Game Playing with Monte-Carlo Tree Search

Proof of Concept: Bandit Problem

Alexandra Danciu

What are Bandit Problems?

Bandit Problems are statistical problems that involve trying to gain as many points over a limited time as possible. There will be multiple places to visit that include an unknown number of points each. The goal is to find a balance between exploring and exploiting.

Exploring means going out to new places and figuring out how many points they can give you, most of the time the points will vary at different times so the average number of points would be best.

Exploiting is going to the place where you know the average number of points given is high and you are likely to gather the most points here.

One example would be going to a restaurant to eat. Say you had 300 days, and you wanted to go out to eat each night at 3 different restaurants. Each restaurant will give you a certain amount of happiness to go eat there each night. One restaurant could give you an average of 10 happiness with a deviation of 2.5 each way, a second restaurant could give you 8 happiness on average with a 2 deviation each way, and the third restaurant offers an average of 5 happiness with a 12.5 deviation.

You want to have a balance between going to a restaurant and trying its menu and seeing how much happiness it gives you and going to your favourite restaurant that you are more confident will make you happy.

Naïve Strategies

There are a number of different algorithms you can use to solve the bandit problem, however, the efficiency of some of them compared to other, better, algorithms are lacking.

Explore Only

One of these less efficient algorithms that still gets the job done is Explore Only. With this algorithm the entire 300 days is spent exploring and picking random restaurants to dine at. By the end, you have spent a considerable amount of time at the wrong restaurants and have lost some points on the way. Each restaurant will end up having an equal 100 days spent at them.

Exploit Only

Another inefficient, naïve algorithm would be Exploit Only, which is similar to Explore Only where it does only one-half of the explore/exploit in the problem.

With this algorithm you will use the first three days to visit each restaurant once and which restaurant gave you the highest happiness level you will visit for the rest of the 297 days.

While this may seem like an ok algorithm, the happiness points given by each restaurant have a deviation. On day one you may visit the best restaurant, number 1, and get a low deviation that gives you 7 instead of the average 10, and then you go to restaurant 2 on day 2 and get an 8. You would then continue to go to Restaurant 2 for the remainder of the days and would only be visiting the second-best restaurant, lowering your chances for higher total happiness points.

E-Greedy

This is one of the better algorithms that could be used to generate decent results.

With this algorithm you start by denoting an E value, say E = 10%.

What this means is that each day, there is now a 10% chance that you will visit a random restaurant and a 90% chance that you will visit the restaurant that has historically given you the most happiness points. These exploit days, the 10% chance that you visit a random restaurant, gives you more evidence of which restaurant is the best. It will either ensure that you switch restaurants to the best one in the case that the first time you went to one a deviation happened, or it will make you more confident that the restaurant you are currently going to is the best one.

The issue with this algorithm is that its performance depends on the chosen epsilon value and the value of the possible deviations.

Upper Confidence Bound (UCB)

This is one of the best algorithms used to solve bandit problems. It is great at balancing exploration of new arms or arms with low confidence and exploiting arms with known high values or arms with high confidence values as they have been visited frequently. This algorithm works similarly to E-Greedy. The strategy is that:

At each time T, pick point R, such that

A math equation with numbers and symbols

Description automatically generatedmean of R + √ 2 \* current time T / number of times visited R

With this algorithm each time an arm needs to be selected, each arm’s UCB value is calculated using the equation above, and the highest value is selected. This works similarly to E-Greedy in the way that, the arm with the highest value is visited more frequently as it gives back the most points but the calculation of the UCB also means that other arms are visited at regular points and in a fair way; it is not left to random chance. The success of E-Greedy depends highly on the value of E, which UCB does not struggle with.

A screenshot of a computer screen

Description automatically generatedCode

The coding for this proof of concept started with the UML design above. We have the BanditArm class that creates instances of the arms equipped with points and numbers of visits and some methods to get and set variables. These arms represent the restaurants in the example used above. Then, we have the four BanditSolver…() classes that represent the four different algorithms used to solve the bandit problem. In this basic version of the UML all of the classes had the same basic variables and methods that later changed as I coded. Finally, we have the main class that initialises all of the classes and sets up the arms and prints out the results of running each solver class.

