A static method for computing the score of a Go game.

Andrea Carta
http://www.micini.net/

Abstract

There are two approaches that may be used in order to let a software compute the score of a Go game. The most common is the so-called "dynamic" approach, implemented by every software capable of playing: "dynamic" means that further moves are played at the end of the game in order to capture any dead stone, to fill dame points and so on, until the score is easily computed just by counting the remaining empty points on the goban. The other one is the so-called "static" approach: "static" means that the final position is instead evaluated without playing any further move; for example, to determine if a group of stones is dead or alive, eyes, liberties and other properties are counted and properly processed by the evaluation algorithm. The static approach was first theorized in 2003 by professor Hyun-Soo Park [PLK03], who eventually developed a method [Par07] for computing the score of a game and claimed good results for the time — 2007 — but only described it in broad outline.

In this paper a similar, albeit more radical and extensive approach to the problem is shown instead: not only how to decide life or death for any group of stones is clearly explained, but also all the issues that often arise at the end of a game are addressed. Otherwise such issues — dame points, kos, seki, potential snapback positions, forced connections — could make it impossible to compute the score accurately. The algorithm, despite being very complex and certainly not suitable for any software capable of playing (which should make use of the dynamic approach instead), greatly helps the understanding of the basic patterns of the game; also, it's usually very fast as it does not require — as in the dynamic approach — to play further final moves.

1 Introduction

Computing the score at the end of a Go game, especially if close, is not a trivial matter, and the mere existence of "counting methods" proves that (see, for example https://senseis.xmp.net/?JapaneseCounting). Automatic computing is even more difficult, and that's why any software capable of playing implements the so-called "dynamic" approach, which consists of playing further moves after the end of a game in order to remove any dead stone, fill dame points and kos and eventually counting the remaining empty points on the goban in order to compute the exact score. Such a process may be long, as a lot of moves may be required to reach the proper final position (assuming the game has not been resigned too early). For example,

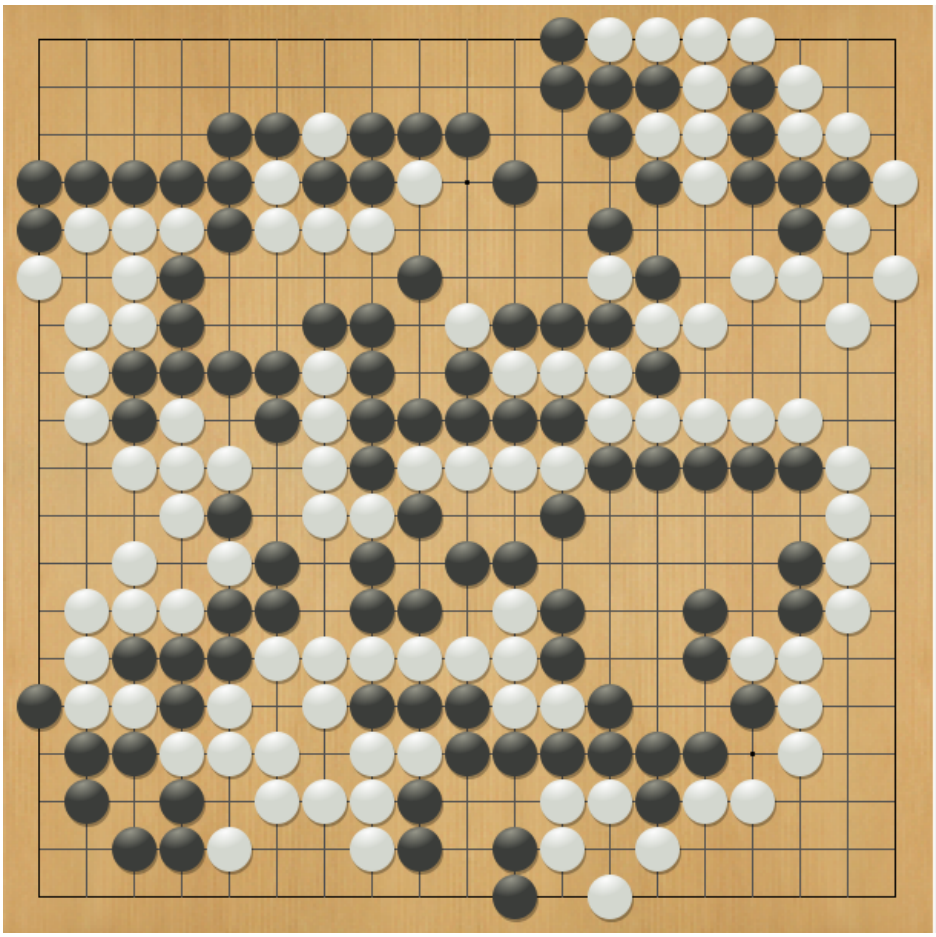

Figure 1: The final position of the 2nd game in the match AlphaGo vs. Lee Sedol.

the outcome of the second game of the match between AlphaGo and Lee Sedol — resigned by White after 211 moves — doesn't look so unreadable to the human eye (figure 1), yet GNU Go — whose scores are especially reliable — has to play 36 more moves before being able to compute the score. More further moves are played, more unlikely the score will be: it's easily verifiable that this game's GNU Go's evaluation — 6.5 points for Black — strongly depends on such moves, and could change if they were played in a different order (although the outcome of the game would not likely change). That's the reason why a "static" approach has been theorized many years ago by professor Hyun-Soo Park [PLK03], in order to avoid the uncertainty due to the moves played after the end of the game, and determine once and for all which stones are dead, how to count dame points and so on.

