GROUP SIZE COUNTING PROBLEM IN COMPUTER GO HAS A WAY TO AUTONOMOUSLY COUNT THE SIZE OF GRAPHS IN A GRID WORLD

ABSTRACT
Computer Go has been a new unconquered challenge to the Artificial Intelligence (AI) and Computational Intelligence (CI) societies since 1997 when Garry Kasparov, the human Chess champion, was defeated by the IBM supercomputer Deep Blue. Computer Go is not only an unsolved problem by itself, but also serves as an excellent test bed for a novel technique to apply it to a practical problem since Go’s clear-cut objective and the vast amount of sub-problems help to test out the novelty’s practicality. We propose a novel, distributed approach to count the size of groups of connected stones on a Go board. This scheme guarantees to answer the correct group size on a Go board in an autonomous and scalable manner. We also discover the solution in computer Go works to autonomously count the size of graphs in a grid world as well.

1. INTRODUCTION
Computer Go [1,2] is a field of study created to develop a computer intelligence that generates a sequence of moves that successfully plays a game of Go. Go, also called Baduk in Korea and WeiQi in China, is a two player board game of which the Western counterpart is Chess; it is as popular or perhaps more popular in East Asia than Chess in the Western world and is gaining more popularity in other regions. Proving the popularity in Asia, this game debuts in the sixteenth Asian Games that will be held in Guangzhou, China, November 2010. Computer Go’s technical significance is rediscovered in quest of a more complicated problem than Chess after IBM supercomputer Deep Blue defeated the human Chess Master Garry Kasparov in May 1997. It is far more complex than computer Chess; the game tree size of a 19x19 Go board is estimated to be between $10^{575}$ to $10^{630}$ [3,4] while that of a chess board is $10^{120}$. The implication is that computer Go cannot beat a human champion with the fastest computer to date. This fact makes computer Go as a novel test bed for computer intelligence. Currently, computer Go is considered as one of the most challenging unsolved problems in the Artificial Intelligence (AI) and Computational Intelligence (CI) societies.

Due to the huge scale of computer Go problem, the problem itself is divided into a large number of sub-problems. One of the sub-problems of our interest is to count the size of connected stones or a group of stones, which is essential and important information to judge the life and death of the group. Life and death of a group has a significant impact to the entire game of which the purpose is to gain more territories than the opponent. In this group size counting problem, the knowledge of the entire game rules is redundant, so only the relevant rules are covered in section 2 and 3.

Our purpose in this paper is to develop an algorithm that can solve the problem in an autonomous and scalable manner. Assume a 2D grid world of an unknown size that is filled with a large number of bi-directional graphs $G=\{V,E\}$ of various size and shapes. To visualize a grid world, take a Rastrigin’s function in Figure 1 as an example. While the side view of the function is like a terrain, the top view can be seen as a 2D grid world. Suppose a part of the grid world is filled with vertices on grids or intersections and each vertex allows up to four edges in cardinal directions. Further suppose these vertices are independent processing units that are connected to its adjacent vertices via edges. The rest of the grid world is either empty or no vertex is placed on a grid.

![Fig. 1 Rastrigin's function as an example of a grid world, (a) side view is a 3D world, (b) top view is a 2D world.](image-url)
2. AUTONOMOUS GRAPH SIZE COUNTING PROBLEM

This problem is best explained with a metaphor: imagine a person whose scope and communication range is limited to his or her immediate neighbors, all sitting on a terrain. In other words, this normal person does not possess the ability to look over the entire terrain. However, he/she wants to know how many related people are connected with him/her. In other words, this person wants to know the size of the same group of the same kind on the terrain.

This problem is similar to a task that soldiers in airborne troops jumped to a drop zone from aircrafts try to identify a cluster of soldiers from the same unit and the size of the cluster. Likewise, the same task can be done by a number of processing units such as robots and wireless sensors deployed in a certain region. It should be emphasized that each person is responsible for discovering the group he or she belongs to and the size of the group; therefore, the group size should be counted autonomously.

Fig. 2. Examples of graphs on a 19x19 grid world. (a) $G_{scattered}$ graphs of a single vertex are scattered around (3D), (b) $G_{spiral}$ basic spiral shape (3D), (c) $G_{cross,columns}$ basic shapes that complicates the problem (3D), row and column locates a graph. For example, the cross on the left bottom is denoted as $G_{c(1,1)}$, (d) (e) in a 2D grid world.

More specifically, a grid or a processing unit on a terrain, like ones in Figure 2.a, can exchange information only with the (immediate) neighbors if neighbors exist. The neighbors are the vertices in the four cardinal directions (North, South, West and East in this case). A vertex does not have access to the global information, but wants to count the size of a graph connected to it autonomously.