A screenshot of a computer

Description automatically generated

Above is the final version of the UML that copies the exact layout of the final code for the proof of concept. The main structure change was in the solver classes, the original UML had two classes for selectArm() and updateArm(), however I decided to break this down further into three classes, selectArm(), round() and runRound().

I chose to do this because I thought I was a better general fit for all the different algorithms. selectArm() contains the code for the actual algorithm within each arm. round() simple runs selectArm() and then increments the visits on the arm and adds the points to the total points. And runRound() checks whether the current round is equal to the total number of rounds before running another round.

Aside from the main structure changes more changes were made on a class basis, depending on what the algorithms needed. Such as the E variable for the E-Greedy class, firstRound() for the exploit only class and random was implemented for the explore only and E-Greedy class.

Main was changed appropriately to accommodate the initialisation of all the classes and instances of the arms. Main was mostly used to change the values in the arms and other variables such as the total rounds and E for the tests done for the comparison of efficiency of the algorithms under different states.

Results Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Round | Parameters | 1. Explore  Only | 2. Exploit Only | 3. E-Greedy | 4. UBC | Total Points |
| 1 | Total Rounds: 100  Arms: 4  Arm1: 10, 4  Arm2: 5, 5  Arm3: 7, 3  Arm4: 8, 2  Base,  Normal parameters | Arm1: 24  Arm2: 23  Arm3: 35  Arm4: 18 | Arm1: 1  Arm2: 97  Arm3: 1  Arm4: 1 | Arm1: 83  Arm2: 8  Arm3: 6  Arm4: 3 | Arm1: 96  Arm2: 2  Arm3: 1  Arm4: 1 | 1: 645  2: 444  3: 857  4: 984 |
| Arm1: 26  Arm2: 24  Arm3: 23  Arm4: 27 | Arm1: 97  Arm2: 1  Arm3: 1  Arm4: 1 | Arm1: 84  Arm2: 5  Arm3: 4  Arm4: 7 | Arm1: 90  Arm2: 1  Arm3: 7  Arm4: 2 | 1: 700  2: 883  3: 932  4: 983 |
| Arm1: 21  Arm2: 31  Arm3: 25  Arm4: 23 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 88  Arm2: 3  Arm3: 6  Arm4: 3 | Arm1: 97  Arm2: 1  Arm3: 1  Arm4: 1 | 1: 649  2: 759  3: 964  4: 931 |
| Arm1: 23  Arm2: 25  Arm3: 22  Arm4: 30 | Arm1: 1  Arm2: 97  Arm3: 1  Arm4: 1 | Arm1: 68  Arm2: 6  Arm3: 2  Arm4: 24 | Arm1: 95  Arm2: 1  Arm3: 2  Arm4: 2 | 1: 660  2: 411  3: 842  4: 907 |
| Arm1: 23  Arm2: 21  Arm3: 25  Arm4: 31 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 92  Arm2: 3  Arm3: 2  Arm4: 3 | Arm1: 97  Arm2: 1  Arm3: 1  Arm4: 1 | 1: 662  2: 724  3: 918  4: 949 |
|  |  |  |  |  |  |  |
