Parallel Optimal Weighted Links

Ovidiu Daescu

Department of Computer Science, University of Texas at Dallas,
Richardson, TX 75083, USA
E-mail: daescu@utdallas.edu.

Abstract. In this paper we consider parallel algorithms for computing an optimal link among weighted regions in the 2-dimensional (2-D) space. The weighted regions optimal link problem arises in several areas, such as geographic information systems (GIS), radiation therapy, geological exploration, environmental engineering and military applications. We present a CREW PRAM parallel algorithm and a coarse-grain parallel computer algorithm. Given a weighted subdivision with a total of n vertices, the *work* of the parallel algorithms we propose is only a $O(\log n)$ factor more than that of their (optimal) sequential counterparts.

1 The Weighted Regions Optimal Link Problem

We consider the (2-D) weighted regions optimal link problem, defined as follows: Given a subdivision R of the 2-D space, with m weighted regions R_i, $i = 1, 2, \ldots, m$, and a total of n vertices, find a *link* L such that: (1) L intersects two specified regions $R_s, R_t \in R$ and (2) the weighted sum $S(L) = \sum_{L \cap R_i \neq \phi} w_i * d_i(L)$ is minimized, where w_i is either the weight of R_i or zero and $d_i(L)$ is the length of L within region R_i. Depending on the application, the link L may be (a) unbounded (e.g., a line): the link L "passes through" the regions R_s and R_t; (b) bounded at one end (e.g., a ray): R_s is the source region of L and L passes through R_t and (c) bounded at both ends (e.g., a line segment): R_s is the source region of L and R_t is its destination region. We consider only straight links; the case of links described by some bounded degree curves is left for further study. Let R_L be the set of regions $\{R_{i_1}, \ldots, R_{i_k}\}$ intersected by a link L. Then, w_{i_1} and w_{i_k} are set to zero. This last condition assures that the optimal solution is bounded when a source (and/or a destination) region is not specified (cases (a) and (b)) and allows the link to originate and end arbitrarily within the source and target regions (cases (b) and (c)). See Figure 1 for an example.

The weighted regions optimal link problem is an extension of the optimal weighted penetration problem [5] and arises in several areas such as GIS, radiation therapy, stereotactic brain surgery, geological exploration, environmental engineering and military applications. For example, in military applications the weight w_i may represent the probability to be seen by the enemy when moving through R_i, from a secured source region R_s to another secured target region R_t. In radiation therapy, it has been pointed out that finding the optimal choice

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2073, pp. 649–657, 2001.

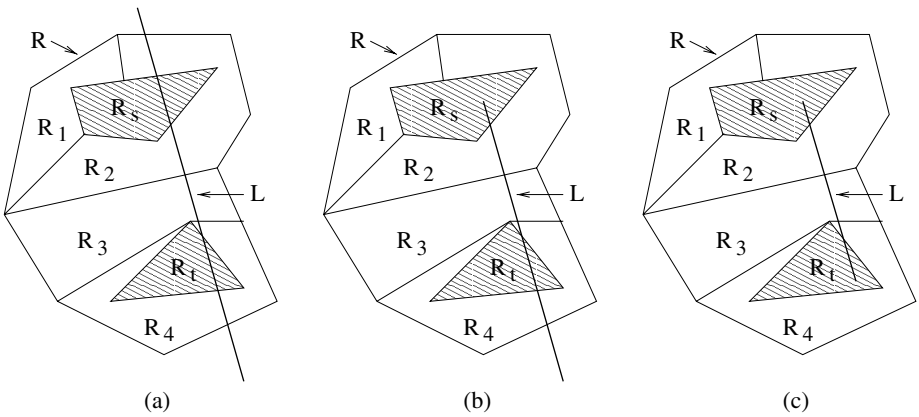

Fig. 1. Illustrating the problem: (a) L intersects R_s and R_t; (b) L originates in R_s and (c) L ends in R_t

for the link (cases (a) and (b)) is one of the most difficult problems of medical treatment optimization [3].

