Improving Average Time Complexity of Tiling Arrays Using A* Algorithm

Mohamed Abd Elhamid Abbas, PhD
Faculty of Engineering
Delta University for science and technology
Gamasa City, EGYPT
mohaym_2000@yahoo.com

ABSTRACT

Genomic tiling arrays are able to inspect the entire genome of random species for which the sequence is known. The plan or assortment of proper oligonucleotide probes for such arrays is however computationally difficult if features such as oligonucleotide quality and cyclic regions are to be considered. Prior works devise the minimal tiling path problem for the selection of oligonucleotides using Dijkstra's shortest path algorithm to compute worldwide optimal tiling paths from millions of candidate oligonucleotides on computers. Although Dijkstra's algorithm works well, it's so complicated that it may take a long time for routers to process it, and the efficiency of the network fails. Also, if a router gives the wrong information to other routers, all routing decisions will be ineffective. The main contribution of this paper is to set up a search approach that can decrease the average complexity time of tilling arrays. This goal is achieved by searching for the shortest path to the probes using A* algorithm instead of Dijkstra's algorithm. A* Algorithm is more efficient and it decreases the average time complexity. In A* Algorithm, the run time is much shorter than the Dijkstra's algorithm.

Key Words: Tiling arrays, DNA microarrays, A* Algorithm, Dijkstra's algorithm , Oligonucleotide probes

1. Introduction

Tiling Arrays hybridize labeled target molecules to unlabeled probes fixed on to a solid surface. Tiling arrays vary in the character of the probes. Number of features on a single array can range from 10,000 to greater than 6,000,000, with each feature containing millions of copies of one probe

[1,6]. Tiling arrays can produce an unbiased look at gene expression because previously unidentified genes can still be incorporated. They are quickly becoming one of the most powerful tools in genome-wide investigations. Representing a genome completely opens new routes, such as transcription profiles that do not rely on prior gene prediction. The probe selection for the array needs to be customized, which is economically feasible as several providers offer customer designed arrays. The determination of suitable probe sequences, typically contiguous subsequences of the genomic sequence, however is computationally demanding even for relatively small genome sizes of bacteria if the quality of probes is considered in the design [1,2]. An oligonucleotide is a short nucleic acid polymer, typically with twenty or fewer bases. Although they can be formed by bond cleavage of longer segments, they are now more commonly synthesized by polymerizing individual nucleotide precursors. Automated synthesizers allow the synthesis of oligonucleotides up to 160 to 200 bases [1-6]. The multi-criterion optimization problem of selecting an optimal subset of oligonucleotide probes is formulated from set of candidates as the minimal cost tiling path problem. This problem is equivalent to a shortest path problem, which is solved in prior works using Dijkstra's shortest path algorithm. Prior works concentrated only in finding the shortest path to oligonucleotide probes using Dijkstra's algorithm. A* ,A-star, is another fast algorithm for finding the shortest path but its applications do not exceed engineering science such as town planning and roads. This paper study the using of A* in finding the shortest path between probes in tilling arrays. It analyze the using of this algorithm instead of Dijkstra's algorithm to improve the average time complexity in tilling arrays. Next section presents prior work problem formulation, the concepts of A* and Dijkstra's algorithms in more details. This paper is organized as follow: in section

2 prior works are introduced , section 3 presents the suggested approach for finding shortest path to oligonucleotides, section 4 presents the simulation analysis and the results of the proposed approach and finally a conclusion is extracted from the results.

2. Prior Works

Prior works formulated the problem of finding the shortest path between the start oligonucleotide probe and the destination one as finding the shortest path in a graph as depicted in Figure 1[1, 7]. The set of vertices are the probes {v1, . . ., vn} with special nodes 0 and n + 1, which are virtual probes. It is required to tile before the start and after the end of the sequence. For an edge (va, vb), with va ≥1, vb ≤ n Where n is the total number of vertices in the graph. The total weight is computed as w(va, vb). The weights w(0, va) and w(va, n+1) are defined as d(0, va). Prior work depends to find the shortest path using Dikstra's shortest path algorithm [1]. Another algorithm that find the shortest path is A* algorithm. It is an efficient more than Dikstra's algorithm but its applications appears only in engineering science such as civil engineering, town planning and roads. Next subsections depict these algorithms in details.

