WALTZ: a strong Tzaar-playing program

Tomáš Valla[1] * and Pavel Veselý[2]

[1] Faculty of Information Technology, Czech Technical University in Prague, Czech Republic. tomas.valla@fit.cvut.cz
[2] Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic. vesely@kam.mff.cuni.cz

Abstract. Tzaar is an abstract strategy two-player game, which has recently gained popularity in the gaming community and has won several awards. There are some properties, most notably the high branching factor, that make Tzaar hard for computers. We developed Waltz, a strong Tzaar-playing program, using enhanced variants of Alpha-beta and Proof-number Search based algorithms. After many tests with computer opponents and a year of deployment on a popular board-gaming portal, we conclude that Waltz can defeat all available computer programs and even strong human players. In this paper we describe Waltz, its performance and an enhancement of Proof-number Search developed for Waltz that can be also used in other domains than Tzaar.

1 Introduction

Tzaar is a relatively new game, which was invented by Kris Burm and published in 2007. Despite being so young, Tzaar has won quite a lot of awards, most notably the Games Magazine's award "Game of the Year 2009" [19], "Spiel des Jahres" Recommendation in 2008 [21], and earned nominations to several other awards. Tzaar is also highly rated by the gaming community, for example on the popular server BoardGameGeek.com it has the second highest rating among abstract games. It is a part of the Project GIPF, a set of six abstract strategy two-player games. The first game of the project, also called GIPF, was played on Computer Olympiad [20] in 2001.

There are several properties that make Tzaar a hard game to play for computers. Most notably it is the high branching factor (see Section 1.3). Even in the endgame there is usually more than one solution to a threat, thus algorithms based on threats like Dependency-based Search [1] or Lambda Search [10] are not effective. We cannot also easily decompose the game into independent parts (unlike Amazons), thus standard techniques from combinatorial game theory are not applicable. Therefore, writing a strong Tzaar playing program is a challenge. We address this challenge by developing Waltz,[3] a strong program able to defeat all other Tzaar programs that we are aware of, and also—which is more important—match up with and defeat even strong human players.

We have installed several playable "robots" on the popular board-gaming portal Boitejeaux.net [18], where some very strong players are playing. The details

* This work was supported by the Centre of Excellence—Inst. for Theor. Comp. Sci. (project P202/12/G061 of GA ČR)

[3] The name stands for the recursive acronym *Waltz ALgorithmic TZaar*.

about Waltz performance against both computer and human opponents can be found in Section 4.

The algorithms employed in Waltz are based on Alpha-beta pruning and Proof-number Search (PNS), together with many enhancements, see Section 2 for more details. We chose and tuned these algorithms and their enhancements after numerous statistical experiments and play-outs with other Tzaar playing programs, humans, and different versions of Waltz.

We also developed an enhancement of PNS for Waltz called Heuristic Weak PNS. See Section 2.2 for its description.

This paper was preceded by the thesis of Veselý [13], which, although slightly outdated, contains a lot of details that are omitted here. Waltz, the thesis, and other information can be downloaded from our website [11].

1.1 Tzaar Rules

Tzaar is a modern abstract strategy two-player game with full information, bearing a distant similarity to Checkers in some sense.

The board for Tzaar is hexagonal and consists of 30 lines that makes 60 intersections. There is a missing intersection in the center of the board. In the starting position there are 30 white and 30 black pieces, one at each intersection. Each color has pieces of three types: 6 are *Tzaars,* 9 are *Tzarras* and 15 are *Totts.* See Fig. 1 for illustration.

The initial placement could be random or players can use a fixed starting position which is defined in the official rules [2].

Pieces can form *stacks,* that means, towers of pieces of the same color. In the beginning, all stacks on the board have height one. A stack is one entity, thus it cannot be divided into two stacks. The type of a stack is the type of its top piece.

White player and black player take turns, white has the first turn. Each player's turn consists of two moves. There is an exception in the very first turn of white player, as his turn consists only of the first move.

The first move of each turn must be a *capture.* The player on turn moves one of his stacks along a line to an intersection with an opponent's stack. A stack cannot jump over other stacks or over the center of the board. Only a jump over and arbitrary number of empty intersections is allowed. No stack may end the jump on an empty intersection. A captured stack must have height at most the height of the capturing stack. Captured pieces leave the board.

The second move of a turn can be another capture move, or a stacking move, or a pass move. *Passing* means that the player on turn does not move with any stack. During the *stacking move* the player jumps with his stack on some other stack of his color. The height of the resulting stack is the sum of both stacks heights. The type of the resulting stack is determined by the piece on the top.

A player loses when the last stack of one of the three types is captured, or if he cannot capture in the first move of his turn. A draw is not possible.