| 2 | Total Rounds: 500  Arms: 4  Arm1:10, 4  Arm2: 5, 5  Arm3: 7, 3  Arm4: 8, 2  A lot more rounds run. | Arm1: 119  Arm2: 123  Arm3: 124  Arm4: 134 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 497 | Arm1: 459  Arm2: 12  Arm3: 11  Arm4: 18 | Arm1: 495  Arm2: 1  Arm3: 1  Arm4: 3 | 1: 3522  2: 3777  3: 4672  4: 4640 |
| Arm1: 122  Arm2: 146  Arm3: 99  Arm4: 133 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 497 | Arm1: 110  Arm2: 47  Arm3: 315  Arm4: 28 | Arm1: 492  Arm2: 2  Arm3: 1  Arm4: 5 | 1: 3448  2: 3827  3: 3569  4: 4693 |
| Arm1: 105  Arm2: 134  Arm3: 147  Arm4: 114 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 497 | Arm1: 182  Arm2: 134  Arm3: 147  Arm4: 37 | Arm1: 478  Arm2: 2  Arm3: 2  Arm4: 18 | 1: 3488  2: 3712  3: 3622  4: 4698 |
| Arm1: 105  Arm2: 134  Arm3: 147  Arm4: 114 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 497 | Arm1: 182  Arm2: 134  Arm3: 147  Arm4: 37 | Arm1: 478  Arm2: 2  Arm3: 2  Arm4: 18 | 1: 3488  2: 3712  3: 3622  4: 4698 |
| Arm1: 132  Arm2: 105  Arm3: 128  Arm4: 135 | Arm1: 497  Arm2: 1  Arm3: 1  Arm4: 1 | Arm1: 82  Arm2: 12  Arm3: 390  Arm4: 16 | Arm1: 492  Arm2: 2  Arm3: 2  Arm4: 4 | 1: 3564  2: 4581  3: 3567  4: 4777 |
|  |  |  |  |  |  |  |
| 3 | Total Rounds: 20  Arms: 4  Arm1:10, 4  Arm2: 5, 5  Arm3: 7, 3  Arm4: 8, 2  A lot fewer rounds run. | Arm1: 9  Arm2: 3  Arm3: 4  Arm4: 4 | Arm1: 17  Arm2: 1  Arm3: 1  Arm4: 1 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 17 | Arm1: 13  Arm2: 1  Arm3: 4  Arm4: 2 | 1: 126  2: 154  3: 149  4: 174 |
| Arm1: 7  Arm2: 3  Arm3: 1  Arm4: 9 | Arm1: 17  Arm2: 1  Arm3: 1  Arm4: 1 | Arm1: 3  Arm2: 2  Arm3: 14  Arm4: 1 | Arm1: 5  Arm2: 2  Arm3: 12  Arm4: 1 | 1: 179  2: 172  3: 136  4: 152 |
| Arm1: 6  Arm2: 5  Arm3: 5  Arm4: 4 | Arm1: 17  Arm2: 1  Arm3: 1  Arm4: 1 | Arm1: 1  Arm2: 1  Arm3: 17  Arm4: 1 | Arm1: 14  Arm2: 1  Arm3: 1  Arm4: 4 | 1: 120  2: 146  3: 144  4: 185 |
| Arm1: 3  Arm2: 5  Arm3: 6  Arm4: 6 | Arm1: 1  Arm2: 1  Arm3: 17  Arm4: 1 | Arm1: 8  Arm2: 2  Arm3: 2  Arm4: 8 | Arm1: 17  Arm2: 1  Arm3: 1  Arm4: 1 | 1: 125  2: 139  3: 150  4: 175 |
| Arm1: 4  Arm2: 6  Arm3: 8  Arm4: 2 | Arm1: 1  Arm2: 1  Arm3: 17  Arm4: 1 | Arm1: 16  Arm2: 2  Arm3: 1  Arm4: 1 | Arm1: 1  Arm2: 1  Arm3: 2  Arm4: 16 | 1: 119  2: 109  3: 158  4: 139 |
|  |  |  |  |  |  |  |
| 4 | Total Rounds: 100  Arms: 8  Arm1:10, 4  Arm2: 5, 5  Arm3: 7, 3  Arm4: 8, 2  Arm5: 6, 4  Arm6: 3, 3  Arm7: 4, 2  Arm8: 1, 5  Double the number of arms. | Arm1: 16  Arm2: 15  Arm3: 12  Arm4: 15  Arm5: 13  Arm6: 9  Arm7: 4  Arm8: 16 | Arm1: 93  Arm2: 1  Arm3: 1  Arm4: 1  Arm5: 1  Arm6: 1  Arm7: 1  Arm8: 1 | Arm1: 82  Arm2: 3  Arm3: 3  Arm4: 3  Arm5: 1  Arm6: 1  Arm7: 2  Arm8: 5 | Arm1: 90  Arm2: 2  Arm3: 1  Arm4: 3  Arm5: 1  Arm6: 1  Arm7: 1  Arm8: 1 | 1: 570  2: 897  3: 887  4: 912 |