On the other hand, this graph size counting problem is easy when an arbiter in the sky has the top view of the region. However, lack of the arbiter makes the problem challenging. The challenges for an autonomous approach include: (1) guaranteeing the correctness of the answer or the graph size of any shape, (2) scaling the region of the interest, and (3) finding the answer within the bounded time limit. When the first and second challenges are rephrased together, finding a solution that guarantees to count the size of a bi-directional graph of any shape at any size is a challenge.

From our experience, one shape is easier and the other is more difficult to find the solution. Figure 2 illustrates various shapes of graphs. The simplest case is $G_{scattered}$ in Figure 2.a that graphs of a single vertex scattered all around the grid world. Each vertex looks around its adjacency to find out it is the only vertex. A simple spiral graph $G_{spiral}$ in Figure 2.b is one of the challenging basic shapes. This spiral graph has 199 vertices and 198 edges. Therefore, there are 198 hops between the first and the last vertices. Even with the correct set of procedures to count the graph size, not all vertices may find the correct answer if the procedures are stopped in slight doubt of the method before the time that spans all 198 hops.

Figure 2.c and 2.d illustrate some of more challenging basic shapes. Note Figure 2.d is a 2D version of Figure 2.c. just to explain a 3D grid world is not different from a 2D grid in our case. From now on, a 2D grid world will be employed for simplicity, but it can also represent a 3D grid world.

To begin with, a cross shape $G_{c(1,1)}$ is more challenging than a straight line because of an intersection. Say a solution for $G_{c(3,1)}$ is found; nonetheless, the solution may fail for a cross shape with a loop $G_{c(3,2)}$. A newer solution considering $G_{c(3,2)}$ may fail for a cross shape with multiple loops $G_{c(3,3)}$ because of the multiple loops. Furthermore, dealing with a larger loop in a square shape $G_{c(2,2)}$ is another story because all the autonomously counting vertices should stop the task once counting is done. On the other hand, a cross-in-a-square shape $G_{c(2,1)}$ can be more complicated than fully connected square shape in $G_{c(1,1)}$. The last basic shape in Figure 2 worth mentioning is a shape with multiple branches $G_{c(1,3)}$. To gain insights for the challenge, look at a disconnected square shape $G_{c(2,2)}$ first. With an incorrect solution, the upper right and the lower left parts of $G_{c(2,2)}$ may count separately. Back to the shape with multiple branches $G_{c(1,3)}$, the counting process becomes chaotic when each branch counts separately because a rule to merge them toward the correct answer is necessary.

It should be stressed that the shapes in Figure 2 are examples of basic shapes that can be combined to further complicate the shape of a graph. To begin with, free your imagination about the size of the grid world in Figure 2. Figure 2 is a 19x19 grid world just to match the Go’s full board size. Ideally, the solution should not be bound by the size of a grid world. And then imagine a complicated spiral graph on a much larger grid world with multiple branches, small and large loops, intersections and so on. Finding a solution for all these cases is challenging because the solution should, ideally, work for any shape at any size of grid world.

3. GROUP SIZE COUNTING PROBLEM IN COMPUTER GO HAS A WAY TO THE SOLUTION

We switch our attention from the autonomous graph size counting problem to the group size counting problem in computer Go. The latter is a relatively simple sub-problem in computer Go to obtain essential information for determining
life-death of a group of stones. Figure 3.a shows a snapshot of a
game play in Go and Figure 3.b is a part of the raw board
status. A popular board representation in computer Go is to use
-1 for white stones, 0 for empty intersections, and 1 for black
stones, so Figure 3.b is represented as Figure 3.c.

The goal of the group size counting problem (in computer
Go) is to obtain the size of a group or connected stones in the
same color. According to Go rules, connectivity of a stone is
defined vertically and horizontally, not diagonally. Therefore,
there are three black groups in Figure 3.b: vertical straight line
of three stones (left), horizontal straight line of two stones
(middle), and a group of a single stone (right). Figure 3.d shows
the counted group sizes for Figure 3.b. The group sizes of the
entire board in Figure 3.a is omitted for the space constraint.
Note a white group of one stone differentiates its color by
negating the sign or -1.

There are four reasons why we relate the graph size
counting problem to the group size counting problem. First, the
latter is a special case of the former. The difference is the size
of the grid world is bound to the board size and there exists
only two kinds (black and white). During a game play, stones
are sequentially placed on intersections of the board. However,
the board status forms a 2D grid world with vertices. Secondly,
our attention to this problem is originated from computer Go.
Thirdly, the game plays result in various shapes on the board,
so it is convenient to test the solution with unexpected shapes.
Lastly, relating the two problems set an example to apply
lessons learned from computer Go for other related issues.

