# High Level Mission Assignment Optimization

Oren Gal1
Technion. Israel Institute of Technology, Haifa
orengal@technion.ac.il

Corresponding author email: orengal@technion.ac.il

***Abstract*— Our research deals with Weapon-Target Assignment (WTA) problem which is a combinatorial optimization one that known to be NP-complete. The WTA is an optimization problem focus on setting the best assignment of weapons to targets, to minimizing the total expected value of the surviving targets. In this paper, we compared different methods of Z3 and Simulated Annealing (SA) to solve the WTA problem and suggest a novel algorithm based on Deep Q Networks (DQN) method. The main advantage of the DQN algorithm since we can learn in advance what is the optimal action for every space in short amount of time. Moreover, in real time the actual assignment of weapon can be done in a lot shorter amount of time then in the SA algorithm and the Z3 as can be seen in the results presented in this paper.**

***Keywords*— Optimization, Weapon-Target Assignment, Deep Q Networks.**

## 1. Introduction and Motivation

The weapon-target assignment (WTA) is an NP-complete combinatorial optimization problem. It aims to find the assignment of weapons to target which minimizes the function below [1]:

(i). $f(\pi) = \sum_{i=1}^{n} v_i \left(1 - \mathrm{P}_{\mathrm{ij}}\right)^{x_{i,j}}$

(ii). s.t $\sum_{i=1}^{n} x_{ij} = 1, j = 1, \ldots, m,$ and $x_{ij} \in \{0,1\}$

Where v_(i )is the value of the target i, p_ijis the probability of weapon j to destroy target i and x_ijis an indicator of the assignment-1 if the weapon j is assigned to the target i and 0 otherwise.

Z3 as explained by L. M. de Moura and N. Bjørner [2] and by Nikolaj Bjørner, Leonardo de Moura, Lev Nachmanson, and Christoph Wintersteiger [3] is a state-of-the-art method solving NP-complete constrained optimization problems such as the WTA problem. Z3 treats the problem as a SAT problem. Z3 returns the exact solution to the optimization problem but as it turned out. For WTA case, it only effective on small size problems. Thus, heuristic methods such as SA are the preferred solution for the problem. Simulated Annealing (SA) is an effective heuristic algorithm for solving the WTA problem. It can be used to solve many problems like the WTA. However, it takes a lot of time to converge.

Deep Learning is used today to solve a wide range of problems such as object detection in pictures and voice to text problems. DQN is a Deep Learning method based on Q learning used to solve a multi-agent's problem that has a big state-action space as described by Minih et al (2015) [4]. They also shown that DQN is a powerful method for solving games. They used DQN to solve 49 Atari 2600 games. They found it outperformed all previous methods explored and achieved a comparable level to the humans' experts.

In this paper we developed DQN architecture and simulation for the specific WTA problem, and compared the results to the exact method Z3 and the SA algorithm. Simulations showed a major advantage to our DQN algorithm dealing with WTA problem.

## 2. Related work

Sonuc, Sen, and Bayir [1] offered to use parallel calculations in the GPU to speed up the calculation of the SA algorithm. The parallel calculations used a technique called multi-start where they used multi-start points and run the SA algorithm in parallel threads in the GPU. The threads are divided into groups and at the end of the run, they take the solution with the smallest f value for each group. All the best solution from the blocks are transferred to the CPU. Then the global best f value is found by comparing the best solutions from all the blocks. Their solution offers a significant speedup of the SA.

In this paper, however, we wanted to find a method to use instead of the SA all together. Thus, we used the original SA for comparison in order to check the performance of our suggested solution which is the DQN algorithm.

The DQN algorithm was first introduced by Minih et al (2015) [4] for learning policy from a high dimension input. They tested it on Atari games and won against the existing methods on 49 games that were tested. They also discussed the instability in the algorithm cause by correlation in the sequence of the observation and the fact that small changes to Q can significantly change the policy. They offered two ideas for solving these problems. The first method is updating the target network only after multiple iterations and not on every iteration. This method solves the major changes to the policy because of small changes in Q. The second method is using experience replay which save the state action pairs from N recent iteration and randomly select K pairs from the memory. By adding the experience replay they handled the correlation problem. Both of those adjustments were used in this paper.

3. **Methods**

## SA Algorithm

The SA algorithm calculate f on a vector s were the locations are the targets and the objects in these locations are the weapons.

