Documentation for Interestingness

Matt Johnson

Version History

2006.12.29 Version 1.0 Initial Document

Contents

1 Motivation and Introduction

Some games are more interesting than others. It's easy to figure out 1,2,...10, and once the trick is learned, forcing a win (or recognizing a loss) is trivial and boring. Within a game, some positions are more interesting than others. An Achi position with remoteness 1 is certainly a less-than-intriguing place to start, but other Achi positions (with higher remoteness) can be more interesting. We want to use Gamesman to explore the interestingness of games and positions.

We defined an interestingness metric and created a simple solver to assign interestingness values to positions. Interestingness is a scalar value defined at each position in a game based on the graph structure around that position. It is intended to measure how interesting a game would be to a human if played from a particular position. Interestingness can also be used to generate "Mate-In-N" puzzles, in which a human player is presented with an interesting winning position of remoteness N.

2 Defining Interestingness

2.1 Criteria to Consider

We believe interestingness should measure the graph complexity around a position. It should increase somehow with the number of moves (branching factor) at a position and also with a position's remoteness. Our basic principle is that the winning path should be "hidden" from the player; it should not be trivial to select the best move, or (similarly) to evaluate the children of a position.

It is clear that interestingness should be defined at winning positions because a human player would be interested in finding the hard-to-see winning move sequence (as in Mate-In-N puzzles). However, a human may be bored at a lose position against the perfect computer; there exists no winning move from the position, so whatever the human does s/he will lose the game. Still, interestingness should be defined at lose positions to provide complexity information for the accumulated interestingness at win positions; we use lose position interestingness to "bubble up" information about the interestingness of subsequent win positions (the rest of the "Mate-In-N" puzzle's position sequence). The lose position interestingness can again be thought of as a measure of complexity.

We came up with several criteria for the interestingness metric:

- Should increase with the proportion of "misstep" moves to the total number of moves at a position.
- For a win, should increase with interestingness of lose children. We might want to only consider a subset of lose children, such as only the lowest remoteness ones that would be relevant to "Mate-In-N," but we want all lose children to be complex so that they are hard to evaluate by eye.
- A win's interestingness cannot depend on its win children since those dependencies could be circular in loopy games.
- A lose position interestingness should capture the interestingness of its win children, so it can be something like an average, but it could be over all win children or a subset (such as lowest remoteness, which would make sense for "Mate-In-N").
- Should generally increase with remoteness so that longer chains of interesting positions are more interesting.
- A boring position could lead to an interesting sequence of positions. The sequence including the boring position should have lower interestingness than the shorter sequence, but the boring position should not be assigned a boring interestingness. Boring positions should still reflect some interestingness of subsequent positions.

2.2 Immediate and Accumulated Interestingness

Given the above criteria, we have can separate the definition into two parts:

- **Immediate Interestingness:** "How hard is it to find a winning move from this winning position?" Immediate interestingness only takes into account the moves available from a position (edges from a node) and not any interestingness of child positions, so thus it only measures the "immediate" complexity of the graph. Since missteps are only possible from win positions, this part of the definition only applies to win positions.
- **Accumulated Interestingness:** This part of the definition allows interestingness of children to contribute to the interestingness of the parent, which could be win- or lose-valued. In the context "Mate-In-N" puzzles, this factor would assign higher interestingness to a position that has interesting child (and grandchild, etc.) positions and thus select for puzzles that "stay interesting." However, we don't necessarily want interestingness always to increase with remoteness.

Both parts apply to win positions because immediate interestingness is only defined for win positions, which makes sense with our general idea that a position is interesting if it is hard to find the one (or few) winning move(s). Lose-valued positions only accumulate interestingness from their win children (for their win parents), and so immediate interestingness is not relevant to lose positions.