1.2 Strategies

In this section we discuss some common heuristic strategies how to play Tzaar. These observations are based on authors' experiences from numerous play-outs with both human and computer opponents. We use these strategies to construct

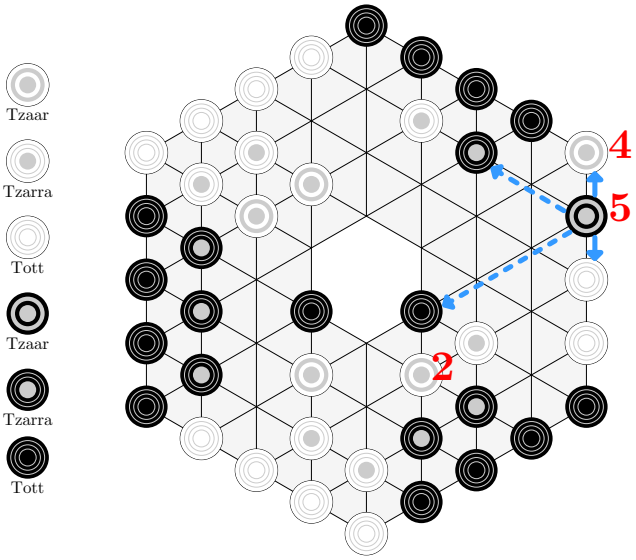

Fig. 1: The Tzaar board with a sample position and piece types on the left. The possible moves of the black Tzaar stack in the second move of a turn are marked by arrows, the dashed arrows represent stacking moves and the numbers denote the stack heights greater than one.

the evaluation function of Waltz, see Section 2.1. However, as these strategies are based on heuristic arguments, there are of course positions where they do not yield good results.

The first move is always a capture move, but it often depends which type of piece is captured. A good move usually consists of capturing piece type t such that the opponent does not have a high stack of type t, and as a secondary condition such type t that there are not many stacks of type t. In a typical game the player starts by creating a stack of Tzaar, it is thus convenient to capture Tzarras.

In the second move of a turn a player has three possibilities:

- *Capturing* again (so called *double-capture move*). This is appropriate if the opponent is running out of pieces of a certain type (he should not have a high stack of that type), or if a high stack can be captured. Height of the double captured stack should be greater than two, because by capturing stacks of size two, a player may lose capturing possibilities. Moreover, double capturing two pieces with height one usually leads to a loss because of no capturing possibilities.
- *Stacking* is often the most reasonable move, because it makes one of the stacks more powerful and more safe against opponent's stacks. The other reason is that the opponent loses capturing possibilities, thus it is more likely that he will run out of captures and lose in the endgame.
- *Passing* occurs rarely during the game. It is worth playing only in the endgame when stacking and capturing are not possible or would result in a loss. See Figure 2 for an example of such position.

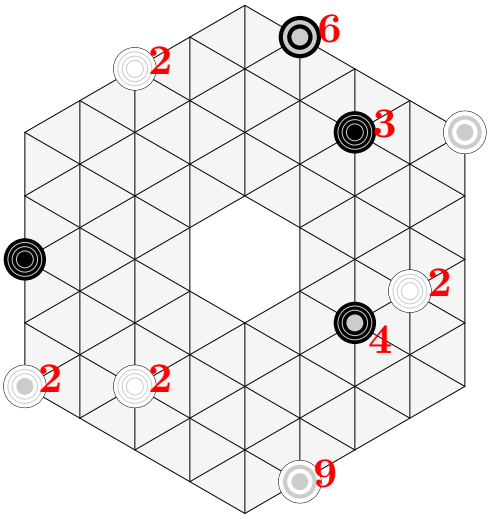

Fig. 2: In this position, black player is on turn. After the last black Tzaar stack captures the white Tzaar piece in the right corner, the only move not leading to a loss for black is the pass move.

There are generally two stacking strategies:

1. Creating one high stack which is powerful and can capture all opponent's stacks, or which forces the opponent to raise his highest stack.
2. Creating more lower stacks, usually of height two, although it is safer when some of them have height at least three.

It is not known to us which strategy is better. Using the first strategy the player can quite easily threaten or even capture small opponent's stacks, but using the second strategy it is sometimes impossible for the opponent to create a new stack (it would be captured immediately) and the opponent can lose because of it. The second strategy is more reasonable during the endgame, since it decreases the opponent's capturing possibilities. Also, having a stack much higher than all other opponent's stacks is worse than having more lower stacks.

These strategy observations were mostly about material; now we give some positional strategy tips:

- Keeping high stacks inside the board, not on the border. Stacks inside are able to move to any direction and thus they threaten large part of the board. Moreover, during the middle game a stack placed inside the board can nearly always escape from a threat. The worst positions are the six corners of the board.
- Limiting moving possibilities of an opponent's high stack, i.e., moving pieces away from lines containing an opponent's high stack.
- Isolating a small stack (preferably of size one) such that there are no other pieces on the same lines as the isolated stack. The reason is that the player cannot run out of the type of an isolated piece, thus the type is safe.
- Isolating own high stack is not good, because the stack cannot be used for capturing opponent's stacks.