In computational geometry, there are a few results that consider weighted region problems, aiming to compute or approximate an optimal shortest path between pairs of points [1,15,16,17]. Mitchell and Papadimitriou [17] first considered the problem of computing an approximate geodesic shortest path between two points in a weighted planar subdivision. Their algorithm runs in $O(n^8 B)$ time and $O(n^4)$ space, where B is a factor representing the bit complexity of the problem instance, and approximates the optimal solution within an $(1+\epsilon)$ factor. Later, Mata and Mitchell [16] presented an approximation scheme for computing approximate shortest paths in a weighted polygonal subdivision. In $O(kn^3)$ time, they create a graph of size $O(kn)$ for $(1+\epsilon)$ approximate shortest paths, where k depends on ϵ and $0 \leq \epsilon < 1$: by varying the parameter k that controls the graph density, one can get arbitrarily close to the optimal solution.

The optimal link problem however, has a different structure than the shortest path problem. The few papers that consider finding an optimal link either discretize the problem (e.g., see [14]) or consider some simplified versions (e.g., see [18]). Important steps towards solving the optimal weighted penetration problem have been made very recently in [5,7], where it has been proved that the 2-D problem can be reduced to a number of (at most $O(n^2)$) subproblems, each of which asks to minimize a 2-variable function $f(x, y)$ over a convex domain D, where $f(x, y)$ is given as a sum of $O(n)$ terms. These subproblems can be generated sequentially in $O(n^2)$ time and thus the bulk of computation consists of solving the optimization problems. To compute the optimal solution for each subproblem, a global optimization software has been used in [5]. As the number of terms in f increases (i.e., > 100), such global optimization software performs badly in both time and memory usage. Since in practical applications, having

hundreds and even thousands of terms in the objective function is not an uncommon case, sequentially solving the global optimization problems (GOPs) on a single processor seems impractical. Instead, one can take advantage of the fact that, once the feasible domain and the objective function for each subproblem have been produced, the GOPs are independent and can be solved in parallel. After all GOPs are solved, the optimal solution can be obtained by a simple minimum selection.

It would then be of interests to efficiently produce the set of GOPs in parallel. We consider this problem and present the following results: (1) We give an $O(\log n)$ time, $O(n \log n + k)$ processors algorithm in the CREW PRAM model, where k is the total complexity description for the feasible domains of the GOPs ($\Omega(n^2)$ in the worst case). The algorithm is based on the arrangement sweeping techniques of Goodrich *et al.* [13]. Our parallel algorithm implies an optimal output sensitive $O(n \log n+k)$ time sequential algorithm for generating all GOPs, by using the optimal segment arrangement construction in [4]. (2) We show that, if at most n processors are available, all GOPs can be generated using $O(n^2 \log n)$ work. This algorithm is targeted to coarse-grain parallel computer models, consisting of a relatively small set of nodes (up to a few thousand), where each node has its own processor, with fair computing power, and a large local memory, allowing to store all data involved in (sequentially) solving the problem. In contrast, in a fine-grain computing model, one would allow only constant local memory, but unrestrict the number of processing nodes available.

2 Useful Structures

The optimal link problem can be reduced to solving a number of (at most $O(n^2)$) GOPs. Since each GOP can be solved using available global optimization software, we are only concerned with efficiently generating the GOPs. We start by describing the structure of a GOP.

Let L be a link intersecting the source and target regions R_s and R_t. Let S be the set of line segments in the subdivision R and let $S_{st} = \{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\}$ be the subset of line segments in S that are intersected by L. Consider rotating and translating L. An event e_v will occur when L passes a vertex v of R. Such an event corresponds to some line segments (with an endpoint at v) entering or leaving S_{st}. As long as no event occurs, the formula describing the objective function $S(L)$ does not change and has the expression $S(L) = \sum_{i=i_1}^{i_{k-1}} w_i * d_i$, where d_i is the length of L inside region R_i and s_i, s_{i+1} are on the boundary of R_i. We refer the reader to [5,9] for more details.

