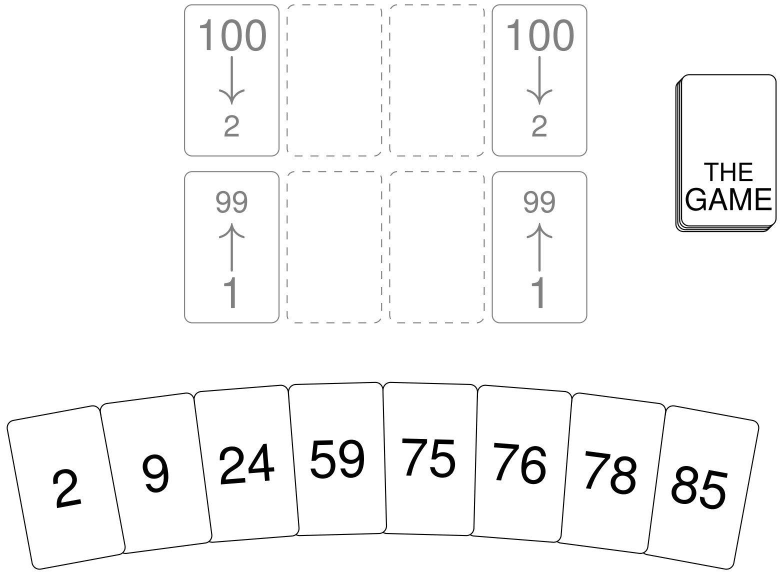
October 10, 2019



@Chair of Algorithms and Complexity
@Department of Informatics
Technical University of Munich

The Game (by Steffen Benndorf)

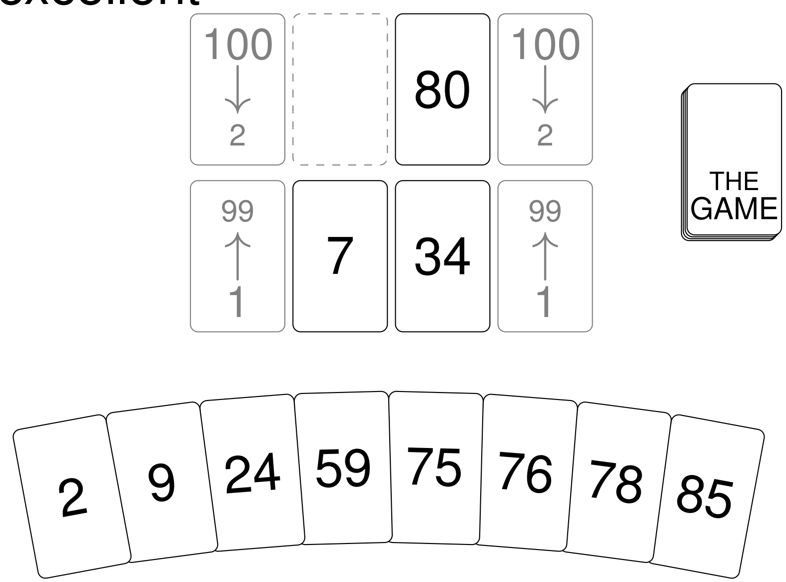
Initial deck: $D = \{2, 3, \ldots, 99\}$
Hand cards: 8
Take turns until no cards can be played any more
Backwards trick ± 10
Play at least 2 cards a turn before drawing
Goal: Lay all cards (difficult)
$\Rightarrow$ Results < 10 are excellent



@Chair of Algorithms and Complexity
@Department of Informatics
Technical University of Munich

The Game (by Steffen Benndorf)

Initial deck: $D = \{2, 3, \ldots, 99\}$
Hand cards: 8
Take turns until no cards can be played any more
Backwards trick ± 10
Play at least 2 cards a turn before drawing
Goal: Lay all cards (difficult)
$\Rightarrow$ Results < 10 are excellent



Motivation

- Simple game but seems difficult to win (no obvious tactics).
- What is the best or a good strategy for “The Game”?
- How do these strategies perform?

