Advanced Algorithms and Programming Project

Johana.A Emma.A

June 16, 2020

1 Discrete Unit Disk Cover Problem

Question 1 - Greedy Algorithm Description

Algorithm 1: Greedy Algorithm

```
Input: a set of $n$ aligned points $P$ and a set of $m$ circles $Q$
Output: a set of circles $Q^* \subseteq Q$
1  begin
2      $remainingPoints \leftarrow$ list of all points in $P$ sorted from left to right
3      $circles \leftarrow$ list of all circles in $Q$ sorted from left to right
4      $Q^* \leftarrow \varnothing$
5      while $remainingPoints \neq \varnothing$ do
6          $p \leftarrow$ the leftmost point in $remainingPoints$
7          $c \leftarrow$ the righmost circle that contains $p$ in $circles$
8          $Q^* \leftarrow Q^* + \{c\}$
9          $remainingPoints \leftarrow remainingPoints -$ all points covered by $c$
10     return $Q^*$
```

Here is the output returned by the implemented algorithm for some instances of the problem in 1 dimension :

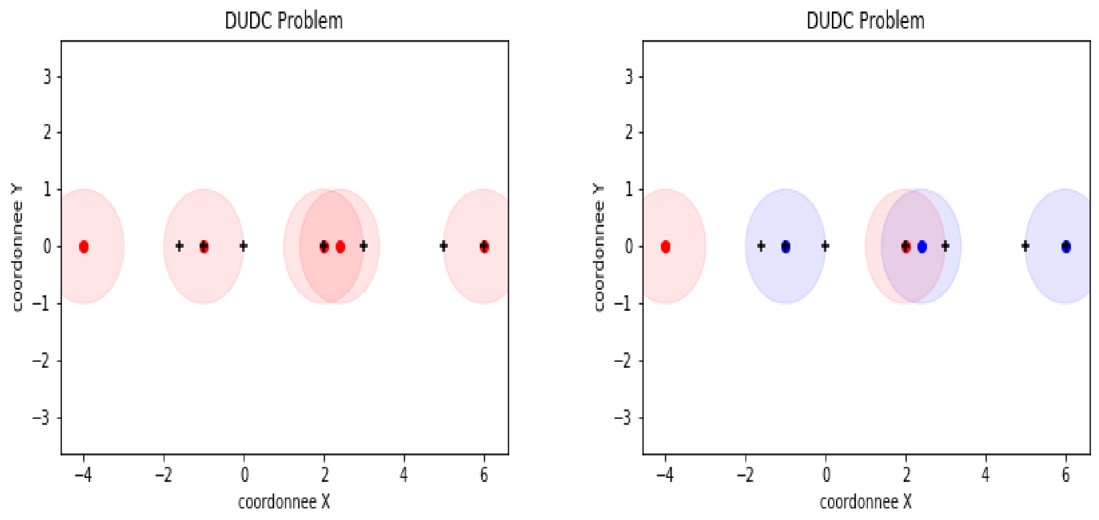


Figure 1: DUDC Exemple 1

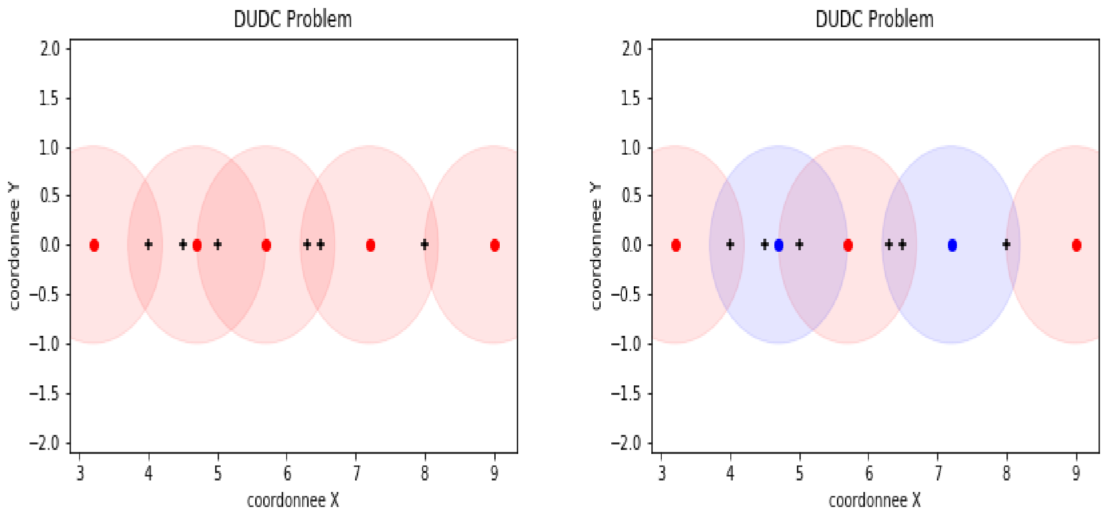


Figure 2: DUDC Exemple 2

Proof of Optimality:
Let P, Q be an instance of the DUDC problem in 1 dimension such that at least one solution exists, i.e. all points in P are covered by at least one circle in Q. Let $OPT \subseteq Q$ be an optimal solution to the DUDC problem for this instance, i.e. a set of circles that covers all points in P of minimum size. Let Q^* be the output of the previously described greedy algorithm. Consider the following procedure :

- Let $circles$ be the list of circles in OPT ordered from left to right and $C = \varnothing$.
- For each circle c in $circles$, consider p the leftmost point covered by c. If $c \notin C$, replace c in OPT by the rightmost circle in Q that covers p and add this circle to C.

At the end of this procedure, OPT is still a valid solution (all points in P are covered) with the same size since each circle that is removed from OPT in the procedure is replaced by exactly one circle. OPT is also now equal to the output of our algorithm Q^*. Therefore $Card(Q^*) = Card(OPT)$ and Q^* is optimal.

Question 3 - Prove DUDC problem is NP-hard

In order to prove that DUDC problem is NP-hard, we will prove that there exists a polynomial reduction from the Minimum Vertex Cover problem to DUDC problem. We know that this problem is NP-complete on discrete graphs.
