線形分数関数最適化とその応用

Danny Z. Chen[1], Ovidiu Daescu[1], 戴 陽[2], 加藤 直樹[3], Xiaodong Wu[1], Jinhui Xu[1]

[1]Department of Computer Science and Engineering, University of Notre Dame
[2] 東京工業大学情報理工学研究科数理・計算科学専攻
[3] 京都大学工学研究科建築学専攻

n 個の線形制約の下で d 変数の m 個の線形分数関数の和を最大化する問題を考察する。とくに $d = 2$ に対して従来のアルゴリズムを改良する。その改良の鍵となる部分問題は動的に変化する制約条件下のパラメトリック最適化問題であり，これを $O((m+n)\log(m+n))$ 時間で効率的に解くアルゴリズムを開発した。計算実験の結果，従来のアルゴリズムに比べて計算時間がかなり改良されたことを検証した。

Optimizing the Sum of Linear Fractional Functions and Applications

Danny Z. Chen[1], Ovidiu Daescu[1], Yang Dai[2], Naoki Katoh[3], Xiaodong Wu[1], Jinhui Xu[1]

[1] Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA
[2]Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 O-okayama, Meguro-ku, Tokyo 152, Japan
[3] Department of Architecture and Architectural Systems, Kyoto University Kyoto, 606-8501 Japan.

The problem of optimizing the sum of m linear fractional functions (SOLF) in a fixed dimension d, subject to n linear constraints, arises in a number of theoretical and applied areas. This paper presents an improved algorithm for solving the SOLF problem in 2-D. A key subproblem to our solution is the *off-line ratio query* (OLRQ) problem, which computes the optimal values of a sequence of m linear fractional functions (called ratios), with the ratios subject to a dynamically changing feasible domain defined by $O(n)$ linear constraints. Based on useful geometric properties and the parametric linear programming technique, we develop an algorithm that solves the 2-D OLRQ problem in $O((m+n)\log(m+n))$ time. Our OLRQ algorithm can be easily implemented and is robust. More importantly, it enables us to speed up every iteration of a known iterative SOLF algorithm in 2-D, from $O(m(m+n))$ time to $O((m+n)\log(m+n))$. Implementation results of our improved SOLF algorithm have shown that in most cases our algorithm outperforms the commonly-used approaches for the SOLF problem.

1 Introduction

The problem of optimizing the sum of linear fractional functions (SOLF) is defined as follows:

$$\max_{(x_1,\ldots,x_d)\in S} f(x_1,\ldots,x_d) = \sum_{i=1}^{m} \frac{n_i(x_1,\ldots,x_d)}{d_i(x_1,\ldots,x_d)}$$

such that for each $i = 1,2,\ldots,m$, $n_i(x_1,\ldots,x_d)$ and $d_i(x_1,\ldots,x_d)$ are linear functions in a fixed d-D space R^d, $d_i(x_1,\ldots,x_d) \neq 0$ for any $(x_1,\ldots,x_d) \in S$, the feasible domain S is defined by n linear constraints (i.e., half-spaces in R^d), and $S \neq \phi$. Each linear fractional term $\frac{n_i(x_1,\ldots,x_d)}{d_i(x_1,\ldots,x_d)}$ is called a *ratio*.

The SOLF problem appears in the algorithmic solutions for several geometric optimization problems. In layered manufacturing, Majhi *et al.* [14] showed that the length-optimal supports for a simple non-convex polygon can be obtained by solving the 1-D SOLF problem on an interval. Arkin *et al.* [1] formulated the problem of finding a minimum-area star-shaped or monotone polygon that contains a simple polygon, which appears in material layout [1], [6], [16] (e.g., cloth manufacturing) and manufacturing [14], as one of optimizing the sum of 2-D fractional polynomials of degrees 3 and 2 under linear constraints. We will show that the objective function of this problem can actually be simplified to the 2-D SOLF form. For the problem of finding optimal penetrations among weighted regions [3], we shall show that the L_1 and L_∞ versions of this problem have the 2-D SOLF problem as a key subproblem. The SOLF problem also arises in other areas. For example, in combinatorial optimization, a class of problems on finding a structure with the maximum (or minimum) mean-weight cost can be solved by optimizing some linear fractional functions [18]. In operations research, it was shown in [8], [19] that many economic applications (e.g., maximization of productivity, return on investment, and return/risk) can be reduced to solving the SOLF problem.

Quite a few solutions have been given for the SOLF problem [1], [8], [12], [14], [20]. It is interesting to note that the SOLF related problems were originally studied in economic applications, where the number of variables is usually much bigger than the number of fractional terms. As a result, many previous SOLF algorithms were designed to target problems with only a few fractional terms (less than 10) for a reasonable running time [8], [12], [20]. On the other hand, there are some general-purpose heuristic packages which can generate local optimal solutions [10], [11], [15]. However, they usually run in a long time yet obtain solutions without quality guarantee.