$\Rightarrow$ **General question:** How to win “The Game”?

Approach Used

- **Positional evaluation** similar to chess.
- **Idea:** Find the best valued position out of all positions in the game tree.
- Strategy is then defined by a **value function**.
- My game tree covers all positions at depth 2 (No special search algorithm is used).
- The win rate of a strategy is then approximated by simulating a sample of games, instead of theoretically calculating it (difficult to do).

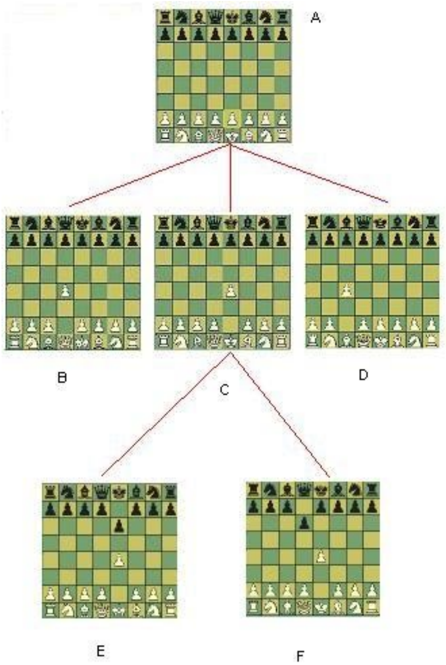


Image source: `https://www.sites.google.com/site/qgchess/chess-algorithms`

Strategy (Positional Evaluation)

Full sample: $X = (X_1, \dots, X_n)$

Procedure 1 Simulate a random game

Input: $v : S \to \mathbb{R}$

Output: Sample of cards remaining at the end of the game X_i

$s \leftarrow$ choose uniformly at random from S_{init}

while $T_{t_{min}}(s) \neq \emptyset$ **do**

$s \leftarrow \mathrm{argmax}_{s' \in T_{t_{min}}(s)}\{v(s')\}$

$s \leftarrow$ refill hand from deck for s

end while

$s \leftarrow$ play longest possible move sequence for s

return $|C \setminus L(s)|$

$$m(X) = \frac{1}{n} \sum_{i \in [n]} X_i \qquad w(X) = \frac{1}{n} |\{i \in [n] \mid X_i = 0\}|$$

@Chair of Algorithms and Complexity
@Department of Informatics
Technical University of Munich

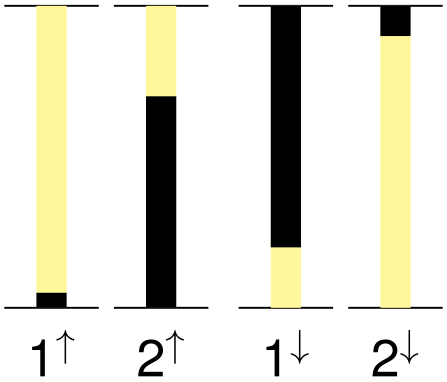
Perceived Capacity

Perceived capacity function:

$$c_p(s,p) = \begin{cases} c^{\uparrow} - p_c(s,p), & p \in P^{\uparrow} \\ p_c(s,p) - c^{\downarrow}, & p \in P^{\downarrow}. \end{cases}$$

Value function:

