Automatic Intersection Extraction and Building Arrangement with StarCraft II Maps

Yuanbin Cheng
University of Southern California
Los Angeles, California
yuanbinc@usc.edu

Yao-Yi Chiang
University of Southern California
Los Angeles, California
yaoyic@usc.edu

ABSTRACT

In StarCraft, buildings arrangement near the intersections is one of most the critical strategic decisions in the early stage. The high time complexity of the buildings arrangement makes it difficult for AI bot to make the real-time decision. This paper presents an approach to analyze the intersection in StarCraft II maps. We propose a radar-like algorithm to automatically detect the intersection and use a designed heuristic search algorithm to arrange the building for building the wall. Our method can obtain the optimal solution while meeting the real-time requirement.

CCS CONCEPTS

• **Information systems** → *Spatial-temporal systems*;

KEYWORDS

Spatial Analysis, Heuristic Search, StarCraft

1 INTRODUCTION

StarCraft II is one of the most popular and successful real-time strategy (RTS) games in the world. From the artificial intelligence (AI) perspective, StarCraft II is the game with multiple agents (player can control hundreds of the units in game), imperfect information ("fog-of-war" cover the unvisited region of the map), vast and diverse action (Units can move to every point reachable in map, different unit has different abilities), which make StarCraft II a challenging playground for researchers to build the AI bot to automatically play the game [2].

Spatial analysis of the game map is important for the AI bot to make critical strategic decisions such as building planning and route planning. Building arrangement often plays a vital role in StarCraft II, especially for Terran and Protoss (two races in the game). It is a common strategy to arrange the buildings to block the narrow intersections for defending the opponent's army in the early stage of the game.

Blocking narrow intersections brings many challenges to the AI bot. First, it needs to determine the position of the critical intersection, which separates the different areas in the map. Second, AI bot need to make the arrangement decision for blocking the intersection in real-time. That means the bot should have the ability to determine the position of the critical intersection and arranging the buildings in short response time.

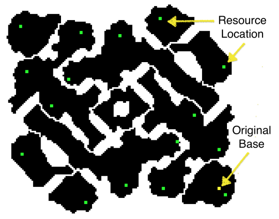
This paper presents an approach to determine the intersection in StarCraft II maps automatically in real-time. Our method initiates a radar-like algorithm to determine the intersection position. After finding the intersection, a building-arrange algorithm based on A* algorithm identifies the best locations of the buildings. The exposure is the boundary of the buildings that enemy units can directly attack, the shorter length of the exposure; the fewer enemy units can attack the block. The building-arrange algorithm can make sure the arrangement result has the shortest length of the exposure. Moreover, compared to the general search algorithm (BFS, DFS), the building-arrange algorithm can significantly reduce the searched space to shorten the response time.

2 APPROACH

During the game, some information can be derived from the StarCraft II in-game API,[1] including locations of the player and enemy's original base, resources locations on the map, and two grid maps. In Figure 1, the first grid map represents the buildable grids of the map. The second grid map in figure 2 indicates the grids that units can pass through it.

2.1 Intersection Detection

Our method first identifies two types of intersections that are the narrow passable place separate the different areas in the map.



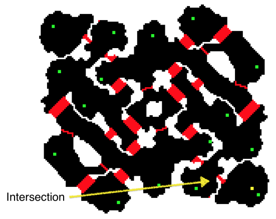


Figure 1: Buildable Grid Map **Figure 2: Passable Grid Map**

2.1.1 Detecting the Intersections Between Different Heights. Many essential intersections appear between two planes with different heights. For detecting this type of the intersection, our method extracts the areas in the map that units can pass through while the buildings cannot be built, which is the common feature of this kind of intersection area, by comparing the difference between two grid maps. Figure 2 displays the result of this kind of intersection.

2.1.2 Detecting the Intersection of Narrow Terrain. Another kind of the critical intersections appears in the narrow terrain area connecting the different areas in the map. We propose a radar-like

[1]https://github.com/Blizzard/s2client-api

algorithm to detect this type of intersections. This algorithm iteratively searches for intersections by increasing the search radius from the start point, typically the locations of base and resource. This algorithm obtains the reachable boundary at each iteration step and recorded the minimize over the previous steps. After the total iteration, the algorithm finds the connected components of the minimize boundary as the candidate intersections. Figure 3 shows some internal states and Figure 4 present the result. In this situation, there is only one candidate intersection.

