Journal of Korea Multimedia Society Vol. 20, No. 4, April 2017(pp. 696-703)
https://doi.org/10.9717/kmms.2017.20.4.696

스도쿠 풀이에서 욕심쟁이 기법과 가지치기를 이용한 완전이진트리 생성 기법

김태석†, 김종수††

A Method to Expand a Complete Binary Tree using Greedy Method and Pruning in Sudoku Problems

Tai Suk Kim†, Jong Soo Kim††

ABSTRACT

In this paper, we show how to design based on solving Sudoku problem that is one of the NP-complete problems like Go. We show how to use greedy method which can minimize depth based on tree expansion and how to apply heuristic algorithm for pruning unnecessary branches. As a result of measuring the performance of the proposed method for solving of Sudoku problems, this method can reduce the number of function call required for solving compared with the method of heuristic algorithm or recursive method, also this method is able to reduce the 46~64 depth rather than simply expanding the tree and is able to pruning unnecessary branches. Therefore, we could see that it can reduce the number of leaf nodes required for the calculation to 6 to 34.

Key words: Sudoku Solver, Algorithm, Artificial Intelligent, Strategy, Complete Binary Tree Search, AlphaGo

1. 서 론

NP 완전문제임이 증명된 스도쿠 게임은 구글의 신경망 네트워크를 가지는 알파고와 다르게 현재의 데스크탑 컴퓨터의 자원을 이용해서 주어진 문제를 빠른 시간 내에 해결할 수 있는 알고리즘 구현을 위한 좋은 소재로 사용될 수 있다. 수학자 오일러의 라틴 방진을 응용한 스도쿠는 약 6.67×10^{21}개의 경우의 수를 가지는 복잡한 문제이면서, 바둑과 같이 상대방이 필요한 턴 방식의 게임이 아니므로 해당 기능 구현이 필요 없다는 장점이 있다[1-4].

또 다른 NP 완전문제인 바둑의 경우에는 오픈된 소스인 Fuego, GnuGo, Pachi, 그리고 Aya와 같은 인공지능이 있으며, 이러한 인공지능들의 추정 기력은 대략 7급 이상인 것으로 알려져 있다. 분산 환경에서 1,920개의 CPU와 289개의 GPU로 구성된 구글의 딥마인드 알파고는 머신러닝과 몬테카를로 트리 순회 기술을 조합한 알고리즘을 사용하여 (2.08×10^{170})/4개의 경우의 수를 가지는 바둑에서 인간의 지능을 뛰어넘는 인공지능을 만들어냈다. 알파고에서는 트리 전개를 최소화하기 위한 기법으로 심층 신경망 기술을 적용하여 주요 핵심기술인 정책 네트워크와 가치 네트워크 구현하였는데, 초기 신경망을 구축하기 위한 방법으로 3천 만 수 정도의 바둑 프로 기사들

※ Corresponding Author : Tai Suk Kim, Address: Dept. of Computer Software Engineering, Dong-eui University 176 Eomgwangno Busan_jin_gu, Busan 614-714, Korea, TEL : +82-51-890-1707, FAX : +82-51-890-1724, E-mail : tskim@deu.ac.kr
Receipt date : Dec. 17, 2016, Revision date : Mar. 6, 2017
Approval date : Mar. 27, 2017
† Dept. of Computer Software Engineering, Dong-Eui University
†† Dept. of System Management, Korea Lift College (E-mail : seatree@klc.ac.kr)

의 기보를 데이터베이스로 사용하였으며, 어느 정도 안정화되어서는 2개의 알파고가 서로 대국하여 만들어진 기보를 사용하여 다시 머신 러닝을 강화하도록 학습시켰다[5-7].

NP 문제를 해결하기 위한 인공지능은 반드시 최적의 해법을 구해야 할 필요는 없기 때문에 짧은 시간에 해를 구할 수 있는 알고리즘의 개발이 중요하다고 할 수 있다. 본 논문에서는 NP 완전문제인 스도쿠의 풀이기 위해 트리순회와 잘 알려져 있는 풀이 방법을 구현한 알고리즘과 욕심쟁이 기법(greedy method)을 이용하여 트리를 순회를 최소화하기 위한 예를 보인다.