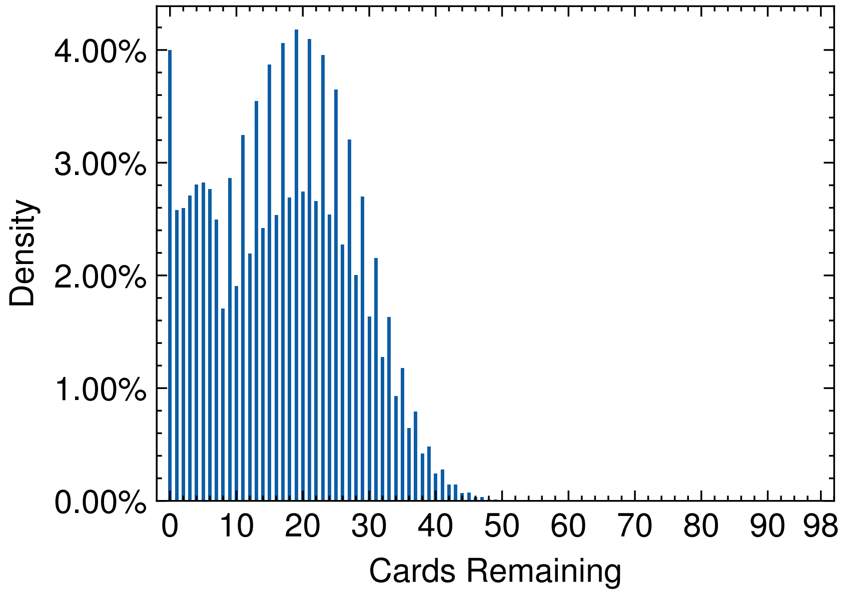
$$v(s) = \sum_{p \in P} c_p(s,p),$$



@Chair of Algorithms and Complexity
@Department of Informatics
Technical University of Munich

Perceived Capacity - Results

Sample size $n = 1\,000\,000$.
Games won: 4.0%
Cards remaining: 17.1
$\Rightarrow$ Very simple strategy already relatively high chance of winning



@Chair of Algorithms and Complexity
@Department of Informatics
Technical University of Munich

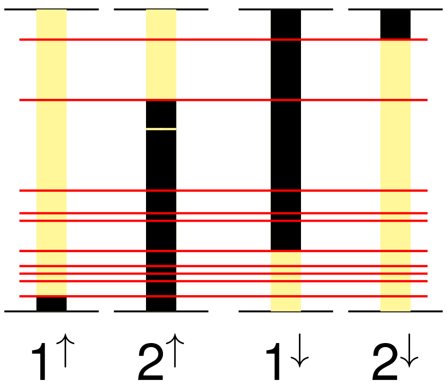
Real Capacity

Real capacity function:

$$c_r(s, p) = |\{c \in C \setminus L(s) \mid playable(s, c, p)\}|,$$

Value function:

$$v(s) = \sum_{p \in P} c_r(s, p),$$

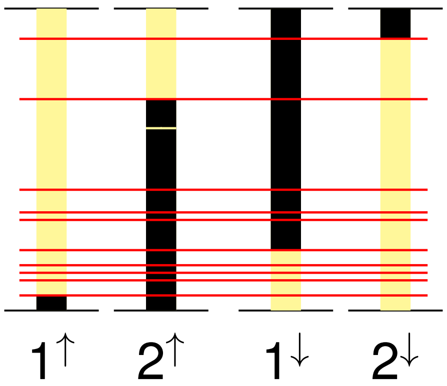


@Chair of Algorithms and Complexity
@Department of Informatics
Technical University of Munich

Real Capacity - Results

Sample size $n = 1\,000\,000$.
Games won: 3.4% (vs. perceived capacity 4.0%)
Cards remaining: 17.9 (vs. perceived capacity 17.1)

Why is this worse than the perceived capacity?
$\Rightarrow$ Probably because of skipping over important cards.



Weighting Capacities

Idea: Change growth rate of the piles by weighting the capacities differently.