Let $H = \{l_1, l_2, \ldots, l_n\}$ be a set of n straight lines in the plane. The lines in H partition the plane into a subdivision, called the *arrangement* $A(H)$ of H, that consists of a set of convex regions (cells), each bounded by some line segments on the lines in H. In general, $A(H)$ consists of $O(n^2)$ faces, edges and vertices and it can be computed in $O(n^2)$ time and $O(n)$ space, by sweeping the plane with a pseudoline [11].

For case (a) of the optimal link problem (the link L is a line), using a point-line duality transform that preserves the above/bellow relations (i.e., a point p above a line l dualizes to a line that is above the dual point of l), all lines intersecting the same subset of segments $S_{st} \in S$ correspond to a cell in the dual arrangement $A(R)$ of R, defined by $H_R = \{l_1, l_2, \ldots, l_n\}$, where $l_i \in H_R$ is the dual of vertex $v_i \in R$. The case of a semiline (case (b) of the link problem), and that of a line segment can be reduced to that of a line, by appropriately maintaining the set of line segments intersected by L and dropping those that arise before a segment in R_s or after a segment in R_t. This can be done sequentially in constant time, by extending the data structures in [5,9]. We leave the details to the full paper.

Generating and sweeping the entire arrangement however, as proposed in [5], may not be efficient since many cells of $A(R)$ may correspond to set of links that do not intersect R_s and/or R_t. Rather, we would like to compute only the cells of interest. Assume that R_s and R_t are convex (the results can be extended in the same complexity bounds to the nonconvex case, by observing that a line intersects a region R_i if and only if it intersects the convex hull of R_i; more details in the full version). Using a point-line duality transform that maps the line $y = mx + p$ in the (x,y) plane to the point (m, p) in the (m,p) plane, the set of lines intersecting R_s (resp., R_t), define a "strip" region D_{R_s} (resp. D_{R_t}) in between two m-monotone, unbounded and nonintersecting chains.

The set of lines intersecting both R_s and R_t thus correspond to the common intersection of D_{R_s} and D_{R_t}. Let k_s and k_t be the number of vertices of R_s and R_t, respectively. Let $D_{st} = D_{R_s} \cap D_{R_t}$.

Lemma 1. *D_{st} is a (possibly unbounded) region bounded by two m-monotone chains with a total of $O(k_s + k_t)$ vertices.*

Proof. D_{R_s} has k_s vertices, each vertex corresponding to a line supporting a boundary segment of R_s. Similarly, D_{R_t} has k_t vertices, each vertex corresponding to a line supporting a boundary segment of R_t. Since there are only $O(1)$ common tangents to R_s and R_t, the pairs of chains defining the boundaries of D_{R_s} and D_{R_t} intersect $O(1)$ times, and the proof follows. □

An example is given in Figure 2, where D_{st} is the quadrilateral with vertices A,B,C and D.

Lemma 2. *The lines in $A(R)$ have at most $O(n)$ intersections with the chains bounding D_{st}.*

Proof. Only $O(1)$ lines tangent to R_s and R_t can pass through a point p. Then, the dual line of p can intersect the chains bounding D_{st} only $O(1)$ times, from which the proof follows. □

Thus, computing the cells of the arrangement defined by $A(R)$ that correspond to set of lines intersecting both R_s and R_t reduces to computing the arrangement of $O(n)$ line segments in D_{st} (some of these line segments may in fact be semilines, but this does not influence the overall computation).