2. 관련연구

2.1 P-NP 문제

1971년 스티븐 쿡의 정리 증명 절차의 복잡성(The Complexity of Theorem Proving Procedures)이라는 문제에서 처음 제안된 P-NP 문제의 NP는 비결정론적 튜링 기계를 사용하여 다항 시간 내에 답을 구할 수 있는 문제 집합으로 정의된다. NP 완전 문제가 2차 혹은 선형 시간 안에 풀릴 수 있는지 아닌지를 묻는 문제가 클레이 수학연구소에서 발표한 7개의 '밀레니엄 문제' 중 하나이며 컴퓨터 과학에서 중요한 위치를 차지하고 있다[8].

NP-완전(NP-complete, NP-C, NPC)은 NP 집합에 속하는 결정 문제 중에서 가장 어려운 문제의 부분집합으로, 모든 NP 문제를 다항 시간 내에 NP-완전문제로 환산할 수 있다. NP완전 문제의 예는 해밀턴 경로 문제, 외판원 문제, 그래프 색칠 문제, 시간표 짜기 문제들이 있으며, 스도쿠와 바둑게임도 이에 속한다. 컴퓨터과학자들은 NP-완전문제를 실용적인 관점에서 해결하기 위해서 다항방정식을 찾는 대신 근사 알고리즘(approximation algorithm)이나 휴리스틱 알고리즘을 사용하거나 욕심쟁이 알고리즘을 사용하여 훨씬 적은 양의 계산으로 빠르게 정답을 찾는 방법을 사용한다[9].

2.2 기존 연구

스도쿠를 풀이하기 위한 프로그램의 대표적인 예는 호도쿠(Hodoku)와 재귀적 호출을 사용하는 방법이 있다. 먼저 스도쿠 문제의 풀이 방법을 도와주기 위해 제작된 호도쿠는 다양한 풀이 방법을 구현하고 있다. 호도쿠에서 사용되는 휴리스틱 알고리즘이 구현된 클래스를 Fig. 1에서 볼 수 있다.

스도쿠 풀이를 위해 사용될 수 있는 알고리즘의 구현하기 위하여 strategy 패턴을 사용하였고, Abstract Solver를 확장한 SimpleSolver, ChainSolver, ColoringSolver, 그리고 FishSolver와 같은 다양한 메서드들이 구현되었음을 볼 수 있다. 총 23개의 클래스를 사용하여 47개의 풀이 방법을 구현하고 있다[10]. 호도쿠 외에도 스도쿠 풀이기를 위한 방법으로는 재귀적 방법이 있다. 이를 응용하는 풀이 예는 “http:// sunnyholic.com/81” 에 소개되어 있다[11]. 이 연구는 재귀적 알고리즘을 적용하기 전에 탐색횟수를 줄일 수 있도록 행, 열 그리고 그룹에서 탐색대상이 되는 숫자가 가장 작은 세트부터 숫자를 채워나가는 욕심쟁이 기법을 구현하고 있다. GUI는 구현되지 않았으며, 컴퓨터의 계산능력을 충분히 활용하여 시간복잡도를 최적화 시켰다. 재귀적호출을 사용하는 방식을 의사코드로 나타내면 Table 1과 같다.

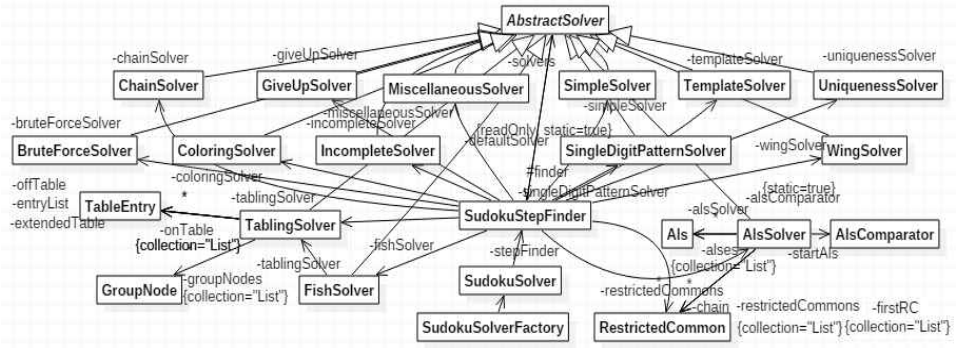


Fig. 1. class diagram for solving method.