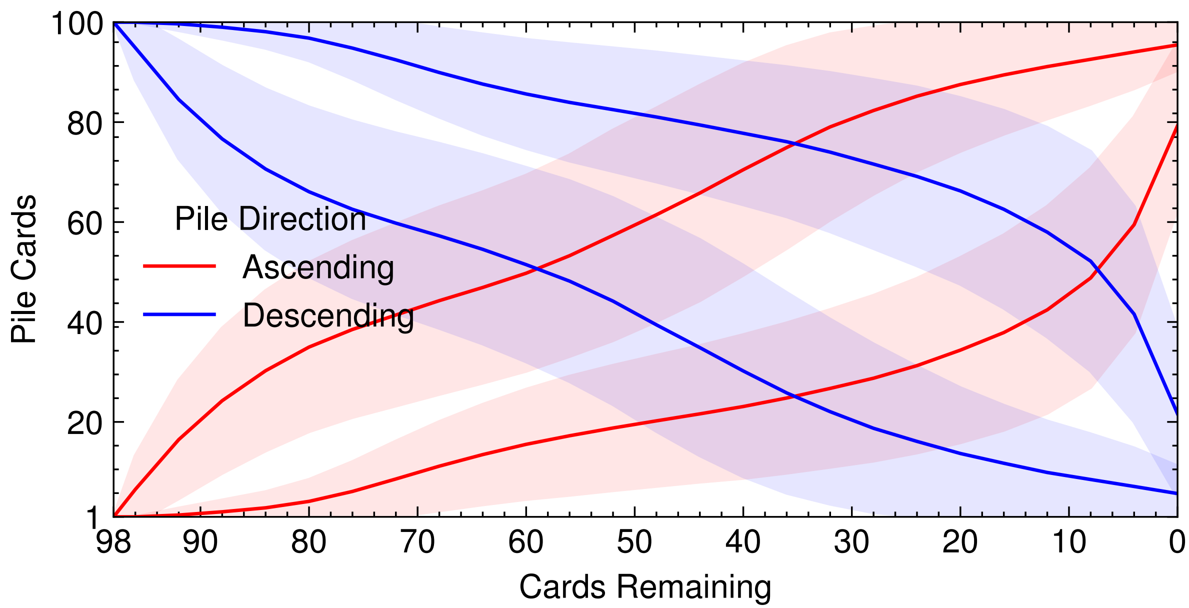


Figure: Average course of the pile cards for won games using the perceived capacity strategy.

Weighting Capacities

Weights $\omega : P \to \mathbb{R}^+$ multiplied with the capacities.

Value function:

$$v(s) = \sum_{p \in P} \omega(p) \cdot c(s, p),$$

with either $c = c_p$ or $c = c_r$.

@Chair of Algorithms and Complexity
@Department of Informatics
Technical University of Munich

Weighting Capacities - Results

Sample size $n = 1\,000\,000$.

Weighted perceived capacity:

Good weights: $\omega(1^\uparrow) = 0.675 = \omega(1^\downarrow), \omega(2^\uparrow) = 1 = \omega(2^\downarrow)$

Games won: 5.2%

Cards remaining: 15.4

Weighted real capacity:

Good weights: $\omega(1^\uparrow) = 0.6 = \omega(1^\downarrow), \omega(2^\uparrow) = 1 = \omega(2^\downarrow)$

Games won: 6.8%

Cards remaining: 14.9

Why does weighting increase the chance of winning?
$\Rightarrow$ Better distribution of cards might lead to a lower risk of large steps.
Why is the real capacity suddenly superior?
$\Rightarrow$ Better distribution of gaps between the cards
$\Rightarrow$ Better decisions possible when skipping cards

Penalize Playability

Why are large steps bad? Why is skipping over important cards bad?
$\Rightarrow$ Playability of remaining cards decreases.

Penalty function:

$$f : \{0, 1, \dots, p_\uparrow + p_\downarrow\} \to \mathbb{R}_0^+$$

with $f(p_\#(s, c))$ as penalty for a card c and
$p_\#(s, c) = |\{p \in P \mid \mathit{playable}(s, c, p)\}|$.

