Application Of Branch And Bound Algorithm In Character Balancing For Online Game

M Aditya Farizki 13516082
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516082@std.stei.itb.ac.id

***Abstract*—Game balancing has been a main problem for some many game developer. With the many stats and variation in characters ability in a game, balancing a character has become harder than ever. Introducing variation in the game by releasing new character which is a central part of a game development has been stagnant for some games. Many games also does not realize that their characters are too weak in a point of a game to the point that the characters cannot develop further in a match. It is necessary to develop a certain method or way to ensure that the balancing process can fix the overpowered and underpowered character problem. Branch and bound implementation for the given problem can inform us on when a character is strong and when the character is too weak, which will help a lot in the balancing process. Even though that the result of the program does not match linearly with the win rate of each character, but the result is good enough to the point that it can be a consideration to the balancing process.**

Keywords—character, balance, strength, branch, bound, cost, game.

I. Introduction

Character balancing has been a big problem for most player versus player game in the world. The demand of new character for every season has made the previous or old character become either obsolete or suddenly too powerful. Some games have been reported to lose players or even gone bankrupt because of the lack of understanding in character balancing division.

The essence of releasing new character or new item is to introduce variation in the game. But more problem raise as more characters get released. Some characters become too powerful and some characters get too powerless or useless. In most player versus player games, players spend money on each character, either it's for cosmetics or to get the access to play the character. The payment method may be in the form of in game currency or even real money.

One example of such game is DoTA2 or Defense of The Ancient 2. DoTA2 has been reported to gradually lose their player over time. The developer and creative team are having trouble releasing new character, or specifically in this game called heroes, without making the old heroes obsolete, useless or too powerful.

The problem in character balancing are not only how strong a character can get, or how strong a character initially is, but also how strong is a character at a given point in game compared to another characters. Some game characters like Caitlyn from League of Legends is a really strong character (League of Legends use the term Champion to refer to their in game characters), but the problem is she is really weak in mid game (when the game has last only 15 – 25 minutes) that most players cannot play her into her full potential. This is the main problem that this paper will address.

II. Game Balance

In game design, balance is the concept and the practice of tuning a game's rules, usually with the goal of preventing any of its component systems from being ineffective or otherwise undesirable when compared to their peers. An unbalanced system represents wasted development resources at the very least, and at worst can undermine the game's entire ruleset by making important roles or tasks impossible to perform, (Newheiser, Mark (9 March 2009). "Playing Fair: A Look at Competition in Gaming". Strange Horizons. Archived from the original on 12 March 2009.).

Game Balance is an extremely important factor in a non "pay to win" game to give the player the sense of fairness. Game also needs to give the player the illusion or the impression that the game is winnable, it needs to give the player a motivation to keep playing, no matter it is for beginner, amateur, or professional player. An unbalanced game will destroy the motivation to play because an overwhelmingly strong characters can be abused to win games.

There's another issue in balancing a game. A game's learning curve influence on the balancing process is very significance. Some characters may feels really unbalanced for beginner players but actually balanced for amateur and professional players. Some characters also may be considered as niche pick, where the condition to play that character is very specific and many players disregard such fact.

To measure how balanced a game is or rather how balanced the characters and items in the game is, we can see through the win rate and performance rate of each character. Many games provide an API to get match data. From that data we then can calculate the average win rate of each character and the items they use.

A character can be called perfectly balanced if and only if its win rate is 50%, which mean using that character in overall has the same chance to win or lose the game. To accommodate real life condition we will say a character is justifiably balanced if

and only if its win rate is 50% ± Δ, where the delta factor will be decided based on the game.

For further analysis, this paper will use a game called League of Legends, a Multiplayer Online Battle Arena game (MOBA) where there are 10 players, each with a single character fight on two teams of 5 trying to destroy the enemy's base. The reason to use this game as a basis are :

1. Huge player base

   A post from Forbes website shows that the number of League of Legends player in 2014 was at least 67 million players.

