Exploration in NetHack Using Occupancy Maps

Jonathan Campbell
School of Computer Science
McGill University, Montréal
jcampb35@cs.mcgill.ca

Clark Verbrugge
School of Computer Science
McGill University, Montréal
clump@cs.mcgill.ca

ABSTRACT

Roguelike games generally feature exploration problems as a critical, yet often repetitive element of gameplay. Automated approaches, however, face challenges in terms of optimality. This paper presents an approach to exploration of roguelike dungeon environments. Our design, based on the concept of occupancy maps popular in robotics, aims to minimize exploration time, balancing coverage with resource cost. Through extensive experimentation on NetHack maps we show that this technique is significantly more efficient than simpler greedy approaches. Results point towards better automation for players as well as heuristics for fully automated gameplay.

CCS CONCEPTS

•Applied computing → Computer games; •Computing methodologies → Planning under uncertainty;

KEYWORDS

Roguelikes, Exploration, Occupancy Maps

ACM Reference format:
Jonathan Campbell and Clark Verbrugge. 2017. Exploration in NetHack Using Occupancy Maps. In *Proceedings of FDG'17, Hyannis, MA, USA, August 14-17, 2017,* 4 pages.
DOI: 10.1145/3102071.3106345

1 INTRODUCTION

In *roguelikes*, a popular subset of Role-Playing Games (RPGs), exploration of game space is a key game mechanic, essential to resource acquisition and game progress, but also a major source of resource cost. In this work we present a novel algorithm for exploration of an unknown environment that aims for an efficient, balanced approach to exploration, considering the cost of further exploration in relation to the potential benefit, as well as factoring in the relative importance of different areas (rooms versus corridors). Our approach is inspired by a variation of occupancy maps, adapted from robotics into video games [5]. With this method we can control how the space is explored, following a probability gradient that flows from places of higher potential benefit.

We compare this approach with a simpler, greedy algorithm, applying both to levels from the canonical roguelike, *NetHack*. This environment gives us a realistic and frequently mimicked game context, with uneven exploration potential (rooms vs. corridors), critical resource limitations (each move consumes scarce food resources), and a non-trivial, dungeon-like map environment. Our algorithm shows improvement in overall efficiency compared to the greedy approach.

Specific contributions of this work include:

- We heavily adapt a known variation on occupancy maps to the task of performing efficient exploration of dungeon-like environments.
- Our design is backed by extensive experimental work, validating the approach and comparing it with a simpler, greedy approach.

FDG'17, Hyannis, MA, USA

DOI: 10.1145/3102071.3106345

2 BACKGROUND & RELATED WORK

Automated exploration or mapping of an environment has been frequently studied in several fields, primarily including robotics and with respect to the problem of graph traversal, with the latter having some connections to video games.

Exploration in robotics branches into many particular sub-problems. Good surveys of robotic exploration algorithms can be found in [6] and [8]. One popular algorithm for exploring an unknown environment is called occupancy maps. This algorithm maintains a grid of cells over a space, with each cell representing the probability that the corresponding area is occupied (e.g., by an obstacle or wall). This structure offers an easy way to store observations of the environment and determination of whether an area is impassable, traversable, or as of yet unknown [9, 10].

A representation of the learned map must then be leveraged to decide where to move next for efficient exploration. Yamauchi described a strategy using occupancy maps to move towards the closest frontier [12]. Gonzàlez-Baños and Latombe discussed taking into account both distance to a frontier and the utility of that frontier (a measure of the unexplored area potentially visible from that position) [2].

Some work on exploration has also been done with respect to video games. Chowdhury looked at approaches for computing a tour of a known environment in the context of exhaustive exploration strategies for non-player characters in video games [1]. Hagelbäck and Johansson explored the use of potential fields to discover unvisited portions of a real-time strategy game map in order to create a better computer AI for the game [3]. Our work, in contrast, focuses on uneven exploration in sparse, dungeon-like environments, where complete methods compete with critical resource efficiency.

