Solving online feasibility problem in constant amortized time per update

Lilian Buzer*†

Abstract

We present a deterministic algorithm for solving the two and three-dimensional online feasibility problem. Insertion of a new constraint is processed in constant amortized time. Our method is adapted from the offline linear deterministic Megiddo algorithm for linear programming. As in his prune and search technique, our method and its time bound extend to higher dimensions but involve the same large constant. Our online version that processes a feasibility problem achieves an optimal time bound relative to previous methods [2, 6, 8, 9]. As an application, it is well suited for the problem of digital object recognition.

1 Introduction

Historically, working in the field of computational geometry, Megiddo gave the first deterministic algorithm for LP whose running time is linear in the number of constraints when the dimension is fixed [8, 9]. For dimension d, the time complexity of his method in $O(2^{2^d}.n)$ was next improved by Clarkson [3] and Dyer [4] to $O(3^{d^2}n)$. Eventually, Dyer and Frieze [5] proposed an approach that yields an $d^{O(d)}n$ time algorithm.

In recent years, no progress has been made on this front. Nevertheless new developments occurred in randomized and parallel algorithms with several simpler and more practical methods [7, 10]. A comprehensive survey of this field can be found in [1].

In the sequel, we work in a d-dimensional Euclidian space. A set of n given inequalities define a convex polytope $S \subset \mathbb{R}^d$ of *feasible solutions*. The *feasibility problem* consists in determining whether S is empty or not. When we specify a linear *objective function* $f(x) = c \cdot x$ and look for a point $x^* \in S$ where this function reaches its minimum, we solve a *linear programming problem*.

2 Motivation

When we process digital images, we often obtain a set of disconnected pixels corresponding to the contour of a shape. Most of the time, we approximate the given pixels by Euclidian geometric primitives that retain the global shape of the pixels by minimizing an approximation error. But, using digital primitives we can build an *exact* approximation whose drawing on the image exactly matches the given pixels. At the beginning of such a covering process, no pixel traversal order is generally defined. A strategy dynamically decides which pixel will be inserted into the object being recognized. For such greedy recognition algorithms, our method offers an interesting technique to efficiently recognize digital objects with two or three intrinsic parameters. For example, determining whether a digital segment covers a set of points is equivalent to a system of two-dimensional linear constraints. We only want to know if a valid segment exists, thus we are faced with a feasibility problem.

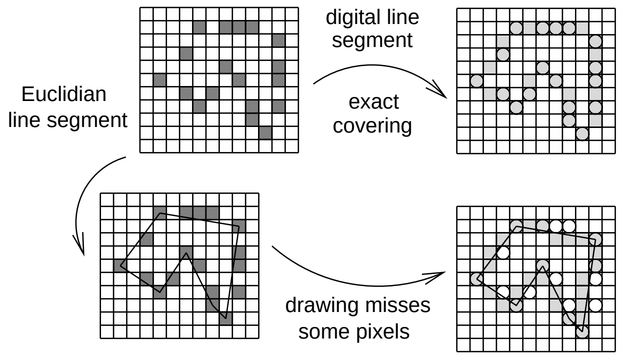

Figure 1: An example of exact approximation

3 Offline Megiddo algorithm for linear programming

Without loss of generality, we rotate the coordinate system in such a way that the objective function is equal to $f(x) = x_d$ (note that this transformation is always possible). For presentation convenience and wlog, let us suppose that x^* is a unique point and that we only have constraints that select the upper part of the space relative to the x_d-axis. Therefore, we have:

$$\text{Minimize } x_d \text{ so that } x_d \geq \Sigma_{j=1}^{d-1} a_{ij}.x_j + b_i, 1 \leq i \leq n \quad (1)$$

Complexity. This technique eliminates constraints that are not tight at x^*. At each iteration, a constant fraction α_d of the constraints is eliminated in $O(n)$ time. The runtime $T(n)$ of the algorithm verifies $T(n) = O(n) + T(\alpha_d.n)$ and this implies that $T(n) = O(n)$.