Let $G = (V, E)$ be a discrete graph. The purpose of the Minimum Vertex Cover problem is to find a subset $V' \subseteq V$ of vertices such that all edges in E are covered by vertices in V', i.e. for all $e \in E$, at least one of the endpoints of e is in V'.
To solve the Minimum Vertex Cover problem on G, we can represent G as a set of unit circles Q and a set of points P in the 2-dimensional plane, where each circle in Q corresponds to a vertex in V and each point in P corresponds to an edge in E. P and Q should have the following property : each circle q in Q should cover all points that correspond to edges covered by the vertex associated to q, and only those points.
If we solve the DUDC problem on (P, Q), we obtain a solution for the Minimum Vertex Cover problem : if Q^* is the set of circles solution of the DUDC problem on (P, Q), we can derive $V' \subseteq V$ the solution to the Minimum Vertex Cover problem from Q^* since each circle in Q^* is associated to a vertex in V.
Therefore since we can make a polynomial reduction from an NP-complete problem (in our case the Minimum Vertex Cover problem) to the DUDC problem, the DUDC problem is NP-hard.

Question 4 - Branch and Bound Algorithm Description

For the Branch and Bound algorithm, we consider the list $circles$ of available circles ordered by number of covered points in descending order. Each node of the state space tree will consist of a set of taken circles and a set of passed circles. In the root node (i.e. the initial promising node), both these sets are empty.
For each node, we compute a lower bound using the following procedure :

Algorithm 2: Bounding Procedure

```
Input: a list of remaining circles $C$ that have neither been taken nor passed of size
       $m$ sorted by descending order of number of covered points, and a set of
       remaining points $P$
Output: an estimation of the minimum number of circles to take in order to cover
        all points
1  begin
2      $LB \leftarrow$ number of circles already taken
3      $nbPoints \leftarrow$ size of $P$
4      for $i = 0$ to $m$ do
5          $c \leftarrow C[i]$
6          $nbPoints \leftarrow nbPoints -$ number of points covered by $c$
7          $LB \leftarrow LB + 1$
8          if $nbPoints \leq 0$ then
9              break
10     if $nbPoints > 0$ then
11         $LB \leftarrow +\infty$
12     return $LB$
```

