Polynomial Time Approximation Scheme – By GeeksForGeeks

It is a very well-known fact that there is no known polynomial time solution for NP Complete problems and these problems occur a lot in real world (See this, this and this for example). So there must be a way to handle them. We have seen algorithms to these problems which are p approximate (For example 2 approximate for Travelling Salesman). Can we do better?

Polynomial **T**ime **A**pproximation **S**cheme (PTAS) is a type of approximate algorithms that provide user to control over accuracy which is a desirable feature. These algorithms take an additional parameter $\varepsilon > 0$ and provide a solution that is $(1 + \varepsilon)$ approximate for minimization and $(1 - \varepsilon)$ for maximization. For example consider a minimization problem, if ε is 0.5, then the solution provided by the PTAS algorithm is 1.5 approximate. The running time of PTAS must be polynomial in terms of n, however it can be exponential in terms of ε

In PTAS algorithms, the exponent of the polynomial can increase dramatically as ε reduces, for example if the runtime is $O(n^{(1/\varepsilon)!})$ which is a problem. There is a stricter scheme, **F**ully **P**olynomial **T**ime **A**pproximation **S**cheme (FPTAS). In FPTAS, algorithm need to polynomial in both the problem size n and $1/\varepsilon$.

Example (0-1 knapsack problem):

We know that 0-1 knapsack is NP Complete. There is a DP based pseudo polynomial solution for this. But if input values are high, then the solution becomes infeasible and there is a need of approximate solution. One approximate solution is to use Greedy Approach (compute value per kg for all items and put the highest value per kg first if it is smaller than W), but Greedy approach is not PTAS, so we don't have control over accuracy.
Below is a FPTAS solution for 0-1 Knapsack problem:

Input:

W (Capacity of Knapsack)
val[0..n-1] (Values of Items)
wt[0..n-1] (Weights of Items)

1. Find the maximum valued item, i.e., find maximum value in val[]. Let this maximum value be maxVal.

2. Compute adjustment factor k for all values

```
    k  = (maxVal * ε) / n
```

3. Adjust all values, i.e., create a new array val'[] that values divided by k. Do following for every value val[i].

```
    val'[i] = floor(val[i] / k)
```

4. Run DP based solution for reduced values, i,e, val'[0..n-1] and all other parameter same.

The above solution works in polynomial time in terms of both n and ε. The solution provided by this FPTAS is (1 – ε) approximate. The idea is to rounds off some of the least significant digits of values then they will be bounded by a polynomial and 1/ε.

Example:

```
val[] = {12, 16, 4, 8}
wt[]  = {3, 4, 5, 2}
W = 10
ε = 0.5

maxVal = 20 [maximum value in val[]]
Adjustment factor, k = (16 * 0.5)/4 = 2.0

Now we apply DP based solution on below modified
instance of problem.

val'[] = {6, 8, 2. 4}  [ val'[i] = floor(val[i]/k) ]
wt[] = {3, 4, 5, 2}
W = 10
```

How is the solution (1 – ε) * OPT?

Here **OPT** is the optimal value. Let **S** be the set produced by above FPTAS algorithm and total value of S be val(S). We need to show that

```
       val(S) >= (1 - ε)*OPT
```

Let **O** be the set produced by optimal solution (the solution with total value OPT), i.e., val(O) = OPT.

```
       val(O) - k*val'(O) <= n*k
       [Because val'[i] = floor(val[i]/k) ]
```

After the dynamic programming step, we get a set that is optimal for the scaled instance
and therefore must be at least as good as choosing the set S with the smaller profits. From that, we can conclude,

```
      val'(S) >= k . val'(O)
              >= val(O) - nk
              >= OPT - ε * maxVal
              >= OPT - ε * OPT [OPT >= maxVal]
```

```
                >= (1 - ε) * OPT
```

Sources:
http://math.mit.edu/~goemans/18434S06/knapsack-katherine.pdf
https://en.wikipedia.org/wiki/Polynomial-time_approximation_scheme