*A2SI Laboratory, ESIEE, 2 bd Blaise Pascal, Cité Descartes, BP 99, 93162 Noisy-Le-Grand Cedex, France, buzerl@esiee.fr
†Unité Mixte CNRS-UMLV-ESIEE, UMR 8049

Deletion criterion. Under the previous assumption (1), if we take two constraints $x_d \geq \Sigma_{k=1}^{d-1} a_{ik}.x_k + b_i$ and $x_d \geq \Sigma_{k=1}^{d-1} a_{jk}.x_k + b_j$, then the equation $\Sigma_{k=1}^{d-1} a_{ik}.x_k + b_i = \Sigma_{k=1}^{d-1} a_{jk}.x_k + b_j$ describes a hyperplane which divides the space into two domains of domination. Thus, if we could tell on which side of this hyperplane the optimal solution lies, we could then drop one of the two constraints. Such a hyperplane is called a *separating line* (SL) in the two-dimensional case and a *separating plane* (SP) in the three-dimensional space.

3.1 The two-dimensional case

We hereafter describe the inner loop of Megiddo two-dimensional algorithm (see Fig. 2):
1- Coupling: we couple constraints and create SLs.
2- Choosing a test line: as the SLs are vertical, we can class them as their abscissa. We compute in linear time a median γ for these values and choose the line $x_1 = \gamma$ as our test line.
3- Testing: we intersect in linear time all the constraints with the test line and obtain a point P that lies on the border of S. Moreover, we know the left and right slopes around P. When no decreasing slopes exists, P is the minimum and the problem is solved. Otherwise, one property of the convex space of feasible solutions is that only one decreasing slope may exist. Thus, this decreasing slope indicates on which side of the test line the optimum lies.
4- Pruning: the way we have defined the test line allows to immediately deduce the optimum location relative to one half of the SL. We then apply the deletion criterion on the associated couples and reject one quarter of the constraints. Thus, we have $\alpha_2 = \frac{3}{4}$.

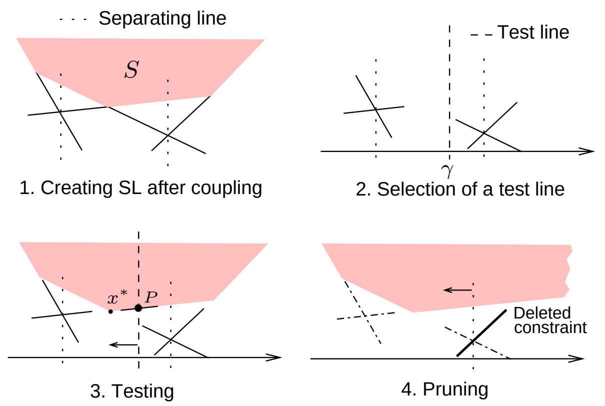

Figure 2: Steps of Megiddo two-dimensional algorithm

3.2 The three-dimensional case

In the following, we explain the main ideas used to extend the previous method to the three-dimensional case.

Point location: suppose we have two planar lines L_1 and L_2 of opposite slopes. Suppose we know the position of a point relative to the vertical and horizontal lines passing trough the intersection of L_1 and L_2. Then we can locate this point relative to one of the two lines L_1 and L_2 (see Fig. 4).
A two-dimensional search problem. By definition, the separating planes (SPs) are vertical. We can project them onto the plane (x_1, x_2) and represent them by a set of lines L. Now, our problem consists in a two-dimensional search problem where we want to locate the projection of x^* relative to some lines of L. We compute in linear time the median of the slopes of the lines in L. For presentation convenience, we rotate the plane (x_1, x_2) in order to make the x_1-axis correspond to this median. Thus, one half of the lines have a positive slope and the other half have a negative slope. We obtain a partition of L into two subsets L_- and L_+ of the same size. We couple each line of L_- to a line of L_+. Let us consider the vertical lines (VLs) that pass through the intersections of these couples. We compute a median of these VLs and test it. We then know the location of x^* relative to one half of the VLs. We now turn to the horizontal lines (HLs) that pass through the intersections of the couples of lines for which the position of x^* is known relative to the VLs. We compute a median line for the HLs and test it. Thus, we obtain the information relative to one half of the HLs and so we know the location of x^* relative to one quarter of the couples. According to the point location technique (see Fig. 4), we locate x^* relative to one eight of the SPs and can delete $\alpha_3 = \frac{1}{16}$ of the constraints. These steps are shown in Fig. 5.