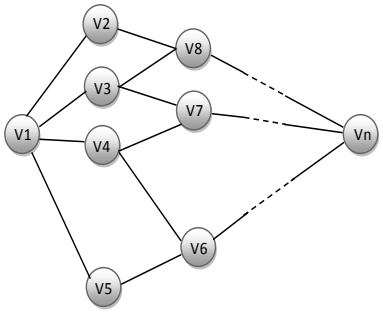

Figure 1: A graph represents a group of oligonucleotide probes

2.1 The Dijkstra's shortest path algorithm

The Dijkstra's shortest path algorithm is the most commonly used to solve the single source shortest path problem today. For a graph G(V, E), where V is the set of vertices and E is the set of edges, the running time for finding a path between two vertices varies when different data structure are used. The run time of first for loop is O(V).

In each iteration of the while loop the total run time is O(E). Figure 2 depicts the steps of algorithm in details [7-13].

```
for each u G:
d[u] = infinity;
parent[u] = NIL;
End for
d[s] = 0; // s is the start point
H = {s}; // the heap
while NotEmpty(H) and
targetNotFound:
u = Extract_Min(H);
label u as examined;
for each v adjacent to u:
if d[v] > d[u] + w[u, v]:
d[v] = d[u] + w[u, v];
parent[v] = u;
DecreaseKey[v, H];
```

Figure2: Dijkstra's algorithm [7]

2.2 A* shortest path Algorithm

The A* algorithm integrates a heuristic into a search procedure. Instead of choosing the next node with the least cost, the choice of node is based on the cost from the start node plus an estimate of immediacy to the destination. In the Searching, the cost of a node V could be calculated as: f(V) = distance from S to V + estimate of the distance to D.

f(V) = d(V) + h(V,D), f(V) $= d(V) + \sqrt{(x(V) - x(D))^2 + (y(V) - y(D))^2}$

Where x(V), y(D) and x(V), y(D) are the coordinates for node V and the destination node D. The shortest path search starts from start point and expands node that goes towards the destination. Therefore, the run time is much shorter than the Dijkstra's algorithm. Figure 3 depicts the steps of algorithm in details [7-13].

```
for each u G:
d[u] = infinity;
parent[u] = NIL;
End for
d[s] = 0; f(V) = 0; H = {s};
while NotEmpty(H) and
targetNotFound:
u = Extract_Min(H);
label u as examined;
for each v adjacent to u:
if d[v] > d[u] + w[u, v] , then
d[v] = d[u] + w[u, v];
p[v] = u;
f(v) = d[v] + h(v, D);
DecreaseKey[v, H];
```

Figure3: A* Algorithm [7]

3 Suggested Approach for Finding Shortest Path to Oligonucleotides

Previous researches suggest finding the shortest path to oligonucleotide nodes using Dijkstra's algorithm. A* algorithm is another one that is

more efficient and has less running time than Dijkstra's algorithm but its applications are in limited range. These applications appear in civil engineering, town planning and shortest paths for roads. It has no applications in medical informatics or in bioinformatics. This paper suggest to use A* algorithm instead of Dijkstra's algorithm in tilling arrays. And study the effect of this replacement in decreasing the average time complexity of tilling arrays. Next subsections study this effect in details and the ability of making benefit of A* algorithm in medical informatics science.

3.1 Suggested approach using A* algorithm

The proposed approach assumes oligonucleotide probes as a graph with multiple nodes. The nodes of the graph are signified as levels, from level 1 to level k. The symbol m represents the number of oligonucleotide nodes in level1 in the oligonucleotide graph. The symbol n represents the total number of the oligonucleotide nodes in all levels of the suggested oligonucleotide graph. The number of nodes between level 1 and level k equals (n-m) as depicted in figure 4. The heuristic procedure operates on number of nodes less than (n-m). The idea of the suggested approach is starting search operation from start node to find the nearest nodes that lie in level 1. Another search operation starts from every node in this level to find the shortest path to the destination using the heuristic procedure of A* algorithm.

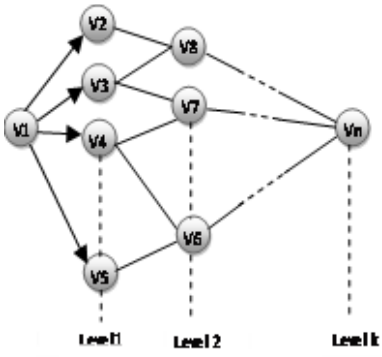

Figure 4: Suggested search approach in oligonucleotide graph

This procedure calculates approximately the distance from every node in level 1 and the destination one. During these calculations, the nodes of the paths are registered in pointers. The smallest distance and its pointer are determined. The related path of this smallest distance is chosen as the shortest path to the oligonucleotide destination node.