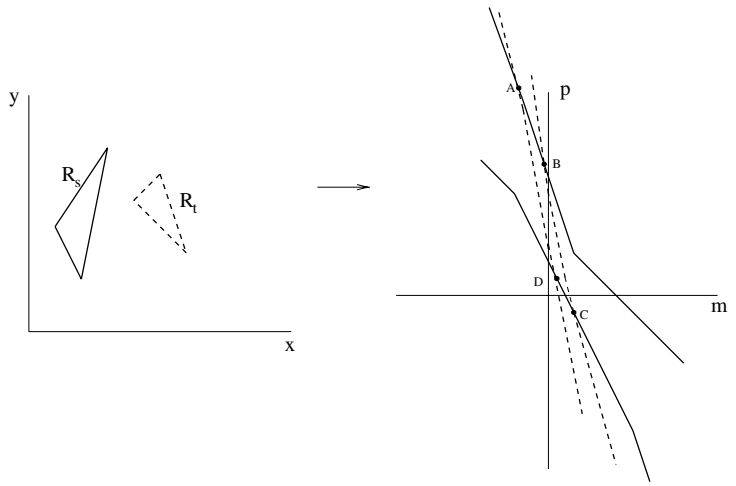

Fig. 2. The line transversals of R_s, R_t dualize to quadrilateral D_{st}=ABCD

3 Parallel Solutions

In this section we present two parallel solutions for the optimal link problem. The first algorithm uses the CREW PRAM model of computation. Recall that in this model processors act synchronously and may simultaneously access for reading the same memory location on a shared memory space. To obtain output sensitive algorithms, we use the paradigm in [13]: the pool of virtual processors can grow as the computation proceeds, provided that the allocation occurs globally [12]. Given a subdivision R with a total of n vertices, the algorithm we present runs in $O(\log n)$ time using $O(n \log n + k)$ processors, where k is the size of the output (the total description complexity for the feasible domains of the GOPs to be solved), and it could be $\Omega(n^2)$ in the worst case. If the traditional CREW PRAM model is used, our solution would require $O(n^2)$ processors.

As outlined in the previous section, to compute the feasible domains for the GOPs it suffices to compute the cells in the arrangement $A(D_{st})$ of $O(n)$ line segments in D_{st}, where each line segment has its endpoints on the boundary of D_{st}. Further, in order to produce the corresponding objective functions, with each cell C of $A(D_{st})$ we must associate the subset of line segments in S that are intersected by a line whose dual is a point in C. This computation may be regarded as a set of queries on the line segments in S.

The algorithm we present follows the one in [13], where the following segment intersection problem has been considered and solved: given a set of line segments in the plane, construct a data structure that allows to quickly report the segments intersected by a query line. Their algorithm is based on a parallel persistence data structure termed *array-of-trees* and on fast construction of line arrangements. The main idea in [13] is to build the arrangement, an operation sequence σ for

that arrangement, and then use the array-of-trees data structure to evaluate the sequence. A reporting query can then be answered in $O(\log n)$ time per query, resulting in a $O(\log n)$ time, $O(n^2)$ processors CREW PRAM algorithm.

The main difference in the algorithm we present is in defining and handling the operation sequence σ. Given the nature of the optimal link problem, a vertex of the subdivision R may in fact be the endpoint of multiple line segments (e.g., $O(n)$ such segments). Then, while crossing from one cell to an adjacent one, many line segments may enter or leave the set S_{st} and thus many *enable/disable*-like operations in [13] would be associated to such crossing. Rather than defining the enable/disable operations on individual segments, we define these operations on subsets of segments in S. Doing this, in order to maintain the processing bounds, we must be able to obtain these subsets in constant time per subset. Fortunately, this can be done by extending the data structures introduced in [9, 5] for the optimal penetration problem. We only mention here that, if not given as part of the input, the additional data structures can be easily computed in parallel in $O(\log n)$ time using $O(n)$ processors. Knowing the number $d(v)$ of edges adjacent to each vertex $v \in R$ and using these structures, we can assign $O(d(v))$ processors to handle an event at v in constant time. Observe that, since R is a planar subdivision, we have $\sum_{v \in R} d(v) = O(n)$.