A node is considered to be promising if its lower bound is not $+\infty$ and if it is not higher than the bound of an already found solution. Until there is no promising node left, we consider the most promising node, i.e. the node with the lowest bound. If it is not a solution node (i.e. if the circles in its set of taken circles are not enough to cover all points), we perform a branching.
For the branching, we consider the first circle c in $circles$ which has neither been taken nor passed in the current state (if there is no such circle, it means that the problem doesn't have a solution). We then create two child nodes from the current node, the left child node having c in its set of taken circles, and the right child node having c in its set of passed circles.
The optimal solution is found when at least one solution has been found and there is no promising node left. If there are several solutions, the optimal solution is the one with the lowest number of taken circles.

Here is the output returned by the implemented algorithm for some instances of the problem in 2 dimensions :

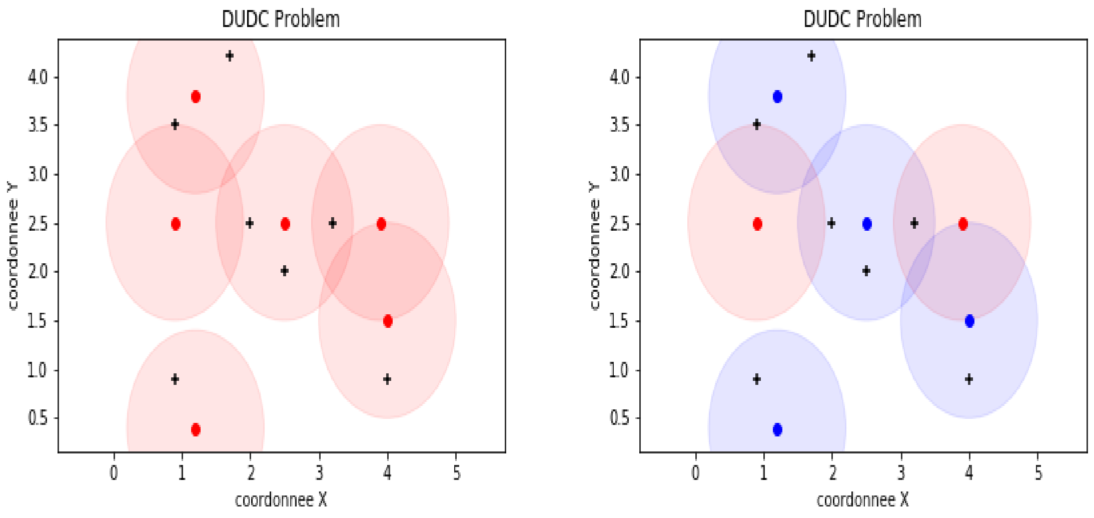


Figure 3: DUDC Exemple 3

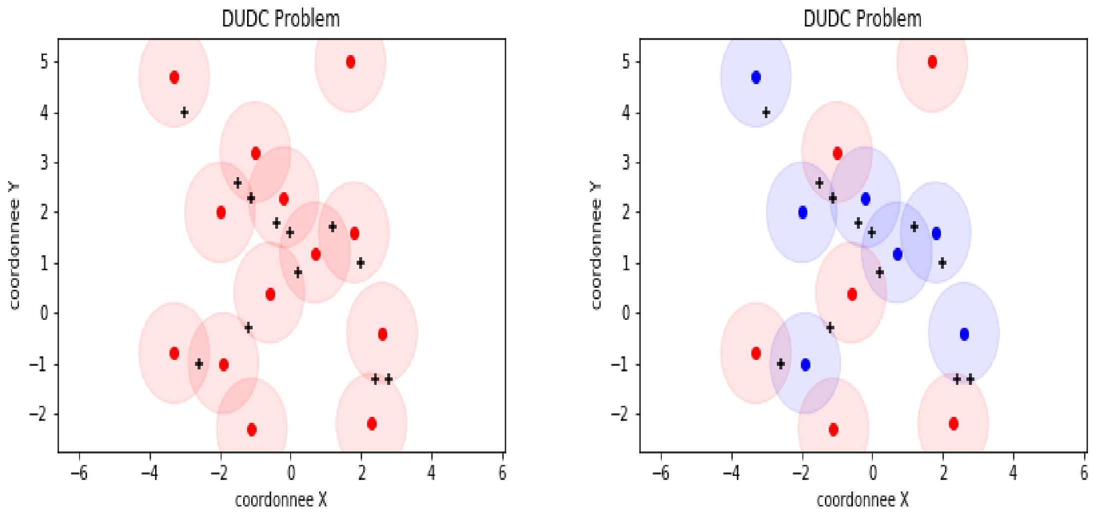


Figure 4: DUDC Exemple 5

2 Upper Envelope of Some Linear Functions

Question 1 - Divide and Conquer algorithm Description

The general idea of this algorithm is to divide the initial list of linear functions into two lists and recursively compute the upper envelope of these sub-lists (the base case being when there is only one function in the list). Then, the two resulting envelopes are merged by doing a vertical line sweep from left to right on the two envelopes, and adding at each step parts of the envelope that is above to the final solution.