4. Simulation Analysis and Results

Validation and analysis of the proposed approach is done via simulation, the simulator is built based on comparing the average time complexity between the proposed approach using A* algorithm and Dijkstra's algorithm. These algorithms are represented as code segments using C# language as depicted in figures 5 and 6. These segments represent the execution statements of the two algorithms. There is a famous rule in calculating the run time from the source code of for loop with size n in [13-21] as follow: Assignment: 3n+2, subtraction: n+1, additions: 3n, division: n, comparisons: n+1 and return: 1. this rule of calculating the average time complexity in are applied for the both segments.

```
for (int v = 0; v < n; v++)
        {
          if (w[u, v] < 0)
            continue;
          if (D[v] < 0) {
            D[v] = D[u] + w[u, v];
            continue;
          }
       if (D[v] > (D[u] + w[u, v]))
            D[v] = D[u] + w[u, v];
        }
```

Figure 5: Dijkstra's algorithm code segment

```
for (int v = 0; v < m1; v++)
        {
          if (w[u, v] < 0)
            continue;
          if (D[v] < 0) {
            D[v] = D[u] + w[u, v];
            continue;
          }
       if (D[v] > (D[u] + w[u, v]))
            D[v] = D[u] + w[u, v];
            F[v] = d(v) + h(v,D)
        }
```

Figure 6: A algorithm code segment*

The results are depicted in Tables 1 and 2 at constant

Table 1: average time complexity for both algorithms at constant m =10

n	15	20	25	30	35	40
A*_algorithm	438	508	578	648	718	788
Dj_algorithm	432	572	712	852	992	1132

Table 2: average time complexity for both algorithms at constant m =30

n	50	75	100	125	150	175
A*_algorithm	1348	1698	2048	2398	2748	3098
Dj_algorithm	1412	2112	2812	3512	4212	4912

alues of m. These results show that searching operation for the shortest path using A* consumes less average complexity time than searching using Dijkstra's algorithm at high values of n. Number of oligonucleotide probes ,n, should be greater than 2.2 m (n > 2.2m) to increase average time complexity of Tiling Arrays Using A* Algorithm. This means when the number of oligonucleotide probes increases, A* Algorithm will be more efficient in searching. Practically this case is suitable because number of tiled probes can reach millions of probes. Figures 7, 8 depict the average time complexity for both algorithms at different values of n at constant values of m. The relation between the average time complexities for Each algorithm and number of probes takes the behavior of straight line. The difference in the slope between the two lines increases with increasing the number of n probes.

5. Conclusion

Tilling array plays an important role in representing the genome of species. It analyzes number of oligonucleotide probes. During this analysis, switching between different probes are vital operation. Searching for the next probe during analysis to reach the destination one is a part of its role. The main contribution of this paper is to suggest a search approach that can decrease the average complexity time of tilling arrays. This goal is achieved by

searching for the shortest path to the probes using A* algorithm instead of Dijkstra's algorithm. The results depict that A* algorithm decreases the average time complexity more than Dijkstra's algorithm especially at large number of probes. Practically this condition is existed in tilling arrays that contains millions of oligonucleotide probes.

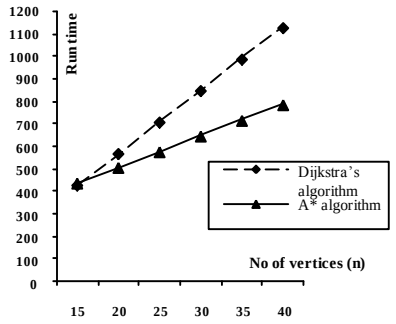

Figure 7: Average time complexity at constant m =10

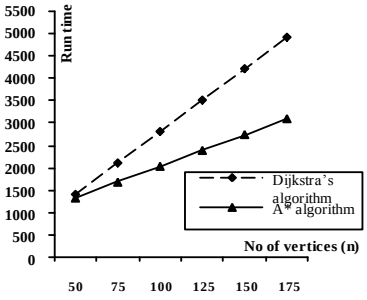

Figure 8: Average time complexity at constant m =30

References:

[1] Alexander Schliep, Roland Krause," Efficient Computational Design of Tiling Arrays Using a Shortest Path Approach", WABI 2007, LNBI 4645, pp. 383–394, 2007.