Lemma 3. *The feasible domains and the objective functions for the GOPs associated with the region D_{st} can be generated in $O(\log n)$ time using $O(n \log n + k)$ processors, where k is the size of the output.*

Proof. We give an algorithm that constructs the GOPs in the claimed time and processor bounds. The algorithm proceeds as follows. (1) Construct the arrangement of line segments inside D_{st}. This can be done in $O(\log n)$ time with $O(n \log n + k)$ processors, using the algorithm in [12]. We then compute a spanning tree for this arrangement and an Euler tour of this tree, as in [13]. While computing the Euler tour, we use an extension of the data structures in [9, 5] to produce the operation sequence σ for the tour. Since the enable/disable operations in σ add only constant time, this computation can still be done in $O(\log n)$ time using $O(k/\log n)$ processors. Constructing the array-of-trees data structure and answering reporting queries can be done as in [13]. Then, the claimed processing bounds follow. □

We mention here that an $O(\log n)$ time, $O(n^2)$ processors algorithm can be obtained by associating an enable/disable operation with each line segment involved in a crossing at a node v (i.e., to $O(d(v))$ segments) and applying the algorithm in [13].

The second algorithm we present uses a coarse-grain parallel computer model of computation. In this model, a relatively small number of processors are available and each processor has a large amount of local memory available, thus being able to store all data involved in (sequentially) solving the problem, much like a personal computer. In particular, such a processing element would be able to store the region R and its dual arrangement, as well as all data that is required in the process of generating and solving a GOP. If at most n processors are available, we present a simple yet efficient algorithm that generates all GOPs using

$O(n^2 \log n)$ work and with practically no communications between processors. The GOPs can be solved locally or they can be sent for solving to some external processing clusters, as in [10].

We make the following assumptions for our model: (1) processors are connected and can communicate via a global data buss or a communication network that allows efficient data broadcasting (i.e, feed the subdivision R to all processing elements) and (2) processors are numbered and each processor knows its order number.

The algorithm we present is based on computing the portion of an arrangement of lines that lies in between two vertical lines. At the start of the algorithm, each processing element stores the subdivision R and the set of lines in $A(R)$ (following a broadcasting operation), and knows its order number. Since each processor will perform similar computation, it then suffices to discuss the computation involved at only one of them, say the k-th processor P_k.

At processor P_k, the algorithm will compute the GOPs associated with the portion of the arrangement $A(R)$ that is in between the vertical lines L_{k-1} and L_k passing through the $(k-1)n$-th and kn-th leftmost intersection points of the lines in $A(R)$. We denote these two points as p_{k-1} and p_k. First, the algorithm finds the lines L_{k-1} and L_k by computing the points p_{k-1} and p_k. These points can be computed in $O(n \log n)$ time each using the algorithm in [8]. Next, the algorithm computes the intersection points of the lines in $A(R)$ with L_{k-1} and L_k and runs a topological sweep algorithm [2] to produce the GOPs inside the parallel strip. Sweeping the strip, as well as generating the corresponding objective functions, can be done altogether in $O(n \log n)$ time, which follows from [9,5]. Alternatively, we can obtain the same results using the (optimal) sequential version of the CREW PRAM algorithm above (i.e., by computing a line segment arrangement inside the strip and traversing that arrangement). Finally, the last step of the algorithm consists of a maximum selection among the optimal solutions stored "locally" at different processing elements, in order to obtain the optimum over all GOPs. These can be done using $O(n)$ broadcasting operations, starting at processor P_1, with the overall optimum computed at processor P_n. Thus, we have the following lemma.

Lemma 4. *In the proposed coarse-grain computing model, the feasible domains and the objective functions for the GOPs can be computed in $O(n \log n)$ time using $O(n)$ processors.*

Corollary 1. *If only p processors are available, where $p \leq n$, the feasible domains and the objective functions for the GOPs can be computed with $O(n^2 \log n)$ total work.*

There are two important features of our solution that should be noted here. First, the approach we propose allows for scalability in solving the GOPs. That is, after a GOP is produced, it can be solved either locally or it can be sent to some external processing cluster, that would in turn compute and return the optimal value for that GOP. Second, once the initial setup for the computation has been completed, it takes constant time to generate a new GOP; since the

objective function of a GOP could have $O(n)$ terms, this implies that all GOPs in a strip can be generated in time comparable to that required to perform a single evaluation of a GOP's objective function, and justifies the proposed coarse-grain model of computation.

