The Game of Deployment

Defining a game for studying sensor deployment problems.

Brian Carter
Chippewa Software Technology
Louisville KY, USA
briancarter@chipsofttech.com

Rammohan Ragade
Computer Engineering and Computer Science
University of Louisville
Louisville KY, USA
ragade@louisville.edu

Games are often used as a tool for teaching the elements and strategies of a problem. The objective of this paper is to define the sensor deployment problem as a game. The rules, game mechanics, and attributes are defined to abstract real life deployment situations while lowering the learning time to allow study of complex scenarios. This paper defines the fundamentals of the game and challenges the reader to study, solve, and define new strategies for sensor deployment problems. It is a fun and simple game, which has an element of entertainment, yet provides a foundation and framework for studying sensor deployment problems.

Sensor, deployment, sensor deployment problem, game of deployment, optimization, board game

I. INTRODUCTION

Games give us a fun means of taking on real life challenges without all the baggage. We can then apply what the game taught us to real life situations because our capacity for analysis has been honed and expanded.

The challenge of sensor deployment in sensor networks is a problem that is perplexing [1]. What if there could be a game that would train a player's mind to consider the facets of the real life problem, with the objective of inventing and resolving sensor deployment challenges? What if students became players in the sensor deployment work? This is our objective.

We have invented The Game of Deployment. This paper presents its concepts and rules, and invites you to play along: either create a deployment challenge, or invent a solution. Our game is a puzzle, a problem that tests the ingenuity of the solver.

The focus of this paper is sensor deployment, but the game may be used to study advanced sensor network topics, such as topology control and power scheduling, without the complexities found in past studies [2, 3].

II. THE GAME OF DEPLOYMENT

Envision a set of sensors and an infinite square grid with some unit square tiles. The sensors are heterogeneous, having varying capacities. A finite set of the unit square tiles on the grid are marked as requiring coverage. Consider:

How can I place a finite number of heterogeneous sensors on the square grid such that all tiles requiring coverage are satisfied while optimizing the cost?

We take this problem and define it as a board game. A game in which pieces are placed, removed, or moved on a pre-marked surface or "board" according to a set of rules.

The rules of board games range from simple to complex, and the length of time needed to learn and master a game varies greatly. Learning time does not necessarily correlate with the number or complexity of rules; such games as chess and checkers have simple rules yet the revolutions of play result in complex scenarios [4].

Some board games simulate a life activity such as a treasure hunt [5], and some merely train one to strategize or to improve one's powers of concentration and perhaps, collaboration skills.

The Game of Deployment has simple rules, yet play leads to complex scenarios. It closely represents the real-life need to deploy sensors in an environment, but strips away the complexities that obscure the fundamental goal of sensor deployment: Cover a given area while minimizing costs.

Board games often have a jargon all their own, along with general terms that describe their mechanics and attributes. The Game of Deployment has the following definitions and jargon:

- Tile – A physical unit on a game board delimited by a distinct border. A unique, atomic position on the board on which a game piece may be located while in play. The tile is the base element used to build boards and sensors; is a unit square.
- Game Board (or simply, board) – the surface on which one plays the game; composed of tiles. There are light tiles and dark tiles; dark tiles require coverage.
- Game Piece – A player's representative(s) on the game board. An individual game involves commanding multiple pieces and as in chess, each has a unique designation and value. Unlike chess, movement of pieces is a function of evaluating connection properties, and strategies and tactics are based on efficiently completing a network.
- Blocking Piece – A blocking piece is used by the board designer to disallow the deployment of sensors to given

tiles. It is a tool for reconfiguring a board and introducing variations to game play.