[2] Kane, M.D., et al," Assessment of the sensitivity and specificity of oligonucleotide (50mer) microarrays" .Nucleic Acids Res. 28(22), 4552–4557 (2000)

[3] Matveeva, O., et al, "Thermodynamic calculations and statistical correlations for oligoprobes design", Nucleic Acids Research 31(14), 4211–4217 (2003)

[4] He, Z., et al., "Empirical Establishment of Oligonucleotide Probe Design Criteria", Applied and Environmental Microbiology 71(7), 3753–3760 (2004)

[5] Pozhitkov, A., et al." Tests of rrna hybridization to microarrays suggest that hybridization characteristics of oligonucleotide probes for species discrimination cannot be predicted", Nucleic Acids Res. 34(9) (2006)

[6]http://en.wikipedia.org/wiki/Oligonucleotide

[7] Liang Dai," Fast Shortest Path Algorithm for Road Network and Implementation", Carleton University,School of Computer Science,COMP 4905,Honours project, Fall Term, 2005.

[8], Yu-Li Chouy H. Edwin Romeijnz Robert L. Smithx. Approximating Shortest Paths in Large-scale Networks with an Application to Intelligent. Transportation Systems. September 27, 1998

[9] nx,z zjc Faramroze Engineer. Fast Shortest Path Algorithms for Large Road Networks. Department of Engineering Science. University of Auckland, New Zealand. http://www.esc.auckland.ac.nz/Organisations/ORSNZ/conf36/papers/Engineer.pdf

[10], Roozbeh Shad, Hamid Ebadi, Mohsen Ghods. Evaluation of Route Finding Methods in GIS Application. Dept of Geodesy and Geomatics Eng. K.N.Toosi University of Technology, IRAN.

[11], Fu, Mengyin, Li, Jie, Deng, Zhihong. "A practical route planning algorithm for vehicle navigation system". Proceedings of the World Congress on Intelligent Control andAutomation (WCICA), v 6, WCICA 2004 - Fifth World Congress on Intelligent Control and Automation, Conference Proceedings, 2004, p 5326-5329

[12] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest and Charles E. Leiserson. Introduction to Algorithms, 2nd edition, McGraw-Hill Higher Education, 2001, ISBN: 0070131511

[13] Mark Burgin,"Algorithmic complexity of recursive and inductive algorithms", Volume 317, Issue 1-3 (June 2004) table of contents, Super recursive algorithms and hypercomputation ,Pages: 31 - 60 ,ISSN:0304-3975

[14] Kannangara, C. S. et al, "Computational Complexity Management of a Real-Time H.264/AVC Encoder ", Circuits and Systems for Video Technology, IEEE Transactions volume 18, 2008

[15] Tianshi Chen," On the Analysis of Average Time Complexity of Estimation of

Distribution Algorithms", 2007 IEEE Congress on Evolutionary Computation (CEC 2007)
[16] H. M¨uhlenbein and G. Paaß, "From recombination of genes to the estimation of distribution i. binary parameters". In Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature – PPSN IV: 178–187, 1996.
[17] H. M¨uhlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm, I.: continuous parameter optimization." Evolutionary Computation 1(1): 25–49, 1993.
[18] Eppstein, D.: Sequence comparison with mixed convex and concave costs. J. Algorithms 11(1), 85–101 (1990)
[19] Galil, Z., Park, K.: A linear-time algorithm for concave one-dimensional dynamic programming. Inf. Process. Lett. 33(6), 309–311 (1990)
[20] Aggarwal, A., Klawe, M., Moran, S., Shor, P., Wilber, R.: Geometric applications of a matrix searching algorithm. In: SCG '86: Proceedings of the second annual symposium on Computational geometry, New York, NY, USA, pp. 285–292. ACM Press, New York (1986)
[21] Aggarwal, A., Schieber, B., Tokuyama, T.: Finding a minimum weight k-link path in graphs with monge property and applications. In: SCG '93: Proceedings of the ninth annual symposium on Computational geometry, New York, NY, USA, pp. 189–197. ACM Press, New York (1993)