CMIT Project B : Small Switching Components

Chenghao Dong Shiyi Ruan JinCheng Rao

September 22, 2023

1 Problem Sketch

Given $n \in \mathbb{N}$, find a list of pairwise non-collinear vectors $(\alpha_i, \beta_i)^T \in \mathbb{Z}^2$, $i \in \{1, ..., n\}$, such that the bivariate polynomial

$$F(x, y) = \prod_{i=1}^{n} f_{\alpha_i,\beta_i}(x, y) = \sum_{i=0}^{M} \sum_{j=0}^{N} \gamma_{i,j} x^i y^j$$

with

$$f_{\alpha_i,\beta_i}(x, y) = \begin{cases} x^{\alpha_i} y^{\beta_i} - 1 & \text{if } \alpha_i > 0,\ \beta_i > 0 \\ x^{\alpha_i} - y^{-\beta_i} & \text{if } \alpha_i > 0,\ \beta_i < 0 \\ x - 1 & \text{if } \alpha_i = 1,\ \beta_i = 0 \\ y - 1 & \text{if } \alpha_i = 0,\ \beta_i = 1 \end{cases}$$

has the number of terms $\Gamma = \sum_{i,j} |\gamma_{i,j}|$ to be as small as possible.

It is currently known that general choices of the (α_i, β_i) result in $\Gamma = 2^n$, which is the largest possible value. It is also known that a value smaller than $2n$ is not possible. Optimal values for $1 \leq n \leq 6$ are known, which are equal to $2n$, except for the case $n = 5$ where it is 12. Open are the cases $n \geq 7$, where the smallest known examples for $n = 7,\ 8,\ 9,\ 10$ have $\Gamma = 20$, 24, 36, and 40, respectively.

The problem is related to the application of discrete tomography of finding the smallest number of switching components within a fixed number of directions. To be more specific, for existing terms $\gamma_{i,j} x^i y^j$, leave a white point at the coordinate (i, j) if $\text{sgn}(\gamma_{i,j}) = 1$, or a black point at the same position if $\text{sgn}(\gamma_{i,j}) = -1$, and then the tomography may raise opposite conclusions in the directions set by $\{(\alpha_i, \beta_i)\}_{i=1}^{n}$ (Alpers, 2018).

In our study, we attempt to substitute algebraic analysis with a perspective from data science and machine learning. By utilizing four common algorithmic frameworks, ranging from heuristic algorithms to reinforcement learning, we tried to offer a new aspect for addressing this specific problem.

Alpers, A. (2018) *On the Tomography of Discrete Structures: Mathematics, Complexity, Algorithms, and its Applications in Materials Science and Plasma Physics.* Habilitation Thesis. University of Liverpool. Available at: https://livrepository.liverpool.ac.uk/3085601/.

2 Heuristic Methods

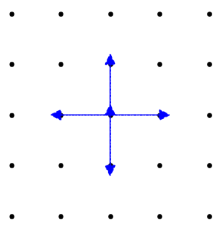
2.1 Simulated Annealing

Several heuristic algorithms were applied to solve the problem. The first one we tried was the simulated annealing. The basic framework of simulated annealing is: **initialize the solution - perturb the solution - accept the new solution which has a lower fitness value (or higher based on the direction of the optimization), or accept a less satisfactory one according to a certain probability - update the temperature and loop until the break condition is met**. To be more specific, we modeled our problem as following:

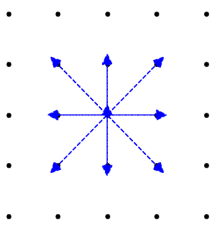
First, we set the fitness value of a given solution $s_t = \{(\alpha_i^t, \beta_i^t)\}_{i=1}^n$ as the relative "distance" between itself and the optimal one:

$$f(s_t) = \frac{\Gamma_t - 2n}{2^n - 2n} \quad \text{s.t.} \quad \Gamma_t = \Gamma(s_t) = \sum_{i,j} |\gamma_{i,j}^t| \quad \text{for} \quad F_t(x,y) = \prod_{i=1}^n f_{\alpha_i^t, \beta_i^t}(x,y) = \sum_{i,j} \gamma_{i,j}^t x^i y^j$$