   (https://www.forbes.com/sites/insertcoin/2014/01/27/riots-league-of-legends-reveals-astonishing-27-million-daily-players-67-million-monthly/#4a9b1ce46d39)

   Some other sources said that currently League of Legends have more than 100 million players.

2. Accurate and easily available match data

   Riot Games (League of Legend developer) officially released an API to get match data from their server. Some websites also provide full data description and analysis to be used. This paper will use data provided from http://www.champion.gg (accessed on 11th May 2018) as basis. The website has been proved to be accurate by the gaming community itself.

3. Scheduled and frequent new character release

   League of Legends releases at around 5 new characters per year which is considered to be a lot for a MOBA game, compared to 1 from DoTA2. League of Legends also has a scheduled update and balance patch at every end of the year so the change can be measured easier.

For League of Legends the Δ value to decide how balanced a character is will be determined by the following function

$$\triangle = f(n) = \frac{n}{20}\%$$

Where

n = the number of character of the role

20 is arbitrary number taken from the given statistic that it is very common for players to feel that the character is balanced around that score. It is also intuitive to think that the more characters are there for a role then the more biased the win rate will be.

For example if there are 40 characters for mage roles, then every character in the role is justifiably balanced if and only if their win rate is between 48% and 52 %.

III. Branch And Bound Algorithm

Branch and bound is an optimization algorithm used to minimized the number of state to be search for from a given number of possibilities. To put it simply, branch and bound algorithm is breadth first search with least cost search involved.

Common application of branch and bound algorithm is in pathfinding problem such as A* algorithm, 0/1 knapsack problem, 15 puzzle problem, Feature selection in machine learning, etc.

The idea of branch and bound is rather than expanding a state based on their order of expansion, the algorithm expand a state based on the cost of expanding that state. The cost function is a combination of the previous cost plus expected cost to reach the goal, or mathematically written as :

$$\hat{c}(i) = \hat{f}(i) + \hat{g}(i)$$

Where

$$\hat{c}(i) = cost\ to\ expand\ state\ i$$

$$\hat{f}(i) = cost\ to\ reach\ state\ i$$

$$\hat{g}(i) = expected\ cost\ to\ reach\ goal\ state$$

Below is the pseudocode for common branch and bound algorithm

```
PriorityQueue Q
SetOfOptions Option <- [options for a given
problem]
SetOfNode expand(Node N, SetOfOptions O){
  SetOfNode temp
  For(Options Op in O){
    temp.add(Make a new state based on option
Op and Node N and then use CostFunction to
calculate the cost of the new Node)
  }
  return temp
}
Node CurrentState <- InitialState
Node GoalState <- Goal
Q.insert(InitialState)
While(CurrentState != GoalState){
  Node ExpandState <- Q.dequeue()
  for(Node temp in expand(ExpandState,
Option){
    Q.enqueue(temp)
  }
  CurrentState = Q.dequeue()
}
```

The expected cost to reach the goal (or sometimes called heuristic function) state function is different for each problem. In the A* algorithm the expected cost is the direct length from the current point to the goal, in 15 puzzle problem the expected cost is the Manhattan distance of each tile to where they're supposed to be.

We use priority queue data structure to make the process of sorting every node based on its cost easier, we assume that the priority queue has method called enqueue() to insert a new Node into the queue and retain its rules where the item is sorted based on the cost, we also assume that the priority queue has a method called dequeue where the queue will return the Node with smallest cost value and then erase it from the queue. The CostFunction(Node N) part is relative to what kind of problem is being solved by the algorithm. SetOfOptions Option is the set of possible option that can be taken. The expand function is a function to expand a Node based on a given SetOfOptions, the function will also decide if a Node can be expanded to a certain state or not, for example in 15 puzzle problem case, the expand function will decide if the hole can be moved to a given direction or not, if the puzzle is on the bottom left part of the puzzle then the expand function will not expand to bottom and left option. Then we declare the initial state and the goal state and insert the initial state to the queue. Now the program will expand the Node with the smallest cost and then switch the current Node to the Node with smallest cost until current Node is the goal state.