In geometric applications, however, the dimension of the SOLF problem is often low (e.g., 2 or 3), while the number of terms in the objective function can be quite big (hundreds or even thousands) [1], [3], [14]. In consequence, most previous approaches have difficulties in dealing with SOLF problems of this nature. A commonly-used approach by known geometric algorithms is to reduce the SOLF problem to computing all real roots of a system of high degree polynomial equations [1], [14]. For an objective function in d-D (i.e., having d variables) consisting of m linear fractional terms, there can be $O(m^d)$ roots. The time for computing all the roots essentially depends on the conditioning of these roots. To compute an ill-conditioned root, substantially long precision is needed in order to attain a certain degree of accuracy, thus increasing the computation time significantly. Clearly, the situation worsens as the number of ill-conditioned roots increases. Another drawback of the root-finding approach is the cost and possible numerical errors of the construction of the polynomial system from the SOLF problem. For example, even in 2-D, each coefficient in the polynomial system is computed from $O(m)$ coefficients of the SOLF problem (by performing $O(m^2)$ multiplication and addition operations). Hence, it is likely that numerical errors are introduced to the coefficients of the polynomial system, making the solution for the SOLF problem incorrect. Further, the overhead of $O(m^3)$ time for creating the polynomial system can be quite significant for a large m.

Falk and Palocsay [8] gave an interesting iterative algorithm (we call it the FP algorithm) for solving the general SOLF problem in any fixed dimension. In this paper, we present an improved solution in 2-D. By exploiting useful geometric properties and using the parametric linear programming technique, we are able to speed up the 2-D FP algorithm considerably.

A key subproblem to our algorithm is the following 2-D **off-line ratio query** (OLRQ) problem:

Given M linear ratios $r_1, r_2, \ldots, r_M$ and a set C of N linear constraints in 2-D, such that $N \geq M$ and the feasible domain defined by the constraints in C is nonempty, find the maximum value of each ratio r_i subject to the constraints in $C - C_i$, where $C_i \subset C$, $|C_i|$ is a constant, the denominator of each r_i is non-zero on each of its feasible points, and for any $i, j \in \{1, 2, \ldots, M\}$, $C_i \cap C_j = \phi$ if $i \neq j$.

It turns out that the OLRQ problem constitutes a key step to the iterative FP algorithm [8]. We treat the computation on each ratio r_i as a *query*, and maintain a data structure for processing such queries (on the dynamically changing constraints).

In [8], the computation on one ratio is done independently of another; further, such a computation in 2-D is transformed to a 3-D linear programming (LP) problem. Thus, solving the 2-D OLRQ problem in the fashion of [8] would take $O(MN)$ time (by using Megiddo's 3-D LP algorithm [?]). By combining the parametric linear programming technique [19] with some interesting geometric observations, we develop a rather simple $O((M+N)\log N)$ time algorithm for the 2-D OLRQ problem. Our algorithm is robust, as ensured by a robustness theorem.

By making use of our OLRQ algorithm and by other modifications, we reduce the time bound of each iteration of the iterative FP algorithm in 2-D [8] from $O(m(m+n))$ to $O((m+n)\log(m+n))$. Our preliminary implementation results on the improved SOLF algorithm show that, in most cases, our approach outperforms the root-finding approaches and some commonly-used global optimization methods. Furthermore, since our approach directly computes one optimal value instead of finding all the real roots, in some degree it may avoid the problem of handling many ill-conditioned roots, and it appears to use less space than the usual root-finding approaches.

As applications, we show that the problem of computing a minimum-area star-shaped (resp., monotone) polygon to cover a simple polygon [1], [6], [16] (called the star-cover or monotone-cover problem) and the problem of determining an optimal penetration among weighted 2-D regions [3] (called the penetration problem) under the L_1 and L_∞ metrics can all be reduced to solving a set of $O(n^2)$ (resp., $O(n)$) instances of the 2-D SOLF problem. Each such problem instance involves a sum of $O(n)$ linear fractional functions over a convex domain, and thus is solvable by our 2-D SOLF algorithm. By using topological peeling [4] to traverse an arrangement of $O(n)$ lines, the $O(n^2)$ SOLF instances can be generated one by one in $O(n^2)$ time and $O(n + S_I(n))$ space, where $S_I(n)$ is the maximum space used by the SOLF algorithm on each problem instance.

2 The Off-Line Ratio Query Problem.

In this section, we show how to solve the 2-D off-line ratio query (OLRQ) problem in $O((M+N)\log N)$ time. Our algorithm, based on the parametric linear programming technique, is actually quite simple. We will first sketch our ideas for handling a single ratio query over a convex polygonal domain by parametric linear programming, and then describe a data structure for efficiently processing the sequence of M ratio queries.