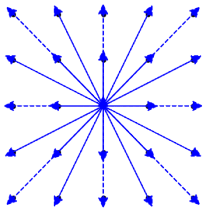
For a given solution s_t, we generate a new one s_t' by randomly updating several vectors within s_t based on the pre-selected **stencil** to achieve the perturbation. The **stencil** is an user-set parameter controlling the updating range of the selected vector, as well as the searching speed of the whole algorithm. Below are three types of the stencils we used in our trail.



(a) Cross stencil

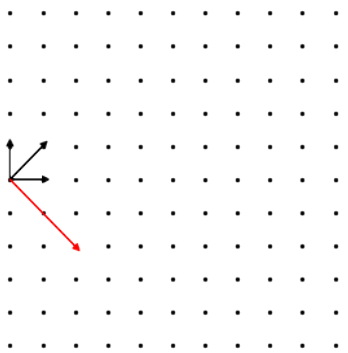


(b) 3×3 stencil

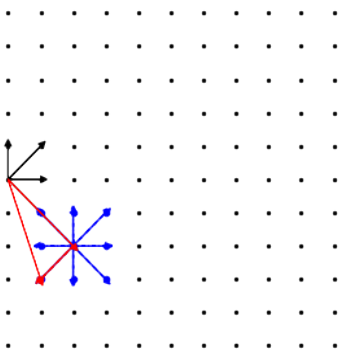


(c) 5×5 stencil

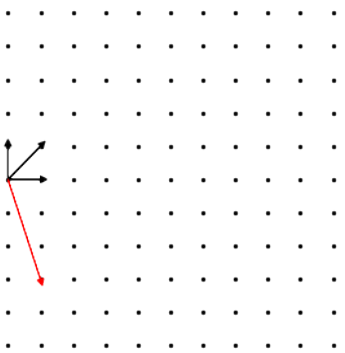
The procedure of a single perturbation based on a 3×3 stencil has been shown in the following sketch. Here, the vector $(2, -2)$ was randomly selected from the initial solution set $s_t = \{(0,1), (1,0), (1,1), (\mathbf{2}, \mathbf{-2})\}$ and replaced by $(1, -3) = (2, -2) + (-1, -1)$, which provided us with a new candidate $s_t' = \{(0,1), (1,0), (1,1), (\mathbf{1}, \mathbf{-3})\}$ for the final solution.



(d) Before perturbation



(e) Perturbation



(f) After perturbation

The benefit of using a stencil for generating new solutions is that it ensures an opportunity to obtain any possible vectors after sufficient iterations. The method is also particularly useful for finding different optimal solutions, as many of them may only have minor differences.

Moreover, in the context of this problem, we also need to impose limitations on the initial temperature of the annealing process. Given that this problem is a discrete optimization and that the fitness values are highly unstable near the optimal solution, setting the initial temperature too high may cause the annealing algorithm to frequently accept the worse result in the early stage, making it extremely difficult to converge early on.

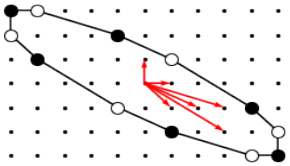
Suppose we could only accept our acceptance probability $A_T(s_t, s'_t)$ to be less than p for an increase in Γ greater than d, i.e., $\Delta\Gamma_t = \Gamma(s'_t) - \Gamma(s_t) \geq d$. Then, the initial temperature T_0 should be bounded from above, as:

$$A_T(s_t, s'_t) = exp\left(\frac{-(f(s'_t) - f(s_t))}{T}\right) \leq p \Longrightarrow T \leq -\frac{\Delta f_t}{lnp} \Longrightarrow T \leq \frac{\Delta\Gamma_t}{-lnp \cdot (2^n - 2n)}$$