Table 1. pseudo code for the recursive call method

```
function solver(){
  //find for the group with the smallest number
  group = findMinGroup();
  //If the found group does not exist, return
  if (group == null) return;
  if (group.getRow() > 9 ) return; //Other error handling

  for number = 1 to 9 {
    //Skip numbers where input is impossible
    if not group.getTrueNumbers(number) continue;

    //Assign numbers to the corresponding columns
and rows
    group(row, col) = number;

    //recursive call and check group, if solved then exit
    if (slover()) return true;
  }

  group(row, col) = 0;              //Clear preset value
}
```

3. 스도쿠 풀이를 위한 알고리즘 설계

3.1 완전이진트리 탐색의 응용

수학자들에 의해, 문제풀이에 대한 다항방정식을 정의할 수 없는 NP 완전 문제임이 증명된 스도쿠 문제를 컴퓨터로 풀이하는데 있어서 좋은 방법 중에 하나는 CPU의 초고속 연산능력을 최대한 활용할 수 있는 재귀적인 호출을 이용하는 것이지만, 구현되는 코드에 불필요한 코드가 추가되면 재귀호출횟수만큼 소요시간이 증가한다는 단점이 있다. 스도쿠와 같은 NP 완전 문제를 풀이하기 위한 또 다른 방법은 각각에 해당하는 경우를 계산하기 위한 트리전개를 하는 것이며 이 경우 트리 전개를 최소화하는 방법에 대한 연구가 필요하다.

트리는 전개하는데 있어서 네트워크로 연결된 다수의 컴퓨터 자원을 활용할 수 있는 경우에는 완전이진트리 탐색을 활용할 수 있다. 이미 출제된 스도쿠 문제는 2개 이상의 부분문제로 분할하여 각 부분 문제의 해결안을 바탕으로 최종 해결안을 도출할 수도 있다. Fig. 2는 제안된 방법이 완전이진트리를 전개하는 방식을 보여준다.

수행횟수 1번째인 Depth1은 휴리스틱 알고리즘이 적용되어 계산된 단계이며, 주어진 문제에서 더 이상 적용할 수 있는 휴리스틱 알고리즘이 없는 경우, 완

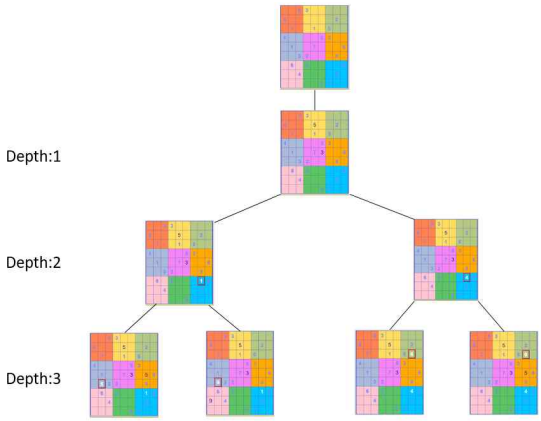


Fig. 2. The complete binary tree expansion of proposed algorithm.

전이진트리를 전개한다. 주어진 스도쿠 문제를 부분문제로 분할하면 통상적으로 원래의 문제와는 입력크기만 다를 뿐 동일한 문제가 된다. 일반적으로 완전이진트리의 시간복잡도 계산은 다음과 같이 이루어진다.

처음 입력된 개수를 n이라고 하면,

첫 번째 이진트리전개에서 좌우로 반씩 나누어짐으로 $\frac{n}{2}$이 되며, 두 번째 시행 후에는 또 다시 반으로 나누어짐으로 $\frac{1}{2}\times\frac{n}{2}$이 된다. 세 번째 수행에서도 반으로 나누어짐으로 $\frac{1}{2}\times\frac{1}{2}\times\frac{n}{2}$이 되는데, K 번 수행 후의 탐색할 자료의 수는 수식 1과 같이 일반화된다.