Algorithm 3: Divide and Conquer

```
Input: a set of linear functions $F$ of size $n$
Output: an upper envelope of $F$ composed of a list of functions $F^*$ and a list of
        breakpoints $B^*$
begin
    if $n \leq 1$ then
        $F^* \leftarrow F$
        $B^* \leftarrow \varnothing$
        return $F^*$, $B^*$
    $F_1^*, B_1^* \leftarrow UpperEnvelope(F[: \frac{n}{2}])$
    $F_2^*, B_2^* \leftarrow UpperEnvelope(F[\frac{n}{2} :])$
    $F^* \leftarrow [\,]$
    $B^* \leftarrow [\,]$
    $Intervals \leftarrow$ all intervals between points in $B_1^* \cup B_2^* \cup \{-\infty, \infty\}$ from left to
      right
    for $I \in Intervals$ do
        $f_1 \leftarrow$ function in $F_1^*$ contributing to its envelope on $I$
        $f_2 \leftarrow$ function in $F_2^*$ contributing to its envelope on $I$
        $intersection \leftarrow$ x-coordinate of the intersection between $f_1$ and $f_2$
        if $f_1$ and $f_2$ don't intersect on $I$ then
            $f \leftarrow$ function that is above on $I$ among $f_1$ and $f_2$
            $F^* \leftarrow F^* + [f]$    if $f$ is not already in $F^*$
            $B^* \leftarrow B^* +$ breakpoints associated to $f$ that are reached in $I$ and not
              already in $B^*$
        else
            $f \leftarrow$ function that is above before $intersection$
            $g \leftarrow$ function that is above after $intersection$
            $F^* \leftarrow F^* + [f, g]$    if $f$ is not already in $F^*$
            $B^* \leftarrow B^* + intersection$ and breakpoints associated to $f$ and $g$ reached
              in $I$ that are not already in $B^*$ ordered from left to right
    return $F^*$, $B^*$
```

Here is the output returned by the implemented algorithm for some instances of the problem :