| Arm1: 16  Arm2: 12  Arm3: 8  Arm4: 14  Arm5: 16  Arm6: 11  Arm7: 13  Arm8: 10 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 1  Arm5: 1  Arm6: 1  Arm7: 93  Arm8: 1 | Arm1: 58  Arm2: 1  Arm3: 29  Arm4: 2  Arm5: 2  Arm6: 5  Arm7: 2  Arm8: 1 | Arm1: 91  Arm2: 1  Arm3: 1  Arm4: 3  Arm5: 1  Arm6: 1  Arm7: 1  Arm8: 1 | 1: 552  2: 341  3: 818  4: 846 |
| Arm1: 10  Arm2: 9  Arm3: 9  Arm4: 16  Arm5: 16  Arm6: 10  Arm7: 16  Arm8: 14 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 1  Arm5: 1  Arm6: 1  Arm7: 93  Arm8: 1 | Arm1: 81  Arm2: 2  Arm3: 3  Arm4: 3  Arm5: 3  Arm6: 2  Arm7: 3  Arm8: 3 | Arm1: 74  Arm2: 2  Arm3: 2  Arm4: 16  Arm5: 3  Arm6: 1  Arm7: 1  Arm8: 1 | 1: 558  2: 350  3: 873  4: 877 |
| Arm1: 13  Arm2: 15  Arm3: 11  Arm4: 15  Arm5: 13  Arm6: 12  Arm7: 11  Arm8: 10 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 1  Arm5: 1  Arm6: 1  Arm7: 1  Arm8: 93 | Arm1: 13  Arm2: 1  Arm3: 1  Arm4: 2  Arm5: 4  Arm6: 76  Arm7: 2  Arm8: 1 | Arm1: 77  Arm2: 6  Arm3: 4  Arm4: 6  Arm5: 3  Arm6: 1  Arm7: 1  Arm8: 2 | 1: 488  2: 185  3: 419  4: 863 |
| Arm1: 10  Arm2: 8  Arm3: 17  Arm4: 12  Arm5: 21  Arm6: 12  Arm7: 12  Arm8: 8 | Arm1: 93  Arm2: 1  Arm3: 1  Arm4: 1  Arm5: 1  Arm6: 1  Arm7: 1  Arm8: 1 | Arm1: 2  Arm2: 1  Arm3: 2  Arm4: 88  Arm5: 3  Arm6: 1  Arm7: 1  Arm8: 2 | Arm1: 88  Arm2: 1  Arm3: 6  Arm4: 1  Arm5: 1  Arm6: 1  Arm7: 1  Arm8: 1 | 1: 512  2: 968  3: 708  4: 894 |
| 5 | Total Rounds: 100  Arms: 4  Arm1: 10, 4  Arm2: 20, 5  Arm3: 30, 3  Arm4: 40, 2  Greatly increase the distance between points given. | Arm1: 17  Arm2: 30  Arm3: 31  Arm4: 22 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 5  Arm2: 5  Arm3: 5  Arm4: 85 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | 1: 2546  2: 3917  3: 3654  4: 3895 |
| Arm1: 28  Arm2: 21  Arm3: 26  Arm4: 25 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 4  Arm2: 2  Arm3: 6  Arm4: 88 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | 1: 2419  2: 3914  3: 3744  4: 3901 |
| Arm1: 26  Arm2: 23  Arm3: 25  Arm4: 26 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 1  Arm2: 2  Arm3: 5  Arm4: 92 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | 1: 2499  2: 3915  3: 3831  4: 3869 |
| Arm1: 25  Arm2: 23  Arm3: 29  Arm4: 23 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 2  Arm2: 2  Arm3: 3  Arm4: 93 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | 1: 2502  2: 3907  3: 3812  4: 3870 |
| Arm1: 29  Arm2: 24  Arm3: 21  Arm4: 26 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 3  Arm2: 7  Arm3: 3  Arm4: 87 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | 1: 2407  2: 3875  3: 3733  4: 3881 |