In the full paper, we will show that the algorithm above can be extended to compute only the GOPs corresponding to the portion of the arrangement $A(R)$ that lies inside the region D_{st}, with each processing element solving about the same number of GOPs. However, we expect such an approach to be slower in practice when compared to the algorithm above, due to the increased complexities of the data structures involved, which may considerably add to the values of the constants hidden in the big-Oh notations.

References

1. L. Aleksandrov, M. Lanthier, A. Maheshwari, and J.-R. Sack, "An ϵ-approximation algorithm for weighted shortest paths on polyhedral surfaces," *Proc. of the 6th Scandinavian Workshop on Algorithm Theory*, pp. 11-22, 1998.
2. T. Asano, L.J. Guibas and T. Tokuyama, "Walking in an arrangement topologically," *Int. Journal of Computational Geometry and Applications*, Vol. 4, pp. 123-151, 1994.
3. A. Brahme, "Optimization of radiation therapy," *Int. Jouurnal of Radiat. Oncol. Biol. Phys.*, Vol. 28, pp. 785-787, 1994.
4. B. Chazelle and H. Edelsbrunner, "An optimal algorithm for intersecting line segments in the plane," *Journal of ACM*, Vol. 39, pp. 1-54, 1992.
5. D.Z. Chen, O. Daescu, X. Hu, X. Wu and J. Xu, "Determining an optimal penetration among weighted regions in two and three dimensions," *Proceedings of the 15th ACM Symposium on Computational Geometry*, pp. 322-331, 1999.
6. D.Z. Chen, O. Daescu, Y. Dai, N. Katoh, X. Wu and J. Xu, "Optimizing the sum of linear fractional functions and applications," *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms*, pp. 707-716, 2000.
7. D.Z. Chen, X. Hu and J. Xu, "Optimal Beam Penetration in Two and Three Dimensions," *Proceedings of the 11th Annual International Symposium on Algorithms And Computation*, pp. 491-502, 2000.
8. R. Cole, J. Salowe, W. Steiger and E. Szemeredi, "Optimal Slope Selection," *SIAm Journal of Computing*, Vol. 18, pp. 792-810, 1989.
9. O. Daescu, "On Geometric Optimization Problems", PhD Thesis, May 2000.
10. O. Daescu, "Optimal Link Problem on PIMs", Manuscript, January 2001.
11. H. Edelsbrunner, and L.J. Guibas, "Topologically sweeping an arrangement," *Journal of Computer and System Sciences* Vol. 38, pp. 165-194, 1989.
12. M Goodrich, "Intersecting Line Segments in Parallel with an Output-Sensitive Number of Processors," *SIAM Journal on Computing*, Vol. 20, pp. 737-755, 1991.
13. M Goodrich, M.R. Ghouse and J. Bright, "Sweep methods for Parallel Computational Geometry," *Algorithmica*, Vol. 15, pp. 126-153, 1996.
14. A. Gustafsson, B.K. Lind and A. Brahme, "A generalized pencil beam algorithm for optimization of radiation therapy," *Med. Phys.*, Vol. 21, pp. 343-356, 1994.
15. M. Lanthier, A. Maheshwari, and J.-R. Sack, "Approximating weighted shortest paths on polyhedral surfaces," *Proc. of the 13th ACM Symp. on Comp. Geometry*, pp. 274-283, 1997.

16. C. Mata, and J.S.B. Mitchell, "A new algorithm for computing shortest paths in weighted planar subdivisions," *Proc. of the 13th ACM Symp. on Comp. Geometry*, pp. 264-273, 1997.
17. J.S.B. Mitchell and C.H. Papadimitriou, "The weighted region problem: Finding shortest paths through a weighted planar subdivision," *Journal of the ACM*, Vol. 38, pp. 18-73, 1991.
18. A. Schweikard, J.R. Adler and J.C. Latombe, "Motion planning in stereotaxic radiosurgery," *IEEE Trans. on Robotics and Automation*, Vol. 9, pp. 764-774, 1993.
19. J. Snoeyink and J. Hershberger, "Sweeping Arrangements of Curves," *DIMACS Series in Discrete Mathematics*, Vol. 6, pp. 309-349, 1991.