TGA Manual

Tomas Kozelek

May 24, 2006

1 About

TGA is a program solving life and death problems in Go (for more information about Go see [2]). Program is designed to solve problems where one group (*defender's*) is rather closed from the rest of the board by the opponent's secure group (*attacker's*) and it's life is in question (it means that it cannot connect outside). Search algorithm is able to deal with such peculiarities as for example: seki, ko (direct ko, approach move ko, multi step ko), double ko, bent four in the corner, under the stones problems. I estimate program's overall solving strenght to be at low dan level (around 1d). However, it showed some pleasant results in solving rather very difficult (5d, 6d) problems. Program might be very useful in Go problems analyses even for low dan players and for all kyu players.

2 Installation & miscellanous

So far program is accessible to Linux/Unix users only. Installation is pretty simple : copy archive file `tga.tar.gz` to desired location on your computer, then extract it and follow instructions in ./INSTALL file (classical `make` command covers whole installation). Program uses no special libraries and for successfull installation is sufficient the presence of `gcc` compiling program with standard libraries. Program is distributed under GNU/GPL free of charge.

After the installation the directory structure (besides others) contains:

tga is a program itself.

clear_sgf is a bash script removing all sgf files from tga directory (useful after TGA is ran in multiple position mode).

src is a directory with program source code.

obj is a directory with program object modules.

data is a directory with a small Go problems collection on which program was tested (problems are categorized according to difficulty for the program).

license is a directory containing general information on GNU/GPL.

sgf_files_list is a list of problems on which was program tested (this file was used as input in TGA multiple position mode).

ABOUT is a simple text file with information about project.

3 Input

3.1 Options

Since there is no need of interactive communication between user and program, TGA is a "command line" program. It is launched by typing `./tga ...-i input_position_filename` in program directory , where ... represents program options and input_position_filename represents problem to be solved in SGF format (see [3]) with appropriate marking (see §3.2). There is no configuration file, therefore requested non-default options must be included in every launch. Options might be written as usual (e.g. -nab) or in GNU format (e.g. –no_alpha_beta). Here is a list of all possible options and a detailed explanation of their meanings:

-nab (*–no_alpha_beta*) Deactivates alpha beta prunning (slows down the search). Alpha beta prunning significantly reduces number of nodes neccessary to be searched in order to retrieve a result, thus reducing the search time.

-af (*–attacker_first*) Searches only for the result when attacker moves first in the given position.

-df (*–defender_first*) Searches only for the result when defender moves first in the given position.

-to (*–testing_output*) Useful testing informations are included in every node's info of the sgf output (testing).

-nkh (*–no_ko_handling*) If this option is switched on, algorithm doesn't try to solve ko. It has mainly testing purposes and given results might be wrong (testing).

-ta (*–testing_area*) No search is made and main program line is switched to the testing area, therefore program won't give any reasonable results (testing).

-ps (*–pruned_sgf*) Sgf output is pruned significantly. If attacker moves first, only variants ending with death of a defenders group or with a ko are included in sgf output. If defenders moves first, only variants ending with live of his group or with a ko are included.

-mp (*–multiple_pos_mode*) Switches on multiple position mode. Instead of one input position given after *-i* option program expects list of positions (path to single position on single line). This is useful for statistic testing and comparison with other programs. Naturally sgf output is generated for every problem into adequate file (transformed input filename).

-ntt (*–no_t_tables*) Deactivates transposition table (slows down the search). Transposition tables cause repeated positions in search to be solved in instant time from cached previous results thus shortening search time a lot.

-nst (*–no_save_t_tables*) Deactivates saving transposition tables afrer first search (slows down the search). Saving transposition tables after first part of search (i.e. defender to move in the position) speeds up second part of the search because of larger information base. When either *-af* or *-df* are switched (not both together) then saving transposition tables has naurally no effect.

-id (*–iterative_deepening*) Activates iterative deepening depth first search (slows down the search). This is rather obsolete (in TGA) method for solving under the stones problems. Generally problems are solved in longer time with this option.

-ndh (*–no_dynamic_heuristics*) Prevents using dynamic heuristics. This option has mainly testing purposes (testing).

-nsh (*–no_static_heuristics*) Prevents using static heuristics. This option has mainly testing purposes (testing).

-nus (*–no_under_the_stones*) No under the stones analysis is performed and algorithm is prevented to play under larger captured blocks of defender's stones. This might cause some problems to be solved incorectly (testing).

-od (*–sgf_output_depth*) `number` No sgf output is produced when actual position is (in the search tree) deeper than `number`. This helps to significantly reduce sgf output in very large problems. Also this option is useful when user is interested only in several first moves of the solution.