The algorithm works the following way [1]:
**Inputs:** Probabilities matrix P and values vector V
**Steps:**

```
1. Randomly choose the allocation of weapon (called S) to target and calculate its f_new value.
2. While T>t_final:
    a. Create a copy of S and randomly swap 2 weapons.
    b. Calculate the f for the new s
    c. If the swap improves f sufficiently:
        i. S=copy of S
        ii. f=f_new
    d. If f_new<f_best:
        i. f_best=f_new
        ii. s_best=s_new
    e. T=T*alpha
```

Sufficient improvement is when P(f, fnew, T) > random(0,1) where for replacing weapons q and r:

$$P(f, f_{new}, T) = \begin{cases} 1 \; if \; \Delta f < 0 \\ \exp\left(-\frac{\Delta f}{T}\right) \; otherwise \end{cases}$$

$$\Delta f = f - f_{new}$$

$$= v_q * (p_{qq} - p_{rq}) + v_r * (p_{rr} - p_{qr})$$

**Z3 Algorithm**
Z3 is a high-performance theorem prover developed at Microsoft Research. Z3 is used in many applications such as software/hardware verification and testing, constraint solving, analysis of hybrid systems, security, biology (in silico analysis), and geometrical problems [2]. One of its' uses is solving a constrained optimization problem like the WTA by treating the problem as a SAT problem which Z3 knows how to solve. The bigger the amount of targets the bigger the amount of constraints the problem have and the longer it takes to solve it. We tried to use the Z3 algorithm to solve our optimization problem. We tested it on instances with values in the range of 10 to size*10+10 in jumps of ten and randomly generated probabilities.

**Q-Learning**
We treat the WTA problem as a game were each state is an allocation, actions are areas for choosing weapons to swap from and the reward function is a variation of f.

Q-learning learns the action-value function Q(s, a): how good to take any action at a particular state.

In a round k, for every state s we calculate target where R(s, a, s') is the reward from the transition to the state s' using the action 'a'. Then we update the Q function [3].

$$target = R(s, a, s') + \gamma * \max_{a'} Q_k(s', a')$$

$$Q_{k+1}(s, a) = (1 - \alpha) Q_k(s, a) + \alpha * target(s')$$

In this problem, the state-action space is too big to learn using Q-learning. Therefore, we are going to use DQN.

**DQN:**

**States-** Our implementation uses a sequence as a state where the location in the sequence is the target and the number in that location is the weapon.

**Actions-** For a problem instance with size larger than we treat the sequence as if it splits to groups of ten locations and group of leftovers. Every action is either a swap between two weapons in the same group or two weapons in different groups.

**Reward Function-** We use the opposite function to the one described earlier because in DQN we maximize a reward function and not minimize it [1].

$$f(\pi) = \sum_{i=1}^{n} v_i \left(\mathrm{P_{ij}}\right)^{x_{i,j}}$$

s.t $\sum_{i=1}^{n} x_{ij} = 1, j = 1, \ldots, m$, and $x_{ij} \in \{0,1\}$

**Pseudocode for training DQN [3]:**

```
1. Initialize replay memory to capacity N.
2. Initialize the neural networks of Policy and Target (In our case the main layer is an attention neural network)
3. For episode= 1,…,M do:
    a. Create a new probability matrix and values vector. Initialize sequence of weapons target allocations s1
    b. For t=1,…,T do:
```

i. With probability, $\boldsymbol{\varepsilon}$ select a random action $a_t$ otherwise select the optimal action according to the Policy neural network. We decrease epsilon in every run from 0.9 to 0.1 using the equation:$0.1 + 0.8/exp(-steps/Edecay)$ where Edecay controls the rate of the decrease.

ii. Execute action $a_t$ and observe the new state and the reward from the transition.

iii. Store the new transition $(s_t, a_t, s_{t+1}, r_t)$ in the replay memory

iv. Sample minibatch of B transitions from the replay memory

v. Compute Q($s_t$, a) - the model computes Q($s_t$), then we select the columns of actions taken. These are the actions which would've been taken for each batch state according to Policy

vi. Compute V($s_{t+1}$) for all next states. Expected values of actions are computed based on the "older" Target; selecting their best reward.

$$expectedQ = r_{batch} + V(s_{t+1}) * \gamma$$

vii. Compute Huber-Loss function:

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 \ \ for\ |a| \le \delta \\ \delta\left(|a| - \frac{1}{2}\delta\right), otherwise \end{cases}$$

Where $a = expectedQ - Q(s_t, a)$

viii. Perform Gradient Descent on the Huber-Loss function

ix. Update Policy with the new weights

x. For every C step Target=Policy

Parameters values used in the simulation were N=1000; M=40; T=500; B=10; C=10, $\gamma$=0.99; Edecay=500.

## 4. Experiments Results

**Z3 Limits:**

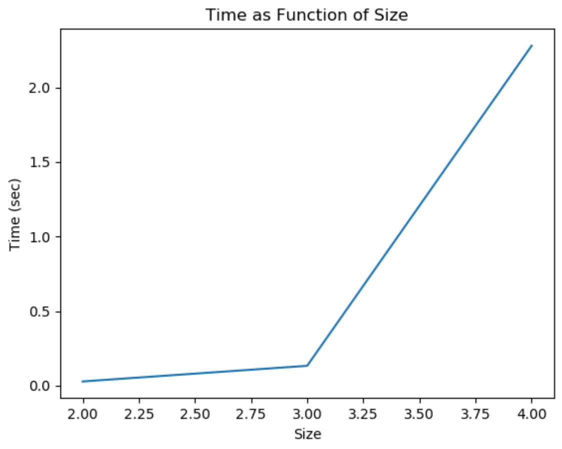

Figure 1: Run time of Z3 as function of the targets amount.

