Soft and Hard Connectivity in Go

Keh-Hsun Chen
Department of Computer Science
University of North Carolina, Charlotte, NC 28223, USA
e-mail: chen@uncc.edu

Abstract

We study connectivity of stones on a Go Board, including influence and heuristics based soft connectivity and pattern and search based hard connectivity. Stone connectivity provides a Go program with an understanding of a fundamental aspect of board configurations. It is essential in positional evaluation, candidate move generation, and move selection.

Keywords: Computer Go, Heuristics, Patterns, Search

1. Introduction

Connectivity is an important aspect of the game of Go. Connectivity gives the stones on the Go board vitality and strength, so they can engage in fighting and pursuing territories. There are two types of connectivity in Go – soft connectivity and hard connectivity. Two or more blocks of stones of one color are said to have hard connection if no matter how the opponent tries to disconnect, the player of the color is able to connect them into one block. Two or more blocks of stones of one color are said to have soft connection if the opponent may be able to cut them but has to pay a higher price in doing so than the gain from the separation. We discuss influence based soft connectivity and its heuristic refinements in Section 2, a set of basic rules for hard connectivity in Section 3, pattern based soft/hard connectivity in section 4, and search based hard connectivity in section 5. The paper concludes with final remarks in section 6 and references in section 7.

2. Soft Connectivity

At opening and early mid-game stages of a Go game, there are many softly connected blocks on the board. Since cutting them usually cost the cutter a higher price than the gain, it is safe to use soft connectivity to identify groups in the first half of a Go game. A common example is a single space jump in the middle of the board as in Fig. 1.

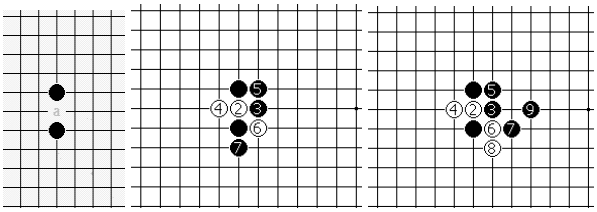

Fig. 1 Two Black stones form a single space jump, if White tries to cut, Blacks has plenty of options to choose a favorable response – each of the last two figures can be produced in 4 different orientations. Black can make a most favorable choice according to the surrounding situation.

If White play a stone at a, the two stones can be separated. But Black can choose from large number of options to fight or to exchange peacefully. Rarely cutting in the center of a single space jump is advantageous to the cutter, so we can practically consider the two stones as one unit when there are no opponent stones close by. In subsection 2.1, we shall discuss an influence-based connectivity recognition, in section 2.2 ~ 2.5, we shall discuss three heuristic refinements to this soft connection recognition.

2.1 Recognizing Soft Connectivity through Stone Influences

Every live stone on the board radiates influences across the board. This influence is at maximum at its immediate neighboring spaces and it decays as distance increases [2]. Black and White stones radiate influences of opposite sign. These influences of stones at empty grid points are accumulated algebraically. The accumulation of stone influences creates potential fields on the board. Two blocks of stones are considered to have soft connection if there is a path on the empty spaces of board between the two blocks along which the cumulative influence is above some threshold. For example, Fig. 3 shows the sum of influences of the black and white stones in Fig.2. If we use 35 as the threshold, then the program recognizes some soft connections in Fig. 4 – stones marked by the same number are considered to have soft connection among them.

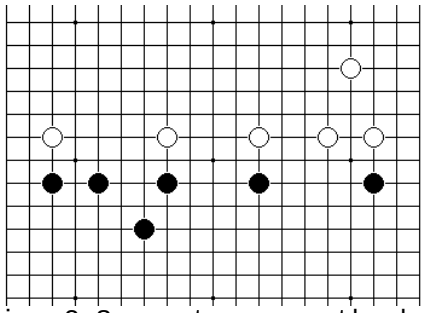

Fig. 2 Some stones on the board

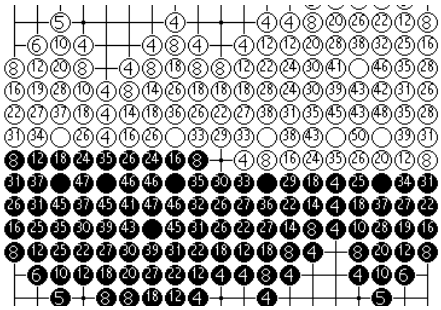

Fig. 3 The cumulative influences at neighboring empty grids.