$$\left(\frac{1}{2}\right)^{K} n \tag{1}$$

탐색이 끝나는 k번째에 남는 자료의 숫자를 생각해 볼 때, 최악의 탐색결과는 수식 2와 같이 자료가 1개 남을 때 까지 탐색하는 것이다.

$$\left(\frac{1}{2}\right)^{K} n \approx 1 \tag{2}$$

양변에 2K를 곱하면

$$2^{K} \approx n \tag{3}$$

이 되며, 다시 양변에 2를 밑으로 하는 로그를 취하면 완전이진트리의 탐색 시간복잡도 T(n)은 수식 4과 같이 계산된다.

$$T_{(n)} = \log_2 n \tag{4}$$

시간복잡도 Big O 표기법으로는 수식 5가된다.

$$O(\log n) \tag{5}$$

완전이진트리에서는 각각의 잎 노드를 네트워크 망에 연결된 컴퓨터로 분산하여 계산을 수행할 수 있도록 구성할 수도 있는데, 앞의 식들에서 깊이가 수행횟수 n를 나타냄으로, 네트워크로 분산된 개별 완전이진트리의 수행횟수는 $n-d_n$이 되며 따라서 시간복잡도 $T'_{(n)}$ 은 수식 6과 같이 줄어든다.

$$\begin{aligned} T'_{(n)} &= \log_2(n-d_n) \\ O(\log(n-d_n)) \end{aligned} \tag{6}$$

3.2 풀이방법들과 욕심쟁이 기법의 적용

제안된 방법은 기존에 알려진 휴리스틱 알고리즘과 욕심쟁이 기법을 사용하여 트리 전개를 최소화하는 기법을 사용한다. Fig. 3에서 제안된 방법을 사용하여 스도쿠 문제를 풀이하기 위한 전체 순서도를 볼 수 있다.

초기에 문제가 출제되면 전처리 과정을 거친 후, Hidden Single과 Naked Pair로 알려진 알고리즘으로 부분 정답을 찾아내거나 전개가 불필요한 가지를 친다. 스도쿠 방진 전체가 임의로 선택된 숫자로 채워지면 정답 검증기를 사용하여 정답을 검증한 후, 정답이면 결과를 출력하여 종료하고, 정답이 아닐 경우에는 다시 문제를 초기화한 후, 트리를 생성하고 최소 깊이 전개를 위해 욕심쟁이 기법을 사용하여

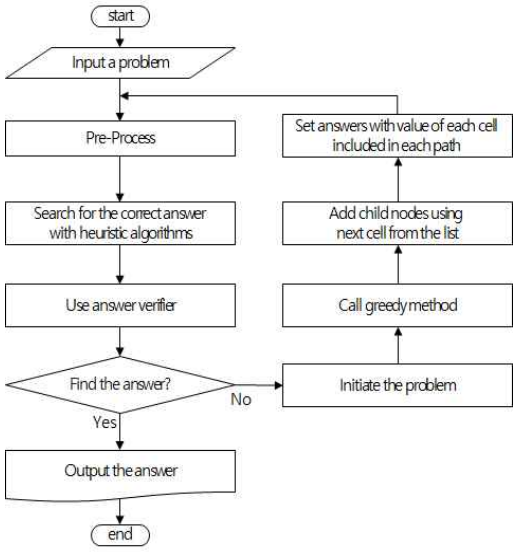


Fig. 3. Flow diagram of proposed method.

전처리된 셀들 중에서 시간복잡도를 최소화할 수 있는 Cell을 구한 후, 해당 Cell에 포함된 숫자들을 사용하여 자식 노드로 전개한다. 정답검증기의 주요 역할은 현재 정답이 검증되고 있는 잎 노드가 속한 경로가 앞으로 계속해서 깊이 전개를 해야 하는지 가치를 평가하는 것이며, 가치 평가의 결과는 현재 잎이 정답인 경우 스도쿠 방진을 리턴하고, 추가적인 깊이 전개가 더 필요한 경우 욕심쟁이 기법으로 찾은 Cell을 이용하여 가장 깊은 잎 노드를 대상으로 자식 노드를 추가해 나간다. 해당 잎 노드가 오답인 경우 더 이상 깊이 전개가 되지 않도록 가지를 친다. 다음으로 전개된 잎 노드를 사용하여 새로운 스도쿠 문제를 생성한 후, 전처리 과정과 정답검증기를 거치는 호출을 정답을 찾을 때까지 계속한다.