2 Limits of Park's approach

Professor Park's approach was based upon the following elements:

1. grouping the stones into "strings", that are stones physically connected: in figure 2 there are three black strings shown, numbered 1 to 3;

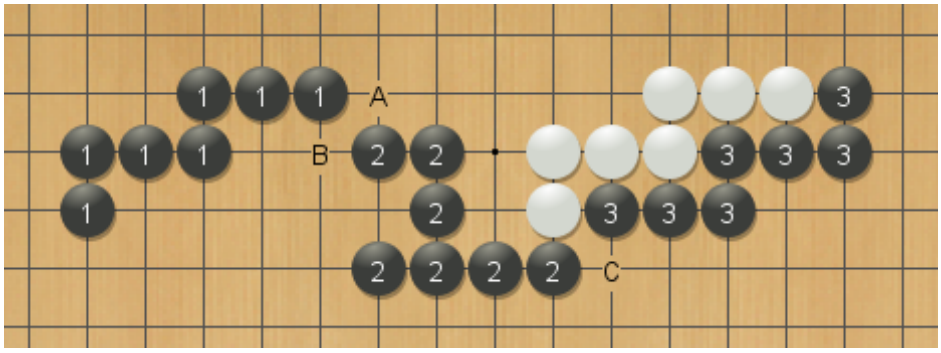

Figure 2: Three black strings.

2. counting strings' properties, such as eyes, eyelikes/special eyes (empty points almost surrounded by friendly stones and that could possibly become eyes), extension points, connection points (empty points between two strings that could be used to connect them), group territory (territory under the influence of the strings belonging to a "group", which is a collection of strings whose connection cannot be prevented, as in figure 2, where strings 1 and 2 can be connected anyway because if white A, black B and vice versa; strings 3 does not belong to the group

because White, unless the white stones may be killed, will play C and prevent the connection between strings 2 and 3);

3. assigning a score to each string, depending on the aforementioned properties; a huge table, listing all possible combinations of the properties and the resulting score (from 100 — certainly alive — to 508 —certainly dead) was provided: this score was named "stability" because, in a way, alive strings are "stable" and vice versa;

4. making use of a recursive algorithm (presented in a subsequent paper [Par07]) for computing the score of a game: with each pass of the algorithm the "deadest" strings are removed from the goban and the remaining ones' score is computed again (because removing a string usually improves the stability of the surrounding ones), until no dead strings remain. Eventually dame points are filled and the game's score is computed in the usual way, just by counting the remaining empty points on the goban.

Although remarkable, Park's work was far from exhaustive: the algorithm presented in [Par07] was just outlined, without the necessary details, and he never wrote any software despite claiming good results on a large number of games (362) found on the British Go Association's website[1]. He compared his algorithm's outcome — a mean error of 4.15 points — with CGoban's and HandTalk's, two of the most famous Go playing softwares back then, and was satisfied with what looked like a successful comparison (the softwares' mean error being 8.66 and 5.96 respectively). But not only this outcome is unsatisfying in absolute terms (an error of 4+ points is not negligible): the aforementioned lack of details makes it impossible to check the results and try to reproduce the outcome on a different set of games (for example the 624 games provided by Dave Dyer [Dye07] on which we tested the new algorithm). Even trying to guess the missing details is futile, as the same problem occurs before, when computing strings' stability: in [PLK03] a detailed method to determine the properties of a string is only provided for eyes (shown in figure 3), eyelikes and special eyes, and only works for eyes — of course the easiest property to determine — as the examples shown in the paper are far from clear and not free from inconsistencies. No method at all is provided for counting extension and connection points, and the examples shown are trivial (figure 4), with no enemy stones nearby disrupting the extensions/connections, as it's common during a game. Once again, no method is provided for the essential task of identifying the territory under

[1] Also, these games don't seem to be online anymore.

```
int IsEye(int posit, int string_id) // posit is EP point. and
string_id is the number of string id.
{ int x, y;
 int count;
 x = posit /100; y = posit % 100;
 count = 0;
 if ( y > 0){ if(BoardPoint[x][y-1].GetStringId(x,y-1) ==
string_id) count++;}
 else if ( y == 0) count++;
 if ( x < 18){ if (BoardPoint[x+1][y].GetStringId(x+1,y) ==
string_id) count++; }
 else if ( x == 18) count++;
 if ( y < 18){ if (BoardPoint[x][y+1].GetStringId(x,y+1) ==
string_id)  count++;}
 else if ( y == 18) count++;
 if ( x > 0){ if (BoardPoint[x-1][y].GetStringId(x-1,y) ==
string_id) count++;}
 else if ( x == 0) count++;
 if ( count == 4 ) return 1;
 else return 0;// not eye.
}
```

Figure 3: How to determine if a point on the goban is an eye.

