Extended Abstract
The 23rd Thailand-Japan Conference on
Discrete and Computational Geometry, Graphs, and Games
Department of Mathematics, Chiang Mai University

Computing the Probability of Striking a Battleship

Jack Spalding-Jamieson[1]

[1]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada
jacksj@uwaterloo.ca (J. Spalding-Jamieson)

Abstract

A set of n non-overlapping rectangular 'battleships' with unit width are placed on an $H \times W$ 2D square grid uniformly at random. Given the state of an in-progress game, and an initial assumption of uniform probability over all legal configurations of the ships, our goal is to determine the probability that each square contains a ship. We describe and implement a practical solution to this problem, and extend our implementation to run on GPU.

Keywords: battleship, graph, clique, probability, GPU.

2010 MSC: Primary 68W05; Secondary 91A60, 05C85, 68W10.

1 Introduction

Battleship is a game played between two people. Each player starts by placing 'ships' into hidden locations on a 2D square grid unique to their own ships, and then players take turns attempting to 'attack' locations on the others' grid.

In [4], Fiat and Shamir describe an overarching strategy that bounds the worst-case number of attacks required, in the case of one ship. In this paper, we attempt to evaluate local choices within a game of battleship. Our goal is to quickly compute the exact probability of striking a ship with an attack, for each possible next choice of square point to attack. A method for computing or estimating these probabilities has previously been explored [3], but it is much slower than the techniques described here.

1.1 Probability Queries

We consider only one side in the game of battleship. A defending player places n unit-width ships $S=\{s_1,...,s_n\}$ with lengths $L=\{l_1,...,l_n\}$ onto an $H \times W$ square grid, such that no two ships occupy the same grid square. Note that we allow the ships to touch, but we will not make special use of this fact. An attacking player is allowed to perform queries, called **attack**s on squares, with at most one query per square. Each query on a square r has three possible responses.

If the attack **missed**, there is no ship at the sqaure.

If the attack **hit**, there is a ship at the square, and some part of the ship remains undiscovered.

If the attack **sunk**, there is a ship at the square, and all parts of the ship have now been discovered. The specific ship itself is identified in the query response, which we say has been sunk at square r.

The attacking player wishes to minimize the number of queries needed to sink all ships.

In this work we attempt to practically solve the following problem: Assume the ships were placed using a uniform distribution over all possible legal configurations of ship placements. Given a series of queries made by the attacker so far, and their responses, determine the probability that each square of the grid contains a ship.

2 Cliques and the Placement Graph

For a set of ships L and a grid size $H \times W$, we construct **placement graph** G_1 with the following vertices and edges: For each ship s, and each possible placement p of s onto the grid (a location and orientation), create a vertex $v_{s,p}$. Create an edge for each pair of vertices $v_{s,p}, v_{s',p'}$ with $s \neq s'$ where s placed at p would not intersect with s' placed at p'. The configurations of all ships correspond to n-cliques in G which cover all squares of the grid that have received hit responses.

Then, given a list of queries and responses so far, we can construct the **valid placement subgraph** G_2 which is an induced subgraph of G_1 with the vertices $v_{s,p}$ such that:

- No square r that has received a miss response is inside the placement p of s.
- If a ship s has already been sunk at a square r, then $v_{s,p}$ covers only r and squares which received hit responses prior to the sunk response of r.

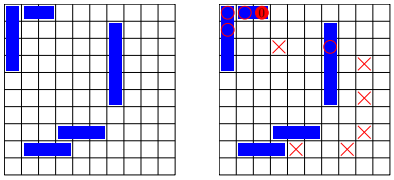

Figure 1: Left: A configuration of ships on the 10×10 grid. Right: The same configuration after some attacks. The 6 $\times$ symbols are misses. The 4 empty circles are hits. The filled circle with a 0 inside is a sink of ship 0. Note that the order of the attacks is not specified here.

