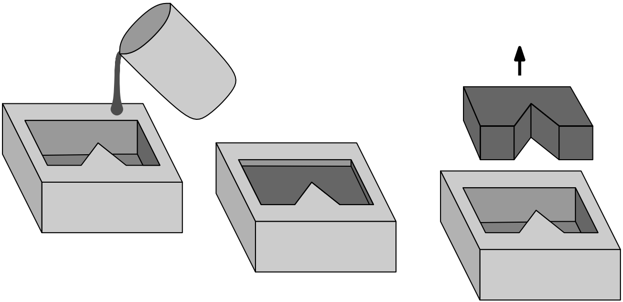
Linear Programming (chapter 4)

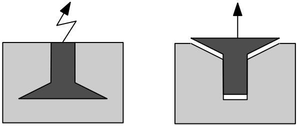
Elisha Sacks

Casting



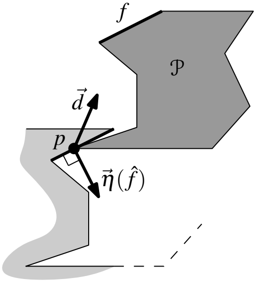
- Pour hot material into a mold.
- The material cools and hardens to form a part.
- Remove the part from the mold.

Castable



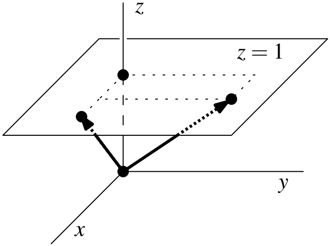
- Task: find a direction for extracting a part from a cast.
- The top facet of the part has outward normal $(0, 0, 1)$.
- The part is castable if it can be removed from the cast by pulling the top facet.
- The motion is linear with direction d such that $d_z > 0$.
- The algorithm tries each facet as the top facet.
- It reports a facet with its direction or reports failure.

Casting Constraints



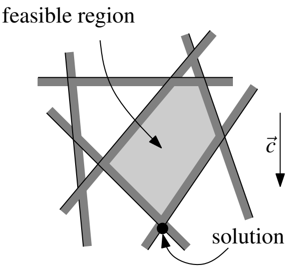
- A part facet f with normal η defines a cast facet $\hat{f}$ with normal $\eta(\hat{f}) = -\eta$.
- The facet $\hat{f}$ blocks motion in the η half plane.
- The constraint is $d \cdot \eta \leq 0$.

Problem Formulation



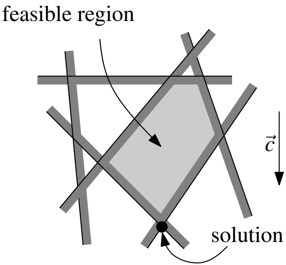
- The motion direction d satisfies $d_z > 0$.
- We normalize it as $d = (d_x, d_y, 1)$.
- The constraint for a facet is $d_x \eta_x + d_y \eta_y + \eta_z \leq 0$.
- We seek a d that satisfies all the facet constraints.

Feasible Region



- Each constraint restricts d to a half space in the xy plane.
- The intersection of the half spaces is the feasible region.
- The book computes the feasible region in $O(n \log n)$ time with a sweep line algorithm.
- We will find a feasible point in expected $O(n)$ time.

Linear Programming Formulation



Find a feasible p that maximizes $\vec{c} \cdot p$ with $\vec{c}$ arbitary, or report that none exists.

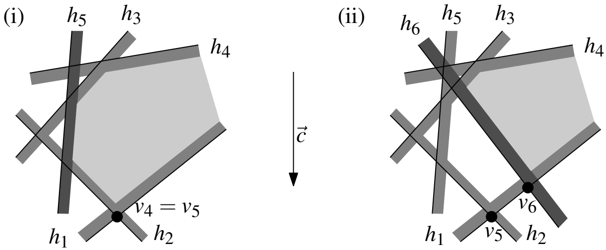
- It is convenient to enclose (d_x, d_y) in a bounding box.
- This is reasonable because tiny casting angles are impractical.
- The textbook shows how to solve unbounded linear programs.