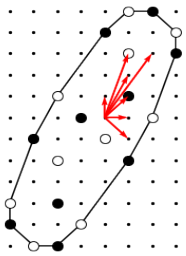
for all T and $\Delta\Gamma_t \geq d$. That is :

$$T_0 = T_{max} \leq \frac{d}{-lnp \cdot (2^n - 2n)}$$

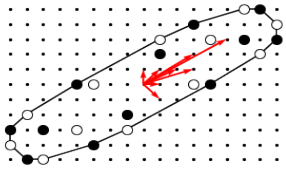
After having all of these parameters set properly, our algorithm could provide a series of reasonably small solutions within a relatively short period of time for n_s which are not too large (usually ranges from several to tens of minutes, depending on the scale of the problem). Although for n equals to 7, 8, 9, 10 the method failed to provide any better solutions, we did manage to find some potential candidates which lead to a same number of terms as the best result we could currently know (That is 20 terms for 7 directions, 24 for 8, 36 for 9 and 40 for 10, we even tried $n = 11$ and get a final result of 76 terms). To be more specific, we found 14 different solutions for $n = 7$, 10 for 8, 4 for 9 and only 1 for 10.



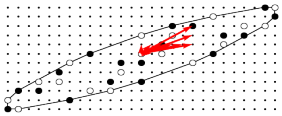
$n = 6$, 12 terms



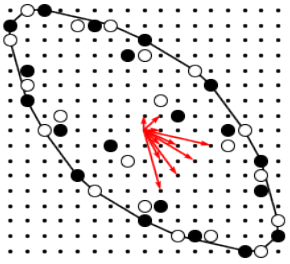
$n = 7$, 20 terms



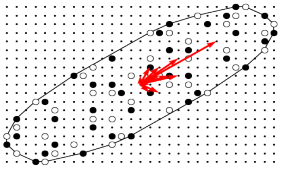
$n = 8$, 24 terms



$n = 9$, 36 terms



$n = 10$, 40 terms



$n = 11$, 76 terms

2.2 Discrete Particle Swan Optimization

Another heuristic method we tried was the discrete version of the particle swan optimization called the Integer and Categorical PSO (ICPSO) proposed by Strasser et al. (2016), where a particle p's position is represented by the probability distributions of the corresponding random variables.

To be more specific, a solution of our problem $s_p = \{(\alpha_{p,i}, \beta_{p,i})\}_{i=1}^{n}$ was uniquely defined by its position

$$\mathbf{X}_p = [\mathcal{D}_{p,1}\ ,\ \mathcal{D}_{p,2}\ ,\cdots,\ \mathcal{D}_{p,n}]\ \text{ s.t. } \mathcal{D}_{p,i} = \left(d_{p,i}^{k,l}\right)_{k,l}$$

where $d_{p,i}^{k,l}$ corresponds to the probability that the i_{th} vector takes on value (α_k, β_l) for particle (solution) p. Usually, the indexes k, l will be set as bounded to ensure a finite optimization. The velocity of the particle is a tensor with the exactly same dimension as its position $\mathbf{X}_p$, defined as

$$\mathbf{V}_p = [\phi_{p,1}\ ,\ \phi_{p,2}\ ,\cdots,\ \phi_{p,n}]\ \text{ s.t. } \phi_{p,i} = \left(\psi_{p,i}^{k,l}\right)_{k,l}$$

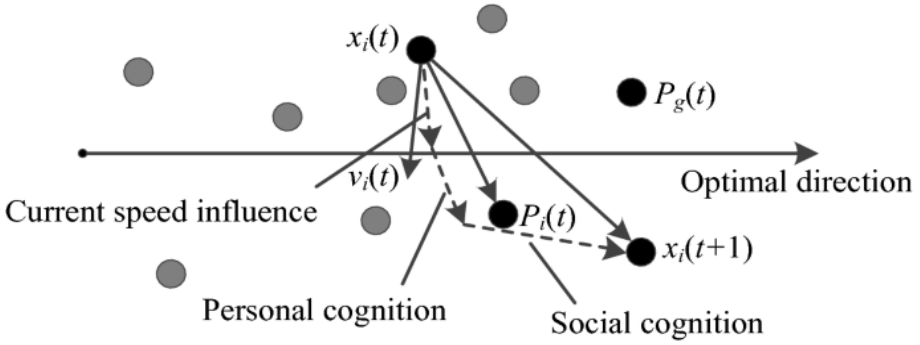
where $\psi_{p,i}^{k,l}$ are quantities used to adjust the particle's probability distributions.