4. THE PROPOSED SOLUTIONS TO THE GROUP
SIZE COUNTING PROBLEM

We propose, in this section, both centralized and
distributed solutions for the group size counting problem in
computer Go. The perspective of the centralized approach is
that of an arbiter on sky with top view of the Go board.
Namely, a global view over a board like Figure 3 is available.
In contrast, the visibility for the distributed approach is limited
locally to the immediate neighbors. Figure 4 shows the
viewpoints of intersections in Figure 3.b. Figure 3.b shows the
global viewpoint of the centralized approach and Figure 4
shows the local visibility of the distributed approach. In the
following sub-sections, we start from the centralized scheme
because the distributed scheme is influenced by the centralized
one.

Fig. 4 An example of the myopic visibility of the distributed approach.
Each cross-like board segment corresponds to a viewpoint of each
point in Figure 3.b. The square in the center of a board segment is the
board status s of the intersection itself and the rest of four statuses are
that of neighbors S. The convention to list the order of the neighbor is
S=[top, down, left, right].

4.1. The proposed centralized scheme with function
recursion

The proposed centralized algorithm counts the group size
like a human does and the idea is implemented elegantly with
function recursion. The idea is to focus on how human moves
one’s attention from point to point. Given a board status in
Figure 5.a, how does a human count the groups size?

Fig. 5 A board status, board representation, and target group size on a
6x6 Go board. The board representation B and the group size G are
bxb matrices where the board size b is 6 in this case.

Say one starts from the top left corner; skip the first row
and the attention or eyeball stops on the black stone on row 2
marked with square. Move the attention down to the black
stone with triangle, revert the attention to the stone with square,
and then move to the black stone with circle. In this way, a
group of three black stones is counted. For the group of white
stones, human’s eyeballs move as follows.
“1→2→3→2→4→5→6→7→8→9”. Notice stone 2 is visited
twice. It should not be counted on the second visit, but
returning to stone 2 keeps the trail in order to count the group
size. In this manner, human can count the group of nine white
stones. Let us call this process counting phase.

On the other hand, our goal is to obtain the group size of
the entire board as given in Figure 5.c, so each element of the
matrix G should have the counted group size. Back to the
human analogy, the counting phase ended on stone 9. The
group size g, or g=9 in this example, is delivered in the reverse
order of the trail in the counting phase. More specifically, the
order is “9→8→7→6→5→4→2→3→2→1”. Along this
backward trail, a stone on each location can update its estimate
of the group size. We call this process group size propagation phase.

The proposed centralized scheme in Figure 6.a implements the fore mentioned idea with function recursion. There are some overheads (compared to human’s eyeball tracking) inevitable to implement the idea as a computer algorithm such as line 1 that checks the color and the visit mark of the location. Basically, the function in Figure 6.a counts up the current estimate of the group size $g$ (line 2) if this location $(x', y')$ in the dummy board status $D$ is the same color $p$ and not visited earlier $V(x', y') 
eq 1$ (line 1). If this location is not visited, mark a visit (line 3) and look around the neighbors (line 4-7) to recursively perform the counting phase. Note function recursion has nice inherent features to spawn child functions (line 4-7) when a (parent) function is called. When no child functions are necessary, it automatically closes the spawned functions in the reverse order that functions are called. The reason why our solution is so elegant and simple is because this solution with function recursion automatically takes care of the forward and back trails.

COUNT_GROUP_SIZE_RECURSIVELY(p, x', y', D, V, g)
1 // $p=D(x', y')$ and $V(x', y')!=1$
2 $g=0$
3 $x'=x$ // Initialize $G$ to a $b\times b$ matrix with zeros
4 $y'=y$ // Initialize $G$ to a $b\times b$ matrix with infinities
5 $p=D(x', y')$ // Up
6 $p=D(x', y'-1)$ // Down
7 $p=D(x'-1, y')$ // Left
8 $p=D(x'+1, y')$ // Right
9 return $g$

(a) Pseudo code of our centralized scheme. $p$ is the color index of a player; $x'$ and $y'$ are the row and column of matrices $D$ and $V$. $x'=x+1$, $y'=y+1$. $D$ and $V$ are $(b+2)\times (b+2)$ matrices; $g$ is the current estimate of the group size.

Fig. 6 The proposed centralized scheme implemented with function recursion. The dummy board $D$ is the board status $B$ in Fig. 5.b wrapped with infinities.