Occupancy Maps in Games. Using the aforementioned occupancy maps from robotics as inspiration, Damián Isla created an algorithm to search for a moving target in a video game context [4]. This algorithm maintains a discretized grid of probabilities over a space (e.g., game map), with each probability representing the confidence of that area containing the target. When the searcher does not observe the target at any timestep, all cell(s) in the current field of view have

their probabilities set to 0, since there is complete confidence that those cells do not contain the target. All probabilities in the grid then diffuse to their neighbours, to account for the possibility that the target has moved in unseen areas. Diffusion for each cell n at time t is performed according to the following formula (assuming each cell has four neighbours):

$$P_{t+1}(n) = (1-\lambda)P_t(n) + \frac{\lambda}{4}\sum_{n' \in neighbours(n)} P_t(n')$$

where $\lambda \in (0,1)$ controls the amount of diffusion.

Our implementation of occupancy maps borrows some concepts from Isla's formulation, namely, the idea of diffusion, which is repurposed for an exploration context.

NetHack. NetHack is a popular roguelike created in 1987 and is used as our experiment environment. Gameplay occurs on a 2D text-based grid of size 80x20, wherein a player can move around, collect items, fight monsters, and travel to deeper dungeon levels. Levels in NetHack consist of large, rectangular rooms connected by maze-like corridors and are for the most part procedurally generated. The player starts in a random room with the rest of the map hidden, and must explore to uncover more. An example of a typical Nethack map is presented in Figure 1.

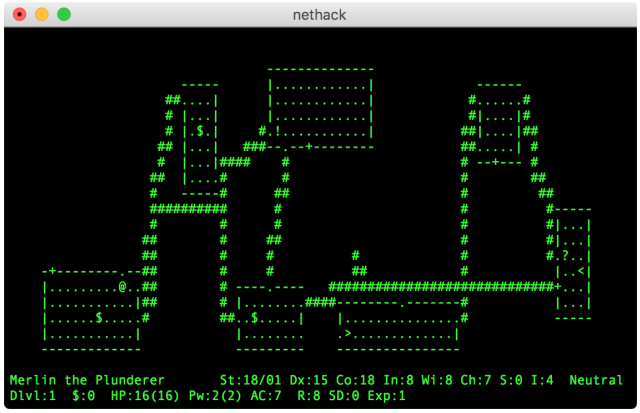

Figure 1: A game of NetHack where the player ('@' character, currently in the bottom-left room) has explored most of the level. A typical NetHack map is composed of corridors ('#') that connect rectangular rooms. Room spaces ('.') are surrounded by walls ('|' and '-'), and unopened doors ('+'), which could lead to other, unvisited rooms. The bottom two lines contain information about the player's current attributes and statistics.

Movement in NetHack is turn-based (each move taking one turn), and the more turns made, the more hungry one becomes, until starvation (death) occurs. Hunger can be satiated by food, randomly and sparingly placed within the rooms of a level [11]. Since food does not regenerate, a player must move to new levels at a brisk pace.

In this context, it is critical to minimize the number of turns spent on exploration, in order to preserve food resources. Rooms are critical to visit since food and helpful items can be found in them, but conversely, the corridors that connect the rooms have no intrinsic value. Some may lead to dead-ends or circle around to already visited rooms. Exploring all corridors of a level is typically considered a waste of valuable actions.

3 EXPLORATION APPROACH

Here we present two algorithms: a trivial greedy approach which guarantees complete coverage of the space, as well as a nuanced approach based on occupancy maps, which will only visit useful frontiers and do so in a specific order. First, we discuss the exploration environment.

Environment

A modified version of NetHack is used to test our exploration algorithms. Game features that might confound experiment results were removed, including monsters, starvation, weight restrictions, and certain dungeon features that introduce an irregular field of view. The maps used in testing are those generated by NetHack for the first level of the game. The same level generation algorithm is used throughout a large part of the game so this does not limit generality, although later levels can contain special, fixed structures.

The algorithms below use the NetHack player field of view. When a player enters a room in NetHack, they are able to immediately perceive the entire room shape, size, and exits (doors). In corridors, knowledge is revealed about only the immediate neighbours to the player's current position. Our algorithms will gain the same information.

Greedy algorithm

A greedy algorithm is used as baseline for our experiments, which simply always moves to the frontier closest to the player. This type of approach is often formalized as a graph exploration problem, where we start at a vertex v, learn the vertices adjacent to v, move to the closest unvisited vertex (using the shortest path) and repeat [7]. The algorithm terminates when no frontiers are left. We also take into account the properties of the NetHack field of view as described above (when we enter a room, all positions in the room are set to visited, and its exits are added to the frontier list).