2.1 Parametric Linear Programming over a Convex Polygonal Domain.

We first consider the following problem.

Problem 1 *Maximize* $r_i(x,y) = \frac{n_i(x,y)}{d_i(x,y)} = \frac{ax+by+c}{dx+ey+f}$, *subject to* $(x,y) \in S$ *in 2-D, where* S *is defined by* N *linear constraints,* $S \neq \phi$, *and* $d_i(x,y) \neq 0$ *for any point* $(x,y) \in S$.

For this problem, we further assume that the common intersection S of the N half-planes (specified by the N linear constraints) is already computed and is represented as an $O(N)$-vertex convex polygon (possibly unbounded with some vertices at infinity). We also assume that a simple $O(N)$-time preprocessing has been done on S to associate certain information with the vertices of S (this will be discussed later).

Since the domain S is convex, the maximum value of $r_i(x,y)$ is actually attained at a vertex of S, as shown in [19]. This immediately implies a brute-force method: First evaluate the value of $r_i(x,y)$ at every vertex of S, and then take the maximum among the resulted $O(N)$ candidate values. This method takes $O(N)$ time once S is available, but using it to solve the OLRQ problem would yield an expensive $O(MN)$ time algorithm. To obtain a better solution, we consider the parametric version of the objective function $r_i(x,y) = \frac{n_i(x,y)}{d_i(x,y)} = \frac{ax+by+c}{dx+ey+f}$ on S, i.e.,

$$f_i(\lambda) = \max_{(x,y)\in S}\{n_i(x,y) - \lambda d_i(x,y)\},$$

The function $f_i(\lambda)$ is a parametric linear function. It is clear that for any given value λ', the value of $f_i(\lambda')$ is the solution to an LP problem $P(\lambda')$ on S whose objective function is $n_i(x,y) - \lambda' d_i(x,y)$.

It is known [19] that maximizing $r_i(x,y)$ on S is equivalent to computing the root λ^* of $f_i(\lambda) = 0$ (i.e., the maximum value of $r_i(x,y)$ occurs at λ^*). To compute λ^* efficiently, we make use of the fact that the curve of $f_i(\lambda)$ is monotone and piecewise linear [18]; moreover, for all values of λ corresponding to each linear piece of $f_i(\lambda)$, $f_i(\lambda)$ actually attains its values at the same vertex of S, say v. The union of all λ values (at vertex v) for a linear piece of $f_i(\lambda)$ forms an interval on the λ-axis, denoted by v_λ. Also, note that the slope of the parametric line $n_i(x,y) - \lambda d_i(x,y)$ changes monotonically with respect to λ.

For a given λ, the slope s_λ of the objective function $n_i(x,y) - \lambda d_i(x,y)$ of the LP problem $P(\lambda)$ determines $O(1)$ vertices on the boundary of S such that each of them has a tangent line with slope s_λ. Obviously, one of them is the optimal point for $P(\lambda)$, i.e., the vertex of S at which $f_i(\lambda)$ attains its value. The slopes of all tangent lines at any vertex v of S clearly form an interval, which we denote by I_v (I_v may contain $+\infty$ or $-\infty$). Hence, to decide whether a vertex v of S is one of the $O(1)$ candidates for the optimal point of $P(\lambda)$ (i.e., at such a candidate vertex $f_i(\lambda)$ is evaluated for its value), one only needs to check whether I_v contains s_λ. Due to the monotonicity, every slope interval I_v corresponds to one λ interval v_λ.

Lemma 2.1 *Let* v_{upper} *and* v_{lower} *be two vertices on the upper and lower boundaries of* S *respectively, such that* $I_{v_{upper}}$ *and* $I_{v_{lower}}$ *both contain the slope* $\frac{-d}{e}$. *Then the values in* v_λ *change monotonically while visiting the vertices* v *by walking counterclockwise along the boundary of* S *either from* v_{upper} *to* v_{lower} *or from* v_{lower} *to* v_{upper}.

Lemma 2.1 and the properties of $f_i(\lambda)$ imply that we can perform a binary search on the vertices of S to find out at which vertex the root λ^* of $f_i(\lambda)$ lies. Suppose the search is visiting a vertex v for a value λ (i.e., v_λ contains λ). Without loss of generality (WLOG), we assume that v is on the upper boundary of S. To compute $f_i(\lambda)$ correctly, we need to find the vertex v' on the lower boundary of S, such that v'_λ also contains λ. To be able to locate v' quickly, we assume that a preprocessing on the vertices of S has already associated with v the list of all of its antipodal vertices [17], in the counterclockwise order. Actually, to represent the antipodal vertex list of v, it is sufficient to store the starting and ending vertices $lapd(v)$ and $rapd(v)$ of the list (called *extreme antipodal vertices*). Given S, this preprocessing can be easily done in $O(N)$ time [17]. With the antipodal vertex information already available for each v, the algorithm below computes the root λ^* of $f_i(\lambda)$ in $O(\log N)$ time. Let the vertices of S be ordered counterclockwise along its boundary and $next(v)$ and $pred(v)$ denote the successor and predecessor of a vertex v in this order.