|  |  |  |  |  |  |  |
| 6 | Total Rounds: 100  Arms: 4  Arm1: 10, 20  Arm2: 5, 15  Arm3: 7, 18  Arm4: 8, 21  Greatly increase the deviation. | Arm1: 24  Arm2: 29  Arm3: 28  Arm4: 19 | Arm1: 1  Arm2: 97  Arm3: 1  Arm4: 1 | Arm1: 74  Arm2: 4  Arm3: 4  Arm4: 18 | Arm1: 79  Arm2: 11  Arm3: 1  Arm4: 9 | 1: 1085  2: 271  3: 967  4: 643 |
| Arm1: 22  Arm2: 25  Arm3: 26  Arm4: 27 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 72  Arm2: 2  Arm3: 4  Arm4: 22 | Arm1: 1  Arm2: 2  Arm3: 1  Arm4: 96 | 1: 655  2: 1003  3: 883  4: 592 |
| Arm1: 20  Arm2: 27  Arm3: 28  Arm4: 25 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 57  Arm2: 23  Arm3: 4  Arm4: 16 | Arm1: 74  Arm2: 1  Arm3: 22  Arm4: 3 | 1: 769  2: 444  3: 915  4: 949 |
| Arm1: 35  Arm2: 22  Arm3: 17  Arm4: 26 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 4  Arm2: 3  Arm3: 3  Arm4: 90 | Arm1: 1  Arm2: 1  Arm3: 76  Arm4: 22 | 1: 614  2: 870  3: 875  4: 762 |
| Arm1: 24  Arm2: 26  Arm3: 29  Arm4: 21 | Arm1: 97  Arm2: 1  Arm3: 1  Arm4: 1 | Arm1: 46  Arm2: 3  Arm3: 5  Arm4: 46 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | 1: 624  2: 1125  3: 686  4: 884 |
|  |  |  |  |  |  |  |
|  | Parameters | Round 1 | Round 2 | Round 3 | Round 4 | Round 5 |
| Total Rounds: 100  Arms: 4  Arm1: 10, 4  Arm2: 5, 5  Arm3: 7, 3  Arm4: 8, 2  E = 10 | Arm1: 3  Arm2: 4  Arm3: 2  Arm4: 91  Total: 747 | Arm1: 3  Arm2: 3  Arm3: 91  Arm4: 3  Total: 662 | Arm1: 87  Arm2: 3  Arm3: 5  Arm4: 5  Total: 846 | Arm1: 52  Arm2: 3  Arm3: 2  Arm4: 43  Total: 832 | Arm1: 69  Arm2: 2  Arm3: 26  Arm4: 3  Total: 870 |
| E = 20 | Arm1: 26  Arm2: 4  Arm3: 11  Arm4: 59  Total: 799 | Arm1: 71  Arm2: 5  Arm3: 10  Arm4: 14  Total: 875 | Arm1: 90  Arm2: 3  Arm3: 3  Arm4: 4  Total: 917 | Arm1: 77  Arm2: 14  Arm3: 2  Arm4: 7  Total: 850 | Arm1: 41  Arm2: 4  Arm3: 51  Arm4: 3  Total: 749 |
| E = 30 | Arm1: 62  Arm2: 3  Arm3: 14  Arm4: 21  Total: 913 | Arm1: 63  Arm2: 7  Arm3: 22  Arm4: 8  Total: 766 | Arm1: 50  Arm2: 9  Arm3: 9  Arm4: 32  Total: 825 | Arm1: 60  Arm2: 5  Arm3: 7  Arm4: 28  Total:917 | Arm1: 72  Arm2: 13  Arm3: 9  Arm4: 7  Total: 840 |
| E = 40 | Arm1: 9  Arm2: 60  Arm3: 20  Arm4: 11  Total: 563 | Arm1: 68  Arm2: 11  Arm3: 10  Arm4: 11  Total: 780 | Arm1: 61  Arm2: 17  Arm3: 14  Arm4: 8  Total: 824 | Arm1: 65  Arm2: 7  Arm3: 13  Arm4: 15  Total: 821 | Arm1: 56  Arm2: 10  Arm3: 19  Arm4: 15  Total: 855 |
| E = 50 | Arm1: 36  Arm2: 20  Arm3: 7  Arm4: 37  Total: 790 | Arm1: 57  Arm2: 14  Arm3: 6  Arm4: 23  Total: 874 | Arm1: 45  Arm2: 15  Arm3: 23  Arm4: 17  Total: 758 | Arm1: 61  Arm2: 12  Arm3: 14  Arm4: 13  Total: 657 | Arm1: 32  Arm2: 47  Arm3: 12  Arm4: 9  Total: 672 |