After initializing the positions and velocities of all potential particles, they can then be updated using the following expressions

$$\mathbf{V}_p := \omega\mathbf{V}_p + U(0,\phi_1)\cdot(\mathbf{pBest} - \mathbf{X}_p) + U(0,\phi_2)\cdot(\mathbf{gBest} - \mathbf{X}_p)$$

$$\mathbf{X}_p := \mathbf{X}_p + \mathbf{V}_p$$

For now it seems that the speed of such algorithm is pretty slow and it always get stuck in a less satisfactory result compared with the ones we want. Perhaps the parameters in our trail for this model require further adjustment. However, considering the the strong interpretability of this model, as well as its extensive application in discrete optimizations, we still believe that it holds significant potential and theoretical feasibility for solving our problem.



(a) PSO velocity update (Sun et al., 2020)

Strasser, S., Goodman, R., Sheppard, J. and Butcher, S. (2016) 'A New Discrete Particle Swarm Optimization Algorithm', *GECCO'16: Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pp. 53-60. Available at: https://doi.org/10.1145/2908812.2908935.

Sun, M., Li, Q., Zhu, J. and Zhang, Y. (2020) 'Particle Swarm Optimization Algorithm Based on Graph Knowledge Transfer for Geometric Constraint Solving', *CENet2018: The 8th International Conference on Computer Engineering and Networks*. Available at: https://doi.org/10.1007/978-3-030-14680-1_50.

3 Reinforcement Learning

3.1 Environment Design

Reinforcement learning involves several important ingredients. Among them, the challenging part for this task is to determine our set of states and actions. The algorithm we need to design should guide us towards finding an optimal solution to some extent, which means minimizing the number of terms in the final polynomial $F_n(x, y)$. We thus designed a mechanism similar to the greedy algorithm. Assuming that we already have an optimal polynomial F_t for $t < n$, the idea is to build on this by selecting $(\alpha_{t+1}, \beta_{t+1})$, such that the next recursive polynomial

$$F'_{t+1} = F_t \cdot f_{\alpha_{t+1},\beta_{t+1}}(x, y)$$

has the fewest possible number of terms. We'll repeat this process until we obtain F'_n, aiming to make it as close as possible to the actual optimal polynomial F_n for $t = n$. The reason why we refer to this algorithm as "greedy" is that at each step, we will only consider the optimal result achievable given the current state. However, we must stressed that this approach might not necessarily lead us to the global optimal solution in the end. Within this framework, we might be able to start defining the states of our agent at each step during learning, as well as our available actions. To be more specific, let's say the polynomial at time t can also be expressed as

$$F_t(x, y) = \sum_{i=0}^{M_t} \sum_{j=0}^{N_t} \gamma_{i,j}^{(t)} x^i y^j$$

Now we multiply the polynomial with $(x^{\alpha_{t+1}} y^{\beta_{t+1}} - 1)$, which means we choose to set both our next α_{t+1} and β_{t+1} to be positive (resp. take $\beta_{t+1} < 0$), and we will have:

$$\begin{aligned}
F'_{t+1}(x, y) &= F_t(x, y) \cdot (x^{\alpha_{t+1}} y^{\beta_{t+1}} - 1) = \sum_{i=0}^{M_t} \sum_{j=0}^{N_t} \gamma_{i,j}^{(t)} x^i y^j \cdot (x^{\alpha_{t+1}} y^{\beta_{t+1}} - 1) \\
&= \sum_{i=0}^{M_t} \sum_{j=0}^{N_t} \gamma_{i,j}^{(t)} x^{i+\alpha_{t+1}} y^{j+\beta_{t+1}} - \sum_{i=0}^{M_t} \sum_{j=0}^{N_t} \gamma_{i,j}^{(t)} x^i y^j \\
&= \sum_{k=\alpha_{t+1}}^{M_t} \sum_{l=\beta_{t+1}}^{N_t} (\gamma_{k-\alpha_{t+1},l-\beta_{t+1}}^{(t)} - \gamma_{k,l}^{(t)}) \cdot x^k y^l + R_{t+1}(x, y) \\
&= \sum_{k=0}^{M_{t+1}} \sum_{l=0}^{N_{t+1}} \gamma_{k,l}^{(t+1)} x^k y^l
\end{aligned}$$