Occupancy maps

Our main exploration strategy is similar to both the original conception of occupancy maps as a way to store information about obstacles, as well as Damián Isla's formulation for searching for a target. The algorithm is designed to optimize exploration time by minimizing amount of time spent in unhelpful areas (e.g., corridors in NetHack).

The main idea is to use the occupancy map to represent probabilities of (unexplored) room presence. This formulation lets us determine the most useful frontiers to move towards, and thus edge an advantage over the greedy approach, which does not consider frontier utility. We also use a form of diffusion from Isla's formulation in the occupancy map, described below. Figure 2 gives a visualization of a sample occupancy map.

Occupancy map representation. The probability of a cell in the map represents the confidence we have in that area containing an unvisited room. Similar to the occupancy maps of robotics, here we set the cells of observed obstacles to 0. Specifically, whenever we see a room/corridor spot, we add it to our memory; at each timestep, we set the probability of each spot in our memory to 0 in the occupancy map. We must do this for all spots in memory at every timestep since the diffusion step we run (detailed below) may alter them.

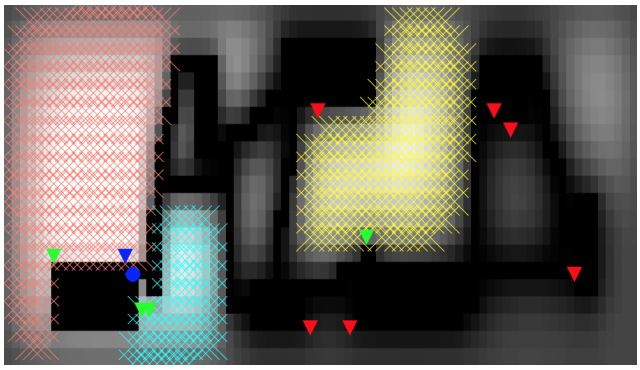

Figure 2: Visualization of an occupancy map corresponding to the NetHack level of Figure 1. Darker areas are less likely to contain an undiscovered room. The player is shown as a blue circle, and current target frontier as blue triangle. Other frontiers are shown as green triangles, while red triangles are frontiers that will not be visited due to being in areas of low probability. Components with neighbouring frontiers are highlighted in a criss-cross pattern.

Diffusion. We use diffusion for two purposes: to influence probability of a cell based on its neighbours in order to better measure its utility, as well as to separate the occupancy map into distinct components of high probability.

Diffusion affects the utility of a frontier. By dispersing the zero probability of visited rooms into surrounding areas, we can more easily identify frontiers that are close to low probability areas. We can then ignore frontiers that are surrounded by cells of low probability (when all of the neighbours of a frontier have a probability below a certain threshold) as seen in figure 2 (red triangles indicate such frontiers).

For extra diffusion, we also diffuse inward from the borders of the occupancy map. Specifically, during the diffusion step, when updating cells that lie on the edges of the map, we treat their out-of-bounds neighbours as cells with a fixed low probability. Diffusing in this manner tends to increase separation of components of high probability (since rooms/corridors rarely extend to the edge of the map). More importantly, it lessens the utility of frontiers that lie near the edges of the map, which are most probably dead-ends.

Diffusion is run at each timestep that a new part of the map (room/corridor) is observed. By diffusing only at these times, probabilities in the occupancy map won't change while we are travelling to a frontier through explored space, and neither will the length of distance travelled have an effect. Probabilities will diffuse at the rate that map spaces are uncovered, and stop when the map is completely known.

Planning with occupancy maps. The knowledge inside the occupancy map must now be exploited to explore a level while minimizing number of actions taken. The idea is to split the map into components of high probability, choose the best component (based on distance and/or utility), and then move to a frontier neighbouring that component.

Splitting the map into components of high probability is performed to understand which distinct areas of the map are interesting. Components are retrieved by running a depth-first search on the occupancy map and considering any unvisited cell with probability above a threshold to be passable. We also consider cells that have less than a certain number of passable neighbours to be impassable themselves, further separating components by eliminating narrow alleys that could otherwise connect two disparate components.

Some components are ignored due to size or neighbour limitations. If a component is smaller than the minimum room size, it is impossible for a room to be there. Likewise, if a component has no neighbouring frontiers, it cannot contain a room since there is no access point.