– Limiting opponent's capturing possibilities and also preparing own capturing possibilities during the endgame.

The black player has a small advantage, because he is stacking first. Hence he can often threaten white player by attacking white stacks and white player should create his first stack as far from black player's stack as possible.

1.3 Game Properties

We estimate the maximum height of a stack. Observe that before a stacking move that created a stack of height h, the opponent must have captured at least $h-1$ pieces. There should be two pieces of another types present and there are 30 pieces of each color, the maximum number of captures is 13 as at least two other pieces must be present, so the maximum stack height is 14.

The *state space complexity* is the number of game positions reachable from any starting position. There are $\binom{60}{v}$ different choices of fields for stacks, where v is the number of free fields. Let k be the sum of heights of all white stacks on the board, i.e. the number of white pieces, and analogously ℓ for the black color. Both numbers are bounded from above by the number of necessary captures before exactly v free fields appeared on the board. Thus $k, \ell \leq 30 - \lfloor \frac{1}{4} v \rfloor$ (at least one fourth of moves must capture a white stack).

Let there be s white stacks, so the number of black stacks is $60 - v - s$. We know that $s \leq \min(k, 58 - v)$, since there are k white pieces and there must be two black stacks on $60 - v$ occupied fields on the board. The number of different stack heights for s stacks with k white pieces is $\binom{k-1}{s-1}$; the number of different choices of fields for white stacks is $\binom{60-v}{s}$ and 3^s is the number of different types of white stacks. Similar formulas holds for black player. This gives us the upper bound on the number of possible states:

$$\sum_{v=1}^{55} \binom{60}{v} \sum_{k=2}^{30-\lfloor \frac{1}{4} v \rfloor} \sum_{s=2}^{\min(k,58-v)} \binom{60-v}{s} \binom{k-1}{s-1} 3^s \cdot \sum_{\ell=60-v-s}^{30-\lfloor \frac{1}{4} v \rfloor} \binom{\ell-1}{59-v-s} 3^{60-s-v}$$

$$\doteq 9.17 \cdot 10^{57}$$

Let us now take symmetries into account. The position can have 6 equivalent rotations. The position may also have 6 isomorphic mirrors by 6 axes (between opposite corners of the board and between centers of opposite sides). Mirroring twice by any two axes results in a rotated position, thus there are 12 isomorphic positions. We use Burnside's Lemma (the orbit-counting theorem) to count the number of distinct positions.

For each symmetry we estimate the number of fixed points, i.e., positions that are the same after applying a symmetry. The identity has clearly $9.17 \cdot 10^{57}$ fixed points. Rotation symmetries (except identity) have at most $(14 \cdot 6 + 1)^{10} \doteq 1.97 \cdot 10^{19}$ fixed points, since the maximal height is 14, Tzaar has six types of pieces and the six triangles with 10 fields that lie between the side of the board and the side of the empty part in the middle must be the same. Mirroring by axes has at most $4.83 \cdot 10^{28}$ fixed points which we obtain using similar formula as for the state space complexity. From Burnside's Lemma we get the upper bound on the number of distinct positions reachable from any starting position:

$$(9.17 \cdot 10^{57} + 5 \cdot 1.97 \cdot 10^{19} + 6 \cdot 4.83 \cdot 10^{28})/12 = 7.64 \cdot 10^{56}$$

This is an upper bound on the number of positions that can be reached from all starting positions altogether, but some positions can be obtained from more than one initial position.

The number of different starting positions is $60!/(15! \cdot 9! \cdot 6!)^2 \doteq 7.13 \cdot 10^{40}$. Using Burnside's Lemma to deal with symmetries we get $5.94 \cdot 10^{39}$ different starting positions. The number of fixed points is zero for rotation symmetries, since the number of black Totts is not divisible by six. For mirroring symmetries the number of fixed points is at most $30!/(8! \cdot 5! \cdot 3!)^2 \doteq 3.15 \cdot 10^{17}$.

Let us now estimate the number of endgame positions. We count the number of positions with six stacks of different types or colors—if there are two pieces of the same color and type, the position is won by one of the players. We observe that the number of positions with more than six stacks is higher. The number of positions with exactly six stacks is the number of different choices of six fields on the board multiplied by the number of permutations of six stacks and the number of different stack sizes for each piece type and for each player. The maximum sum of stack sizes for a player is 16, because there should be a capture before each stacking. Therefore, the number of endgame positions is

$$\binom{60}{6} \cdot 6! \cdot \left(\sum_{i=3}^{16} \binom{i-1}{2} \right)^2 \doteq 1.13 \cdot 10^{16},$$

where i denotes the sum of stack heights for one player.