- Sensor –A sensor is a game piece composed of nine tiles, configured 3x3. Any tile can be marked as having sensing abilities. One tile is marked as the deployment tile and takes up one physical space on the board. Other games could have different preset layouts and number of tiles, hence the heterogeneity. There will be inherent orientation issues with such layouts.
- Base Station – A base station is a game piece with additional capabilities. A game must have at least one base station. Only sensors that have a connected communication path to a base station are counted in calculating the score.
- Connected path – Two tiles are considered connected if there exists a path from north to south or east to west. Diagonal connections do not form a path. Two connected tiles must have a common side without space between the tiles.
- Deployment – A sensor takes up one physical space on the board when placed. The tile that consumes a space on the board is indicated with an "Id" sensor type on its surface.
- Coverage – A sensor coverage tile is indicated with a "+" symbol, or, in the case of the ID tile, with an ID that is additionally a "+". In other words, in Table 1, Sensor 8 has two coverage tiles, expressed as "8" and "+". The differences are: The "+" may be overlapped and counts as 5 units; the "8" counts as 10 units with 5 for its ID and 5 for its "+" coverage tile. Each heterogeneous sensor will have a unique footprint.
- Communication - Two sensors can communicate if their tiles form a connected path.
- Score (or fitness) – The score in the game is a simple calculation to add the cost of all sensors used to satisfy all constraints. The lowest score wins.

III. SIMPLIFIED EXAMPLE

Rather than jumping into the details, let us start with a simple question.

Given a one-dimensional board, how can we place a finite number of sensors resulting in the lowest cost while covering all dark tiles?

The first step is for the board designer to define the board. Figure 1a shows a one-dimensional board and Figure 1b shows three dark tiles that require coverage. For simplicity, we have omitted the base station piece. One sensor is available that covers one tile (indicated with a "1") at a cost of one unit (Figure 1c).

It not difficult to see, that three sensors of type "1" are required to satisfy the coverage requirement at a cost of three units (Figure 1d). Any fewer sensors would not cover all required tiles, and any more would result in a higher cost.

Like most games, The Game of Deployment allows multiple player gameplay. Players can compete or they can cooperate on teams. In a simple game, a player creates a board, and his/her opponent submits a solution. In complex team play, a set of players creates the board. The other players or set must come up with a solution that tallies a low score. Such competition and team play ideally will engender better strategies, tactics and results.

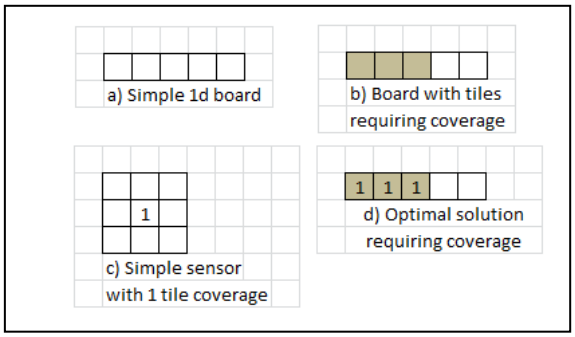

Figure 1. Simple example for game of deployment.

IV. GAME MECHANICS

The initial step in The Game of Deployment is creating a board from tiles. As in the game of checkers, the tiles are light or dark; unlike checkers, the pattern of light and dark tiles can be in any arrangement. The dark tiles require coverage.

The next step is to define the available sensors. Sensors are made up of tiles that may or may not provide coverage.

Play is initiated as the first sensor is placed or deployed to the board, and completed when the tiles requiring coverage are covered.

A. *Board*

The board is a set of unit square tiles arranged in a 1d, 2d, or higher dimensional arrangement. Our game doesn't restrict the board to a standard layout; it supports a surface that changes based on the problem. The only constraint in placing the tiles on the board is that there must be a path connecting every tile (no isolated tiles). We say that the arrangements of tiles on the board are connected if for any two tiles there exists a path from one tile to the other that stays completely within the board surface. Diagonally touching tiles are not considered to be connected.

Samples of boards are shown in Figure 2. In our examples, 1-dimensional and 2-dimensional boards are displayed, but again, n-dimensional boards could be considered, using the same rules and constraints. Dark tiles require coverage.

B. *Sensors*

A sensor is defined as an arrangement of connected tiles on a grid. Each tile is the same size as a board tile; thus, each tile on the board and the sensors are 1-unit squares. For a given problem, the available sensor types are defined showing their

sensing abilities and their corresponding costs. For this paper, Table I shows all available sensor types and their attributes or unique footprint.

A sensor is created from a sensor type and is identified by the sensor type identification number (ID). When playing the game, any number of sensors can be created from a sensor type. Unlike real-life board games, our game has an unlimited number of game pieces.

When the sensor is deployed, the ID is displayed on the board to allow the player to visualize the sensor type used. A sensor takes up one physical space on the board indicated by the tile with the ID. Its other eight tiles do not take up physical space and can overlap other sensor tiles.