3.3 전처리기 구현

스도쿠를 풀이하기 위한 첫 번째 단계는 각각의 정답이 주어진 각 셀을 제외하고 정답이 아닌 셀에 대하여 정답이 될 수 있는 숫자들의 집합을 찾아내는 것이다. 제안된 방법에서, Cell 클래스는 스도쿠 문제를 풀이하기 위한 기본 정보를 가지고 있는데, Cell의 좌표와 입력 가능한 숫자의 집합이다. 스도쿠 문제의 전처리를 위한 사용자 인터페이스는 Fig. 4에 나타냈다.

주어진 문제의 전처리 결과 C(0, 0) = {1, 2, 6, 9}, C(0, 1) = {2, 4, 9}, C(0, 4) = {2, 4, 6, 8, 9}, 그리고 C(0, 5) = {2, 4, 6, 7, 8}의 숫자가 올 수 있음을 볼 수 있다.

3.4 휴리스틱 알고리즘을 이용한 부분 정답 찾기

제안된 방법에서는 스도쿠 풀이를 위해 적용될 수

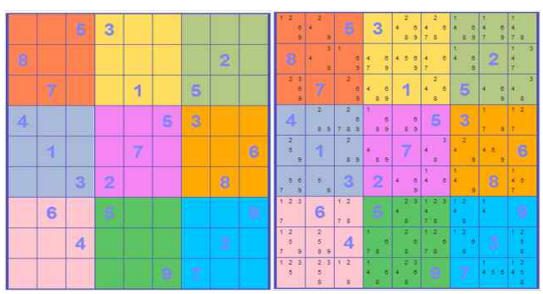


Fig. 4. preprocessing execution (left: before, right: after).

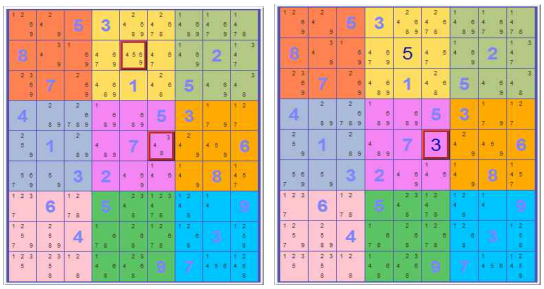


Fig. 5. Hidden Single before (left) and after (right).

있는 알고리즘 중에서 사용 빈도수가 비교적 높은 “Full House/Last Dight”, “Hidden Single”, “Naked Single”, 그리고 “Naked Pair”만을 구현하였다. 이 중 “Full House/Last Dight”과 “Naked Single”은 전처리과정에서 사용되며, 정답을 찾기 위해 “Hidden Single”과 “Naked Pair”가 사용되었다. Fig. 5에서 “Hidden Single”이 수행된 결과를 볼 수 있다.

먼저, C(1, 4)를 살펴보면 C(1, 4)에 포함된 원소는 {4, 5, 6, 9}인데, 여기에 포함된 원소 중에서 숫자 5가 Hidden Single이다. C(4, 5) = {3, 4, 8}에 대해서도 숫자 3이 Hidden Single임을 볼 수 있다. 좌측의 실행 전 그림을 살펴보면 정답이 아닌 전체 셀에 대하여 Hidden Single을 찾아내기 위한 구현을 위해 행과 열 그리고 그룹에 대하여 모두 구현해야 될 것처럼 보이지만 실제로는 행이나 열, 또는 그룹 중에서 한 가지 경우에서만 Hidden Single을 탐색하면 된다는 것을 볼 수 있다. 해당 행과 열 그리고 그룹에서 각각 숫자 5와 3이 제거된 것을 볼 수 있다. 알고리즘을 수행한 결과가 true를 리턴하면 다시 전처리된다.