After taking symmetries into account, we get $9.42 \cdot 10^{14}$ different positions with six different stacks. Note that the number of fixed points is zero for rotations symmetries and for mirroring by axes between opposite sides. For mirroring between opposite corners it is at most

$$6! \cdot \left(\sum_{i=3}^{16} \binom{i-1}{2} \right)^2 \doteq 2.26 \cdot 10^{8}$$

For a lower bound on the state space complexity we can use the number of distinct starting positions which is $5.94 \cdot 10^{39}$. We thus believe that the real state space complexity lies roughly between 10^{45} and 10^{55}

The *branching factor* depends on the starting position. The fixed starting position has the maximum branching factor around 5 500, but there are starting positions with the branching factor up to 10 000. We count positions reachable by two possible ways only once, otherwise the branching factor can be 14 000. During the game, the branching factor is decreasing as the pieces are captured or stacked. Table 1 provides a summary of the minimum, maximum and average branching factor according to the number of stacks on the board, computed statistically from real play-outs.

The *game tree complexity* is usually estimated by multiplying the average branching factor for each turn. For Tzaar we get approximately 10^{79}.

We conclude that Tzaar has much larger state space complexity than GIPF that has roughly 10^{25} different positions [14] and probably slightly larger than Chess that has 10^{46} positions [3]. The game is also harder for computers because of huge number of possible starting positions and more importantly the branching

Stacks	59	55	49	43	37	31	25	19	13	9	7	6
Positions	470	464	456	460	446	427	399	338	170	61	15	7
Min	4961	3962	2732	1732	906	403	139	35	1	1	1	1
Max	9933	7651	6007	4235	2986	2078	1073	476	114	21	4	2
Avg	7497	5965	4463	2971	1978	1117	562	203	37	7	1	1

Table 1: Minimum, maximum and average branching factor according to the number of stacks on the board. The table contains also the number of positions from which the values were obtained. We sampled positions from real games at BAJ [18] and these positions can be downloaded from [11].

factor which is more than 1000 for most of the game. In contrast, GIPF or Chess have average branching factor from 30 to 40. On the other hand, Tzaar games are quite short, typically up to 28 turns of a player, thus the game tree of Chess or GIPF is larger (about 10^{123} for Chess and 10^{132} for GIPF [14]).

2 Algorithms for Tzaar

We now discuss algorithms we have implemented in Waltz. We also describe domain dependent heuristics. Since the game tree properties differ in the middle game and in the endgame, we discuss these parts of the game separately.

Due to the high number of possible starting positions the Opening database technique is not applicable. Similarly, one cannot use the endgame database as even the number of positions with only six different stacks is $9.42 \cdot 10^{14}$ as we counted in Section 1.3. We thus believe that data-base methods are not applicable in Tzaar.

We cannot also easily decompose the game into independent parts, since stacks can jump from one part of the board to another by few moves. Hence standard techniques from combinatorial game theory are not applicable.

Opening has the highest branching factor, but otherwise it is not very different from the middle game. Before the endgame, the attacking player usually cannot capture defender's high stack or even win in a few moves by a threat sequence. Defender can escape with his stack from most threats easily and there are often more different ways to do it. We thus conclude that algorithms based on threats would be ineffective during the opening and middle game, therefore Proof-number Search (PNS) is used only during the endgame.

The most frequently used algorithm in Waltz is Minimax with the Alpha-beta pruning and several enhancements, namely:

- Transposition Table (TT): Used for storing moves from the previous shallower search (the Principal Variation Move, PV) and also because some positions can be reached by a few different move sequences.
- Iterative Deepening (ID): Implemented because of time estimation (how deep may the engine search), and because of PV.
- domain specific Move Ordering (MO): Done by heuristically assigning values to moves and sorting moves according to these values. In most cases, stacking is preferred to capturing.
- History Heuristic (HH): Only for the first move of a turn.
- NegaScout (NS): To quickly find cutoff nodes.

– Randomized Alpha-beta: for the first two moves, Waltz chooses uniformly randomly among moves with a value at least $bestValue - margin$ for a given constant *margin*. These moves are found using a slightly modified Alpha-beta search. See [13] for more details.
– Playing in lost positions: when Waltz finds out that it is in a lost position, it uses the best move in the last iteration of the Iterative Deepening where Alpha-beta has not found out that the position is lost. Thus Waltz plays a move that leads to a loss after the maximal possible number of moves.