2.3 Interestingness Always Between 0 and 1

We keep interestingness normalized by maintaining all interestingnesses between 0 and 1. More accurately, we want to avoid both the 0 and 1 boundaries, so $Interestingness(p) \in [PRIMITIVE\_INTERESTINGNESS, 1)$ for all positions p. Keeping the interestingness confined to such an interval is also handy if we want to mix interestingnesses in products. We display interestingness as a percent (multiplying by 100).

3 Calculation

3.1 At a Primitive

Primitive positions are all equally boring, so they are set to a constant. This constant is the lowest interestingness possible, and is nonzero so as to build accumulated interestingness for parent positions (interestingness cannot be "killed off" in a product because we always keep it positive). Currently, the constant is set to 0.01.

The next sections discuss interestingness for non-primitive positions.

3.2 At a Win

The fundamental interestingness is the immediate interestingness at a non-primitive win. As mentioned before, we want it to be hard to choose the winning move, so we want it hidden in many other options. Thus the immediate interestingness is defined as:

$$ImmediateInterestingness(p) = \frac{W+T}{W+L+T}$$

Where W, L, and T represent the number of win, lose, and tie children of position p (recall that a winning *move* from a winning position is to a lose-valued child, so we want lower $L/(W+L+T)$, or equivalently higher $(W+T)/(W+L+T)$).

It was noticed that moves to simple winning positions are trivial to avoid, and thus they should not contribute to the above factor as they do not properly obscure winning moves. Thus, we take W in the above to be the count of winning child positions with remoteness greater than 1; winning children are only counted in immediate interestingness if they have sufficient remoteness. L and T children are counted regardless of remoteness, though the same argument could be made for ties counted in the numerator.

We also take into account the complexity of child positions through accumulated interestingness. To avoid circular definitions (win positions with another win as its child and parent in a loopy game), we only

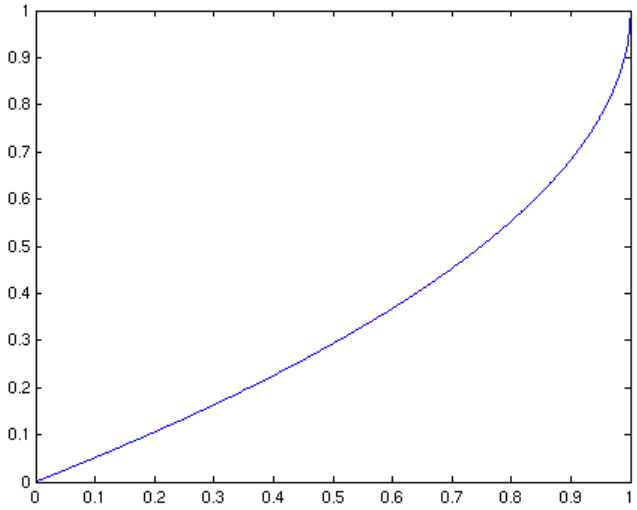

Figure 1: The function applied to "bend down" combined interestingness

consider the lose children (winning moves). Currently we take the arithmetic mean of the interestingnesses of the lose children:

$$AccumulatedInterestingness(p) = \frac{\sum_{c \in \{LoseChildren(p)\}} Interestingness(c)}{|\{LoseChildren(p)\}|}$$

We have considered other methods of accumulation, but in practice the mean has worked the best and it reflects the stated criteria well. It is also an option to take the mean only over a subset of the lose children (e.g. only the lowest remoteness lose children for "Mate-In-N"), but we chose take it over all children so as to maintain general interestingness of all subsequent player-winning games.

We want immediate and accumulated interestingness both to contribute to the total interestingness of a position. However, we want to keep the total normalized. Immediate and accumulated interestingness are combined in two steps:

$$CombinedInterestingness(p) = 1 - (1 - Immediate(p))\dot{(}1 - Accumulated(p))$$

This product allows the combined interestingness to stay normalized and allows both terms to "pull" the result towards the highest interestingness of 1. It is also equal to the sum minus the product. However, we don't want the interestingness to blow up too far, so we reduce the result by applying a function:

$$Interestingness(p) = 1 - \sqrt{1 - CombinedInterestingness(p)}$$

The above function was chosen because it is a simple function that "bends down" the combined interestingness while lowering higher values more than lower values. A plot of the function is in Figure 1.