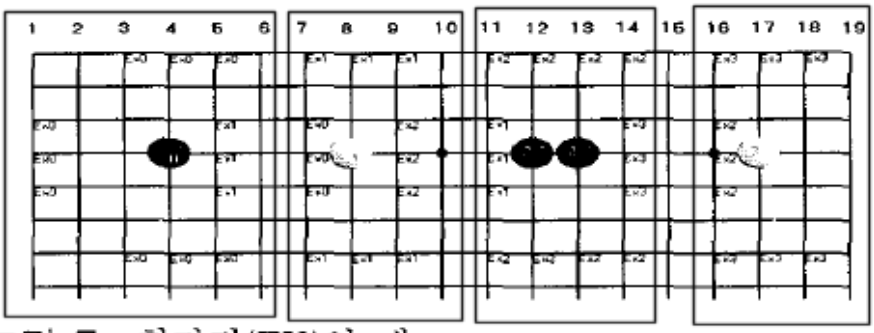

그림 7. 확장점(EX)의 예

Fig. 7. Example of extension-point(EX).

Figure 4: Extension points (for both black and white stones), with each influence zone outlined separately.

the influence of a group of stones: only some examples are shown, as in figure 5, and not much can be inferred from them. Last but not least, the huge table that should determine the stability score of a string given its properties is not that accurate: sometimes a match cannot be found, and occasionally two are found instead. These cases are sporadic, but occur nonetheless, and trying to improve the table to handle them is futile, as the criteria employed

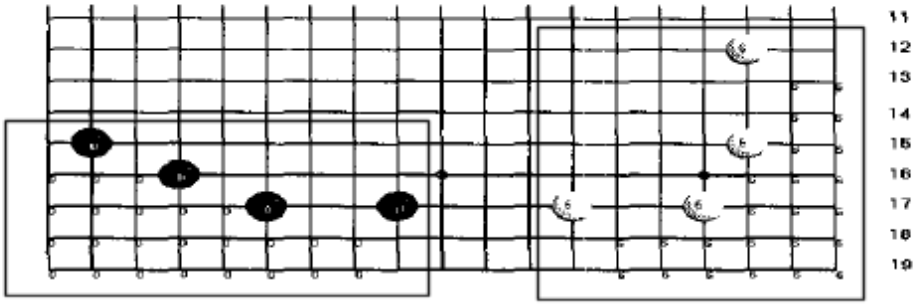

그림 10. 그룹영역(group territory)의 예
Fig. 10. Example of group territory.

Figure 5: Territory controlled by both groups, outlined separately.

in the first place to build it are far from clear.

Eventually the guesses required for writing a program capable of seriously testing Park's ideas are too many; and even guessing everything right would not be enough, as there are many yose's issues that Park never addressed: kos, stones remaining in atari, local snapback positions, forced connections, big eyes[2]. That's the reason why the mean error of his algorithm is more than 4 points. Only seki are addressed, as not only they often occur, but may also change dramatically the outcome of a game (Park wrote another paper notably addressing the seki issue: [PK05]). In the end, a real improvement is possible only if we start from scratch and keep only Park's basic ideas, as will be shown in the next section.

3 The new scoring algorithm

The first thing the algorithm does is to identify the strings: this is done by scanning the whole goban for stones, left to right, top to bottom, and each time a stone is found either it is added to a pre-existing string (if a stone of the same colour does exist either on the intersection immediately above or on the one immediately at left) or it becomes the first stone of a new string.

After that, all the strings are sorted into "groups", that are collections of strings not tangibly connected, but whose connection cannot be prevented. To determine if a connection cannot be prevented, it is important

[2] "Big eyes" are the known configurations including a lot of empty points, that may or may not become two eyes — the "rabbit ears", for example.

to check empty points between the strings involved, looking for "full connection points", as A/B in figure 6 (if White A, Black B and vice versa) as well as "half connection points", as C or D in figure 6 (if Black moves, C or D may be played connecting the two strings; if White moves, C or D may be occupied and the connection prevented). Figure 6 shows the simplest cases, but full/half connection points may have other shapes, for example involving two-points jumps and knight jumps, given that no — or few — enemy stones are found nearby and the points in-between are empty. The presence of either

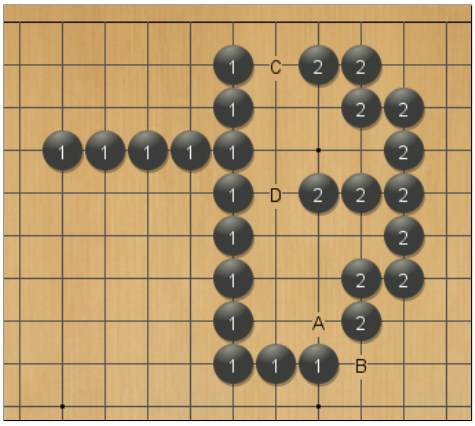

Figure 6: Full and half connection points for strings 1 and 2.