From the above derivation, we discovered that the number of terms of the new polynomial was bounded from below by that of the remainder polynomial $R_{t+1}(x, y)$. To restrict the growth of term numbers, the only way is to focus on terms that provide us with the possibility of cancellations. To ensure this kind of cancellation, it's necessary for us to set

$$\alpha_{t+1} \leq M_t \quad \text{and} \quad |\beta_{t+1}| \leq N_t$$

where $M_t = \sum_{i=1}^{t} \alpha_i$ and $N_t = \sum_{i=1}^{t} |\beta_i|$ as we could also write $F_t(x, y)$ in the form of products $\prod_{i=1}^{n} f_{\alpha_i,\beta_i}(x, y)$.

Basically, the above derivation states that in our environment, the possible candidates for α_s and β_s are bounded from above, and theoretically speaking, the achievable upper bounds keep growing as the selection proceeds. Nevertheless, experiments also showed that sometimes it is enough to set constant bounds M & N, which are relatively small, for both parameters to avoid a waste of time and computing resources. To conclude, the action set in our environment for a given polynomial at state $F_t(x, y)$ is

$$\mathbb{A}_t = \{(\alpha, \beta) : 0 \leq \alpha \leq min(M_t, M) \ , \ |\beta| \leq min(N_t, N)\} \subseteq \mathbb{Z}^2$$

To simplify the model, the reword function is usually set as the "normalized distance" between the number of terms in the final polynomial and that of the least desired one

$$R_t = \frac{2^t - \Gamma_t}{2^t - 2t} \quad \text{s.t.} \quad \Gamma_t = \sum_{i,j} |\gamma_{i,j}^{(t)}|$$

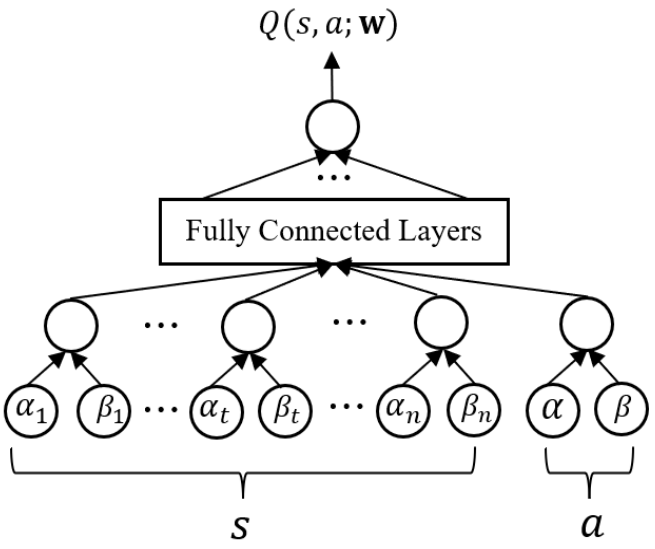
3.2 Deep Q Network

The deep Q network (DQN) is a well-known value-based algorithm for reinforcement learning, which eventually provide us with a marking function $Q(s_t, a_t; \mathbf{w}_t)$ to guide the performance of the agent. The pseudo-code of the algorithm is