Written in one step, we have interestingness defined in terms of immediate and accumulated:

$$Interestingness(p) = 1 - \sqrt{(1 - Immediate(p))\dot{(}1 - Accumulated(p))}$$

We found the above calculation to work well for combining immediate and accumulated interestingness while not allowing the result to "blow up" too quickly.

3.3 At a Lose

As mentioned before, non-primitive loses represent the interestingness of their win-valued children (recall that all of their children are wins). We again accumulate through an average.

$$AccumulatedInterestingness(p) = \frac{\sum_{c \in \{WinChildren(p)\}} Interestingness(c)}{|\{WinChildren(p)\}|}$$

3.4 At Ties and Draws

Ties and draws are inherently boring, and so we set their interestingnesses to be the primitive interestingness. We could also set them to be zero, since their interestingness values do not contribute to win or lose interestingness, but we chose to keep the primitive interestingness as the minimum for all positions.

4 Code and Implementation

The code for solving interestingness resides in **analysis.c**. There are two functions:

- **DetermineInterestingness**: In short, this function's job is to allocate a float array of size gNumberOfPositions, pass it to a helper function that sets up all the interestingnesses, and then store the desired information. This function is the wrapper that is called from the text UI. It does not do the interestingness solving, but instead calls a helper function with an allocated Interestingness array, which the helper function sets up to map each position to a float interestingness value (so after the helper is called, Interestingness[gInitialPosition] is the interestingness of the initial position, etc.). This function uses the array to store any information we want to retain about the interestingness, e.g. the most interesting position overall or a remoteness-based interestingness summary, and saves it to the gAnalysis struct (and thus the analysis database).
- **DetermineInterestingnessDFS**: This function is currently the "solver" for interestingness. It takes in a float array indexed by position and enters the interestingness values into the array using the above calculations. This "solver" function can only operate on games that have already been evaluated and have a database. It uses a simple DFS recursion to carry about the above calculations. It can operate on loopy games because it uses a visited bit to avoid infinite looping, and the definition stated above does not get into any trouble with loops.

The function **DetermineInterestingness** is called from **textui.c** in the text analysis menu. The file **textui.c** also has small code to set the initial position to be the most interesting position.

We calculate interestingness as a value between 0 and 1, but we print out interestingness to the user as a percent.

5 Text Interface

Interestingness can be accessed from the analysis submenu of the post-evaluation game menu. There are currently two options in the menu. Option 's' runs the interestingness solver and stores the required interestingness information. Option 'e' sets the initial position to the overall most interesting position. The latter option can only be executed after the interestingness solver is run, and is meant to test the validity of the metric and be a precursor to the "Mate-In-N" setup.

6 Room for Improvement

- The definition and calculation can certainly be polished and revised. In particular, the way we combine immediate and accumulated interestingness at win positions was empirical and can be improved. The calculation can always be made to reflect a human's perceived "interestingness" more accurately.
- The best way to find problems with the metric and to inform the revision process is to test its performance on games. There should be testing across all solvable gamesman games so that we can verify our interestingness value is reflecting real interestingness accurately.

- The code of the interestingness solver (especially DetermineInterestingnessDFS) was not optimized when written. That code could definitely be optimized for greater speed and possibly lower memory overhead.
- We have not yet added the "Mate-In-N" puzzle-finding feature, where a user would be able to choose a remoteness and have the initial position set to the most interesting position of that remoteness. This feature is an easy addition to the DetermineInterestingness function, and a remoteness-based "most interesting" table can be stored in the analysis struct/database.
- There should be an interestingness summary printed in the analysis menu, so that the results of the analysis solver can be easily viewed.