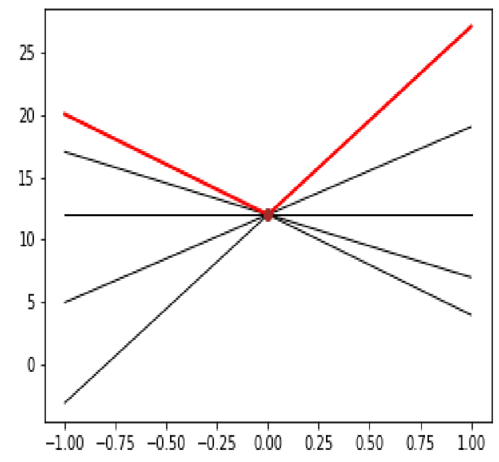


Figure 5: Upper Envelope for Set 10 functions

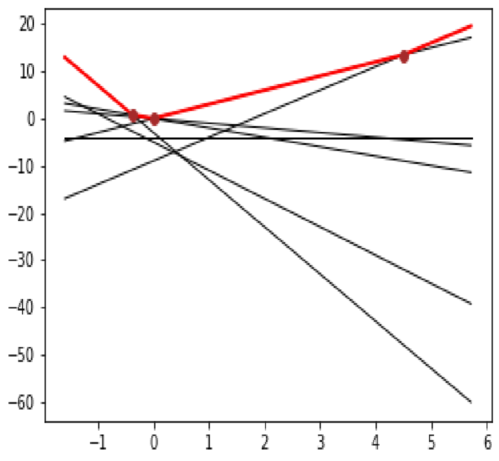


Figure 6: Upper Envelope for Set 7 functions

Question 2 - Time complexity Analysis

The recurrence equation corresponding to the algorithm presented above is

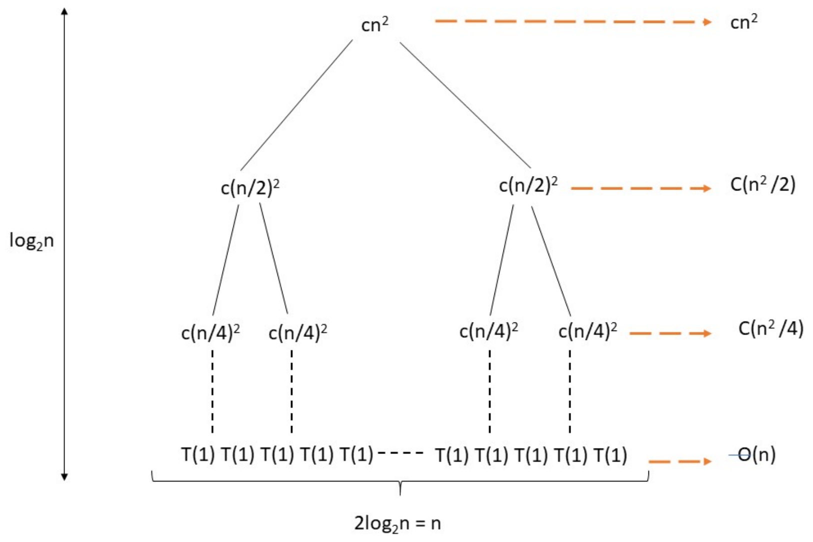
$$\begin{aligned} T(n) = 2 \times T(\frac{n}{2}) + n^2 \\ T(1) = 1 \end{aligned} \tag{1}$$

In fact, the original problem is divided into 2 sub-problems of size $\frac{n}{2}$. Besides, to combine the sub-problems generated, we compare n functions on n intervals created which cost finally n^2.
We construct a recursive tree where each node is the cost of the corresponding sub-

problem. The size of a sub-problems at level i is: $\left(\frac{n}{2^i}\right)^2 = \frac{n^2}{2^{2i}}$
The total number of levels x in this tree is therefore:

$$\frac{n^2}{2^{2x}} = 1 \iff 2^{2x} = n^2 \iff x = \frac{\log(n)}{\log(2)}$$

Adding the root the tree has $\log_2(n) + 1$ levels.
Finally, the number of last level nodes is : $2^x = 2^{\log_2(n)} = n$



$$\begin{aligned}
T(n) &= cn^2 + c\frac{n^2}{2} + c\frac{n^2}{4} + ... + \theta(n) \\
&= \sum_{i=1}^{\log_2(n)-1} (\frac{1}{2})^i cn^2 + \theta(n) \\
&< \sum_{i=1}^{\infty} (\frac{1}{2})^i cn^2 + \theta(n)
\end{aligned}$$

Using geometric series result we have :

$$\begin{aligned}
\sum_{i=1}^{\infty} (\frac{1}{2})^i cn^2 + \theta(n) &= \frac{1}{1-\frac{1}{2}} cn^2 + \theta(n) \\
&= 2cn^2 + \theta(n) \\
&= O(n^2)
\end{aligned}$$

Therefore the complexity of the Divide and Conquer algorithm is $O(n^2)$.