1. Let v_{first} be v_{upper} and v_{last} be v_{lower} (assume v_{upper} and v_{lower} have been identified in the preprocessing).
2. Compute the slope s_f of the line through both v_{first} and $next(v_{first})$ and the slope s_l of the line through both v_{last} and $pred(v_{last})$. Determine the values λ_f and λ_l for s_f and s_l.
3. Compute $f_i(\lambda_f)$ and $f_i(\lambda_l)$. If they both have the same sign, then evaluate $r_i(x, y)$ at v_{upper} and v_{lower} and take the maximum as the solution.
4. Otherwise, compute $f_i(\lambda_{ml})$ and $f_i(\lambda_{mr})$, where λ_{ml} and λ_{mr} are the two extreme λ values associated with the middle vertex v_{mid} of the list of vertices from v_{first} to v_{last}. Determine which half of the vertex list contains the root, and recursively search on the appropriate half.
5. If only one vertex is left in the list, say v, then evaluate $r_i(x, y)$ at v, and also perform a binary search on the vertices from $lapd(v)$ to $rapd(v)$, in order to determine the root.

Lemma 2.2 *The above algorithm correctly computes the root λ^* of $f_i(\lambda)$ in $O(\log N)$ time provided that the operations of finding the median, successor, predecessor, and extreme antipodal vertices can be performed in constant time each, where N is the number of vertices of S.*

2.2 Sequence of Off-Line Ratio Queries.

We now show a sequence of M 2-D ratio queries $r_1, r_2, \ldots, r_M$ (r_i on constraints in $C - C_i$) can be efficiently processed by means of a data structure. Although the M queries are on different domains, these domains differ from each other at only $O(1)$ constraints. Hence, it is possible to maintain an efficient data structure for these changing domains (as well as other useful information such as the antipodal vertex list of each vertex of every convex polygonal domain) in an off-line fashion.

Our OLRQ algorithm is based on the structure of *layers of common intersections of half-planes* in C, which is defined as follows:

1. Let CL_1 be the common intersection of all half-planes in C, and let it be the common intersection of layer 1. (Note that $CL_1 \neq \phi$ since $CL_1 = S$.)
2. Remove from C all the half-planes whose boundaries contain a boundary edge of CL_1.
3. Repeat steps 1 and 2 to define the common intersections of layers 2, 3, ..., until $C = \phi$.

Figure 1 gives an example for the layers of common intersections of C. Note that these layers of common intersections of the half-planes are closely related, through a geometric duality [17], to Chazelle's convex layers of a planar point set [2].

Let $k = 1 + \max_{1 \leq i \leq M}\{|C_i|\}$. It is easy to observe that each of the M domains for the ratio queries contains the common intersection $S = CL_1$ of all half-planes of C; further, all these domains are contained inside the k-th common intersection layer (if it exists). Based on this observation, we only need to make use of the first k (convex) common intersection layers $CL_1, CL_2, \ldots, CL_k$ of the N linear constraints of C. These k convex layers can all be computed in $O(N \log N)$ time by using Chazelle's algorithm [2], with each convex layer being stored in an array. Once this k-layer structure is available, the M domains can be constructed easily, by removing certain specified constraints.

More precisely, for a query r_i, if there is no half-plane hp (i.e., constraint) in C_i whose boundary line $l(hp)$ contains an edge of the boundary $bd(CL_1)$ of CL_1, then the domain of r_i is CL_1. Otherwise, assume the boundary line $l(hp)$ of a half-plane $hp \in C_i$ contains an edge of $bd(CL_1)$. To obtain the domain of r_i defined by $C - C_i$, we need to "expand" CL_1 onto CL_2 by removing $l(hp)$ from $bd(CL_1)$. The removal of $l(hp)$ causes CL_1 to expand along the two neighboring lines, l_{left} and l_{right}, of $l(hp)$ on $bd(CL_1)$. That is, a convex polygon Q which is bounded by $l(hp)$, l_{left}, l_{right}, and possibly $bd(CL_2)$ and which is to the right of $l(hp) \cap bd(CL_1)$ (as we walk along $bd(CL_1)$ counterclockwise) is attached to CL_1. Let l_{left} and l_{right} intersect $bd(CL_2)$ at two points p_l and p_r, respectively. The expanded convex polygon Q is called the *cap* of hp, denoted by $cap(hp)$. Note that l_{left} and l_{right} may intersect each other at a point p before they touch $bd(CL_2)$, and if this is the case, $cap(hp)$ is the triangle $pp_{left}p_{right}$, where p_{left} (resp., p_{right}) is the intersection of $l(hp)$ and l_{left} (resp., l_{right}). If a boundary edge of $D_i = cap(hp) \cup CL_1$ lies on $l(hp')$ of another half-plane $hp' \in C_i$ (e.g., the half-plane bounded by $\overline{jg}$ in Figure 1), then we similarly expand D_i onto CL_2 or CL_3, and so on.