-h (*–help*) Shows short help about program input and options.

Program's variables are implicitly set to perform the most effective search (all following speeding mechanism's are used: alpha beta , transposition tables, transposition tables saving, dynamic and static heuristics). Most often are used following cofigurations of input parameters.

./tga -i `problem_name` Classical cofiguration with all speeding mechanisms switched on.

./tga -ps -i `problem_name` Here moreover pruned sgf output is demanded (for larger problems).

./tga -ps -od 10 -mp -i `problem_list` Here program is in multiple position mode and every single sgf output is pruned and it's depth is limited by constants 10.

3.2 Input sgf file

As an input file for the search (after *-i* option), legal sgf (see [3]) file is expected. Moreover there are special requirements on the input format. Algorithm expects given Go position to be on the Go board of 19x19 size in the root node of sgf tree. That means that position must be created (by editing stones positions) not played out (as it is usual in the game). It is moreover expected that black's stones represent attacker and white's defender. Also following marks are required :

Square to mark empty coordinates on which are both defender and attacker allowed to play and also to mark attacker's unsafe stones i.e. stones that might be captured by the defender (non marked attacker's stones are expected to be safe).

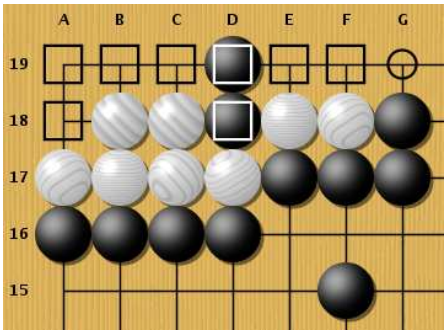

Figure 1: Example of input position

Circle to mark empty coordinates on which is attacker allowed to play but defender is not. By these are meant coordinates on the "edge" of the problem (see Figure 1).

Moreover some comments (according to sgf format) might be added to the problem (in the root node), these will be included in the output. Example of input position is shown in Figure 1.

Sgf source of problem given in Figure 1:

```
(;GM[1]FF[4]CA[UTF-8]AP[CGoban:2]ST[2]
RU[Japanese]SZ[11]KM[0.00]
CR[ga]AW[bb][cb][eb][fb][ac][bc][cc][dc]
AB[da][db][gb][ec][fc][gc][ad][bd][cd][dd][fe]
SQ[aa][ba][ca][da][ea][fa][ab][db])
```

4 Output

4.1 Output sgf files

Besides testing messages which are switched off for usual user, program generates short informing messages. Before the search shows information stored at the problem as comments. After the search informs user of a result of the search, number of searched nodes and time taken in the search. The most important output of the search are output sgf files. Names of output sgf files is generated from transformed input position filename ('/' and '.' are transformed to '_') by adding suffix *at.sgf* (solution with attacker to move) or *de.sgf* (solution with defender to move). In every sgf node there is a short information about depth and status of the node. In nodes where transposition tables cutoff was performed (same position in transposition tables was found) or repetition cutoff was performed (same position along the path to actual node was found) there is a short information about this action. If *-to* is switched on node's description are longer containing information about e.g. potential eyes, possible ko. When *-ps* option is switched on, sgf output is pruned to show only relevant information. Thus when attacker moves first all nodes resulting in the life of the group are pruned and on the other hand when defender moves first all nodes proving the death of the group

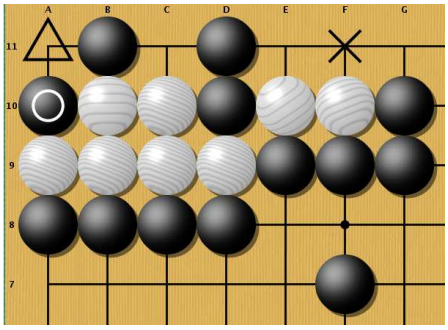

Figure 2: Position in the sgf output.

are pruned. This restricts the number of nodes a lot and together with marking the output and possible output depth limitation (option *-od* number) significantly increases output's legibility. There are three marks used in sgf output:

Circle marks the last played move.

Cross marks the opponent's moves answering last played move.

Triangle marks the coordinate where ko arose in the last turn, therefore opponent cannot play there now.

Example of position in the single node in the sgf output is shown in Figure 2.

References

[1] Tomas Kozelek. Life and death solver in go. Bachelor's thesis, 2006.

[2] International server dedicated to the game of go. http://senseis.xmp.net/.

[3] Official specification of sgf format. www.red-bean.com/sgf.