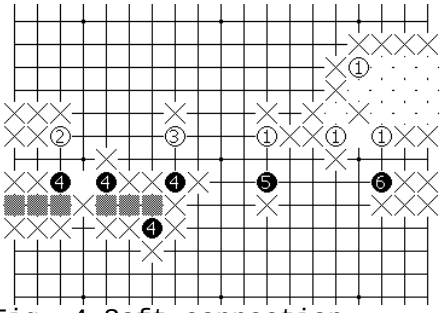

Fig. 4 Soft connection recognition

2.2 Boundary Adjustment

Since the spaces near the border are easier to be influenced by stones in the neighborhood, we factor up (up to 60%) the net influence values on the first three lines along the border. Fig. 5 shows the two black stones on line 2 are considered softly connected since the influence of the two empty points between then have high enough influences after factoring up; but the two black stones near the center are considered separated since the two empty spaces between them do not have high enough cumulative influences.

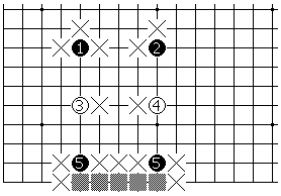

Fig. 5 Effect of boundary adjustment

2.3 Neutralizing Empty Point Adjacent to Live Stones of Both Colors

If an empty board point is adjacent to live stones of both sides then no connectivity can be established through that point regardless its cumulative influence value. Fig. 6 provides an example.

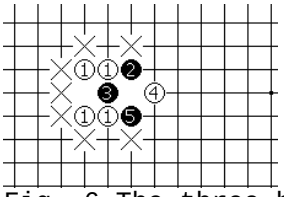

Fig. 6 The three black stones are not considered to have soft link since the strong cumulative influence in black's favor at their common liberties is neutralized by the white stone marked 4.

2.4 Neutralizing Articulation Points

An empty point on the board is called an articulation point if it has 4 empty neighbors and the point itself has a cumulative influence above the threshold but either both the empty points right above and below it have lower than threshold cumulative influences or both the empty points immediately to its right and to its left have such low influences. Usually an articulation point can be attacked and two sides of it will be separated, so we set its cumulative influence to 0 for linkage recognition purpose to avoid bad recognition of soft connection. Fig. 7 shows that point a between two black stones is an articulation point. After resetting the cumulative influence to 0, the two sides of it are considered two separate groups as in Fig. 8.

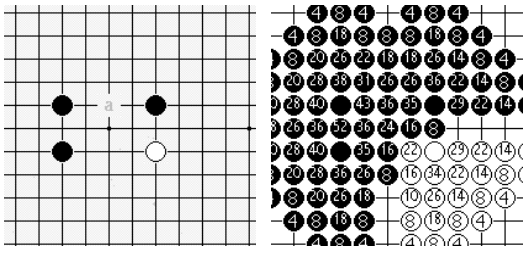

Fig. 7 Influences of the four stones shows that there is a soft link between two top black stones.

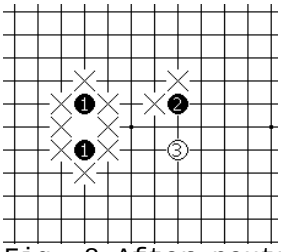

Fig. 8 After neutralizing the articulation point, the program no longer considers top two black stones are linked

2.5 Neutralizing Opponent's Sente Points

An Empty point of the board is said to be a sente point for one side if the opponent must reply to it to avoid a heavy loss, for example a move threatening to capture a valuable block of the opponent's stones is usually sente. For a sente point with open escape path, its cumulative influence should be reset to 0 to reflect the reality that it does not provide connection for the opponent. For example, in Fig. 9, c is a sente point for White, so the Black blocks are not softly connected. If a neutralized point is on line 2 and the point underneath it on line one is also an empty point, then that empty point should also be neutralized.

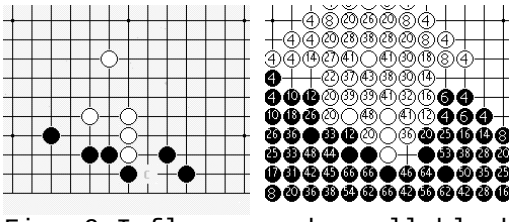

Fig. 9 Influences show all black stones are softly connected.

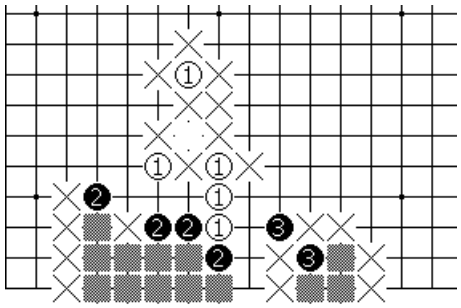

Fig. 10 Correct connectivity recognition by the program after the sente point is neutralized.