Sensing abilities are indicated by tiles with a "+" symbol. When deployed, the tile provides coverage. The tile with the ID also provides coverage; for visual clarity, only the ID is displayed. The game designer may align the tiles in any order and size. Part of the challenge in designing the sensors is keeping the game play balanced such that no single sensor is dominant.

Sensors can communicate directly with any sensor on the board that has a connected path to one of its nine tiles. Communication can exist between sensors across the board by performing multiple hops from one sensor to another. The sensors on the board must form a sensor network. In real-life sensor deployments, information is transmitted to and from backend systems via a multi-hop network of low-powered devices.

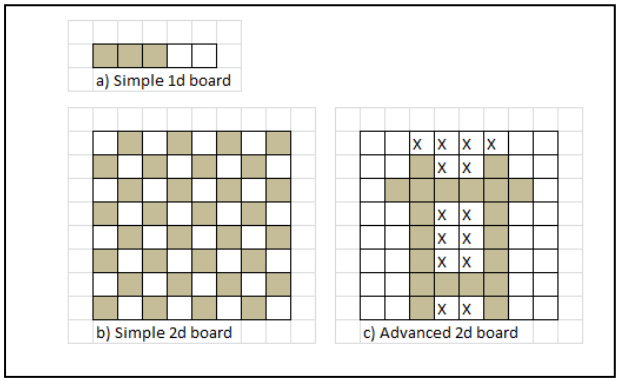

Figure 2. Examples of games boards.

For this paper, there are 16 available sensors types and one base station type available. The name, ID, coverage, and cost are shown in Table I. The cost of a sensor is simply 10 units for the ID and 5 units for each coverage tile. The base station incurs an additional cost of 10 units. The types listed were chosen somewhat arbitrarily.

When setting up a game, other types of sensors can be defined. This may produce unique game play and strategies based on the layout of the board and the pieces' attributes. One could define the game to closely represent a real-life sensor deployment problem. Currently, there are many hardware vendors that provide sensors with varying sensory and communication abilities at different price points.

TABLE I. SENSORS IN THE GAME

Sensor	Table Column Head	
	Coverage	*Cost*
1 2 3 4	1	10
5 6 7 8	2	15
9 10 11 12	3	20
13 14 15 16	4	25
8	5	40

C. *Base Station*

A base station performs a critical role in a sensor network. It is a sensor that is less resource constrained and has additional abilities. All messages and sensory information are transmitted through the base station. The base station communicates commands from the back end command station to all sensors in the network. It has the responsibility to collect all sensory readings and communicate them to the backend systems.

In Table I, the base station is indicated with a capital letter "B". The base station can send and receive information from sensors where a connected path exists. This path can be direct or through a series of hops. In sensor networks, a multi-hop network is established to communicate back to a base station [6]. Each sensor in the network carries the packets through the network to the base station which communicates the sensory information to the backend systems.

Typically, base stations have two communication devices. One, a low-powered radio, is used to communicate to a sensor within a small area. A second piece of communication hardware, normally a cell phone or hard-wired media, sends messages to and receives them from the backend systems.

The cost of a base station is the same as a sensor with an additional cost of 10 units. To simplify the representation on the board, the "#" and "B" are represented together with the symbol "B" having a cost of 20. The base station takes up one physical tile when deployed on the board ("B" tile). As shown in Table I, base stations can have sensing tiles. The cost of a base station is calculated by "B" plus the summation of all coverage tiles ("+") for a cost of 40 units in the example.

D. *Blocking Pieces*

In the game, the board designer can place blocking pieces onto the board. The player cannot deploy sensors on the tiles where a blocking piece is placed. A blocking tile is indicated with an "X" (Figure 2. c). This gives the board designers the ability to introduce challenging and unique configurations to their boards.

The game is played by one player creating the board and placing the blocking pieces and the other player solving the problem at the lowest cost. Other pieces may be introduced to add complexities, for example to block communication, destroy sensors that are deployed at that location, consume coverage, and other attributes that would stump the competitor.