To determine the most promising remaining component, an evaluation function is used that considers component utility and distance to player. Utility is calculated by summing the probabilities of all cells in the component. (The sum is then normalized by dividing by the sum of all probabilities in the map.) To determine distance to player, the component is first matched to the closest frontier on the open frontiers list (by calculating the Manhattan distance from each frontier to the closest cell in the component). Distance is then calculated as: $d(frontier,player)+d(frontier,closest\_component\_cell)$ (the former calculated using A*, and latter using Manhattan distance, since that part of the path is unknown). This distance is then normalized by dividing by the sum of the distances for all frontiers for the specific component under evaluation. With the normalized utility and distance values, we pick the component that maximizes $norm\_prob+\alpha*(1-norm\_dist)$, where α controls the balance of the two criteria.

Once the best component is determined, the algorithm moves to the frontier matched to that component. On arrival, it will learn new information about the game map, update the occupancy map, and run diffusion. Components will be re-evaluated and a new target chosen. Exploration terminates when no interesting frontiers remain.

4 EXPERIMENTAL RESULTS

Results will be shown below for the greedy and occupancy map algorithms as a function of the exhaustive nature of their searching. We will look first at metrics for comparison.

Exploration metrics

To evaluate the presented exploration strategies, we use as metrics the average number of actions per game as well as average percentage of rooms explored. To get a more fine-grained view of exploration which penalizes incomplete room discovery on a level, we also use a third metric which counts partially-explored map runs (runs that fail to explore all rooms on a map) as 0 (with full exploration runs counted as 1). We call this the 'full runs only' exploration metric.

Exhaustive approaches

The result for the greedy algorithm is presented in figure 3, at around 315 average actions per game to explore the complete map (i.e., exploring all rooms and corridors). The result for the fastest occupancy map model with parameters that guaranteed exploration of all rooms is also shown, at about 280 average actions per game.

In the same figure, the result is given for an approximately optimal solution, obtained by running a travelling salesman problem solver on a NetHack map. Maps are translated to graphs with room centroids as vertices and edges as connections between rooms. The length of an edge corresponds to the shortest distance between the two connected room centroids in the map. The initial vertex in the

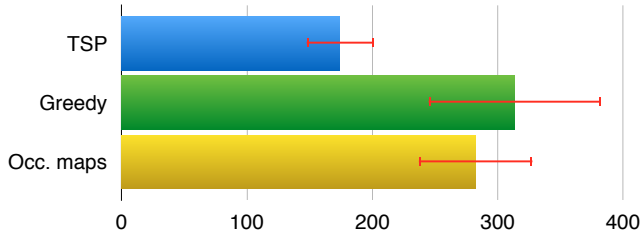

Figure 3: Average number of actions taken by the optimal solution (TSP) and greedy and occupancy map algorithms for complete room exploration with best performing parameters. The average over 200 runs on different randomly-generated NetHack maps is taken. Error bars (standard deviation over all runs) are presented in red.

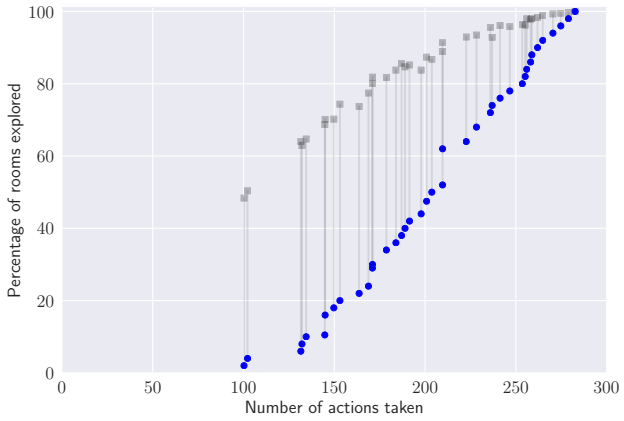

Figure 4: Occupancy map models with parameters that best minimize average actions per game and maximize percentage of rooms explored. Each blue dot represents the average over 200 runs using a different combination of model parameters. The blue dots show the result under the 'full runs only' metric and the corresponding black squares show the total percentage of rooms explored.

graph is set to the player's starting room. The TSP solution guarantees exploration of all rooms (not necessarily all corridors) similar to the occupancy map algorithm. It is an approximation to the fastest way to explore a known NetHack map, since some time could be saved by moving from room exit to room exit instead of centroid to centroid. As seen in the figure, it visits all rooms in about 175 actions on average. The speed difference between it and the greedy algorithm is a result of both inefficient exploration as well as full corridor exploration on the part of the greedy algorithm.