3. Basic Connection Rules

Some hard connectivity can be recognized through simple basic connection rules (BCRs).

BCR1. Two blocks sharing two or more common liberties are connected.

BCR2. Two blocks sharing one protected liberty are connected.

BCR3. Blocks adjacent to a dead opponent block are connected.

Some examples are shown in Fig. 11. Stones marked with the same number are considered connected via basic connection rules.

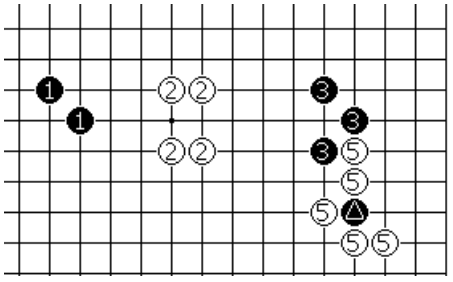

Fig. 11 Hard connectivity recognized via basic rules.

Hard connectivity is not transitive. We shall discuss this issue in section 5.

4. Connection Patterns

Both soft and hard connectivity can be recognized through pattern matching with connection pattern libraries. Fig. 12 provides two examples of hard connection pattern, where a square represents a black stone or an empty space.

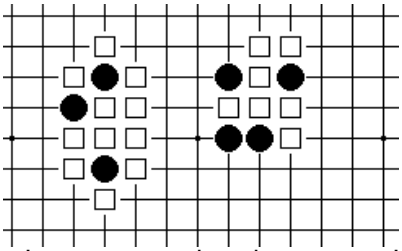

Fig. 12 Two hard connection patterns

5. Connection Search

Heuristics and Patterns can only cover a small portion of connectivity problems. The majority of hard

connectivity needs to be settled through search. On an unlimited size board, theoretically we could embed any hard life-and–death problem as part of a connection problem. Hence we expect the stone connectivity problem in nXn Go is also P-space hard and exponential time complete, just as nXn Go itself. Experience shows that tactic searches in Go, other than the ladder which has a branching factor near 1, do not allow the luxury of complete and result-guarantee techniques in practice [3], or one would get only trivial or restricted cases solved in real time. In the following subsections, we shall describe a very fast and effective heuristic connection search technique.

5.1 Connection Distance

In the heart of the proposed connection search is an elaborate connection distance recognition procedure, which will be used as the evaluation function and the foundation for candidate move generation for the connection search. The procedure tries to find the number of moves needed to link two blocks into a chain by the connection side without any interference from the opponent.

We count the length of a path joining two grid points as follows: if there is already a chain, [2, 3], of our color on the board, going through the chain is free. Moving to an adjacent empty point or a diagonal adjacent empty point with two connection options: if the point is protected, it is free, otherwise the algorithm increases the connection distance by 1. Moving through dead opponent stone is free. Moving through an opponent block that can be captured via a ladder, it increases the distance by 1, otherwise it increases the distance by the number of liberties of the opponent block. Fig. 13 shows the connection distance from the lower leftmost black stone to upper right black stone is 3.

5.2 Shortest Paths and candidate moves

All shortest connection paths between two connection targets can be found using a breadth-first search algorithm. We first mark grid points starting with source as 0, explore all immediate neighbors and diagonal neighbors with two join options, marking them using the scheme described in Section 5.1. To avoid exploring in wrong directions, we can restrict the BFS to the rectangle with just one extra line extension in each direction to the min. rectangle containing the two connection targets. We can trace back from the destination toward source on the marked board to find all shortest paths between the connection targets, see Fig. 13. We can use the empty points along the shortest paths as candidate moves for both sides in the connection search.