E. Deployment, Coverage, and Communication

Deploying a sensor is similar to the game of checkers. A piece can be deployed and moved to any tile on the board (except a tile with a blocking piece "X"). The "#" (ID) or "B" on the sensor represents the base of the sensor and is placed on a board tile overlaying nine tiles (3x3). A sensor coverage tile (whether a "+", "B" or "#") that aligns on top of a dark tile (board tile requiring coverage) satisfies coverage for that board tile.

Two sensors can communicate if their tiles form a connected path. As stated, a connected path must exist from a base station to all sensors on the board. If a sensor provides coverage for a tile but is not connected into the network, its coverage does not count but its cost is included.

V. RULES, CONSTRAINTS, AND SCORE

The Game of Deployment, similar to other board games, comes with a set of instructions. The initial setup in the game requires placing tiles on a 2d surface. Next, a set of sensors are defined. The game is played by placing sensors and base stations onto the board.

With any game, the instructions always list the rules. The Game of Deployment has five rules:

1. A connected path must exist for all tiles on the board.
2. Only one sensor can be deployed to a given tile on the board.
3. At least one base station must be deployed on the game board.
4. Coverage must be satisfied for all tiles on the board marked as requiring coverage.
5. Communication from the base station, along connected paths, to all sensors on the board must be complete.

The game of deployment is classified as an "infinitely long game" [7]. Real-world board games are generally finished in a finite number of moves. We are not so constrained; the game lasts until the player has a winning solution. A winning score is simply a lower cost than any previous solution; lowering the cost of deployment is the essential point of the game.

With large problems, the time to compute the optimal cost may not be realistic, so the game continues.

A. Score (or fitness)

The score is a simple calculation of adding up the cost for all sensors used. The lowest score is the current winner. Only solutions that satisfy all constraints are given a final score.

In optimization problems, there must have a way to distinguish the fitness of all candidate solutions [8]. To work toward a more optimal solution, a fitness value is calculated for all solutions. This value is calculated by adding the cost of all sensors on the board plus a penalty cost of 10 units, typically set to the lowest cost sensor, for each tile not covered. A board with no base station receives a penalty cost of 40 units, typically set to the highest cost base station. Again, a lower score represents a better solution.

VI. EXAMPLE AND CHALLENGES

To introduce The Game of Deployment in more detail, a checkerboard example is given. Many of us have played checkers and are familiar with its game board, so we use it in this example. The goal is to provide coverage for all the dark tiles. Game pieces may be placed in any tile.

Although the checkerboard deployment problem is a relatively simple example for teaching people about the game, it simulates the way deployments are done in the real world.

A. Checker Board Example

This example is modeled after a checkerboard. It is played on a square board of eight rows and eight columns of square tiles (figure 2b). The 64 tiles are light tiles or dark tiles. The dark tiles require coverage. Game pieces may be placed on any tile.

Given an endless supply of each type of sensor and base station, the player places pieces onto the board. Pieces can be removed and added. The available sensors and their corresponding costs are shown in Table I.

Since this is a simple problem it would be pretty easy for a human player to figure out a good strategy. After playing the game for a few sessions, we devised a few. An initial observation is to use Sensors 9-12, since they provide the best cost per tile covered for this problem. Sensors 13-16 have a lower cost per tile but due to their coverage layout, one coverage tile won't be used.

Another strategy is to place the base station on a light tile where it can provide coverage for four surrounding tiles. We found that a quick scan of the board for groupings of sensors 1-4 is a good way to find candidates for replacement. Play a few sessions and see if you can recommend any additional strategies or observations.

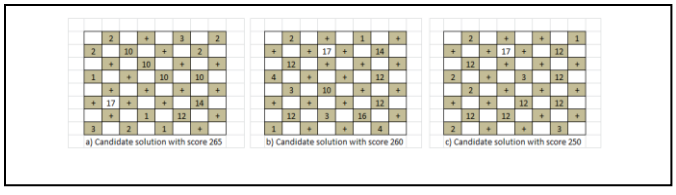

Figure 3. Checkers candidate solutions.

Figure 3 shows a few candidate solutions found by running the problem in our sensor deployment model [1]. We leveraged our past work and used the Genetic Algorithm (GA) to let the computer figure it out for us. We wrote the program following

the game mechanics, rules, constraints, and strategies outlined so far.