Value function:

$$v(s) = -\sum_{c \in C \setminus L(s)} f(p_\#(s', c)),$$

Note that this value function is negative to minimize the penalty.

Penalize Playability - Results

Good penalty function: $f(x) = e^{-1.5x}$
Games won: 9.4%
Cards remaining: 13.2

Value function can give up on a pile for a card as opposed to keeping as many cards playable on as many piles as possible in the capacity strategies.

Penalty with Recoverability

Simplified penalty function:

$$\pi : S \times P \times C \rightarrow [1, \ldots \infty)$$

Value function:

$$v(s) = -\sum_{c \in C \setminus L(s)} \prod_{p \in P} \pi(s, p, c),$$

Example penalty function:

$$\pi(s, c, p) = \begin{cases} 1 & \text{if } \mathit{playable}(s, c, p) \\ \alpha & \text{otw.} \end{cases}$$

Equivalent win rate and cards remaining with $\alpha = 3.5$ to other penalty function.

Recovery Using Distance

Penalty function:

$$\pi(s, c, p) = \begin{cases} 1 & \text{if } \mathit{playable}(s, c, p) \\ \alpha - \beta \cdot e^{\gamma \cdot (|p_c(s,p) - c| - 1)} & \text{otw.,} \end{cases}$$

Results:

With $\alpha = 3.5$, $\beta = 1$, and $\gamma = 0.03$ (there are surely better choices).
Win rate: 12.7% (9.4% without recovery term)
Cards remaining: 11.2 (13.2 without recovery term)

Recoverability Estimation Using Single Bridges

Penalty function:

$$\pi(s, c, p) = \begin{cases} 1 & \text{if } \mathit{playable}(s, c, p) \\ 1 + (\alpha - 1) \cdot (1 - \rho(s, c, p)) & \text{otw.,} \end{cases}$$

$\rho(s, c, p)$ should be the chance of recovering card c onto pile p.
$\Rightarrow$ Difficult to calculate, therefore single bridges estimation.

Single bridges estimation:
$\rho(s, c, p)$ is the probability of being able to recover the card using a single bridge next turn after drawing 2 cards.

$$\rho(s, c, p) = P(\text{drawn}) = \frac{s_b}{|D|} + \frac{d_b}{|D|} \cdot \frac{s_b + 1}{|D| - 1} + \frac{|D| - s_b - d_b}{|D|} \cdot \frac{s_b}{|D| - 1}.$$

Recovery with Single Bridges - Results

Results:

Win rate: 10.8% (9.4% without recovery term)

Cards remaining: 12.8 (13.2 without recovery term)

$\Rightarrow$ Underestimation of the recoverability probability.

Amplified recoverability term:

$$\pi(s, c, p) = \begin{cases} 1 & \text{if } \textit{playable}(s, c, p) \\ 1 + (\alpha - 1) \cdot (1 - \rho(s, c, p)^{\lambda}) & \text{otw.,} \end{cases}$$

Amplified results:

Using $\lambda = 0.2$.

Win rate: 12.2%

Cards remaining: 13.0

$\Rightarrow$ Almost as good as with the distance recovery term (win rate 12.7%).

Recoverability with Fallback

Penalty function:

$$\pi(s,c,p) = \begin{cases} 1 & \text{if } \mathit{playable}(s,c,p) \\ \alpha - \beta \cdot e^{\gamma \cdot (|p_c(s,p) - c| - 1)} & \text{if } \neg \mathit{playable}(s,c,p) \\ & \quad \wedge |p_c(s,p) - c| > b \\ 1 + (\alpha - 1) \cdot (1 - \rho(s,c,p)^{\lambda}) & \text{otw.} \end{cases}$$

Results:

$\alpha = 3.5$, magnitude $\beta = 1$, falloff $\gamma = 0.03$, and power $\lambda = 0.2$.
Win rate: 13.1%
Cards remaining: 12.4