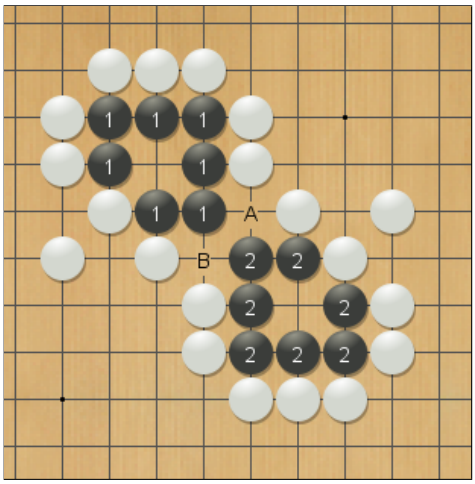

Figure 7: A group made of two strings that cannot be killed despite having only one eye.

a "full connection point" between two strings or at least two "half connection points" is a sufficient condition to consider the strings logically (albeit not tangibly) connected (in figure 6 both conditions are true); a "group" is a collection of strings that are logically connected, standing out as a single one. This idea, that the strings belonging to a group stand out as a single one — except for strings in atari — is a major improvement over professor Park's algorithm, that never contemplated groups except when computing the territory under their influence. Figure 7 shows why this is a major improvement: both black strings encompass a single eye and — according to Park — should be dead, but because of the full connection point at A/B are alive instead, as trying to kill them with White A would be met with Black B, and vice versa.

In the algorithm's next step the strings properties are computed. These properties are: eyes, special eyes, eyelikes (as in Park's algorithm), liberties, territory under the influence of the whole group. Liberties are easy to count,

but they are not all alike, for example:

1. liberties that the opponent cannot occupy without putting him/herself in atari, are worth twice;

2. liberties that can be occupied by a friendly stone without decreasing the overall number, are worth 1/3 more;

3. liberties that can be occupied by a friendly stone, thus increasing the overall number, are worth 1/2 more;

These subtle differences are important in life and death situations: two strings competing for life may have the same number of liberties and, at first glance, it might not be clear which one survives and which one dies; but as liberties are not all alike, the winner is often easy to determine, and sometimes it's not the one with more liberties. An example is shown in figure 8: at first, it looks like the black string (2) has got more liberties — 3 — than the white string (1) — just 2. But the two white liberties L1W and L2W are worth twice (a black stone there would be in atari), occupying L2W would not decrease the overall number, and occupying L1W would increase them: so these two liberties are worth about 5. The three black liberties, L1B, L2B and L3B only are worth 1/3 more, so about 4 overall. This means the black string dies first (as indeed it does).

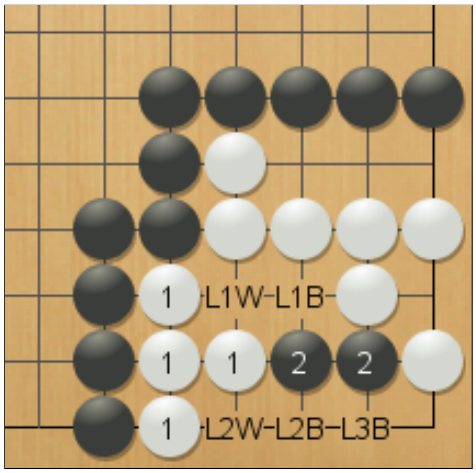

Figure 8: Liberties are not all alike.

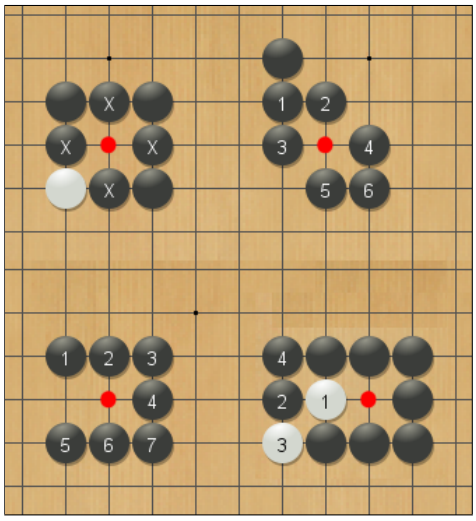

Figure 9: Several eyes.