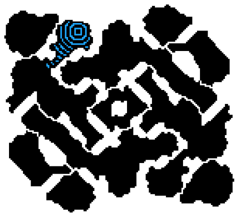


Figure 3: Radar-like Internal states

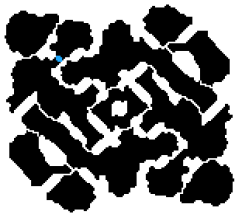


Figure 4: Radar-like Detection Result

2.1.3 Validity of Intersections. After getting the candidate intersections, we need to verify that the detected intersections can separate the map into multiple parts. Our method compares the number of the reachable grids from the start point before and after removing the intersections to determine whether the intersection is valid. If the number is significantly decreased, this intersection is valid.

2.2 Building Arrangement Search

After detecting of the critical intersections in the map, we utilize the building-arrange algorithm based on A* algorithm to find the optimal building arrangement for blocking the intersection.

2.2.1 Border of the intersection. In order to block the intersection, the building must be close to the border, the unpassable areas and unbuildable areas.

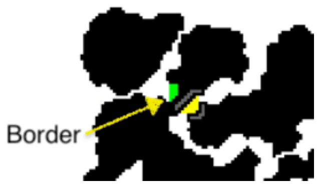


Figure 5: Border and Possible Building Location

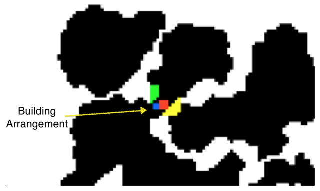


Figure 6: Building Arrangement Result from A* Search

We located the border areas near the intersection by finding all unpassable and unbuildable grids within three grids from the intersection. The green and yellow areas in Figure 5 show the two border areas of the intersection.

2.2.2 Building-arrange Algorithm for Building Arrangement. In StarCraft II, Terran usually use Supply Depot (2×2 building) and Barracks (3×3 building) for blocking the intersection, and Protoss usually use Pylon (2×2 building) and Gateway (3×3 building) for blocking.

The blocking arrangement of the building should have two properties. First, at least one building is close to each of the border; otherwise, the unit can pass through the gap between the border and the building. Second, each building must be adjacent to at least one building; otherwise, the unit can pass through the gap between the buildings.

We designed the building-arrange algorithm based on A* search algorithm to search the building arrangement (as the node) from one side of the border until reaching to another side of the border (goal). Algorithm 1 shows the algorithm process. The *candidate*() obtain the candidate building locations near the current border (Gray color areas in Figure 5). The *minManh*() gain the minimize Manhattan distance between buildings in the current state and another part of the border (the heuristic function). Moreover, the *boundary*() acquires the length of the exposure boundary of the current state's buildings (the cost function).

Algorithm 1 Building-Arrange Algorithm

```
Input: start, goal(), candidates(), boundary(),
1:  if goal(start) = true then
2:      return arrangment(start)
3:  $open \leftarrow start, closed \leftarrow \emptyset$
4:  while $open \neq \emptyset$ do
5:      sort(open)
6:      $n \leftarrow open.pop()$
7:      $candidates \leftarrow candidates(n)$
8:      for all $cand \in candidates$ do
9:          $cand.f \leftarrow cand.boundary + minManh(cand)$
10:         if $goal(cand) = true$ then
11:             return arrangment(cand)
12:         if $goal(cand) = false$ then
13:             $open \leftarrow cand$
14:     $closed \leftarrow n$
15: return $\emptyset$
```

The minimize Manhattan distance is the lower bound of the exposure boundary for any blocked buildings arrangements, which can make sure that the algorithm obtains the optimal solution [1]. The time complexity in the worst case is $O(b^d)$, b is the number of the candidates and d is the number of buildings in the arrangement result. Figure 6 shows the building arrangement result from the building-arrange search.

3 EXPERIMENTS

We conducted experiments on all maps from the StarCraft II Ladder 2017. For the original base location and the closest expand resource location, our approach could find the right intersection and plan the appropriate building locations for all maps. The number of nodes expanded in search for all map is less than one hundred, which meet the real-time requirement. For the start point in the middle of the map, due to the increase in connectivity paths and the variety of the terrain, our approach build lots of redundant buildings for blocking the location and need to expand hundreds of nodes to get the building arrangement.

REFERENCES

[1] P. E. Hart, N. J. Nilsson, and B. Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (July 1968), 100–107.
[2] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. P. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing. 2017. StarCraft II: A New Challenge for Reinforcement Learning. *CoRR* abs/1708.04782 (2017).