Progression

Strategy Name	Win Rate	Cards Remaining	Backwards Trick Usage of Winners
Perceived Capacity	4.0%	17.1	18.1
Real Capacity	3.4%	17.9	16.3
Weighted Perceived Capacity	5.2%	15.4	18.0
Weighted Real Capacity	6.8%	14.9	15.7
Penalize Playability ($e^{-1.5x}$)	9.4%	13.2	15.1
Recovery Using Distance	12.7%	11.2	15.9
Single Bridges	10.8%	12.8	15.9
Single Bridges Amplified	12.2%	13.0	16.6
Recovery with Fallback	13.1%	12.4	16.7

Deeper Search Depth

Extended depth value function:

$$v_d(s) = \max_{\substack{s' \in \bigcup T_i(s) \\ i \in \{0,1,\ldots,d\}}} \{v(s')\}$$

Strategy Name	**Win Rate** $d = 0$	**Win Rate** $d = 1$	**Multiplicative Increase**
Perceived Capacity	4.0%	4.0%	0.6%
Real Capacity	3.4%	3.4%	0.4%
Weighted Perceived Capacity	5.2%	5.3%	2.3%
Weighted Real Capacity	6.8%	6.8%	1.2%
Penalize Playability ($e^{-1.5x}$)	9.4%	16.2%	73%
Recovery Using Distance	12.7%	18.8%	47%
Single Bridges	10.8%	18.2%	67%
Single Bridges Amplified	12.2%	22.6%	85%
Recovery with Fallback	13.1%	23.4%	80%

Deeper Search Depth

Results for $d = 2$:
Small sample size $n = 10\,000$.
Win rate: 29.7%
Cards remaining: 7.1

6 988 out of 10 000 reached the point where less than 10 cards were remaining.
$\Rightarrow$ This strategy gives on average an excellent result.

Possible Improvements (Outlook)

- Find better parameters.
- Search algorithm filtering only relevant positions.

$\Rightarrow$ Extended search depth with larger sample size.

- Improve the recovery term accuracy.

$\Rightarrow$ Amplified double bridges estimation?

Possible Improvements (Outlook)

- Find better parameters.
- Search algorithm filtering only relevant positions.

$\Rightarrow$ Extended search depth with larger sample size.

- Improve the recovery term accuracy.

$\Rightarrow$ Amplified double bridges estimation?

Thank you!

References I

- Benndorf, S. (2012). *Quixx*. URL: boardgamegeek.com/boardgame/131260/qwixx (visited on 09/12/2019).
- — (2015). *The Game*. URL: boardgamegeek.com/boardgame/173090/game (visited on 09/12/2019).
- Birge, J. R. and F. Louveaux (2011). *Introduction to stochastic programming*. Springer Science & Business Media.
- *Chess Algorithms - QGChess* (n.d.). URL: http://www.sites.google.com/site/qgchess/chess-algorithms.
- Gabillon, V., M. Ghavamzadeh, and B. Scherrer (2013). "Approximate dynamic programming finally performs well in the game of Tetris". In: *Advances in neural information processing systems*, pp. 1754–1762.
- Haykin, S. S. et al. (2009). *Neural networks and learning machines/Simon Haykin.* New York: Prentice Hall,
- Knuth, D. E. and R. W. Moore (1975). "An analysis of alpha-beta pruning". In: *Artificial intelligence* 6.4, pp. 293–326.

References II

- Ku, J. S. and M. Rudoy (n.d.). “Complexity of Benndorf’s “The Game””. In: ().
- Schafer, R. W. et al. (2011). “What is a Savitzky-Golay filter”. In: *IEEE Signal processing magazine* 28.4, pp. 111–117.
- Shannon, C. E. (1950). “XXII. Programming a computer for playing chess”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 41.314, pp. 256–275.
- Smith, D. K. (2007). “Dynamic programming and board games: A survey”. In: *European Journal of Operational Research* 176.3, pp. 1299–1318.