Testing. We now explain how we find the location of x^* relative to a test line l during the search problem. Let us consider the SP P associated to l (see Fig. 3). We first solve our LP problem restricted to this plane and obtain a minimum called m^*. As in the two-dimensional case, we know that if a better solution exists, it can only lie on one side of P. We restrict our attention to the constraints C that pass through m^*. Let P_1 and P_2 denote two planes that are parallel to P and that lie on each side of P. We solve two LP problems restricted to P_1 and P_2 under the set of constraints C. If none of the two optima is better than m^*, we have $x^* = m^*$. Otherwise, the lower optimum indicates on which side of P x^* is located.

4 The online feasibility problem

Lemma 1 *When no feasible solutions lie on the tested hyperplanes, we can prune the constraints and obtain an identical set of feasible solutions by adding a constant number of new constraints.*

Proof. No feasible solutions lie on the tested hyper-

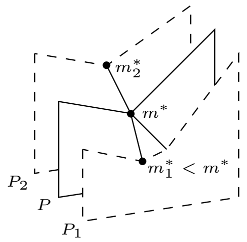

Figure 3: Testing a plane

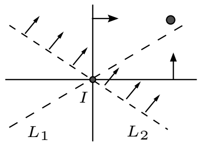

Figure 4: Point location

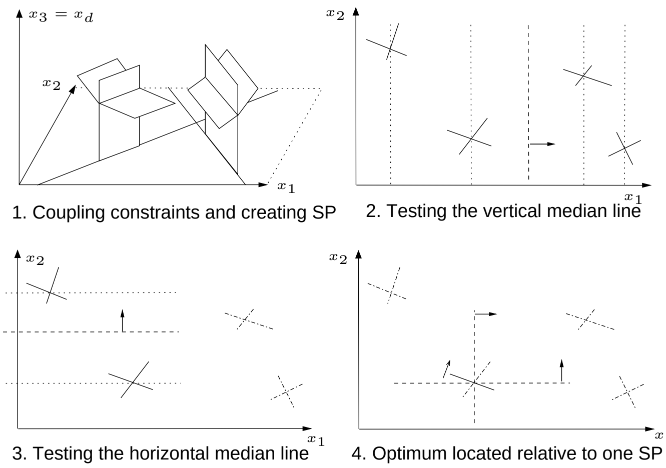

Figure 5: Solving the two-dimensional search problem

planes. Thus, the separating hyperplanes, from where we determine the possible location of the set S of feasible solutions, contain no feasible solutions as well. Therefore, the constraints to be pruned support no face of S and so their deletion have no influence on it. For each tested hyperplane, we must insert one new constraint that selects the half-space containing S in order to maintain valid the system of inequalities. $\square$

The frozen phase. We temporally *freeze* the operation sequence of the third step (testing) of Megiddo algorithm as soon as a feasible solution is found. More precisely, when we enter this modified step, the constraints C of the problem have been coupled and the first test hyperplane is chosen. Let CF denote the constraints that have been inserted since the beginning of the frozen phase. Solving the feasibility problem reduced to this test hyperplane is done by recursively calling a $(d-1)$-dimensional online algorithm. As soon as a feasible solution is found in the $(d-1)$-dimensional problem, the current problem is solved, the program stops and waits for a new constraint. When a new constraint is given, it is not coupled but directly inserted in the $(d-1)$-dimensional problem in order to obtain a new feasible solution. When the $(d-1)$-dimensional problem finds no more feasible solutions, we determine the possible location of the current space of solutions relative to the constraints C and CF. We then compute the next test hyperplane relative to the coupled constraints C only. We proceed in the same way for the following test hyperplanes.