As we can see in Figure 1, for sizes between two and four, the algorithm works fine and takes a very little amount of time. For sizes higher than five it keeps running for hours without results. SA succeeds to converge to the optimal solution (with the same f value as the Z3) in a little amount of time for these sizes

**DQN vs SA on Random Examples:**

We ran the model on a new random problem (new probability matrix and values vector). We check performances on 10 different initial points randomly generated for every problem. We used only one random problem for each targets amount. In every iteration, we got an action according to the model and set s to the best state (the state the maximize the reward or minimize f) that the action could reach. We compared the result at the end of all the iterations to the result from the SA algorithm with different alpha values. The alpha values we used are 0.99,0.999,0.9999,0.99999. The bigger the alpha the more iterations the SA algorithms do and accordingly the better the solution the algorithm returns. I used T=1000 for all iterations.

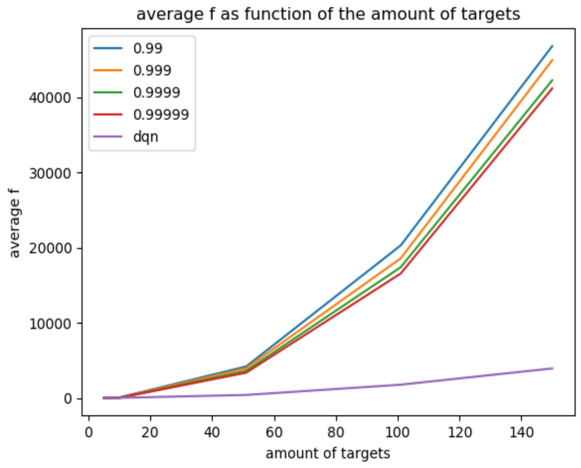

Figure 2: Average f as function of the targets' amount for SA with alpha=0.99,0.999,0.9999,0.99999 and the DQN algorithm. As shown in Figure 2, for big sizes- 51,101,150 DQN shows better results than SA for every alpha.

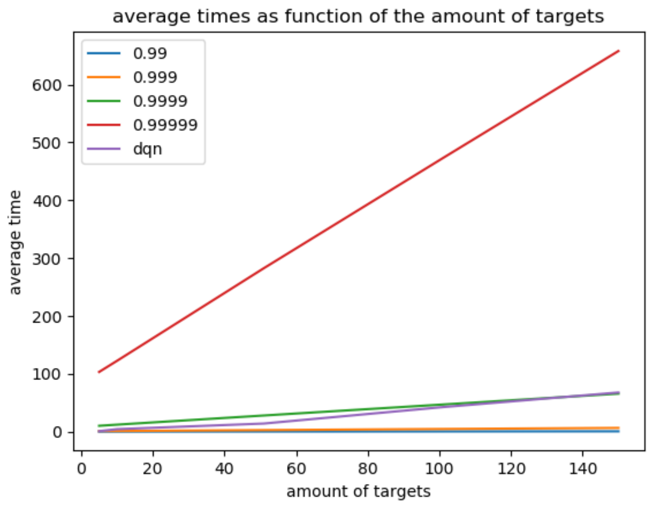

Figure 3: Average run-time of the algorithms as function of the number of targets for SA with alpha=0.99,0.999, 0.9999, 0.99999 and the DQN algorithm.

As shown in Figure 3, it takes less or equal time to SA with alpha=0.9999 and performs better than it. We only considered the test time for each initial point for the DQN because the training time doesn't matter.

## 5. Conclusions and Future Work

As we showed calculating the exact solution to the problem becomes almost impossible for a problem with size five even with a state-of-the-art algorithm such as Z3. Using Deep Q-Learning yields results that are better than the results of the SA algorithm for a shorter amount of time especially for large problems. However, for size F SA algorithm works better and reaches the optimum even for alpha=0.99 and as such is recommended to use for small size problems. We checked the algorithm for a limit group of possible sizes thus more checking is required to determine exactly for what size it is better to use DQN then to use the SA algorithm. For the sizes we checked the advantage is really shown for sizes bigger or equal to 50. The ratio of results to the times it takes to reach them improved using DQN, but it is still very far from the ratio required for real time use.
Recommendations for future work: (1) Check DQN on more problem instances and find its' limits. (2) Check more advance versions of DQN such as Double DQN which also explained in [3]. (3) Find a way to reduce the run time of DQN.

## References

[1] Sonuc, Sen, and Bayir (2017) A Parallel Simulated Annealing Algorithm for Weapon-Target Assignment Problem. In (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 8, No. 4.

[2] L. M. de Moura and N. Bjørner (2008). Z3: An Efficient SMT Solver. In TACAS.

[3] Nikolaj Bjørner, Leonardo de Moura, Lev Nachmanson, and Christoph Wintersteiger: Programming Z3. A tutorial published in Stanford University Site.

[4] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540):529–533.