Algorithm 1: Deep Q Network

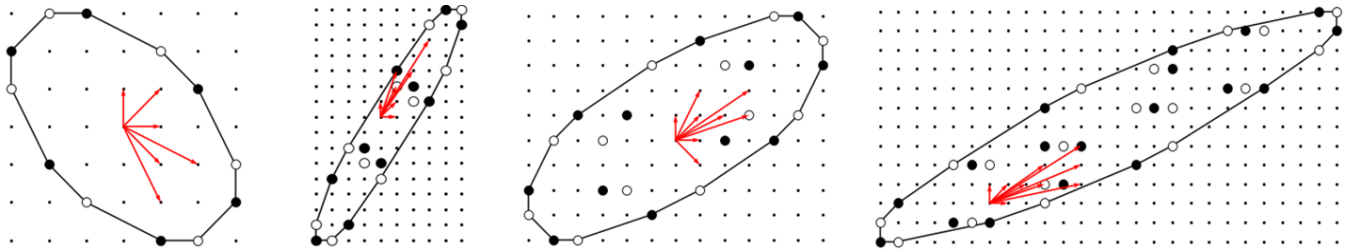
Initialize: Neural network weight parameters $\mathbf{w}$ for $Q(s, a; \mathbf{w})$;
while *episode* $<$ *max_episodes* **do**
  Initialize: Agent state s, total reward R ;
  while *process not done* **do**
    Choose action a with the highest $Q(s, a; \mathbf{w})$;
    Execute action a, observe the reward r and update to the next state s' ;
    Store the transition (s, a, s', r) in a batch D ;
    if *batch_size* $\geq$ *target_size* **then**
      for (s_i, a_i, s'_i, r_i) *in* D **do**
        Compute the target Q-value : $y_i = r_i + \gamma \cdot \max_{\tilde{a}} Q(s'_i, \tilde{a}; \mathbf{w})$;
        Compute the mean squared loss : $L_i = [y_i - Q(s_i, a_i; \mathbf{w})]^2$;
        Update weights $\mathbf{w}$ using the gradient descent optimizer on L_i with the exploration rate ϵ ;
      Clear batch D ;
    $s \longleftarrow s'$;
    $R \longleftarrow R + r$;
  episode $\longleftarrow$ episode + 1 ;

In our case, the structure of the Q network designed for a problem of a final size n could be given as follows; the action a is selected in a way s.t. $Q(s, a; \mathbf{w})$ is maximised under the condition of non-colinearity.



(a) DQN structure

Here are some of the results we get from the model: for $n \leq 9$, $M = N = 5$, the algorithm managed to find some of the solutions that are in accordance with the anticipation; for $n = 10$, $M = N = 5$, the smallest number of terms generated was equal to 76, which is slightly larger than the target number (40) we could expect; for $n \geq 10$, the algorithm requires a large amount of time to finish, which is mainly effected by the expansion of large degree polynomials in the evaluation of rewards. We only tried to train a network for $n = 11$, $M = N = 5$, and got a smallest number of terms equal to 112.