B. Checker Board Program

Let's walk through the process for developing a program. Our first step, as the board designer, is to define the board. An 8x8 integer array is used to define the board and to define which tiles require coverage (simple 0 or 1 approach). Blocking was not applied for this problem but a simple update to the array would allow us to indicate which tiles are blocked.

Our next step is to define the sensors. The specification on what sensors are available is defined in Table I. There is one type of base station available. The sensors are defined using a 17x 3x 3 array where the first dimension represents the sensor ID and the last two dimensions represent the nine tiles of the sensor. Another 17-integer array is used to define the costs.

Next, we get into the specifics of the algorithm chosen to solve the problem. We are using a GA [9, 10], so the chromosome is defined with 64 genes, one for each of the 64 tiles. A given gene has a value of 0, indicating that no sensor is deployed to that particular tile, or a value of 1-17 (17 is B), indicating the type of sensor used.

Creating the fitness function for calculating the score can be the most challenging part of creating the program. The starting point is to use the rules outlined in the score section (section V. *A*). We plan to update the function as we explore the problem.

We won't include the actual program here, however, it follows our previous model with some changes that help to evolve candidate solutions to better, more fit scores. The updates were found after running the program and studying the behaviors. A few observations and strategies were discovered as we played the game, and more were found by watching how the GA uses a few neat tricks. The steps are outlined below.

1. Generate the initial population
2. Select two individuals randomly from the population.
3. Have the two individuals compete in a tournament. Do not change the individual with the best score. Evolve the loser by crossover and mutate variations.
4. Return to step 2 until we have seen enough tournaments.
5. We have finished. Review the candidate solutions from the population. Repeat steps 1-4 until a satisfactory solution is found.

The algorithm does the magic of creating strategies. The parameters used for the GA: population size (30), number of tournaments (300,000), number of sessions (100). For the GA, we set the crossover rate to 90% and the mutation rate to 5%. Other numbers can be used which may produce good strategies. We let a few sessions run for 800,000 tournaments to see if a loser could evolve to a better score.

C. Checker Board Examination

Before we ran this algorithm, we must admit that we tried a brute-force approach. We constructed our own "smart" strategy, so we could see how well the GA would compare with our approach. We played the game as students, learning how our choices affected the score. The more we played, the more we learned about the game.

We knew our strategies were not perfect, but the tricks found by studying the GA on this problem were pretty clever. The algorithm applied strategies that we did not consider. Though a GA does not provide a magic-bullet solution, it may provide further insights into a problem.

After watching the program run and reviewing the results, there are a few things we found. In our initial sessions, we mutated the loser by randomly picking a number 0-17. The number 0 removes a sensor from the tile. The checker board has half of the files marked as requiring coverage. Why were we always deploying a sensor during mutation? Well, it is due to the probability of choosing a sensor is higher; 0 only has a 1 out of 18 chance of getting picked. We updated the mutation function so if mutation occurs in the tournament, to randomly pick a sensor 20% of the time. We started with 50% but found that a lower percentage caused sensors to be removed which lead to better scores. This is partially due to the population creation function following the same approach as our original evolution function (17/18 chance of placing a sensor).

Initially, we only defined one sensor with coverage of one (Sensor 1). We found that Sensor 1 did not appear as often as we would have expected. After running the program a few times, we realized there was another probability issue. This time, there was a balancing issue in how we defined the sensors. We increased the probability of using Sensor 1 by adding 3 additional copies (Sensors 2-4). By having the same number of sensors at each coverage level, each level could be selected uniformly in our selection and mutation function.

Another strategy we found was in respect to a tie score. Initially we randomly chose a winner if one was not found, but we changed this logic. If the individuals do not use the same number of sensors, define the loser as the individual who used the most sensors. Otherwise, make the 2nd individual the loser. We found that this approach allowed for solutions using fewer sensors to evolve.

With the GA, we found that if the penalty for not covering a tile was set too high, the algorithm would favor using more sensors and would not explore removing sensors. Setting it too low, for example below the lowest cost sensor, would leave tiles uncovered. As found in our studies, the GA parameters have a big impact on the performance of the program.

D. Challenges Proposed

In this paper, three boards and a set of available sensors were presented. The challenge proposed is: Play The Game of Deployment on the boards shown in figure 1. Review our solutions for the first two boards and compare them to your own. Did you find a solution with a better score?