In the endgame the branching factor is not so high and threat sequences occur more frequently. There are also fewer solutions to threats, thus threats limit the branching factor and Proof-number Search (PNS) can sometimes be more effective than Alpha-beta search. However, PNS as proposed by Allis [1] consumes a considerable amount of memory. Therefore, we use the Depth-first Proof-number Search (DFPN) [6] with the following enhancements:

– Move Ordering: The same as in Alpha-beta.
– Evaluation Function Based PNS (EFB PNS) [15]: Heuristic initialization of leaves using the evaluation function.
– $1 + \varepsilon$ Trick [7]: To avoid frequent jumping of the search across the tree.
– Weak PNS (WPNS) [4] and Dynamic Widening (DW) [17]: To suppress overestimation of proof and disproof numbers.
– Heuristic Weak PNS (HW PNS): A new enhancement, see Section 2.2.
– Time estimation: How many nodes can DFPN visit within a given time—at first a certain number of nodes is visited and then the number of nodes to visit is estimated.

We note that there are some other algorithms for solving endgame positions. For the Lambda Search [10] we were not able to determine quickly the order of a threat. Since there is usually more than one way to evade a threat, we may conclude that the Dependency-based Search [1] is not suitable for Tzaar.

See Section 3 for an evaluation of how each enhancement improves the search. The detailed description of the algorithms and their enhancements can be found e.g. in [13].

2.1 Evaluation Function

The evaluation of a position in Tzaar is used both by the Alpha-beta search and DFPN. We created the evaluation function according to strategy observations given in Section 1.2. We tuned up its constants by playing with Waltz and by numerous play-outs between different versions of the evaluation function.

In positions with a positive value, white player has an advantage (∞ is a win), and vice versa for black player. We basically use this formula:

$$\begin{aligned}\mathrm{eval}(\mathit{position}) = {} & \mathrm{material}(\mathit{position}, \mathrm{White}) + \mathrm{positional}(\mathit{position}, \mathrm{White}) \\ & - \mathrm{material}(\mathit{position}, \mathrm{Black}) - \mathrm{positional}(\mathit{position}, \mathrm{Black})\end{aligned}$$

The material value for a player is the sum of values of player's stacks:

$$\mathrm{material}(\mathit{position}, \mathit{player}) = \sum_{\substack{s \text{ is a stack} \\ \text{of } \mathit{player}}} \mathrm{heightValue}(s) \cdot \mathrm{countValue}(s)$$

The function heightValue grows rapidly up to 150 for heights less than 4, then stays nearly the same and decreases for stacks higher than 8. The reason is that instead of building very high stacks a player can build more lower stacks, which is usually better. The function countValue is inversely proportional to the count of stacks with the same piece type as the stack s. It is 100 for the count 1, then it decreases rapidly and it is less than 20 for counts higher than 5.

The material value is more important in the first half of the game. The material value together with some positional information is counted incrementally (when a move is executed or reverted), other positional features are counted statically for each leaf node that is not won by a player.

For the positional value the Zone of Control (ZOC) is maintained. It determines how many stacks of a certain type can be captured in one move, no matter who is on turn. It is used also for determining whether a player on turn has lost because of no possible captures.

The positional value for a player is roughly the sum of these bonuses:

- 20 000 000 for an immediate threat: The player is on turn and he can capture all stacks of an opponent's piece type (the player can win).
- 1 000 for a threat, when the player is not on turn.
- 1 000–200 000 if the opponent has few possible captures.
- Value of ZOC: $\sum_{\substack{\text{opponent's}\\ \text{piece type } t}} \text{stacksInZOC}(t) \cdot (1 - \text{count}(t)/\,\text{initialCount}(t))$
- 25 000 for each player's piece type that is "secure"—the player has a stack higher than all stacks of his opponent—and 100 000 if all types are secure.
- 50 000 for stacks with height at least 2 of all types.
- 50 000 if an opponent's valuable stack can be captured.
- 1 000–100 000 for an opponent's high stack that cannot move.
- 10–25 for high stacks not on the margin of the board and −30 for a stack in the corner.

2.2 Heuristic Weak PNS

As positions often occur more than once in a game, the state space is described by a directed acyclic graph instead of a tree. Then DFPN suffers from the *double-counting problem*, when the proof number of a position contains the proof number of another position more than once.

This problem can be addressed by modifying the summation of disproof numbers in OR nodes and proof numbers in AND nodes. Weak PNS [4] proposes taking the maximum disproof number and adding the number of children minus one. Another solution to this problem is described by Kishimoto [5].

We propose a new enhancement based on Weak PNS and the evaluation function. We modify counting disproof numbers in OR nodes (and analogously in AND nodes) in a way similar to Evaluation Function Based PNS. The idea of using the evaluation function is also briefly mentioned by Kishimoto [5].