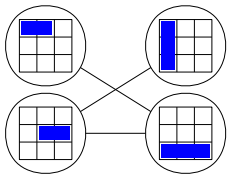

Figure 2: For a 3×3 board with $L=\{2,3\}$, the above diagram represents a small subgraph of G_1.

Using the valid placement subgraph, we can compute our probabilities with a simple algorithm. Let $h(v_{s,p})$ be the number of hit squares s touches, and let H be the total number of hit squares. We iterate through all n-cliques C of G_2. If $\sum_{v\in C}h(v)=H$, then C corresponds to a valid configuration of all ships given all the information we have. For each vertex $v_{s,p}$, we let $f_{v_{s,p}}$ denote the number of valid configurations containing the ship s in placement p. Finally, the total number of valid configurations covering a square r is the sum of $f_{v_{s,p}}$ over all placements p of a ship s that overlap with r. This value is sufficient to compute the exact probability that r appears in a configuration, by dividing by the total number of valid configurations.

3 Clique Iteration and Performance Engineering

The primary performance bottleneck of this approach is the iteration through all n-cliques. Bron and Kerbosch describe a method for iterating through all maximal cliques of general graphs [2]. However, for our n-partite graph, this can be done much more quickly with a simple recursive approach: Iterate through all placements p_1 of s_1. Inside the iteration of p_1, iterate through all placements p_2 of s_2 which do not conflict with the current placement of s_1. Inside the iteration of p_2, iterate through all placements p_3 of s_3 which do not conflict with the current placements of s_2 or s_1. And so on. This pattern can be implemented as a simple recursive function.

An implementation of our simple recursive approach can be refined for much better real-time performance: For each ship s_i, a set V_i of placements of s_i which do not intersect with the current placements of ships $s_1,s_2,...,s_{i-1}$ can be maintained. Furthermore, these sets can be updated quickly too: For each pair of ships s_i and s_j with $i<j$, and every pair of placement p_i of s_i, we can compute a set $B_{i,p_i,j}$ of placements of j which do not overlap p_i. Note that all such sets can be computed prior to the n-clique iteration, and that in practice the computation of these sets is negligible. When choosing a placement p_i for a ship s_i, each set V_j for $j>i$ can now be updated by performing the operation $V_j\leftarrow V_j\cap B_{i,p_i,j}$ By storing each V_i as a bitset, this operation can be performed extremely fast in practice.

This algorithm can also be adapted for a GPU: Pre-compute the first few levels of recursion on CPU, and compute the remaining levels in parallel on GPU.

4 Experimental Results

We evaluated our implementation on the version of the game with $n=5$, $L=\{5,4,3,3,2\}$, and $H\times W=10\times10$. The input with the most valid configurations is the empty board, so it provides an upper bound on performance. We tested on an AMD Ryzen 5 1400 CPU and an NVIDIA RTX 3080 GPU. Our basic single-threaded CPU implementation was able to solve the empty board in 146.37s. Our parallelized implementation using hipSYCL [1] completed in 5.70s on GPU, and in 36.36s using 8 CPU threads.

Acknowledgments. The authors are grateful to the referees for their careful reading of the manuscript and their useful comments.

References

[1] Aksel Alpay and Vincent Heuveline, *Sycl beyond opencl: The architecture, current state and future direction of hipsycl*, Proceedings of the International Workshop on OpenCL (New York, NY, USA), IWOCL '20, Association for Computing Machinery, 2020.

[2] Coen Bron and Joep Kerbosch, *Algorithm 457: Finding all cliques of an undirected graph*, Commun. ACM **16** (1973), no. 9, 575–577.

[3] C. Liam Brown, *Battleship probability calculator: Methodology*, https://web.archive.org/web/20210709154432/https://cliambrown.com/battleship/methodology.php, Accessed: 2021-07-09.

[4] Amos Fiat and Adi Shamir, *How to find a battleship*, Networks **19** (1989), no. 3, 361–371.