Eyes are easy to count once it becomes clear how to identify them. Figure 9 shows some configurations: the simplest one (on the upper left) was

already pointed out by Park: if the 4 "cardinal points"[3] around an empty one belong to the same string, this point is an eye. A second configuration is shown on upper right: the potential eye is real if at least 6 out of the 8 surrounding points are occupied by friendly stones and the remaining two are empty corners (the eye belongs to all the strings possibly involved). On the lower left, the last configuration: 7 out of the 8 surrounding points are occupied by friendly stones, with the last one empty. The reason why this last configuration works (the others are obvious) is shown on the picture's lower right : if White 1, trying to steal the eye, then Black 2, and eventually White 3 is met by Black 4 and vice versa.

A special eye is a point that looks "almost like an eye": it occurs when at least 6 out of the 8 surrounding points are occupied by friendly stones, and at most one corner (not two) is occupied by an enemy stone. An eyelike is a point that "has got eye potential": it occurs when at least 5 out of the 8 surrounding points are occupied by friendly stones, and the remaining ones are empty.

But what is the purpose of special eyes and eyelikes? In some cases they may become true eyes, once the strings are sorted into groups and all their properties are summed up: for example, if two special eyes/eyelikes are contiguous, and at least one of them is a special eye, they become a true eye (for the whole group, not for the single strings); also, if three eyelikes are contiguous, they form a true eye too. To clarify the matter, an example is shown in figure 10: on the left there is a configuration with one special eye ("S", surrounded by 6 friendly stones) and one eyelike ("E", surrounded by 5 friendly stones), contiguous — so, a true eye; on the centre and on the right White does all he/she can in order to steal the eye, but fails; on the far right the stone marked with "X" is missing, so the special eye too becomes an eyelike, and as two contiguous eyelikes cannot become a true eye (three are needed), this time White succeeds.

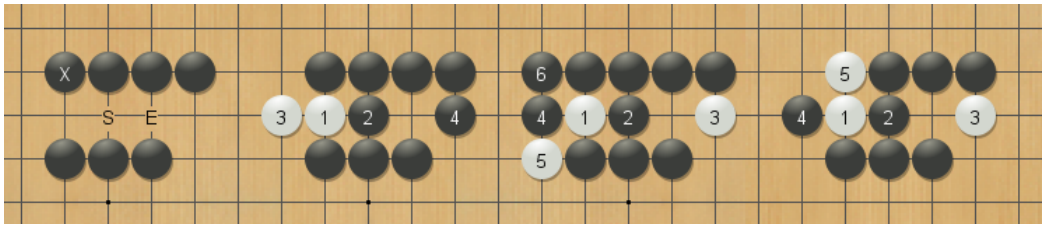

Figure 10: A special eye and an eyelike, contiguous, become a true eye; two contiguous eyelikes cannot.

[3] Given a point on the goban, the "cardinals" are the one immediately above, the one on the left, the one on the right, the one just underlying.

The territory under the influence of a group is computed by means of the famous Bouzy algorithm [Bou03], with 8 dilations and 21 erosions (instead of the more common 5 dilations/21 erosions), and it's quite a simple task.

Contrary to Park's method, connection and extension points are not needed, as they are of little help and are extremely difficult to identify with the necessary accuracy: the properties computed so far are all we need to determine the only thing that matters, that is if a group of strings includes two eyes or not. A group with two eyes is of course alive; if there's only one eye, but also some eyelikes/special eyes (not contiguous) and some territory the group is still alive. Even with no eyes, as long as a large territory (at least six points) is controlled, the group is still alive. When a group is alive, its stability is set to 100; when it's not, its stability is set to a value up to 520, depending on the overall number of eyes, liberties and territory (the exact formula is: 520 - eyes / 2 - liberties * 2 - territory / 2, much simpler than the huge table Park designed). Although a stability > 100 implies the group is dead, this is not always true, as the "deadest" groups are removed first, making it possible for the remaining ones to come back alive (see figure 11: both strings 1/2 look dead, with no eyes, but black's stability is 518, white's is 507, so the black string (1) is removed first, and the white one (2) becomes alive, having now got 5 contiguous special eyes, that are the equivalent of two true eyes. Thus, if Black A White B, killing the black string and leaving a big eye inside the white one).

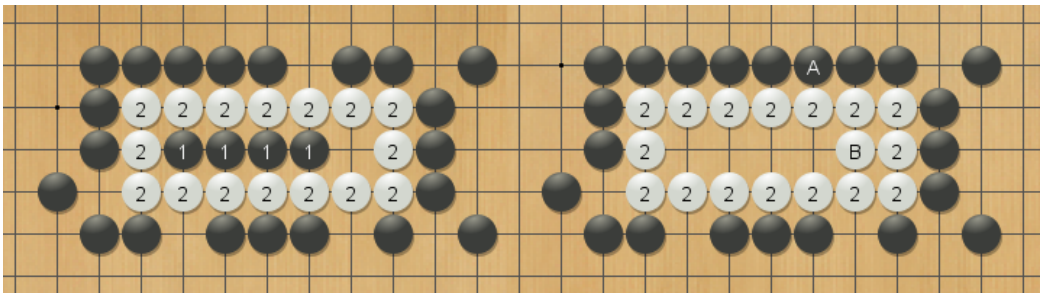

Figure 11: Both strings (1/2) are dead, but the inner one dies first, letting the other live, as shown on the right.