We define the step function similarly to Evaluation Function Based PNS:

$$\text{step}(\mathit{value}) = \begin{cases} 2 & \text{if } \mathit{value} \geq t, \\ 1 & \text{if } -t < \mathit{value} < t, \\ 0 & \text{if } \mathit{value} \leq -t, \end{cases}$$

where *value* is the value of the current position and the threshold t indicates the player's high advantage. The best value for t is at least 10^6 (see Section 3) while a win has value $2 \cdot 10^9$.

We count the disproof number (DN) as $maxDN + h(m-1)\,\mathrm{step}(value)$, where *maxDN* is the maximum disproof number among children, m is the number of moves and $h > 0$ is a constant.

Now we discuss reasons for this modification of Weak PNS. When the player on turn has a big advantage and $value \geq t$, DN is ∞ with a high probability. We can thus set DN to $maxDN + 2h(m-1)$. In the case of a balanced position, we count DN similarly to Weak PNS. Because of this, the parameter h should be close to 1. When the player on turn is in a bad position, we likely do not need to search many positions to disprove the node, so DN is set to *maxDN*.

3 Experiments with Waltz

This section shows the results of search runtime optimization. For parameter tuning and measuring the runtime we use two sets of Tzaar positions. The first set, we call it *MidSet*, consists of 200 middle game positions with exactly 41 stacks on the board. It is intended for testing Alpha-beta.

For experimenting with DFPN we have a set of 713 endgame positions with less than 27 stacks on the board, we call it *EndSet*. Both *MidSet* and *EndSet* are available at [11]. We took these positions from Waltz's games with strong and intermediate players on BAJ. This set contains both easy positions (Waltz solves them quickly) and hard positions (neither DFPN, nor Alpha-beta are able to find a solution within a minute).

We performed the tests on a Dual-Core AMD Opteron 2216 server with 64 GiB of memory, but we used only one of its cores.

For Alpha-beta we measured the efficiency of the Alpha-beta enhancements in the domain of Tzaar by searching each *MidSet* position to the depth of 3 turns. We observed that it is best to use all Alpha-beta enhancements listed in Section 2. Since this behavior occurs also in other games, this approach does not contribute with some new insight, so we omit the exact results. They can be found in [13].

Table 2 shows the importance of enhancements for DFPN. Note that there is nearly no difference between DW, WPNS and HW PNS, and that one single enhancement is still not enough. Surprisingly, sorting moves heuristically using the same algorithm as in Alpha-beta is useful. We ran the tests on *EndSet* positions with the time limit of 60 seconds.

Enhancements	Solved (out of 713)
HW, $1+\varepsilon$ Trick and EFB	484
WPNS, $1+\varepsilon$ Trick and EFB	484
DW, $1+\varepsilon$ Trick and EFB	480
HW, $1+\varepsilon$ Trick and EFB without sorting moves	465
Only $1+\varepsilon$ Trick	343
Without enhancements	336
Only Heuristic Weak (HW)	289
Only Evaluation Function Based (EFB)	289

Table 2: Results of the DFPN search with different enhancements listed in Section 2.

We find it strange that Heuristic Weak PNS does not solve more positions than Weak PNS, but we think that Heuristic Weak PNS can improve solvers in other games.

We experimented also with different sizes of TT and constants used in DFPN enhancements, namely $1+\varepsilon$ Trick, EFB PNS and DW. For each constant we tried different values, run the experiments and counted the number of solved positions from *EndSet*. The results are omitted due to space limitations and can be found in [13].

Heuristic Weak PNS has two parameters: the threshold t for the step function and the multiplier h. From the experiments we observed that the best values are $h = 1$ and $t \geq 10^6$—the value of a position in which a player has a significant advantage.

3.1 DFPN versus Alpha-beta in Endgames

DFPN was designed to find long winning strategies where the player can force his opponent to have only a limited number of possible moves. We tried DFPN on Tzaar endgames, although Tzaar has relatively high branching factor even in endgames. On the other hand, the player can sometimes force his opponent to have a small number of moves.

To decide whether to use Alpha-beta or DFPN in endgames we ran statistical experiments. Using the best possible setting of constants in DFPN, it solved 495 out of 713 positions. Then we tried Alpha-beta (with all enhancements) and it solved 506 positions. There are 20 positions which DFPN solved and Alpha-beta did not, so DFPN is reasonable to use in Waltz.

Hence Waltz try to use DFPN first in the endgame when the number of stacks is at most 23. If it does not succeed because of the time limit or because DFPN found disproof, we run the Alpha-beta search.

4 Results against Computer and Human Opponents

We tested Waltz against other existing programs for playing Tzaar that are available: HsTZAAR [12] and programs of students from University of Alaska [22].[4] See Table 3 for the results.

During the tests, Waltz had a time limit of 30 seconds. Each game started with a random starting position. We performed tests with HsTZAAR on Intel Xeon ES-1620 server with 64 GiB of memory and tests with the other programs on a AMD Turion II P560 Dual-Core notebook with 4 GiB of memory.