Linear Programming

Maximize a linear objective subject to linear inequality constraints.

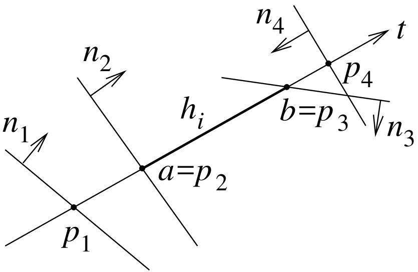
- Widely used in computational science.
- Simplex algorithm and interior point methods are efficient.
- Running time is polynomial in input size, but super-linear.
- Most application involve many variables and constraints.
- Casting involves two variables and many constraints.
- This case has a fast algorithm with expected linear time.
- The approach applies to three or more variables.
- The constant factor grows rapidly with dimension.

Incremental 2D Linear Programming Algorithm



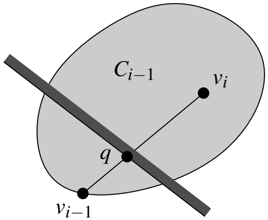
- The initial feasible region is the bounding box C_0 and the initial solution is a corner v_0 that maximizes $f(p) = \vec{c} \cdot p$.
- Each constraint h_i is added and $v_i \in C_i$ is computed.
  - (i) If v_{i-1} is in the h_i half space, $v_i = v_{i-1}$.
  - (ii) Else v_i is the maximum of f on the feasible interval of h_i. If the feasible interval is empty, report failure.
- Case (ii) is a 1D linear program.

Solving the 1D Linear Program



- Let the h_j line have normal n_j and let h_i have tangent t.
- h_j intersects the h_i line in a half line bounded by a point p_j.
- Let a maximize $p_j \cdot t$ among the h_j with $n_j \cdot t > 0$.
- Let b minimize $p_j \cdot t$ among the h_j with $n_j \cdot t < 0$.
- If $(b - a) \cdot t < 0$, the feasible region is empty.
- Else the feasible region is $[a, b]$ and the solution is a or b.

Correctness

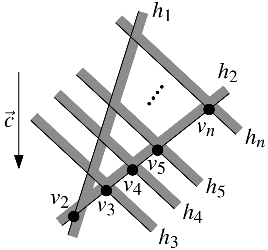


Lemma 4.5 If $v_{i-1} \notin h_i$, either C_i is empty or v_i is on the h_i line.

Proof Assume C_i is not empty and v_i is not on the h_i line.

- v_i is in C_{i-1} because C_{i-1} is a subset of C_i.
- The line segment $v_i v_{i-1}$ is in C_{i-1} by convexity.
- $v_i v_{i-1}$ intersects the h_i line because $v_{i-1} \notin h_i$ and $v_i \in h_i$.
- The intersection point q is in C_i.
- f increases along $v_i v_{i-1}$ because v_{i-1} is its maximum in C_{i-1}.
- $f(q) \geq f(v_i)$ which contradicts the definition of v_i.

Computational Complexity



- Computing v_i takes $O(i)$ time.
- The algorithm is $O(n^2)$ because this can happen at every i.
- The running time depends on the order of the h_i.
- The output is independent of the order.
- In the example, reversing the order leads to $O(n)$ time.
- Inserting the h_i in random order gives $O(n)$ on average.

Expected Running Time

Lemma 4.8 A 2D bounded linear program with n constraints is solved in $O(n)$ randomized expected time.

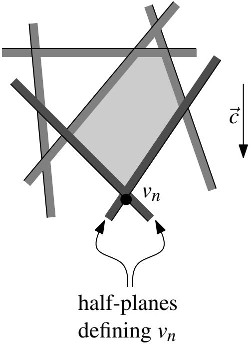
Proof

- The sample space is the $n!$ orderings of $h_1, \ldots, h_n$.
- The distribution is uniform.
- Let X_i equal 1 if $v_{i-1} \notin h_i$ and 0 otherwise.
- The running time for the steps with $X_i = 0$ is $O(n)$.
- We bound the expected value of the steps with $X_i = 1$.

$$E = E[\sum_{i=1}^{n} O(i)X_i] = \sum_{i=1}^{n} O(i)E[X_i]$$

- We will prove that $E[X_i] \leq 2/i$.
- Hence $O(i)E[X_i] = O(1)$ and $E = O(n)$.