Question 4 - Dynamic Programming Algorithm Description

In this algorithm, if the input is a list of linear functions of size n, we compute the upper envelope of the first k functions for $k \in \{1, 2, ..., n\}$, and return the last computed envelope on all the n functions. In order to compute the envelope of k functions when $k \geq 2$, we take the previously computed envelope of $k-1$ functions and update it with the k^{th} function.

Algorithm 4: Dynamic Programming

```
Input: a set of linear functions $F$ of size $n$
Output: an upper envelope of $F$ composed of a list of functions $F^*$ and a list of
        breakpoints $B^*$
begin
    if $n = 0$ then
        $F^* \leftarrow \varnothing$
        $B^* \leftarrow \varnothing$
        return $F^*$, $B^*$
    $envelopes \leftarrow [\,]$
    $F^* \leftarrow F[0]$
    $B^* \leftarrow \varnothing$
    $envelopes \leftarrow envelopes + (F^*, B)$
    for $f \in F[1:]$ do
        $(F', B') \leftarrow$ the last computed envelope $envelopes[-1]$
        $intersections \leftarrow$ intersections between $f$ and $(F', B')$
        if $f$ does not intersect $(F', B')$ then
            $F^* \leftarrow F'$
            $B^* \leftarrow B'$
        else
            $F^* \leftarrow$ functions in $F'$ that are above $f$ + $f$ ordered from left to right
            $B^* \leftarrow$ breakpoints in $B'$ that are above $f$ + $intersections$ ordered from
             left to right
        $envelopes \leftarrow envelopes + (F^*, B^*)$
    return last computed envelope in $envelopes$
```

Here is the output returned by the implemented algorithm for some instances of the problem :

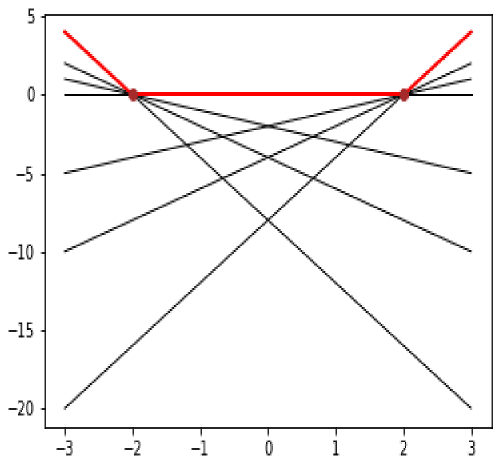


Figure 7: Upper Envelope for Set11 functions

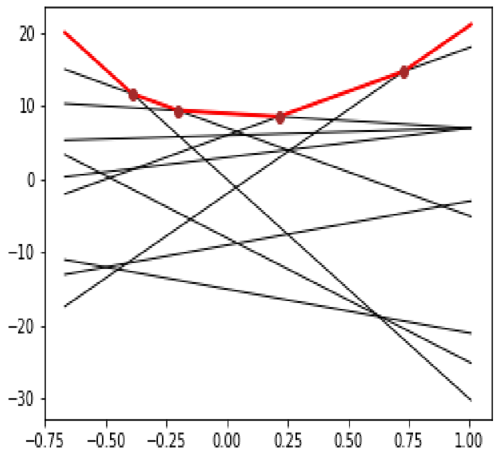


Figure 8: Upper Envelope for Set 2 functions

Question 5 - Time Complexity Analysis

$$\begin{aligned} T(n) &= T(n-1) + O(n) \\ T(1) &= 1 \end{aligned} \tag{2}$$

The resolution comes from the successive replacement method:

$$\begin{aligned} T(n) &= T(n-1) + O(n) \\ &= T(n-2) + O(n-1) + O(n) \\ &= T(n-3) + O(n-2) + O(n-1) + O(n) \\ &= ... \\ &= T(1) + O(1) + O(2) + ... + O(n) \\ &= T(1) + \sum_{k=1}^{n} O(k) \\ &= O\big(\sum_{k=1}^{n} k\big) \\ &= O\big(\frac{n(n-1)}{2}\big) \\ &= O(n^2) \end{aligned}$$

Conclusion

The time complexity of this problem considering the divide and conquer approach and the dynamic programming approach is in the worst case $O(n^2)$.