3.5 최소 깊이 전개를 위한 욕심쟁이 기법

Hidden Single과 Naked Pair를 사용하는 전처리 과정을 거친 후에도 스도쿠 방진 전체가 숫자로 채워지지 않으면, 다음 단계는 욕심쟁이 기법을 사용하여 최소 깊이 전개에 가장 효과적인 cell을 검색한 후, 트리의 전체 잎에 해당 cell의 숫자들을 자식 노드로 추가한다. 이렇게 생성된 잎 노드를 사용하여 스도쿠 방진을 다시 생성하며, 만약 스도쿠 방진이 숫자로 다 채워지면 정답 검증기를 사용하여 정답을 검증하는 과정을 반복한다. 트리 전개에서 최소 깊이를 생성하기 위한 욕심쟁이 기법은 다음과 같다.

method 1: 문제 풀이 시간을 단축하기 위해서 가장 적은 숫자들의 원소를 가지는 셀 선택

method 2: method 1로 선택된 셀이 여러 개가 있을 경우, 해당 셀이 가진 원소가 정답으로 설정되었을 때, 정답이 아닌 셀들이 포함하고 있는 숫자를 가장 많이 제거할 수 있는 셀을 우선적으로 선택

욕심쟁이 기법이 적용되는 과정은 다음과 같다. 앞의 문제에 대해서 먼저 전처리를 거친 실행결과에서 method 1을 적용하면 C(5,1) = {5, 9}와 C(6,7) = {1, 4}의 두 개의 셀이 가장 적은 원소의 개수를 가지고 있다는 것을 볼 수 있다. 후보의 개수가 2개 이상일 경우, method 2를 적용한다. 두 후보에서 각각의 값이 정답으로 설정되었을 때, 정답으로 설정되지 않은 행 열 또는 그룹에서 가장 많은 값을 제거하는 숫자 선택해서 새로 생성되는 depth의 자식 노드로 추가한다. 이 후, 다시 전처리를 수행하며, 스도쿠 방진이 전부 채워졌을 경우 정답검증기를 사용하여 가치평가를 수행하여 불필요한 가지를 친다. 제안된 알고리즘의 의사코드를 Table 2에서 볼 수 있다.

1) Input: cell_List; Tree;
   - 셀에 포함된 원소의 개수가 최소이면서 해당 cell을 정답으로 설정했을 때 다른 cell 들에 있

Table 2. pseudo code for the proposed method

```
function SAI()
  Cell = getGreedyCell()              //call greedy method

  for each leaf in Tree :             //tree traversal
    if (leaf = indeterminate) continue
    if (leaf = impossible) continue
    //get path form selected leaf
    treePath := leaf.getPath()

    while hasNextCell  //create more leaf
      cell.getAnswer()

      pruning with solving methods
      result = answerVerifier()  //call answer verifier
      if (result = true) return sudoku matrix
      else pruning current node  //pruning node

      initiate the problem            //init problem

call next step                        //call next step
```

는 원소들을 가장 많이 제거할 수 있는 cell들의 정보가 있는 List

- Tree: Cell이 가진 원소들을 정답으로 설정하여 만들어지는 각각의 Node로 정답 설정

2) Output: 스도쿠 방진

Fig. 6에 구현된 스도쿠 풀이기가 문제풀이를 위해 완전이진트리를 전개하는 방식을 나타냈다.

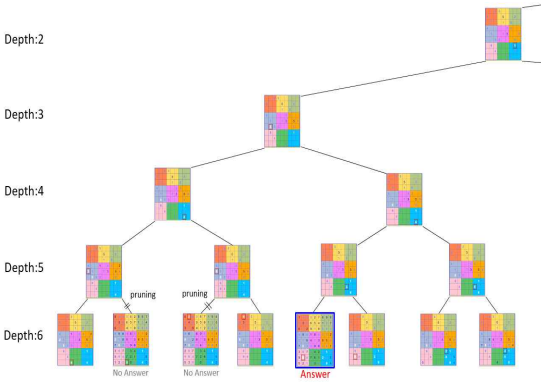


Fig. 6. The result of the complete binary tree expansion for solving the sudoku.

해당 문제에서는 깊이 6에서 정답을 찾았으며, 2번의 가지치기가 있었다.