Leaving the frozen phase. All the test hyperplanes have been processed and thus no feasible solutions lie on them. Therefore, we enter the pruning phase. The constraints to be suppressed in C verify Lemma 1 and so they are inactive. The constraints CF are inserted into the pruned constraints. To finish the process, we add one constraint for each tested hyperplane.

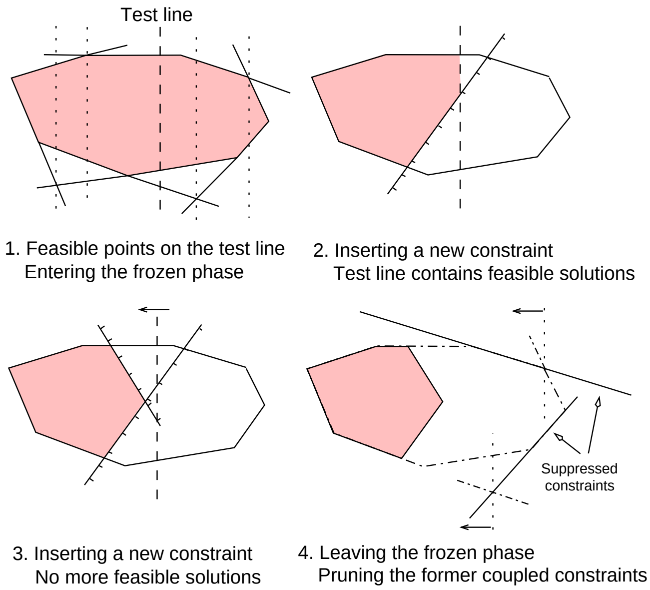

Figure 6: The frozen phase of the incremental method

4.1 The two-dimensional case

We present the frozen phase in the two dimensional case (see Fig. 6). We intersect all the current constraints C with the test line. If we obtain a segment of feasible points, we enter the frozen phase. We intersect each new constraint with the current segment of feasible solutions. We continue until no more solutions lie on the test line and then we leave the frozen phase.

4.2 The three-dimensional case

To test a plane, we use our two-dimensional online method. When we test the first plane, we insert C_1 constraints until no more feasible solutions lie on it. We then determine the location of the current space of solutions S_1 relative to the constraints C and C_1. Thus we deduce the second test hyperplane and launch another two-dimensional problem. After inserting C_2 constraints, no feasible solutions are present on this test plane. We test it relative to the constraints C, C_1 and C_2. We then know the location of the current set of feasible solutions $S_2 \subset S_1$. We prune constraints in C and obtain a new set of constraints C'. The current set of the problem constraints is now given by $C' \cup C_1 \cup C_2$.

4.3 Complexity of the online algorithm

The proof is by induction on the dimension d of the space. Let us first consider the two-dimensional case. Let $(r_i)_{1\leq i\leq k}$ denote the number of constraints remaining in the problem when we enter the i^{th} frozen phase and let $(a_i)_{1\leq i\leq k}$ denote the number of constraints added during this phase. Notice that a_i may be zero. To maintain the system of inequalities, we only insert one constraint relative to the test line. When $r_i \geq 4$, constraints are pruned and $\beta \cdot r_i$ constraints remain in the problem. Thus we have $r_{i+1} \leq \beta \cdot r_i + a_i + 1$. Otherwise, when $r_i < 4$ we have: $r_{i+1} \leq 4 + a_i + 1$. In any case, this inequality holds:

$$r_{i+1} \leq \frac{5}{1-\beta} + \sum_{u=1}^{i} \beta^{i-u} \cdot a_u \qquad (2)$$