It is not hard to see that each expansion (caused by the removal of a half-plane in C_i) can be done in $O(\log N)$ time, since it performs $O(1)$ binary searches on one of the k convex layers (for computing its intersections with two lines). Thus for each query r_i, its $O(k)$ caps can all be obtained in $O(\log N)$ time.

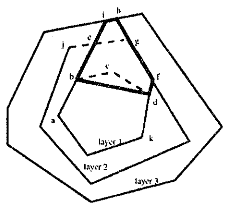

Figure 1: The expansion of a cap $bdfhi$. The dashed lines represent the removed constraints.

After all the caps of r_i are found, we need to compute the antipodal vertices for the vertices on the boundary of the union of these caps and CL_1. The information on such antipodal vertices is needed for handling the query r_i, as shown in Subsection 2.1. We can show that it is possible to compute the antipodal vertices for all the M ratio queries in altogether $O(M + N)$ time.

Theorem 2.3 *The OLRQ problem can be solved in $O((M + N)\log N)$ time and $O(M + N)$ space.*

3 Robustness Theorem for Optimizing Fractional Functions.

In this section, we first prove a robustness theorem for optimizing a fractional function in any fixed dimension d, and then use this theorem to ensure the robustness of our OLRQ algorithm.

Let P be the primal problem of optimizing a fractional function:

Problem 2 *Maximize $r(x) = \frac{h(x)}{g(x)+b}$, subject to $x \in S \subseteq R^d$, where $b \in R$ and S is the feasible domain of x.*

We also assume that $g(x) \geq 0$ for all $x \in S$ and $b > 0$. The essence of these conditions on $g(x) + b$ is to ensure that $g(x)$ does not get arbitrarily close to zero in S, i.e., b is a non-zero constant lower bound on how close the denominator can get to zero. Hence, it will be fine if we have $g(x) \leq 0$ for all $x \in S$ and $b < 0$ (in this case, the conditions of problem P are satisfied since we can simply change the signs for both $h(x)$ and $g(x)+b$). Associated with problem P, we consider its parametric version $Q(\lambda)$ with a linear parameter λ:

Problem 3 *Maximize $h(x) - \lambda g(x)$, subject to $x \in S \subseteq R^d$, where $\lambda \in R$.*

Let $x(\lambda)$ and $v(\lambda)$ be the optimal solution and the objective value of problem $Q(\lambda)$, respectively. Let λ^* satisfy $v(\lambda^*) = \lambda^* b$, i.e., λ^* is the optimal objective value of P. WLOG, assume that $\lambda^* > 0$. Notice that $v(\lambda)$ is nonincreasing in λ due to $g(x) \geq 0$. We shall show the following theorem.

Theorem 3.1 *Let x' be a feasible solution of P such that*

$$v(\lambda') - \delta \leq h(x') - \lambda' g(x') \leq v(\lambda') + \delta, \qquad (1)$$

$$-\delta \leq h(x') - \lambda'(g(x') + b) \leq \delta. \qquad (2)$$

Then x' is an ϵ-approximate solution of problem P, where $\delta = \frac{\epsilon b \lambda^*}{2}$.

Intuitively, the theorem says that if we have an approximate solution x' of $Q(\lambda')$ such that the objective value of $Q(\lambda')$ is almost zero, then x' is an approximate solution of P. This implies a useful robustness property of the linear fractional problem we consider.

Note that, in order to apply Theorem 3.1, we need to know a positive lower bound of the value of λ^* because ϵ is related to the value of λ^*.

Lemma 3.2 *Let $r(x) = \frac{n(x)}{d(x)}$, where $x \in S \subseteq R^d$, be a linear fractional function such that $d(x) \neq 0$ for any $x \in S$. Then $r(x)$ can be represented as a function of the form $\frac{h(x)}{g(x)+b}$, such that $b > 0$ and $g(x) \geq 0$ for any $x \in S$.*