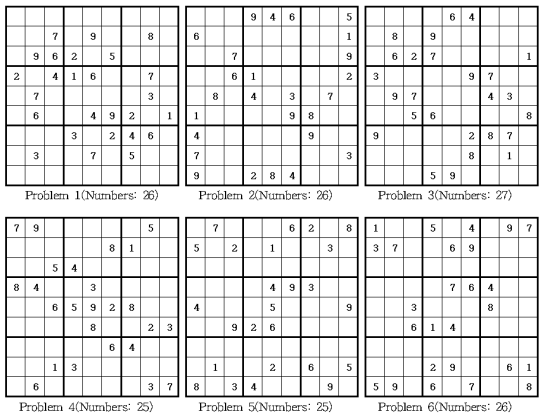
4. 성능 평가

각각의 프로그램에 대한 테스트를 위한 환경은 Table 3과 같다.

호도쿠나 재귀적 호출을 사용하는 방법의 서로 다른 설계 목적을 가지고 있음으로 각각의 구현 방법에 대한 성능평가를 위해 임의로 출제된 난이도가 높은 스도쿠 문제를 대상으로 처리 시간과 주요항목을 측정하였고, 이를 위해 호도쿠와 재귀적호출 방법은 기존의 프로그램에 해당코드를 추가하였다. 주어진 문제는 다음과 같다.

Table 3. Test environment

OS	Windosw10
Processor	Intel Core2 Quad CPU Q9500 2.83GHz 2.83GHz
RAM	4GB
Language	JDK1.8.0_92
IDE	Netbeans IDE8.1



호도쿠, 재귀적 호출 그리고 제안된 방법으로 Problem1을 풀이한 결과를 Table 4에 정리하였다.

총 47개의 알려진 풀이방법으로 구성된 호도쿠는 주어진 문제에서 적용할 수 있는 알고리즘의 탐색시간이 많이 소요되었다. 이것은 같은 조건에서 사용될 수 있는 여러 개의 알고리즘을 모두 탐색하기 때문으로 예상된다. 호도쿠에서 가장 많이 사용되는 풀이방법은 Full House, Naked Single, Hidden Single이였으며, 다음으로 Locked Condidates, XY-Chain 등 이였다. GUI를 구현하지 않고 오직 스도쿠 문제 풀이를 위한 코드로만 구현된 재귀적호출 방법이 가장 빠른 시간을 보였지만, 호도쿠에서 전체 알고리즘의 수나 제안된 방법에서의 깊이와 잎의 곱을 호출횟수로 가정한다면 다른 2개의 방법보다 메서드 호출 횟수가 많은 것을 알 수 있다. 재귀적 방법에서는 불필요한 호출을 최적화하기 힘들다는 단점도 있다.

트리를 전개하면서 “Full House/Last Dight”, “Hidden Single”, “Naked Single” 알고리즘과 욕심쟁이 기법을 사용하는 제안된 방법은 깊이전개를 최소화하고 필요 없는 가지를 쳐냄으로써 계산에 필요한 잎 노드를 많이 줄인 것을 볼 수 있다. 9x9 정형 스도쿠에서 25개의 숫자가 공개된 문제의 경우에서 볼 때, 단순하게 트리만을 전개하는 경우에는 최악의 경우 56 단계의 깊이 전개가 필요하고, 그에 따른 잎도 엄청나게 늘어날 수 있지만, 제안된 방법에서는 주어진 문제들에 대하여 전개된 깊이가 4~9단계로 줄어들었으며, 계산에 필요한 잎 노드가 6~34개로

Table 4. Measuring execution time (unit: milliseconds)

Problem / elapsed time	1 (26)	2 (26)	3 (27)	4 (25)	5 (25)	6 (26)
Hodoku (Algorithm Search time) Algorithm Sub Total)-(Total Algorithm)	0(921) (13)-(68)	0(827) (15)-(80)	0(526) (12)-(70)	0(458) (13)-(81)	0(553) (13)-(73)	0(583) (13)-(71)
Recursive Method (call count)	37 (338)	42 (263)	36 (190)	68 (503)	44 (223)	47 (251)
Proposed Method (depth)-(Total leaves)	281 (4)-(6)	148 (4)-(6)	228 (5)-(12)	210 (5)-(13)	195 (3)-(4)	133 (9)-(34)

줄어든 볼 수 있었다. 구현된 GUI를 코드를 없애고, 가장 많이 적용되는 알고리즘을 추가적으로 구현하고, 욕심쟁이 기법을 최적화하면 계산시간이 더욱 단축될 수 있을 것으로 보인다.