Non-exhaustive approaches

Exhaustive approaches are fine in certain circumstances, but often it's acceptable to sometimes leave 10 or 20% of the map unexplored, especially when there is a cost to movement. Figure 4 gives the results for the best-performing non-exhaustive occupancy map models in terms of fastest time vs. highest room exploration. (A grid search over the parameter space was performed – the models shown lie on the upper-left curve of all models.)

As seen in the figure, there is a mostly linear progression in terms of the two metrics. The relationship between the 'full runs only' metric and total percentage of explored rooms is also consistent, with both linearly increasing.

The specific parameter values that led to the fastest performing exhaustive exploration model were as follows: diffusion factor of 0.5, border diffusion of 0.5 and probability threshold of 0. The parameters for the fastest model at 80% non-exhaustive exploration (the full map being explored about 30% of the time) with 175 avg. actions were: diffusion factor of 0.5, border diffusion of 0 (smaller values diffuse more), and probability threshold of 0.

5 CONCLUSIONS & FUTURE WORK

Automated exploration is an interesting, surprisingly complex task. In strategy or roguelike games, the tedium of repetitive movement during exploration is a concern for players, and offering efficient automation can be helpful. Exploration is also a significant sub-problem in making more fully automated, learning AI, and techniques which can algorithmically solve exploration can be useful in allowing AI to focus more on higher level strategy than basic movement concerns.

In this work we detailed an algorithm for efficient exploration of an unknown environment. Inspired by the occupancy map algorithm of robotics and the similar approach by Damián Isla, we built an algorithm to select frontiers to visit when performing exploration of interesting areas of a map, while also considering complete coverage. Our design notably improves over a more straightforward, greedy design.

Our further work on the occupancy map algorithm aims to increase efficiency in exploration. In particular, a 'local' diffusion of probabilities instead of the current global diffusion may prove fruitful. Further verification of the algorithm on other games would also be interesting.

ACKNOWLEDGMENTS

This work supported by NSERC grant 249902.

REFERENCES

[1] Muntasir Chowdhury and Clark Verbrugge. 2016. Exhaustive Exploration Strategies for NPCs. In *Proceedings of the 1st International Joint Conference of DiGRA and FDG: 7th Workshop on Procedural Content Generation.*
[2] Héctor H. Gonzàlez-Baños and Jean-Claude Latombe. 2002. Navigation Strategies for Exploring Indoor Environments. *International Journal of Robotics Research* 21, 10-11 (2002), 829–848.
[3] J. Hagelbäck and S. J. Johansson. 2008. Dealing with fog of war in a Real Time Strategy game environment. In *Computational Intelligence in Games*. 55–62.
[4] Damián Isla. 2005. Probabilistic Target-Tracking and Search Using Occupancy Maps. In *AI Game Programming Wisdom 3*. Charles River Media.
[5] Damián Isla. 2013. Third Eye Crime: Building a Stealth Game Around Occupancy Maps. (2013). Artificial Intelligence and Interactive Digital Entertainment.
[6] Miguel Juliá, Arturo Gil, and Oscar Reinoso. 2012. A comparison of path planning strategies for autonomous exploration and mapping of unknown environments. *Autonomous Robots* 33, 4 (2012), 427–444.
[7] S. Koenig, C. Tovey, and W. Halliburton. 2001. Greedy mapping of terrain. In *IEEE Int Conf Robot Autom*, Vol. 4. 3594–3599.
[8] S. M. LaValle. 2006. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K. Available at http://planning.cs.uiuc.edu/.
[9] Hans Moravec. 1988. Sensor Fusion in Certainty Grids for Mobile Robots. *AI Magazine* 9, 2 (July 1988), 61–74.
[10] Hans Moravec and A. E. Elfes. 1985. High Resolution Maps from Wide Angle Sonar. In *IEEE Int Conf Robot Autom*. 116–121.
[11] NetHack Wiki. 2016. Comestible — NetHack Wiki. https://nethackwiki.com/wiki/Comestible. (2016).
[12] B. Yamauchi. 1997. A frontier-based approach for autonomous exploration. In *CIRA*. 146–151.