By the way, "deadest" strings cannot be removed at once: before doing such a thing seki and snap-back positions must be dealt with, as they may change the stability of the strings. Seki are not so easy to identify, even for the human eye, but they may be defined as follows: two strings sharing all their liberties (2 to 4, eyes do not count), with at least three stones each. Some examples are shown in figure 12: on the left, the inner string has only got two stones, so this is not a seki; on the right, the positions is very much

alike, but the stones are now three, and a seki occurs.

Snap-back positions involve two strings sharing their only liberty: of course both strings look dead, but if one of them cannot kill the other without being killed in turn then a "snap-back" occurs: a typical example is shown in figure 13, with the white string coming back to life after White 2 despite having being killed by Black 1. When a snap-back occurs, the surviving string will have its stability changed to 100. The same is true for seki, with both strings' stability set to 100.

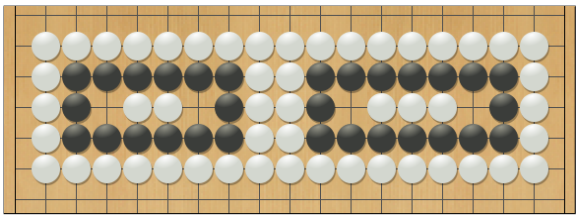

Figure 12: On the left, no seki; on the right, seki.

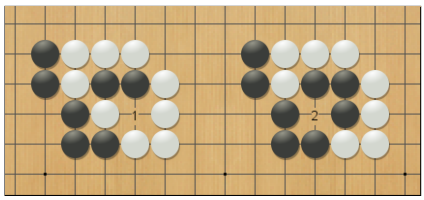

Figure 13: A snap-back position.

Only now it is possible to proceed with the removal of the "deadest" strings, starting with the ones whose stability is set to 520; if there are no such strings, the ones with stability 519 are removed and so on. After each passage the remaining strings' stability is computed again (and seki and snap-backs are again checked) and the removing process restarted, until there are no more strings to remove. At this point it is already possible to compute the score by means of the Bouzy algorithm, but the result would not be much better than Park's (who did not consider snap-backs) and in most games would be wrong, albeit not by much.

To really improve on the result it is mandatory to handle dame/ko filling and above all "forced connections". A forced connection is a point that cannot be left empty as otherwise, after all dame have been filled, either some friendly, alive stones would die or some enemy, dead stones would live: of course this is an issue that both players are well aware of when computing the score by hand and handle without too much thinking, even when leaving dame empty; but a score-counting software is another matter, and needs explicit guidelines, as forced connections are points that cannot be counted as territory, either filled or empty. A first example is shown in figure 14: the point marked with "D" is a dame and, once filled, the point marked with "F" should also be filled at once, to prevent the large black string marked with "1" to be killed. That's why "F" is called a forced connection. A second example is shown in figure 15: the white string marked with "X" is ostensibly dead (its stability is 518); but, should Black do nothing, will come back to

life, first with White 1, then with White 2. That's the reason why Black 2 is mandatory before it's too late, thus becoming a forced connection.

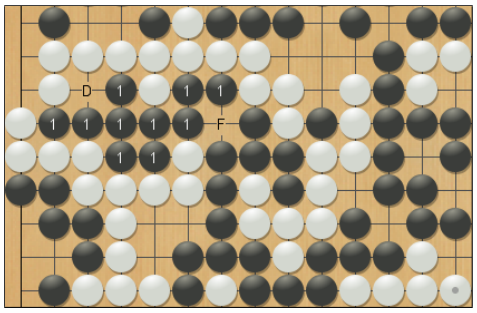

Figure 14: A forced connection, without which the black stones would be killed.

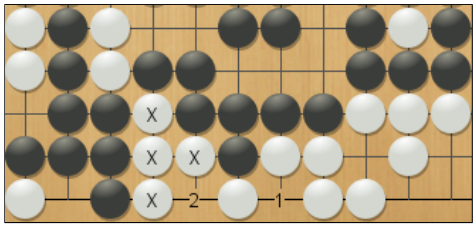

Figure 15: Another forced connection, without which the white stones would survive.

Forced connections are not always easy to see, even for a human player, and the Bouzy algorithm would consider them territory; also, they are the main reason why Park's algorithm, which ignores them, is not that accurate, with a mean error of 4+ points.

So, after having restored the original position — before the removal of dead strings — dame must be located. This task may be accomplished by means, once again, of the Bouzy algorithm: as dame points are not controlled by either player their "intensity" is exactly 0[4].