Backward Analysis

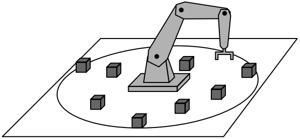


- $E[X_n]$ is the probability that v_n is created when h_n is added.
- This is the probability that v_n vanishes when h_n is removed.
- v_n vanishes if h_n is one of its two defining lines.
- The probability is $2/n$ because the order is random.
- Likewise $E[X_i] = 2/i$.

Computing a Random Permutation

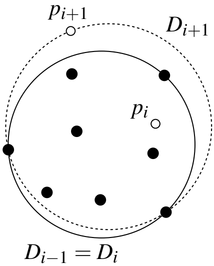
```
unsigned int * randomPermutation (unsigned int n)
{
  unsigned int *p = new unsigned int [n];
  for (unsigned int i = 0u; i < n; ++i)
    p[i] = i;
  for (unsigned int i = n - 1u; i > 0u; --i) {
    unsigned int j = rand()%(i+1);
    swap(p[i], p[j]);
  }
  return p;
}
```

Minimal Disk



- The incremental strategy applies to other tasks.
- Example: find the smallest disk that contains n points.

Minimal Disk Constraint



- Let C_i and D_i be the minimal circle and disk of $p_1, \ldots, p_i$.
- If $p_{i+1} \in D_i$, $D_{i+1} = D_i$.
- Otherwise, $p_{i+1} \in C_{i+1}$.
- Likewise if the circles must contain one or two points.

Algorithm

```
Circle * minDisk (const Points &pts)
{
  unsigned int n = pts.size(),
    *p = randomPermutation(n);
  PTR<Circle> c = new Circle2pts(pts[p[0]],
                                 pts[p[1]]);
  for (unsigned int i = 2u; i < n; ++i) {
    Point *r = pts[p[i]];
    if (PointInCircle(r, c) == -1)
      c = minDiskWithPoint(pts, p, i, r);
  }
  delete [] p;
  return c;
}
```

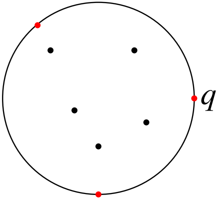
Algorithm

```
Circle * minDiskWithPoint
  (const Points &pts, unsigned int *p,
   unsigned int n, Point *q)
{
  PTR<Circle> c = new Circle2pts(pts[p[0]], q);
  for (unsigned int i = 1; i < n; ++i) {
    Point *r = pts[p[i]];
    if (PointInCircle(r, c) == -1)
      c = minDiskWithTwoPoints(pts, p, i, q, r);
  }
  return c;
}
```

Algorithm

```
Circle * minDiskWithTwoPoints
  (const Points &pts, unsigned int *p,
   unsigned int n, Point *q1, Point *q2)
{
  PTR<Circle> c = new Circle2pts(q1, q2);
  for (unsigned int i = 0u; i < n; ++i) {
    Point *r = pts[p[i]];
    if (PointInCircle(r, c) == -1)
      c = new Circle3pts(q1, q2, r);
  }
  return c;
}
```

Expected Running Time



Theorem 4.15 The smallest enclosing disk of a set of n points is computed in $O(n)$ randomized expected time.

Proof

- minDiskWithTwoPoints is $O(n)$.
- minDiskWithPoint is $O(n)$ excluding minDiskWithTwoPoints.
- p_i costs $O(i)$ if it calls minDiskWithTwoPoints.
- This occurs if p_i is one of the three points on D_i.
- The probability is $2/i$ because q is one of the three.
- Running time is $O(n) + \sum_i \frac{2}{i} O(i) = O(n)$.
- Likewise minDisk with $1/i$ instead of $2/i$.