4.2. The proposed distributed scheme

Assume an autonomous processing unit is placed on an intersection of a board and the distributed scheme is mounted on it. We name this processing unit a cell and a cell’s visibility is myopic as explained in Figure 4. Our goal is to develop a cell that inherently has myopic visibility and it becomes more challenging when this scheme should scale for any size of the board and any shape of a group.

How can this goal be achieved? If a group can count its size and propagate the size like the centralized scheme or if the whole procedure can be done like human’s eyeball tracking, this challenging task can be accomplished in a distributed manner. With this inspiration in mind, take a look at Figure 4 (cells’ local visibilities) and Figure 8 (a tree constructed by the centralized scheme). Each cell and a node have the same knowledge of neighbors’ status. What this indicates is the recursive branching of a tree structure can be mirrored if the spawning and pruning of child nodes are handled properly in a distributed manner.

This inspiration is implemented by the proposed distributed algorithm and so this scheme autonomously discovers where to start, counts the group size, and propagates the estimated group size as function recursion recursively constructs a tree. So one may say the function recursion is implemented in a distributed manner. Each cell performs the human’s eyeball tracking process described in section 4.1 autonomously. Therefore all the cells perform its own eyeball tracking process simultaneously in cooperation with the neighbor cells in the same color (if necessary).

Fig. 7 A simplified example to explain the tree in Fig. 8 constructed by the centralized scheme.

Fig. 8 How the centralized scheme constructs a tree for a board status in Fig. 7.a. In other words, how our centralized solution finds the answer recursively. (a) The full tree for intersection (1,1). Value on a node of the tree denotes the board status of the corresponding intersection. Each node may spawn child nodes, which corresponds to the neighbors. The order of the child nodes is top, down, left and right nodes from the left. A line connecting a parent and children indicates a trail for the eyeball tracking is possible (solid line) or impossible (dashed line). A tilted dashed line on a solid line means the child node is trimmed off. This node is trimmed because it has been visited already according to the visit mark $V$. (b)-(j) How child nodes are spawned and trimmed are explicitly illustrated. Note some steps combine two clock cycles for space constraint. Notice the tree may grow, e.g. (i), after the terminal nodes in (e) disappear in the tree. Note also that Fig. 7 is a simplifying example to explain the spawning and trimming of child nodes. A more complicated shape, say the complicated spiral graph mentioned at the end of Section 2, has a more complicated tree than this example.
Initialization of the internal variables in a cell or processing unit on an intersection

```
C.id←index // Index of the current cell
C.s←D(x+1,y+1) // Board status (of this cell), a scalar value
C.g←0 // Estimate of the group size
C.b←0 // Baton, a flag to activate/hibernate a cell
C.p←0 // phase of the operations, 1 is steady state
C.i←index // (Estimated) root cell id of this cell
C.e←c // Excess waiting time
C.w←0 // (Current) waiting time
C.n←−1 // next cell to pass the baton
C.c1←0 // (neighbor check) completed, 1st circle
C.c2←0 // 2nd circle
C.v←0 // Visit mark for this cell
C.V←zeros(1,4) // Visit mark for the neighbor in N.S.W.E
C.T←[] // Trail information, a stack
```

(a) Initialization of the internal variables in a cell or processing unit on an intersection

UPDATE_CELL(Status_neighbors,ERC.baton)

```
// if this cell's status is 1 or −1
1 C.S←Status_neighbors // In the order of top, down, left, right.
2 if C.p=0 // If this cell is in a non-steady-state
3 then C←DISCOVER_ROOT_CELL(C.ERC)
4 else if C.p=1 and C.b=0 // By-pass a cell w/o a baton
5 then C←COUNT_AND_PROPAGATE_GROUP_SIZE(C)
```

(b) How a cell updates its internal variables at each clock cycle. Relevant lines of codes are grouped to improve the readability of the pseudo code, which means there is a room to improve the efficiency of the code.

COUNT_AND_PROPAGATE_GROUP_SIZE(C)

```
1 if (C.b=1 and C.c1=0) or (C.b=2 and C.c2=0)
2 then if C.v=0 // If this cell has never been visited.
3 then C.v←1 // mark the visit.
4 if C.b=1
5 then C←COUNT_GROUP_SIZE(C,g,c.id,c.d)
6 C.V←zeros(1,4)
7 if leave
8 then return C
9 if C.b=1 then C←1−1
10 then C.v←1−1
11 C.V←zeros(1,4)
12 else // If C.b=2
13 then C.c2←1−1
14 then C.T←[] // Trail is empty.
15 then (C.n,C.T)←POP_STACK(C.T)
16 else if (C.b=1 and C.c1=1) or (C.b=2 and C.c2=1)
17 then C.T←[]
18 then (C.n,C.T)←POP_STACK(C.T)
19 else
20 then if C.b=1
21 then C←C.b−2
22 else
23 then C←−1
24 then (C.n,C.T)←IS_LEAVE_THIS_CELL(C,v,c.id,c.s)
25 if leave
26 then return C
27 return C
```

(c) Procedures for the group size counting and propagation phases in Steady state (C.p=1). Both phases can be written separately, but The pseudo code is shortened by merging the overlapped part.