When all the dame have been located, they are filled with black stones, and the stability of all the strings on the goban is computed again: if some of them are found to be in atari, their last liberty is located and filled with a stone of the same colour: this point is marked as a potential "forced connection" (as in in figure 14). If the string remains in atari despite that, either the last liberty (of course a new one, acquired through the connection) is again filled or some dead enemy string is killed, if possible, in order to acquire more liberties. Either case the connection is found (unless the string in atari is dead, something that can happen only if its stability has been wrongly computed) and marked as potential "forced".

Once all the potential forced connections have been found, the task is repeated again, this time filling the dame with white stones: the forced connection points that are found a second time become real and are marked as such, as they need to be filled whoever occupies the dame. The forced

[4] The Bouzy algorithm computes a score for each empty point on the goban: if this score, called "intensity", is positive, the point is inside black territory; if negative, it's inside white territory. If 0, it's a dame.

connections that are found just once are not real: they may be left empty once the dame are filled with the right colour (something that, once more, the players are well aware of). As seen in figure 15 there are other forced connections to be found: the ones needed to avoid dead strings coming back to life. So the task is repeated once more, this time searching for strings whose stability changes to 100 after filling the dame; if that occurs, these strings need to be killed at once, and their last liberty becomes a forced connection (for the opposite colour). That's not all: although the forced connections do not alter the stability of the strings, they may cause new seki to surface: these cases are infrequent, but occur nonetheless, and after all forced connections have been found a new search for seki is needed. The algorithm's last task is the identification of kos, that are not as important as forced connections, but have to filled anyway: otherwise the Bouzy algorithm will count them as territory (in a way, they are not so different from forced connections). Each open ko is filled with a stone of the same colour, unless the resulting string remains in atari; should that occur, a forced connection saving the string is searched and, if not found, the ko is assigned to the opposite colour, and filled accordingly.

The whole task of looking for forced connections and kos is probably the most complicated part of the algorithm (and many further small details have been left out); not surprisingly, professor Park settled for a less accurate, indeed much simpler algorithm, still able to score most games reasonably well. But this task, although complicated and difficult to implement, is mandatory if we want the static method's outcome to match the dynamic's, as this last is obviously able to figure out, move after move, and subsequently handle, all the issues that take a static method all sort of calculations to be identified and correctly treated.

Once identified all connections, both forced and kos, the position after the removal of dead strings is restored and the strings in seki are reinstated; the Bouzy algorithm computes the score, then all connection points (forced and kos) are subtracted from respective territories/areas. Also, if handicap stones are present and scoring rules are chinese (area), their number must be added to the white area. The resulting score is usually correct, as we'll see in the last section.

4 Conclusion

The algorithm has been turned in a software routine capable of computing the score of a Go game after reconstructing the final position from

the SGF file; this routine is embedded in VideoKifu[5] and is called when the game analysis ends. The routine has been extensively tested on the so-called "Dyer's suite" (downloadable at http://www.real-me.net/ddyer/go/scored-games.zip), which includes 624 games played between japanese professionals. Dave Dyer, now retired, is an engineer specialised in graphical plugins and image manipulation; years ago he tried to write a software capable of playing Go games, but only completed the scoring routine, and tested it on a collection of 2000 games, of which 624 (the "Dyer's suite") have an exact score. This is probably the biggest collection of games suitable for such testing, even bigger than the one employed by Park in 2007[6]. Dyer's routine's results are not much detailed, nor his software is available for further testing. He claims that he *"can calculate the correct score for about 75%. In the other 25%, the most common errors are to be off by 1 point due to a fine point of endgame play. Occasionally, it turns out that territories aren't really final, just determined in the eyes of the pros. Gross errors, where tsumego is judged incorrectly and a wildly inaccurate score is calculated, are very rare."*. We must remember that Dyer's method is dynamic, although many static ideas are present (for example, a "problem solver" determines which strings are alive and which ones are dead), so a very good result, such as the one he claims, is to be expected (but there is the drawback: the method it's very slow, taking several minutes on each game).

When our algorithm is tested on Dyer's suite, the results are as follows:
- 337 games (54%) are scored correctly;
- 223 games (36%) are wrong by 1 point;
- 50 games (8%) are wrong by 2 points;
- 7 games (1%) are wrong by 3 points;
- 7 games (1%) are wrong by 4 or more points.