$n = 6$
12 terms

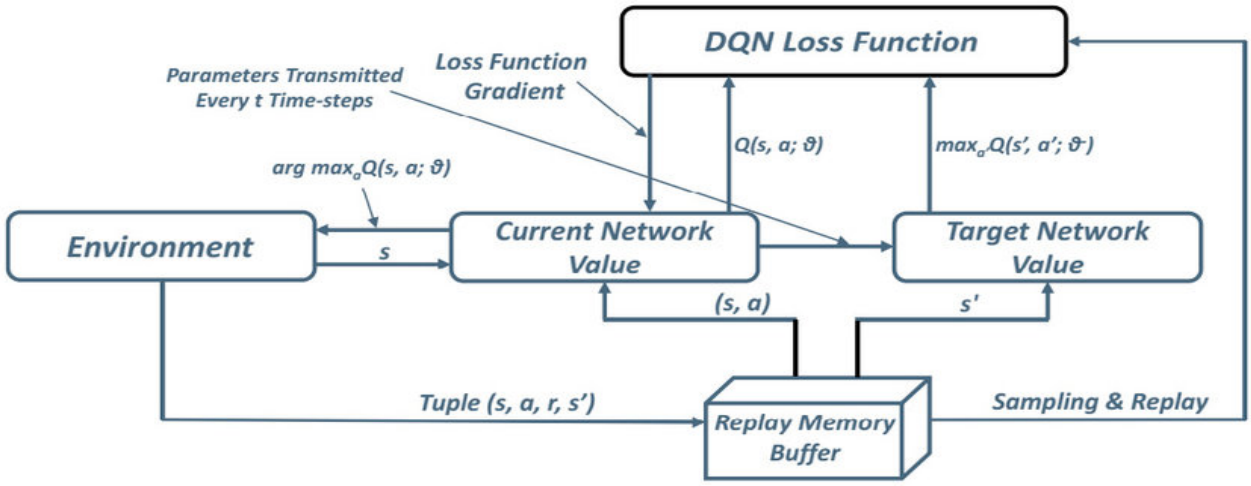
$n = 7$
20 terms

$n = 8$
24 terms

$n = 9$
36 terms

Although in our modelling, the action space is fixed by the upper bounds M and N for simplicity, DQN may be more suitable for similar tasks involving changing action spaces due to its nature of being **value-based** compared to the following CEM. Plus, as "previous experience" is usually utilized in the training of DQN, the performance of the model may turn out to be more **stable** and usually **requires less manual intervention**.

Yet, as we have already mentioned from above, **the operation time of the algorithm explodes** when the size of the problem gets too large. This, however, is mainly **induced by the polynomial expansions** of the reward assignment, and therefore we could hardly put the blame on the algorithm itself. Perhaps the only efficient way for solving this problem is to find a better reward mechanism or to search for more effective polynomial expansion techniques.



(b) DQN flowchart (Muteba et al., 2020)

Muteba, F.K., Djouani, K.D. and Olwal, T. (2020) 'Deep Reinforcement Learning Based Resource Allocation For Narrowband Cognitive Radio-IoT Systems', *Procedia Computer Science*, 175, pp. 315-324. doi: 10.1016/j.procs.2020.07.046.

3.3 Cross-Entropy Method

Apart from learning a deep Q network, we also applied a cross-entropy method to solve this task. Within the method, the neural network does not explicitly learn a value function for the given states but learns a probability distribution called a "policy" to assign a higher probability to the moves that the agent thinks are better.

The crucial emphasis in the cross-entropy method lies on the concept of an "elite batch", which is a subset of the best-performing solutions selected based on their ability to achieve high rewards in each iteration aiming to make the learning more efficiently.

To simplify the model, set the upper bounds for possible α_s and β_s as $M = N = 5$ and obtain a finite action set that is independent from the time index t.

$$\mathbb{A} = \{(\alpha, \beta) : 0 \leq \alpha \leq 5 \ , \ |\beta| \leq 5\} \subseteq \mathbb{Z}^2 \ \text{ s.t. } \ |\mathbb{A}| = (5+1) \cdot (2 \times 5 + 1) = 66$$

All states and actions can now be encoded using a flattened array of length 66 and an integer respectively in the following way :

$$\begin{aligned} \text{reward} \ &: \ R_t = (2^t - \Gamma_t)/(2^t - 2t) \\ \text{action} \ &: \ a_t = i \iff \text{take } (\alpha_i, \beta_i) \in \mathbb{A} \\ \text{state} \ &: \ s_t = [\ \mathbb{I}_{SOL_t}(\alpha_i, \beta_i)\ ]_{i=66} \\ \text{policy} \ &: \ \rho_t = [\ \mathbb{P}\ (\ i \mid s_t\ )\ ]_{i=66} \end{aligned}$$

The pseudo-code for the model is given as :