5. 결 론

본 논문에서 바둑과 같이 NP-완전 문제 중에 하나인 스도쿠를 풀이할 수 있는 풀이기 설계방법을 제안하였다. 아주 복잡한 난이도를 가지는 NP 문제들은 정확한 수학적공식이나 일반적인 해법을 찾을 수 없는 경우가 많다. 특히 복잡도가 높은 많은 문제를 다루어야할 경우, 한정된 자원을 사용하여 최적의 해를 빠르게 도출할 수 있는 방법이 필요하며, 이러한 경우 주어진 시간 내에서 어느 정도 최적한 된 풀이 방법에 빠르게 도달하기 위해서는 컴퓨터의 빠른 계산 능력을 활용하는 것이 좋은 방법이다.

본 논문에서도 많은 계산량을 가지는 게임인 스도쿠 문제의 풀이기를 설계하는데 있어서 트리전개바탕으로 욕심쟁이 기법을 사용하여 완전이진트리에서 전개될 깊이를 최소화는 방법과 자주 사용되는 알고리즘을 사용하여 불필요한 가지를 쳐냄으로써 계산되어야 할 잎 노드를 최소화시키는 방법을 보였다. 향후 일상생활에서 접할 수 있는 다양한 NP-완전문제들에 대하여 최적의 풀이를 수행할 수 있도록 제안된 방법을 응용하고 개선할 수 있는 연구가 지속적으로 필요하다.

REFERENCE

[1] A. Hulpke, P. Kaski and P.R.J. OSTERGARD, "The Number of Latin Squares of Order 11," math.CO., arXiv.org, pp. 1-22, 2010.

[2] T.S. Kim and J.S. Kim, "A Comparative Study of Solver of 9x9 Normal Sudoku," *Proceedings of the Korea Multimedia Society Conference*, pp. 232-237, 2016.

[3] T.S. Kim and J.S. Kim, "A Study on the Undo Function Implementation using the Design Patterns," *Journal of the Korea Industrial Information Systems Research*, Vol. 19, No. 8, pp. 1544-1522, 2016.

[4] J.S. Kim and T.S. Kim, "Design of Network-based Game using the GoF Design Patterns," *Journal of Korea Multimedia Society*, Vol. 9, No. 6, pp. 742-749, 2006.

[5] Google AlphaGo, https://www.wikipedia.org, (accessed Feb., 1, 2016).

[6] David Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, and G.V.D. Driessche et al., "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, Vol 529, pp. 1-37, 2016.

[7] H.J. Yang, H.Y. Jang, and B.T. Zhang, "Monte Carlo Tree Search for a Board Game with Nonconsecutive Turns," *Proceedings of Korea Institute of Information Scientists and Engineers*, pp. 1717-1719, 2014.

[8] P-NP Problem, http://www.wikipedia.org, (accessed Nov., 5, 2016).

[9] Computational Complexity Theory, http://www.aistudy.com, (accessed Nov., 5, 2016).

[10] Hodoku, http://hodoku.sourceforge.net/en/index.php, (accessed Jun., 3, 2016).

[11] Recursive Method for Solving Sodoku, http://sunnyholic.com/81, (accessed Nov., 7, 2016).

김 태 석

1981년 경북대학교 전자공학과 공학사 졸업
1989년 일본 KEIO대학 이공학부 계산기과학전공 공학석사
1993년 일본 KEIO대학 이공학부 계산기과학전공(공학 박사)
1993년 일본 국제전신전화연구소(KDD) 기술고문
1993년 일본 KEIO대학 이공학부 객원연구원
1994년~현재 동의대학교 컴퓨터소프트웨어공학과 교수
관심분야 : 정보시스템, 기계번역, 인터넷비즈니스

김 종 수

1992년 부경대학교 냉동공학 전공 공학사 졸업
2003년 부산외국어대학교 컴퓨터 공학전공 공학석사
2006년 동의대학교 컴퓨터소프트웨어공학전공 공학박사
2012년 동의대학교 산업기술 개발연구소 P.D 연구원
2013년~현재 한국승강기대학 승강기공학부 교수
관심분야 : 소프트웨어 설계, 게임