Theorem 3.1 and Lemma 3.2 can be used to improve the robustness of our 2-D OLRQ algorithm in Section 2. Note that for each ratio $r_i(x, y) = \frac{n_i(x,y)}{d_i(x,y)}$ in our OLRQ problem, $i = 1, 2, \ldots, M$, $d_i(x, y) \neq 0$ for any point (x, y) in the feasible domain S. WLOG, suppose $d_i(x, y) > 0$ and $\lambda_i^* > 0$ over S. Then the minimum value b_i of $d_i(x, y)$ and a positive lower bound λ_i^l of λ_i^* over S for the ratio query $r_i(x, y)$ can be easily obtained in $O(\log |S|)$ time once the feasible domain S for $r_i(x, y)$ is available ($\lambda_i^l = r_i(x^l, y^l)$ such that at the point (x^l, y^l), $n_i(x^l, y^l) = \max_{(x,y)\in S} n_i(x, y)$). Our OLRQ algorithm uses a binary search on the boundary of a convex polygonal domain to determine the root of $f_i(\lambda)$. However, when the search examines a λ value such that $f_i(\lambda)$ is very close to 0, numerical errors could change the sign of $f_i(\lambda)$ and thus misguide the course of binary search. By Theorem 3.1, we know that when $f_i(\lambda)$ is close to 0, λ is close to λ^*. Thus, we can terminate the binary search process and simply use the corresponding boundary vertex as an ϵ-approximate optimal solution.

4 Speeding up the FP Algorithm for the 2-D SOLF Problem.

In this section, we show how the time bound of each iteration of the FP iterative algorithm for the 2-D SOLF problem [8] can be sped up from $O(m(m+n))$ to $O((m+n)\log(m+n))$.

We begin with an outline of the general FP iterative algorithm [8]. Let $\overline{x} = (x_1, x_2, \ldots, x_d)^T \in S \subseteq R^d$. The FP iterative algorithm first transforms the original SOLF problem in d-D (called the X-space) to another problem in m-D (called the R-space), by mapping each ratio $r_i(\overline{x}) = \frac{n_i(\overline{x})}{d_i(\overline{x})}$ in the objective function $f(\overline{x})$ to a corresponding dimension r_i in the new m-D space (i.e., $r_i = r_i(\overline{x})$ is a variable in the R-space). The SOLF problem thus becomes one of optimizing $F(\overline{r}) = \sum_{i=1}^{m} r_i$ in the R-space. It then computes an upper bound u_i for each dimension r_i and a feasible lower bound f_l for $F(\overline{r})$. These u_i, $i = 1, 2, \ldots, m$, and f_l together define a simplex in the R-space containing at least one optimal point. The algorithm iteratively cuts the simplex by reducing the upper bounds u_i, and an optimal solution is obtained when the upper bounds match f_l. When the algorithm cannot lower the upper bounds any further (this is called a *stall* state), it either increases the lower bound f_l to change the domain, or splits the simplex into two and searches on each of them.

The key step in each iteration of the FP algorithm is to compute the upper bound u_i for each ratio r_i, by transforming it to solving a $(d+1)$-D LP problem. Using Megiddo's LP algorithm for any fixed dimension d [?], all m upper bounds can be obtained in $O(mn)$ time. Other steps of the iteration (for updating and representing the new simplex resulted from cutting) take $O(m^2)$ time.

To speed up the FP algorithm, we formulate the operations for computing the m upper-bounds in each iteration as a sequence of m ratio queries, and reduce it to an OLRQ problem with parameters $k = 2$, $M = m$, and $N = |C| = m+n$. The total time for computing the m upper bounds in each iteration is thus reduced from $O(mn)$ to $O((m+n)\log(m+n))$. By using an implicit representation for storing the simplex (instead of an explicit one as in [8]), we are able to carry out the rest of the computation in each iteration in $O(m)$ time. Therefore, the time bound of each iteration of the FP algorithm in 2-D is reduced from $O(m(m+n))$ to $O((m+n)\log(m+n))$.

5 Solving Geometric Optimization Problems as SOLF Problems.

In this section, we discuss two geometric optimization problems which can be reduced to 2-D SOLF problems (and thus are solvable by our improved SOLF algorithm).

The reductions for these problems are all hinged on a traversal of a 2-D arrangement of $O(n)$ lines [7] that defines the domains and objective functions of various instances of the SOLF problem. The arrangement traversal is based on a new technique called topological peeling [4].

5.1 Minimum Area Star Cover and Monotone Cover Problems.

Let P be an n-vertex simple polygon. The star-cover or monotone-cover problem on P is that of computing a star-shaped or monotone polygon P' such that P' contains P and the area of P' is minimized [1], [6], [14], [16]. This problem finds applications in material layout (e.g., cloth manufacturing) and manufacturing. Arkin *et al.* [1] showed that the star-cover (resp., monotone-cover) problem is reducible to solving $O(n^2)$ (resp., $O(n)$) problems of optimizing the sum of $O(n)$ fractional polynomials of degrees 3 and 2 under $O(n)$ linear constraints in 2-D. We are able to reduce the star-cover (resp., monotone-cover) problem to $O(n^2)$ (resp., $O(n)$) instances of the SOLF problem in 2-D. Our star-cover algorithm also improves the space bound of [1]. The details are left to the full paper.