In pathfinding problem, there is an additional requirement for the heuristic function.

If the heuristic function, h always underestimate the true cost (h(n) is smaller than h*(n)), then A* is guaranteed to find an optimal solution → admissible; and also has to be consistent. (Rinaldi Munir, slide kuliah IF2211, " A*, Best FS, and UCS (2018)").

IV. Game Basic Explanation

Generally speaking, how balanced a character is depended on how the players think. Some characters may be balanced on paper but the players may find playing the character still too hard or too easy. There are many cases on how a character ability and statistic is considered to be balanced but a lot of beginner players rant about how unbalanced a character is. One of the case is Rikimaru in DoTA and DoTA2, the characters ability to become invisible for most of the match is deemed to be unbalanced for beginner player, where the amateur and professional layer knows that by simply buying item that can reveal invisible object on the map, they can counter Rikimaru pretty easily and the items are considerably cheap. However the mentioned condition will not be considered as unbalanced in this paper.

The goal of this approach of balance checking is to make sure that a character can keep up with most other character in the same role throughout a match. So the character should not be too weak or too strong in any point of time in the game. The characters will be given points on how much advantage they have (can be negative) compared to other characters in the same role for all point in the match. For the chosen game (League of Legends) we will consider how strong a character is on every level. Since every character in League of Legends can get up to level 18, then there will be 18 steps of comparison. Items are also will be considered in the application of the algorithm. Other than only basic stat like how much hit damage a character has, or how much ability point a character has, or how much health a character has, the calculation will grouped on some categories given by the game itself.

In League of Legends, there are 7 basic stat for any character.

1. Health

   This stat represent how much damage a character can take before they die.

2. Health Regen

   This stat represent how much health gained through time while a character is alive.

3. Attack Damage (AD)

   This stat represent how much damage can a character deal to another object with basic attack. Attack damage is a form of physical attack unless described as magical attack by some passive ability given by the character or item.

4. Armor

   Armor stat represent how much protection a character has against physical attack. The calculation on how much protection a level of armor can give is interpreted to Effective Health.

$$Effective\ Health = (1 + \frac{armor}{100}) \times health$$

   For example if a character has 100 armor points, then the character's effective health against physical attack is doubled.

5. Attack Speed (AS)

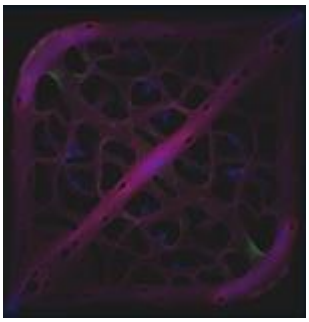
   This stat represents how many basic attack a character can do in a second. So if a character has 0.5 attack speed point, then the character can do 1 basic attack for every 2 seconds. The maximum value for Attack Speed point is 2.50, but some items allow the character to exceed this limitation for a limited period.

6. Magic Resist (MR)

   Magic resist is a similar stat to armor, but magic resist work against magic damage instead of physical damage.

7. Movement Speed (MS)

   The length of two point in the game is represented in the term of "units" the map of the game itself has 19600 units on its diagonal. The movement speed is basically how many units can a character travel for one second. So 325 MS means a character can travel through 325 units per second.



(picture 1, the map of League of Legends, taken from https://boards.euw.leagueoflegends.com on 12th May 2018)

V. Branch And Bound Application

Branch and bound algorithm is used to find the optimum build (item pick) for each champion. Then from the optimum build we can decide if a champion is unbalanced or not. The algorithm will also inform us of the maximum strength of a character for a given state.