Fig. 9. The pseudo code for the proposed distributed scheme. One used for simulations is modified to this version because of space constraint.

Take a simple 2x2 board in Figure 7.a as an example to explain how a tree is constructed by function recursion. The neighbors are checked recursively and the estimated group size is incremented if a neighbor has the same color. Figure 8.a. visualizes this tree construction process. Figure 8.b through Figure 8.j break down the tree construction process because it is hard to visualize how the tree is constructed only with Figure 8.a. Refer to Figure 8 for details.

The difference between the centralized and distributed approaches is worth mentioning. The former knows where to start (a root node) by iteratively assigning a location on a board; in contrast, a group of cells should agree where to start, called a root cell, for the latter. This process for the latter is called root cell discovery and counting and propagation phases come after the root cell discovery phase.

Details on the three phases are as follows. The first phase is to discover the root cell in a group. All the cells maintain internal variables like Figure 9.a. A cell exchanges its own estimate of the root cell (ERC), initialized to its cell index, with neighbors of the same kind or color. If this cell sees a smaller ERC than the current estimate within the scope (neighbors and this cell), update the current estimate to the smallest. Performing this procedure spreads the smallest cell index to the entire group as water poured to the entry point of a dried canal spreads along the waterway of the canal. Note taking the largest ERC works the same way with a minor tweak. Once a cell is sure all the cells in the group shares the same ERC, it switches its internal state from non-steady-state (C.p=0) to steady state (C.p=1). In steady state, all cells within the group share the same ERC. In other words, different groups can be identified by ERC.

While all cells are active to discover the root cell during non-steady-state, only one cell per group is active in steady state. We define a term baton as a signal to activate a cell. The concept is similar to a baton used by relay runners. Only one runner with baton runs in a relay game. Likewise the cell with baton is the only cell activated within the group. Cells pass the baton to take turns. In steady state, group size counting and propagation are done as follows. These two separate tasks are signaled by an internal variable baton index C.b. C.b=1 and C.b=2 mean the cell is in the counting and propagation phases, respectively. For details, refer to Figure 9 for the pseudo code of the distributed scheme.

5. SIMULATION RESULTS

Both centralized and distributed schemes are tested on various settings. These schemes work for all shapes in Figure 2, 5, 7 all work. Figure 10 shows how a raw board status is mapped to cells.

![Fig. 10. Board status to the cellular processing units, (a) Mapping a board status to cells in a cellular structure, the board size b is 19. (b) a part of raw board status, (c) corresponding board representation of (b), (d) counted results for (b).](image_url)
The distributed scheme has a key parameter excessive waiting time $\varepsilon$ that serves as the threshold in the root cell discovery. $\varepsilon$ is introduced as a measure to ensure the ERC spreads among the group. The performance and execution time depend on $\varepsilon$. Figure 11 shows the relationship between the board size $b$ and $\varepsilon$ for fully connected boards. In other words, all the intersections on a board have the same color. The solid line with asterisks is the lower bound of $\varepsilon_g$ that guarantees the correctness of answers. $\varepsilon$ above this line always results in the correct group size while $\varepsilon$ below it does not always lead to the correct answer. Carefully chosen, the correctness and the execution time are trade off.

Equation 1 is the dashed line larger than the lower bound. Note the dashed line becomes linear in log scale.

$$\varepsilon = 2b^2 > \varepsilon_g \text{ or } \log_{10} \varepsilon = \log_{10} 2 + 2 \log_{10} b \quad (1)$$

![Fig. 11 Relationship between the board size and $\varepsilon$ in linear and log scale.](image)

7. CONCLUSIONS

The proposed centralized and distributed solutions for the graph/group size counting problem are successfully developed and tested. The centralized scheme with global view of the interested region always computes the correct answer to the problem. The distributed scheme computes the correct answer when the excess waiting time $\varepsilon$ is set larger than the lower bound $\varepsilon_g$. The proposed schemes scales well and work for any shapes.

8. ACKNOWLEDGMENTS

Authors would like to thank Rui Xu, John Seiffertt, Ganesh Kumar Venayagamoorthy, Sriram Chellapan and Sejun Kim for insightful discussions.

9. REFERENCES