Theorem 5.1 *In $O(n^2)$ (resp., $O(n\log n)$) time and $O(n)$ space, it is possible to reduce the minimum-area star-cover (resp., monotone-cover) problem to $O(n^2)$ (resp., $O(n)$) problem instances of optimizing the sum of $O(n)$ linear fractional functions subject to $O(n)$ linear constraints in 2-D.*

5.2 Optimal Penetration Problem.

The optimal penetration problem [3] is defined as follows. Given a subdivision R with a total of n vertices in 2-D, divided in m regions R_i, $i = 1, 2, \ldots, m$, find a ray L such that L originates from outside R and intersects a specified *target region* $T \in \{R_1, R_2, \ldots, R_m\}$, and such that the weighted sum $S(L) = \sum_{L\cap R_i \neq \phi} w_i * f_i(L)$ is minimized, where $f_i(L)$ is a function associated with the pair (R_i, L) and w_i is a weight factor associated with R_i. Such a ray L is called a *penetration*. The regions R_i are all simple polygons, and the weights of T and the complement $\overline{R}$ of R are zero ($\overline{R}$ is the free space outside R). This problem arises in several applied areas such as radiation therapy, geological exploration, and environmental engineering (see [3] for a discussion of the applications).

Let R_L be the set of regions of R intersected by a ray L and d_i be the length of L within $R_i \in R_L$. As shown in [3], if the length d_i is computed in the L_2 metric, then this problem can be reduced to solving $O(n^2)$ instances of an optimization problem in 2-D, each such problem with an objective function of the form $\sqrt{1+x^2}\sum_{i=1}^{m}\frac{a_i y+b_i}{x-x_i}$, where (x,y) is any point in a convex domain defined by linear constraints. If the L_1 and L_∞ metrics are used, then we can prove the following theorem (the details are left to the full paper).

Theorem 5.2 *In $O(n^2)$ time and $O(n)$ space, it is possible to reduce the optimal penetration problem under the L_1 and L_∞ metrics to $O(n^2)$ problem instances of optimizing the sum of $O(m)$ linear fractional functions subject to $O(n)$ linear constraints in 2-D.*

6 Implementations.

We have implemented the 1-D FP algorithm and the improved 2-D FP algorithm. Our implementations are based on LEDA, and all experiments ran on a SUN ULTRA 30 computer. Because of the space limit, we only give the results for 2-D case.

For our improved 2-D FP algorithm, we tested it using a set of randomly generated problem instances with different numbers of ratios (up to 1000) and different numbers of linear constraints (from 3 to 1000), and compared it with several optimization softwares, such as *Maple, CFSQP* [5], and *GENOCOP III.* So far, much of our efforts is on large-ratio problems. This is motivated by the observations that Maple has difficulties with handling problems with ratio numbers $\geq$ 24, and the local optimization package *CFSQP* fails to obtain good quality solutions for large-ratio problems.

Our experimental results are summarized in Figure 2, where all problems are of ratio numbers larger than 100, and the average execution time is taken over 15 runs on problems with the same numbers of ratios and linear constraints. Only one comparison is made, with *GENOCOP III*, for the 2-D algorithm (a good software package for solving systems of high degree polynomial equations is not yet available to us). Figure 3 shows that our algorithm is roughly 10 times faster than *GENOCOP III* for large-ratio problems.

The experiments seem to suggest that the average execution time of our algorithm is a slowly growing function of the number of ratios. It is also interesting to note that the execution time does not increase too much when more constraints are used. One of our explanations is that since our algorithm improves the time complexity of each iteration from $O(m(m+n))$ to $O((m+n)\log(m+n))$, the influence of an increasing n is not significantly amplified by m. The experiments also show that for most problem instances, the number of iterations is much bigger than the number of R-space simplex splitting, implying that it makes sense to reduce the time complexity of each iteration.

We think the experiments on our 2-D cases are not quite thorough and the result given in Figure 2 is only a preliminary version. More data and a full analysis will be given in the full paper.

Acknowledgements

The authors would like to thank Prof. Leonid Faybusovich for discussing optimization techniques with us and for providing references, Prof. John M. McNamee and Prof. Dario A. Bini for sharing their knowledge on methods and software for finding roots of polynomials, and Shuang Luan for assisting us in preparing some of the experimental data.

References

[1] E. M. Arkin, Y.-J. Chiang, M. Held, J. S. B. Mitchell, V. Sacristan, S. S. Skiena, and T.-C. Yang, *On Minimum-Area Hulls*, Algorithmica, 21:119-136, 1998.

[2] B. Chazelle, *On the Convex Layers of a Planar Set*, IEEE Transactions on Information Theory, 31(4):509-517, 1985.