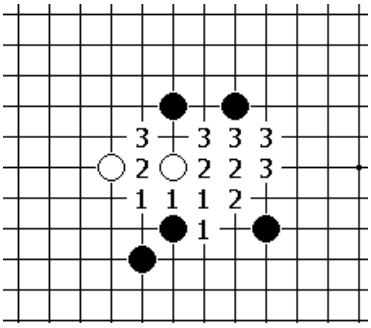

Fig. 13 Connection distance and shortest paths

5.3 Connection Search Algorithm

An iterative deepening strategy is used to control the search depth on the following basic recursive connection search procedure based on a neg-max version of alpha-beta:

```
Short int ConnectSearch (BDPOINT p1, BDPOINT
            p2, short int distLimit, BDPOINT *
            bestMove, LIST * bestSequence, short
            int level, short int alpha, short int beta,
            short int *depth) {

    short int max, eval, dep, staticEval, kt;
    LIST moves, bestSeq;
    BDPOINT p, pBest;

    moves = Empty;
    bestSeq = Empty;
    pBest = Empty;
    *bestMove = Empty;
    Dispose(bestSequence);
    if ((Board[p1] == Empty) || (Board[p2] ==
                Empty)) staticEval = 99;
    else staticEval = ConnectDistance(p1, p2,
                Board[p1], distLimit, Ds,
                level);
    if (ToPlay == Board[p1]) staticEval = -staticEval;
    if ((level >= CnLevelLimit) || (abs(staticEval) >=
    distLimit) || (staticEval == 0) && (level > 1)) {
            *depth = 0;
            Dispose(bestSequence);
            return staticEval;
    }
    Copy(ConnectCandidateMvs, &moves);
    if (moves == Empty) {
            *depth = 0;
            return staticEval;
    }

    max = alpha;
    *depth = 0;
```

```
    kt = 0;
    while (moves != Empty) {
        Pop(&p, &moves);
        kt++;
        if (ExecuteConnect(p)) {
            eval = - ConnectSearch(p1, p2,
            distLimit, &pBest, &bestSeq,
            level+1, -beta, -max, &dep);
            UndoMoveAndKo();
            if (eval > max) {
                max = eval;
                *depth = dep+1;
                *bestMove = p;
                Dispose(bestSequence);
                *bestSequence = bestSeq;
                Push(p, bestSequence);
                bestSeq = Empty;
            }
            if ((level > 1) && (max >= beta))
        Dispose(&moves);
            if (kt >= Maxi(16-level, 2))
        Dispose(&moves);
        }
    }
    Dispose(&bestSeq);
    return max;
}
```

5. 4 Non-transitivity Issue

It is well known that the hard connectivity is not transitive in Go [1, 2]. For example, In Fig. 14, Black stone marked by triangle and Black stone marked by circle are connected and Black stone marked by circle and Black stone marked by square are connected, but Black stone marked by triangle and Black stone marked by square's link can be broken. The connection distance procedure does not deal with this non-transitivity issue, so the evaluation function will return connection distance 0 for transitive basic connection. ConnectSearch procedure resolves this issue by requiring the search procedure to reach search depth at least two before it makes any conclusion. The connection search procedure is able to solve most of non-transtivity connection problems in Cazenave's test suite[1] in a fraction of a second on a 2 GHz Pentium PC, even though the procedure is not specially designed for the non-transitivity type connection problems. Fig. 15 shows ConnectSearch finds a way for White to cut the Black's link.

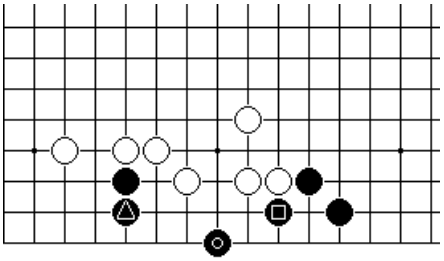

Fig. 14 Connectivity is not transitive.

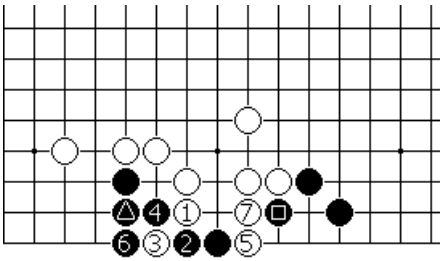

Fig. 15

6. Final Remarks

How to take advantage of the knowledge of connectivity in a Go match by a Go program is not an easy task. Attacking opponent's connection link may not be advantageous – that why many soft links should be left alone unattacked. Good global positional judgment by the program is essential to make good use of a sharp tactic.

7. References

[1] T. Cazenave and B. Helmstetter, Search for transitive connections, Information Sciences, accepted, 2004. http://www.ai.univ-paris8.fr/~cazenave/PapersYears.html.

[2] K. Chen, Group Identification in Computer Go, Go chapter in the book "Heuristic Programming in Artificial Intelligence" edited by D. Levy & D. Beal, 195-210, Ellis Horwood, Fall 1989.

[3] K. Chen, Computer Go: Knowledge, Search, and Move Decision, ICGA Journal, Vol. 24, No. 4, 203-215, December 2001.