A. *Stat Grouping*

For the algorithm implementation, we need to group the explained stat before using them as parameter because the basic stat doesn't represent the full game experience as it is. The grouping of the basic stats are as follow

1. The character's ability to deal damage to another character (Damage)

   The damage will be calculated based on how much damage can character deal in 1 second and how much damage a character can deal in 6 second. The difference is how damage a character can deal in 1 second is mostly called "burst damage" or sudden damage, mostly mage and some marksman character in League of Legend has a high affinity for this kind of damage. The 6 seconds damage is to measure the consistency of damage a character can output in a prolonged fight, this is an important factor for most hitter or shooter character that has high damage but not burst damage. The damage is a combination of damage from ability and basic attack.

2. The character's ability to absorb damage from another character (Toughness)

   Toughness score will be based on 4 stats, health, health regen, armor, and magic resist. The point from this parameter is basically how much damage a character can sustain in 6 seconds. The calculation will use real health rather than effective health.

3. The character's ability to apply harmful effect such as slow, or stun the enemy's character (Crowd Control)

   This is a pretty peculiar parameter to be considered. The calculation for this parameter is by the following rule :

   1. Crowd control value point is based on how far the distance can a character nullify from another characters. If a character can stun 2 enemy's character for 2 seconds, then the character's crowd control point is average character's movement speed (325) times how many character can be stunned at a time.
   2. The point calculation will be multiplied by how long the enemy character will be affected by the ability.
   3. If it is a continuous ability then the time multiplication factor is 5.
   4. If there are multiple abilities, then every ability will count.

4. The character's ability to move around the map and dodge attack (Mobility)

   Mobility score will be based from the character's movement speed and the character's ability to dash or jump from a point to another.

5. Character's ability to provide useful ability for the team (Utility)

   Utility score will be based on how much shield and heal a character can give for 5 seconds in the match. Other ability such as movement speed buff and damage increase will be judged accordingly.

B. *State Definition*

A state in the algorithm implementation is a combination of picked items and the value of all parameters discussed in the previous point. The depth of the tree will represents the level of the character, it is guaranteed that an optimum solution for each character has at least 17 steps to reach the goal (optimum state).

C. *Cost Function*

The goal of the branch and bound application in the character balancing problem is to make sure that a character is viable for most of the time in a match. So it is important to measure how good a character is compared to another character.

The cost function for this problem is defined as follow

$$\hat{c}(i) = \hat{f}(i) + \hat{g}(i)$$

Where

$$\hat{c}(i) = character\ strength\ at\ state\ i$$

$$\hat{f}(i) = character\ cummulative\ strength\ before\ state\ i$$

$$\hat{g}(i) = \bar{f}(i) - \ average\ character\ strength\ at\ the\ same\ level\ and\ role$$

Rather than expanding the state with least cost, the implementation of branch and bound in this problem will expand a state with the most cost. In the program implementation because the value of c function will be multiplied by -1, because the priority queue data structure available in most programming language is sorted ascending and not descending.

The problem raise when we're trying to calculate the g function. The value of average character strength for the given role is not acquired in the initial calculation, we don't have the data for another champion strength in the initial calculation. So we have to settle with given statistic from https://champion.gg. To calculate the initial character strength, the program will use the given highest win rate build from https://champion.gg.

(picture 2, example stat from https://champion.gg for a character)

D. Program Execution And Data Interpretation

To calculate how strong a character is I've created a program that calculate which item a character should take and the program will show the strength of a character at every given point in game.

Here is the program pseudocode that cover the basic logic of the program.

(I've created a database beforehand to store the information of each character and how every item interacts with each character)

```
Create connection with database
Character Char <- fetchCharacterFromDB(get
character name)

PriorityQueue Q
Q.enqueue(Char)
int cost <- 0
int level <- 0
int itemsNumber <- 0
Char.level <- level
Char.cost <- cost
Char.itemsNumber <- itemsNumber
Character currentCharacter <- Char
SetOfInteger avgVal <-
fetchAverageFromDB(getCharRole(Char))

While(currentCharacter.level < 18 or
currentCharacter.itemsNumber < 6){
  For(Char temp in tempChar.pickItems()){
    temp.addCost(avgVal) //this will
calculate the cost for a given state
    Q.enqueue(temp)
  }
  currentCharacter <- Q.dequeue
}
```

The program will show what items should a character buy throughout the game the character strength for each level.