[3] D.Z. Chen, O. Daescu, X. Hu, X. Wu, and J. Xu, *Determining an Optimal Penetration among Weighted Regions in Two and Three Dimensions*, in Proc. of the 15th Annual ACM Symposium on Computational Geometry, pp. 322-331, 1999.

[4] D.Z. Chen and J. Xu, *Peeling an arrangement topologically*, manuscript, 1999.

[5] L. Craig, J.L. Zhou, and A.L. Tits, *User's Guide for CFSQP Version 2.5*, Electrical Eng. Dept. and Institute for Systems Research, University of Maryland, TR-94-16r1, 1994.

[6] K. Daniels, *The Restrict/Evaluate/Subdivide Paradigm for Translational Containment*, in Proc. Fifth MSI Stony Brook Workshop on Computational Geometry, 1995.

[7] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, New York, 1987.

[8] J. E. Falk and S. W. Palocsay, *Optimizing the Sum of Linear Fractional Functions*, Collection: Recent Advances in Global Optimization, C.A. Floudas and P. M. Pardalos (eds.), pp. 221-258, 1992.

[9] S. Hashizume, M. Fukushima, N. Katoh, and T. Ibaraki, *Approximation Scheme for Combinatorial Fractional Programming Problems*, Mathematical Programming, 37:255-267, 1987.

[10] L. Ingber, *Adaptive Simulated Annealing, Optimization Algorithm for Nonlinear and Stochastic Systems*, http://www.ingber.com/#ASA-CODE.

[11] M. Jelasity and J. Dombi, *GAS Genetic Algorithm*, ftp://ftp.jate.u-szeged.hu/pub/math/optimization/GAS/.

[12] H. Konno, T. Kuno, and Y. Yajima, *Global Minimization of a Generalized Convex Multiplicative Function*, Journal of Global Optimization, 4:47-62, 1994.

[13] G. Liotta, F. P. Preparata, and R. Tamassia, *Robust Proximity Queries: An Illustration of Degree-Driven Algorithm Design*, SIAM Journal on Computing, 28:864-889, 1998.

[14] J. Majhi, R. Janardan, M. Smid, and P. Gupta, *On Some Geometric Optimization Problems in Layered Manufacturing*, in Proc. 5th Workshop on Algorithms and Data Structures, pp. 136-149, Springer-Verlag, 1997.

[15] Z. Michalewicz, *GENOCOP III Genetic Algorithm for Constrained Problems*, http://www.coe.uncc.edu/~zbyszek/gcreadme.html

[16] V. Milenkovic, K. Daniels, and Z. Li, *Placement and Compaction of Nonconvex Polygons for Clothing Manufacture*, in Proc. 4th Canad. Conf. Comput. Geom., pp. 236-243, 1992.

[17] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

[18] T. Radzik, *Newton's Method for Fractional Combinatorial Optimization*, in Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science, pp. 659-669, 1992.

[19] S. Schaible, *Fractional Programming*, Handbook of Global Optimization, R. Horst and P. M. Pardalos (eds.), Kluwer Academic Publishers, pp. 495-608, 1995.

[20] H. Yamashita, *Efficient Algorithms for Minimizing the Sum and the Product of Linear Fractional Functions*, Master Thesis, Dept. of Industrial Engineering and Management, Tokyo Institute of Technology, 1997.

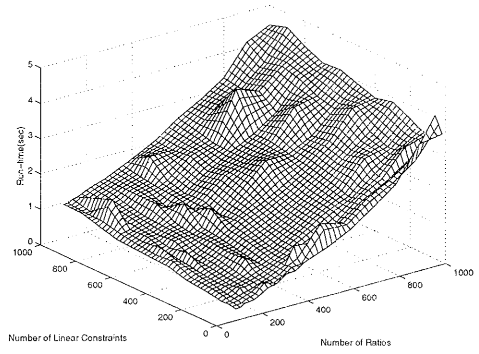

Figure 2: Preliminary result of the improved 2-D FP algorithm for problem instances with numbers of ratios $\geq$ 100.

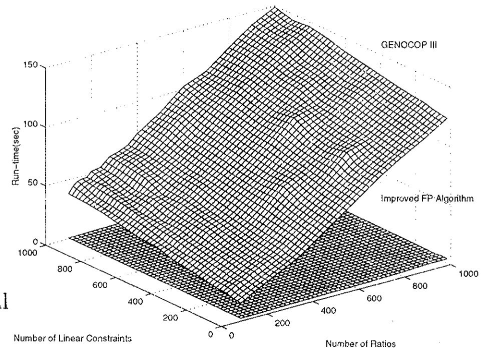

Figure 3: Comparison between *GENOCOP III* and the improved 2-D FP algorithm for problem instances with numbers of ratios $\geq$ 100.