To test Waltz against people we chose the game server Boiteajeux.net (BAJ) [18], since a lot of people play Tzaar there.[5] For each game, an ELO rating is counted.[6]

We created four different versions of Waltz which are described in Table 4. We performed matches between these versions to compare their strength. See Table 5 for the results.

[4] There are also some more programs available, but due to their design it is not possible to run automatic play-outs between them and Waltz.

[5] 299 players have played Tzaar in the last six month till May 10, 2013 and 24 842 Tzaar games were finished on BAJ from October 31, 2008 to May 10, 2013.

[6] For a win a player obtains some ELO points according to his and opponent's ELO and his opponent loses the same number of points. New player receives ELO 1 500.

Program	Wins	Losses	Note
HsTZAAR	479	121	Used with the algorithm `pscout_full_4` on 4 cores.
GreensteinTzaarAI	342	53	We could not set a time limit.
BiTzaarBot	196	5	The time limit was 40 seconds.
Mockinator++	83	2	The time limit was 40 seconds.
Mockinator	82	1	The time limit was 40 seconds.

Table 3: Results of Waltz against other Tzaar-playing programs. Wins and losses are counted from the Waltz's point of view.

Level	Username on BAJ	Time limit [s]	Used algorithms
Beginner	PauliebotBeginner	30	Randomized Alpha-beta with a very simple evaluation function to the depth of two turns of a player for the whole game with a big margin (5000).
Intermediate	PauliebotMedium	30	Randomized Alpha-beta with the full evaluation function to the depth of two and half turns for the whole game with a small margin (20).
Expert	Pauliebot	30	Both Alpha-beta and DFPN with the full evaluation function.
"Unbeatable"	PauliebotUnbeatable	300	Both Alpha-beta and DFPN with the full evaluation function.

Table 4: Versions of Waltz.

	Beginner	Intermediate	Expert	"Unbeatable"
Beginner		224:1140	218:1090	45:351
Intermediate	1140:224		647:1371	84:208
Expert	1090:218	1371:647		98:149
"Unbeatable"	351:45	208:84	149:98	

Table 5: Results of matches between versions of Waltz.

Now we describe how successful Waltz was against human opponents on BAJ. We focus only on the expert and unbeatable versions since the other versions are intended to play weaker.

We released Waltz in the expert version on March 20, 2012 under username *Pauliebot*, and it was under development until April 4, 2012. After that we made only minor updates, mostly improving the evaluation function. On April 24, 2012 we released the other versions of Waltz.

The expert version has played 154 games so far.[7] It won 114 of them and it is the 16th best Tzaar player with ELO 2 068.[8] Most important results of the expert version are in the left part of Table 6. We conclude that the expert version played on the level of best players on BAJ, but sometimes intermediate players were able to defeat it.

[7] Some of these games were played for testing purposes.
[8] ELOs of players and other data in this section were up to the date March 4, 2013.

The unbeatable Waltz version has ELO 2087, the 14th highest, and played 74 games from which it won 46 games.[9] The most important results of the unbeatable version are in the right part of Table 6. The 9 wins against SlowBrain are a great success because SlowBrain is far better than other players. From these results we conclude that more time to search helps Waltz to play better.

Player	Rank	ELO	Wins	Losses
SlowBrain	1st	2 432	1	3
Gambit	2nd	2 229	2	4
Paulie	3rd	2 220	8	3
evrardmoloic	17th	2 062	1	1
mat76	77th	1 690	16	1
Gregg	78th	1 684	14	5
PhilDakota	79th	1 684	14	3

Player	Rank	ELO	Wins	Losses
SlowBrain	1st	2 432	9	16
Paulie	3rd	2 220	1	3
mnmr	6th	2 184	0	1
Zeichner	9th	2 143	1	0
Talisac	13th	2 100	3	0
azazhel	28th	1 931	8	2

Table 6: Some results of the expert (left) and unbeatable (right) versions. Wins and losses are counted from the Waltz's point of view. Note that Paulie is a nickname of one of the authors, not of one of the Waltz's version.

The most frequently appearing reason why Waltz lost games on BAJ was the loss of the last stack of Tzarras. We observed that in two or three last turns of these lost games Waltz had no chance to create a stack of Tzarras which could not be captured by the opponent—Waltz was probably not aware of such an opponent's trap soon enough. Another bad thing in Waltz's behavior during these games was losing quite high stacks (size 3, 4, or even 5) during the middle game.

We thus tried to improve the evaluation function to avoid these problems. The version with the enhanced evaluation function won 136 and lost 102 games against the version with the old evaluation function. On March 4, 2013 we released the version with the enhanced evaluation function.