Comparison of Algorithms

Explore only is an algorithm where an arm is selected at random each round, meaning each arm is visited roughly equally overall. This is arguably the worst algorithm. In all the test conducted above explore only collected around half of the maximum points within each test and all the runs. It beat a few of the other algorithms in test 6 where the deviation on the arms was greatly increased as the point values given would differ greatly from the main value meant for the arm, meaning some of the other algorithms would fail and explore only would work better simply because it explored all arms equally, however, the results for test 6 were wildly different for each run accounting for the wildly different values available.

Exploit only is the second worst algorithm. Its performance greatly depends on which arm gives it the best results during the first round. This algorithm best performance was test 5 where the values for each arm differed greatly and the deviation was low, meaning it was easy for the algorithm to find the highest value arm and stick to it. It scored close to max points on each run. The worst performance for this algorithm was test 4 where more arms with low values and low deviance were added. The algorithm picked the wrong arm on multiple runs and scored extremely low.

E-Greedy is the second-best preforming algorithm. Its performance stayed consistent over all the tests conducted, however, this algorithm is highly dependent on the value used for E. As the second table shows, for the base values used, when E is equal to 10 the values are around 800, but when E is equal to 50 the values end up around 600. When E is 20-30 the best results are produced, giving points around 900. Of course, this is just with the base stats, which E value is best will also depends on the other values such as number of arms, difference in point values and difference in deviance.

The Upper-Confidence-Bound (UCB) algorithm is the best performing algorithm, over all the tests it was near-perfect scoring over all the runs. This is a great algorithm that weighs the values of each arm before choosing, not leaving it to random chance or external values. The test that this algorithm faltered on was test 6, where the deviance values were greatly increased, but even then, its lowest value was about half of the max, and all of the algorithms seemed to struggle with this test.

Bibliography

[ritvikmath](https://www.youtube.com/@ritvikmath)(2020) Multi-Armed Bandit: Data Science Concepts. Available at: <https://www.youtube.com/watch?v=e3L4VocZnnQ>

[ritvikmath](https://www.youtube.com/@ritvikmath)(2020) Best Multi-Armed Bandit Strategy? (feat: UCB Method). Available at: <https://www.youtube.com/watch?v=FgmMK6RPU1c>

[Nikolay Manchev](https://domino.ai/blog/author/nikolay-manchev)(2023) Reinforcement learning: The K-armed bandit problem. Available at: <https://domino.ai/blog/k-armed-bandit-problem>

[Nikolay Manchev](https://domino.ai/blog/author/nikolay-manchev)(2022) Semi-uniform strategies for solving K-armed bandits. Available at: <https://domino.ai/blog/semi-uniform-strategies-for-solving-k-armed-bandits>

samishawl(2020) Upper Confidence Bound Algorithm in Reinforcement Learning. Available at:<https://www.geeksforgeeks.org/upper-confidence-bound-algorithm-in-reinforcement-learning/>

[Ankit](https://www.analyticsvidhya.com/blog/author/ankit2106/)(2023) Reinforcement Learning Guide: Solving the Multi-Armed Bandit Problem from Scratch in Python. Available at:<https://www.analyticsvidhya.com/blog/2018/09/reinforcement-multi-armed-bandit-scratch-python/>