Try increasing the crossover rate. What are your findings? Try increasing the mutation rate. Is the application guessing? Increase and decrease the size of the population and review the results. How about using a different algorithm?

Now, apply your strategy to the board shown in Figure 2c. The board could represent the silhouette of a building, layout of a room, or a river with bridges. Let us know your best score along with the sensors used, the algorithm, strategies, and your observations.

How about playing as the board designer and create a game based on a familiar area to challenge other players? Can you create a game that would engage others? What blocking pieces are you using and what new pieces can you add to increase the challenge?

As part of the game, we are interested in compiling strategies used along with additional challenges. Visit our website (www.TheGameofDeployment.com) for a list of board game examples and submitted solutions. Compare your best scores to what others have found. Submit your ideas, example games, challenges, and solutions so we can evolve The Game of Deployment. We are gathering such statistics as number of unique visitors, number of games added, number of scores submitted, mapping of score evolution and improvements, and other metrics that may help students and researchers. Visit our website for submission timelines and further instructions.

VII. CONCLUSION

In this paper, The Game of Deployment is presented. Its objective is to provide a game for studying the sensor deployment problem. A simple, game-based approach was introduced. This approach lowers the level of entry and abstracts the model so it can be applied to other studies. Our vision is to provide a tool that is fun and engaging for educational activities.

The game is general enough to provide a foundation and framework for future studies. A range of games, strategies, and algorithms can be applied.

We presented only a few simple examples. In current studies, we are exploring large boards, boards that cause multiple base stations to be deployed, boards based on familiar locations, and the use of other evolutionary algorithms [11, 12]. Future studies will survey submissions and present findings of such works.

REFERENCES

[1] B. Carter and R. Ragade. A probabilistic model for the deployment of sensors. Sensors Applications Symposium, 2009. SAS 2009. IEEE, pages 7-12, Feb. 2009.

[2] Ren Hongliang and M.Q.-H. Meng. Game-Theoretic Modeling of Joint Topology Control and Power Scheduling for Wireless Heterogeneous Sensor Networks. Automation Science and Engineering, IEEE Transactions, volume 6, pages 610 – 625, Oct. 2009.

[3] J.R. Marden, G. Arslan, J.S. Shamma. Cooperative Control and Potential Games. Systems, Man, and Cybernetics. IEEE Transactions, volume 39, issue 6, pages 1393-1407, Dec. 2009.

[4] E.J. Hughes. Checkers using a co-evolutionary on-line evolutionary algorithm. The 2005 IEEE Congress on Evolutionary Computation, volume 2, pages 1899-1905, Sept. 2005.

[5] Maiga A. Chang, A. Chang Hsieh. A Treasure Hunting Learning Model for Students Studying History and Culture in the Field with CellPhone. Advanced Learning Technologies, 2006. Sixth International Conference, pages 106-109, July 2006.

[6] Ian F. Akyildiz and Xudong Wang and Weilin Wang. Wireless mesh networks: a survey. Computer Networks, Elsevier, volume 47 (2005), pages 445-487, January 2005.

[7] M.B. Fayek. The Suitable Fitness Test-bed for Competing Candidates in GA. Computational Intelligence for Modeling, Control and Automation, 2005 and Interntional Conference on Intelligent Agents, Web Technologies and Internet Commerce, 2005. IEEE, pages 100-106, Nov. 2005.

[8] M. Hutter, S. Legg. Fitness uniform optimization. IEEE Transactions on Evolutionary Computation, volume 10, pages 568-589, Oct. 2006.

[9] I. Harvey, The microbial genetic algorithm, Evolutionary Computation (1996).

[10] J. H. Holland. Adaption in Natural and Artificial Systems, University of Michigan Press, 1975.

[11] J. Kennedy, R. Eberhart. Particle swarm optimization. IEEE International Conference on Neural Networks, volume 4, pages 1942-1948, Nov/Dec 1995.

[12] Yi Hong-Xia Xiao and Liu Liu Pu-Kun. Intelligent Algorithms for Solving Multiobjective Optimization Problems. Wireless Communications, Networking and Mobile Computing, 2008. WiCom 2008. IEEE, pages 1-5, Oct. 2008.