5 Further Work

There are some other algorithms which we did not implement in Waltz. Monte Carlo Tree Search is probably the most promising approach, and we consider it to be the next direction where we would like to move Waltz's development. Another direction lies in parallelizing Waltz's algorithms, which is a natural step we would like to try. For example the DFPN algorithm can be parallelized by Job-level Proof-number search [16], there are also parallelization approaches proposed by Saito, Winands and van den Herik [9] or Saffdine, Jouandeau and Cazenave [8].

It turned out that the enhancement Heuristic Weak PNS was not better than Weak PNS in the domain of Tzaar, but we leave for a future research whether it can be useful in other domains.

References

1. Allis, L.V.: Searching for Solutions in Games and Artificial Intelligence. Ph.D. thesis, University of Limburg, Maastricht, The Netherlands (1994)

[9] Some of these games were against other Waltz versions—this was done to increase robot's ELO, otherwise strong players would not want to play against an opponent with a low ELO.

2. Burm, K.: Tzaar rules. GIPF project, http://www.gipf.com/tzaar/rules/rules.html
3. Chinchalkar, S.: An upper bound for the number of reachable positions. ICCA Journal pp. 181–183 (1996)
4. Hashimoto, J., Hashimoto, T., Iida, H., Ueda, T.: Weak proof-number search. In: Proceedings of the 6th international conference on Computers and Games. pp. 157–168. Springer-Verlag, Berlin, Heidelberg (2008)
5. Kishimoto, A.: Dealing with infinite loops, underestimation, and overestimation of depth-first proof-number search. In: Fox, M., Poole, D. (eds.) Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010. AAAI Press, Atlanta, Georgia, USA (2010)
6. Nagai, A.: Df-pn algorithm for searching AND/OR trees and its applications. Ph.D. thesis, The University of Tokyo, Japan (2002)
7. Pawlewicz, J., Lew, Ł.: Improving depth-first pn-search: $1 + \varepsilon$ trick. In: Proceedings of the 5th international conference on Computers and games. pp. 160–171. CG'06, Springer-Verlag, Berlin, Heidelberg (2007)
8. Saffidine, A., Jouandeau, N., Cazenave, T.: Solving breakthrough with race patterns and job-level proof number search. In: van den Herik, H.J., Plaat, A. (eds.) ACG. Lecture Notes in Computer Science, vol. 7168, pp. 196–207. Springer (2011)
9. Saito, J.T., Winands, M.H.M., van den Herik, H.J.: Randomized parallel proof-number search. In: van den Herik, H.J., Spronck, P. (eds.) ACG. Lecture Notes in Computer Science, vol. 6048, pp. 75–87. Springer (2009)
10. Thomsen, T.: Lambda-search in game trees – with application to go. In: ICGA Journal. pp. 203–217. Springer (2001)
11. Valla, T., Veselý, P.: Waltz, http://kam.mff.cuni.cz/~vesely/tzaar/
12. Vasconcelos, P.: HsTZAAR, http://www.dcc.fc.up.pt/~pbv/stuff/hstzaar/
13. Veselý, P.: Artificial intelligence in abstract 2-player games. Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic (2012), http://kam.mff.cuni.cz/~vesely/tzaar/thesis.pdf, Bachelor's thesis
14. Wentink, D.: Analysis and Implementation of the game Gipf. Master's thesis, Universiteit Maastricht (2001)
15. Winands, M.H.M., Schadd, M.P.D.: Evaluation-function based proof-number search. In: Proceedings of the 7th international conference on Computers and games. pp. 23–35. CG'10, Springer-Verlag, Berlin, Heidelberg (2011)
16. Wu, I.C., Lin, H.H., Lin, P.H., Sun, D.J., Chan, Y.C., Chen, B.T.: Job-level proof-number search for connect6. In: Proceedings of the 7th international conference on Computers and games. pp. 11–22. CG'10, Springer-Verlag, Berlin, Heidelberg (2011), http://dl.acm.org/citation.cfm?id=1950322.1950324
17. Yoshizoe, K.: A new proof-number calculation technique for proof-number search. In: Proceedings of the 6th international conference on Computers and Games. pp. 135–145. CG '08, Springer-Verlag, Berlin, Heidelberg (2008)
18. Boiteajeux board-gaming portal, http://www.boiteajeux.net/
19. GAMES game awards. Games Magazine, http://www.gamesmagazine-online.com/gameslinks/archives.html#2009awards
20. List of games, ICGA tournaments. [cit. 2013-05-11], http://www.grappa.univ-lille3.fr/icga/games.php
21. Spiel des jahres, awarded games 2008, http://www.spiel-des-jahres.com/cms/front_content.php?idart=925
22. Tzaar - ai game project for 2011, http://www.math.uaa.alaska.edu/~afkjm/cs405/tzaar/