Coupling, computing one median and cutting by one test line can be done in $O(r_i)$ time. During the frozen phase, we intersect each new constraint with the line segment in $O(1)$. Let T_i denote the runtime of our algorithm at the end of the i^{th} frozen phase. T_i verifies:

$$T_i = O(r_i) + a_i \cdot O(1) + T_{i-1} = \sum_{u=1}^{i} [O(r_u) + O(a_u)] \quad (3)$$

Then combining (2) and (3):

$$T_k \leq O(\sum_{i=1}^{k} a_i) + O(\frac{k \cdot 5}{1-\beta}) + \sum_{i=1}^{k}\sum_{u=1}^{i} \beta^{i-u} \cdot a_u \qquad (4)$$

Let $n = \Sigma_{i=1}^{k} a_i$ denote the total number of entered constraints. Note that $k \leq n$. We can thus conclude that:

$$T_k \leq O(n) + \sum_{u=1}^{k}\sum_{i=1}^{k-u+1} \beta^{i-u} \cdot a_u \leq O(n) + \sum_{u=1}^{k} \frac{a_u}{1-\beta} = O(n) \qquad (5)$$

In the d-dimensional case [9], the recurrence equation of Megiddo algorithm satisfies:

$$LP_d(n) = 3.A_d.LP_{d-1}(n) + LP_d(\beta_d.n) + O(n.d)$$

With our version, we have to perform A_d frozen phases before pruning at each iteration. Let a_i^l denote the number of inserted constraints in the l^{th} frozen phase of the i^{th} iteration. With $a_i = \sum_{t=1}^{l} a_i^t$, we obtain:

$$T_d(r_{i+1}) = 3.\sum_{l=1}^{A_d} .T_{d-1}(r_i + \sum_{t=1}^{l} a_i^t) + T_d(\beta_d.r_i + a_i) + O((r_i + a_i).d)$$

Let K denote $1 + 1/(1-\beta_d)$. Notice that our recurrence equation is equivalent to the one of Megiddo algorithm:

$$T_d(n) \leq 3.A_d.T_{d-1}(K.n) + T_d(K.\beta_d.n) + O(n.d)$$

5 Conclusion

We have presented an approach that transforms the deterministic offline linear Megiddo algorithm for linear programming into a deterministic online and linear algorithm that can solve a feasibility problem. Nevertheless, as Megiddo algorithm, this version suffers from an important linearity coefficient when the dimension grows. However, for the two and three-dimensional cases, this technique provides an efficient online method for the feasibility problem. Moreover the complexity of our algorithm being optimal in the number of constraints, our method will lead to optimal bounds for several recognition algorithms.

References

[1] Pankaj K. Agarwal and Micha Sharir. Efficient algorithms for geometric optimization. In *ACM Comput. Surv.*, pp:412-458, 30, 1998.

[2] T.M. Chan. Deterministic Algorithms for 2-d Convex Programming and 3-d Online Linear Programming. In *J. Algorithms*, pp:147-166, 27(1), 1998.

[3] K.L. Clarkson. Linear Programming in O($n.2^{3.d^2}$) time. In *Inform. Process. Lett.*, pp:21-24, 22, 1986.

[4] M. E. Dyer. On a multidimensional search technique and its application to the Euclidian one-centre problem. In *SIAM J. Comput.* pp:725-738, 15, 1986.

[5] M. E. Dyer and A. M. Frieze. A randomized algorithm for fixed-dimension linear programming. In *Math. Program.*, pp:203-212, 44, 1989.

[6] D. Eppstein. Dynamic Three-Dimensional Linear Programming. In *ORSA J. Computing*, pp:360-368, 4, 1992.

[7] K.L. Clarkson. A Las Vegas algorithm for linear programming when the dimension is small. In *Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pp:452-456, 1998.

[8] N. Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. In *SIAM J. Comput.*, pp:759-776, 12, 1983.

[9] N. Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, pp:114-127, 31, 1984.

[10] R. Seidel. Small-Dimensional Linear Programming and Convex Hulls Made Easy. In *Discrete and Computational Geometry*, pp:423-434, 6, 1991.