Here is an example of the program execution

```
C:\Users\adity_000\StimaPaper>python BNB.py
Insert character name :
Ashe

List of Items :
Essence Reaver
Runaan's Huricane
Infinity Edge
Panthom Dancer
BloodThirster
Berseker's Greave

Average strength per level :
level 1 : 420
level 2 : 512
level 3 : 591
level 4 : 697
level 5 : 813
level 6 : 1066
level 7 : 1102
level 8 : 1439
level 9 : 1533
level 10 : 1722
level 11 : 1854
level 12 : 2272
level 13 : 2408
level 14 : 2543
level 15 : 2721
level 16 : 2789
level 17 : 2861
level 18 : 2981
```

```
C:\Users\adity_000\StimaPaper>python BNB.py
Insert character name :
Kallista

List of Items :
Blade Of The Ruined King
Runaan's Huricane
Infinity Edge
Panthom Dancer
BloodThirster
Berseker's Greave

Average strength per level :
level 1 : 430
level 2 : 522
level 3 : 593
level 4 : 685
level 5 : 801
level 6 : 989
level 7 : 1202
level 8 : 1245
level 9 : 1290
level 10 : 1376
level 11 : 1499
level 12 : 1598
level 13 : 1723
level 14 : 1923
level 15 : 2186
level 16 : 2236
level 17 : 2244
level 18 : 2276
```

Ashe is a carry character which means she can carries her team to win the game, so she scales well as the game progress. She's a representation of a well balanced character as her win rate is 50.51% and the delta value for carry role is 2.12%, so that means she's safely inside the balanced range. Some characters have declining average strength after level 14 because the other character in the game get way stronger. The highest average strength level recorded is from the character Jinx where the average strength level at level 19 is 3491, and her win rate is 51.85% which means the character is still justifiably balanced but is rather on the strong side. Jinx's growth is rather exponential because in level 8 her average strength level is still below Ashe's which means she need more time to be strong. the most underpowered character decided from the game is however from the character named Kallista, her strength level throughout the game is almost always lower than Ashe with the strength point at level 18 is 2276 and her win rate is 44.09% which mean she's grossly unbalanced. Some characters also have a spike where their average strength rise really high on a certain level, which mean it's the strongest point for the character in a game and if a character has more than 2 levels of the spike then it can be a justification for saying a character is statistically balanced.

The win rate grows does not exactly match the average strength value because there are some factors that the program does not consider such as how difficult to play a character and how many players play the character, however the result of the program is solid enough to decide that a character is not balanced.

VI. Conclusion

Results from the algorithm does not exactly reflect the win rate of each character. There are also some simplification in the program that may cause the result to have some anomaly for some characters. but it can be concluded that the results is good enough to become a consideration if a character is balanced or not.

Acknowledgement

Firstly the author would like to thank god for his blessing and his giving to the author so that the author is able to finish this paper.

Secondly I'd like to thank my parents and my whole family for supporting me to continue my study in college, and especially to the government program for giving me scholarship.

I'd like to also thank the IF2211 lecturers that has created the lecture material so that it's easily understandable and reachable.

I'd also like to thank my friend Ramon who introduced me to the game and help me to calibrate the calculation in my program.

References

[1] Munir, Rinaldi *Slide of IF2211 : Strategi Algoritma, Branch and Bound*

[2] https://champion.gg, accessed on 11th May 2018.

[3] https://www.geeksforgeeks.org/branch-and-bound-set-1-introduction-with-01-knapsack/, accessed on 12th May 2018.

[4] Clausen, Jens (1999). Branch and Bound Algorithms—Principles and Examples (PDF) (Technical report). University of Copenhagen.

DECLARATION

I hereby certify that this paper is my own writing, neither a copy nor from another paper, and not an act of plagiarism

Bandung, May 13 2018

M Aditya Farizki/13516082