The mean error is 0.83 points: not only this outcome is a huge improvement over Park's results, but it looks so good that it might be possible to employ the algorithm for professional purposes, such as publicly scoring important games and analyse their subtleties. The reason behind the residual error is hinted in Dyer's comment *"a fine point of endgame play... territories aren't final, just determined in the eyes of the pros"*, and an example will clarify the matter. Figure 16 shows the final position of one of Dyer's games, whose official score is B+4. According to the algorithm the score should be B+3 instead: that's because the position in the corner, as depicted in the figure, is not that clear, although the two black strings 1/2 are alive and the two white strings 3/4 are dead (string (4) is even in atari). But, should ever

[5] http://www.oipaz.net/VideoKifu.html
[6] Out of curiosity, Dyer too wrote his routine in 2007.

dame D be filled, the two black strings will die instead: so two forced black connections are needed, both in B1 and, in order to avoid a snap-back, in B2. However, filling dame D would in turn endanger the white stones in the corner, so White too has to force connect in W1 and W2. The algorithm correctly understands the situation, thus subtracting two points from both territories, settling for a final score of B+3. So why did the players agree for B+4? Because they simplified matters, deeming alive the two black strings without dealing with all the forced connections except for the one in W2, too manifest to be ignored. So in the end this point alone was subtracted from white territory, setting the score at B+4. Who is right? The algorithm

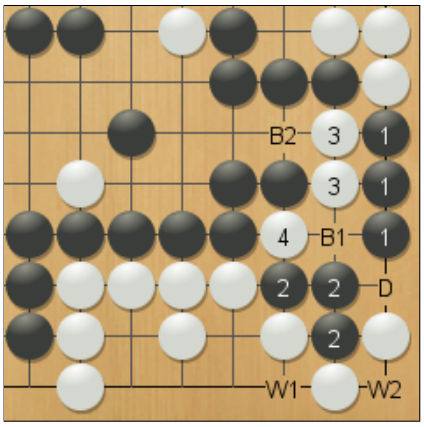

Figure 16: A situation the players evaluated differently.

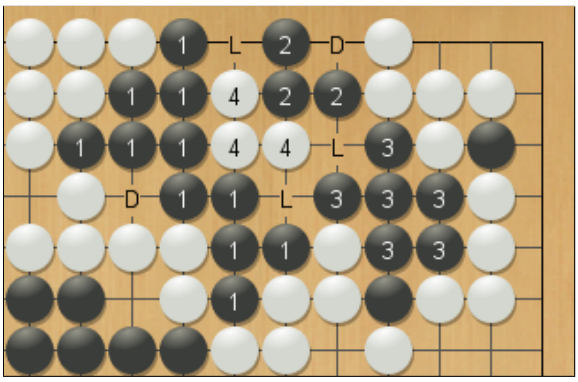

Figure 17: A situation the algorithm fails to evaluate correctly.

or the players? An argument can be made for both cases; Dyer, as it's now clear, thought the players were not taking too seriously the matter of forced connections, and from time to time settled for a questionable score as long as the winner stayed the same (this case being one of many). What matters here is that this is the main cause of the aforementioned mean error of 0.83 points, and it's something that cannot be avoided, as it depends on a different judgement of the position by the players, not on some flaws in the algorithm. There are only 5 games, out of 624, where the algorithm is really wrong: the reasons vary, but the most common is the presence of "multistring seki". What they are is shown in figure 17: black strings 1/2/3, along with white string (4), share 3 inner liberties (L) and — should the outer dame (D) be filled — become part of an enormous seki. As this seki involves more than the usual two strings, it is difficult to identify even for the human player, and it would become exceedingly complicated for the algorithm to detect (and the algorithm is already anything but simple); so, being an extremely rare occurrence, it has not been handled. By the way, the technique the algorithm

employs at the moment for seki's identification could be improved, and it's likely that sooner or later this issue will be fixed[7]: in that case the mean error will decrease at 0.71 points, a value difficult to achieve even by means of the dynamic approach. Also, as pointed at the beginning, the algorithm is fast (3 to 4 seconds for game[8]) and greatly helps to understand the basic patterns of the game — what determines life and death, when connections are mandatory and so on.

References

[Bou03] Bruno Bouzy. Mathematical morphology applied to computer go. *Internationatl Journal of Pattern Recognition and Artificial Intelligence*, 17(2):257–268, March 2003. http://www.mi.parisdescartes.fr/~bouzy/publications/Bouzy-IJPRAI.pdf

[Dye07] Dave Dyer. Scoring Completed Games, November 2007. http://www.real-me.net/ddyer/go/scoring-games.html

[NKM05] Xiaozhen Niu, Akihiro Kishimoto, and Martin Mueller. Recognizing Seki in Computer Go. *Lectures Notes in Computer Science*, 4250:88–103, 2005.

[Par07] Hyun-Soo Park. Score-Counting Algorithm for Computer Go. *Journal of the Institute of Electronics Engineers of Korea*, 44(1):49–55, December 2007.

[PK05] Hyun-Soo Park and Kyung-Woo Kang. Evaluation of Strings in Computer Go Using Articulation Points Check and Seki Judgment. *Lecture Notes on Artificial Intelligence*, pages 197–206, 2005.

[PLK03] Hyun-Soo Park, Doo Han Lee, and Hang Joon Kim. Static Analysis of String Stability and Group Territory in Computer Go. *Journal of the Institute of Electronics Engineers of Korea*, 40(6):392–402, November 2003.

[7] Although it's a very difficult issue, as shown in [NKM05]

[8] On a PC equipped with an Intel® Core™ i5-2500 CPU @ 3.30GHz.