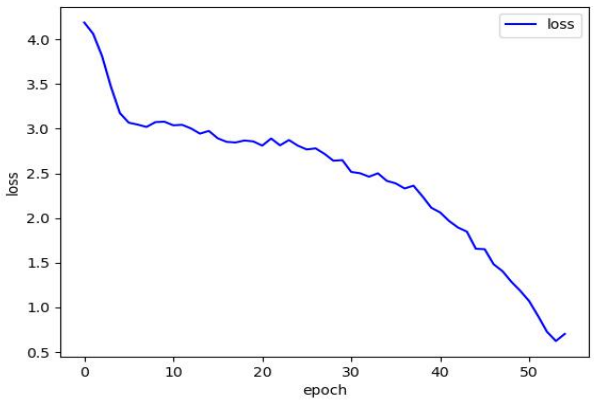
Algorithm 2: Cross-Entropy Method

```
Initialize: k agents and neural network weight parameters ;
while iter_no < iter_num do
    for each agent do
        Step one pace according to the current policy ;
        Update the next state ;
        if process ends then
            Record the sample track in a batch ;
            Record the final step reward ;
            Initialize the agent ;
    if batch_size ≥ target_size then
        Compute batch reward mean and batch reward bound ;
        Choose elite batch according to the reward bound ;
        Train the neural network using the elite batch ;
        if stopping criteria met then
            break
        Clear batch ;
        iter_no ⟵ iter_no + 1 ;
```

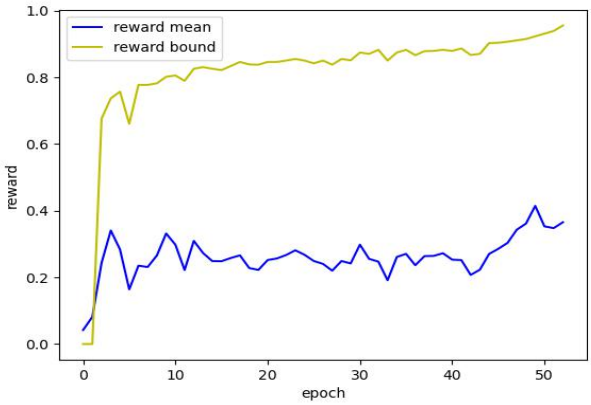
As the size of the problem n gets larger, experiments indicated that the algorithm may have the opportunity to converge in an opposite direction giving rise to a least expected result. This could be

triggered by an accumulation of poor examples at the early stage of the iteration for some episodes, which might bring about a relatively low reward bound, causing the current batch to train the network in a less expected direction.

Therefore, at the end of each episode, the actual reward bound will be set by taking **the maximum of the nominal bound in the current batch as well as the one in the previous batch** to locally adjust the direction of the optimization.



(a) CEM loss



(b) CEM reward

In comparison with the Deep Q Network, the CEM is **policy-based**, **suitable for sparse reward setting**, usually assures **better convergence** and is also **more robust for the choice of hyper-parameters** (Wagner, 2021), which makes it a quite suitable model for the problem we worked on.

Finally, for the results, when $n \leq 9$, $M = N = 5$, the algorithm assures a stable and quick searching for a list of different solutions that is in consistent with the optimal results we know; for $n = 10$, $M = N = 5$, the result is less stable, which might converge to a polynomial that has 40 to 48 terms; setting $n = 11$, $M = N = 7$, a polynomial with 62 terms was found.

3.4 Further Improvements

It also turns out that the above algorithms both suffer from several apparent drawbacks. For instance, only a small number of optimal solutions can be found during each training session, and, both methods are faced with the dilemma of converging to a sub-optimal solution when n gets larger.

One possible strategy for addressing the latter challenge is to add a "retraction mechanism" which allows the agent to discard some of the vectors (α, β) that has already been selected to the solution set SOL_t at time t :

$$\mathbb{A}_t = \{ \text{ add } (\alpha, \beta) : 0 \leq \alpha \leq min(M_t, M) \ , \ |\beta| \leq min(N_t, N)\} \cup \{ \text{ discard } (\alpha, \beta) : (\alpha, \beta) \in SOL_t\}$$

Another suggestion is to use a discounted reward DR_t instead of the original one to encourage the agent to pay more attention to the steps that are closer ($\varepsilon < 1$), or further ($\varepsilon > 1$), to the target step n :

$$DR_t = R_t \times \varepsilon^{n-|SOL_t|}$$

Wagner, A.Z. (2021) *Constructions in combinatorics via neural networks*, arXiv:2104.14516.