

# Basic Approaches to Optimization Problems

**Definition 1.** A minimization problem is specified by a pair  $(\mathcal{L}, f)$  where

$\mathcal{L}$  is a set of **feasible solutions** and

$f : \mathcal{L} \rightarrow \mathbb{R}$  is the **cost function**

$\mathbf{x}^* \in \mathcal{L}$  is an **optimal solution** if  $f(\mathbf{x}^*) \leq f(\mathbf{x})$  for all  $\mathbf{x} \in \mathcal{L}$ .

Is a **maximization problem** any different?

# References

[1] E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. Wiley, 1996.

[2] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

## 1 Optimization Problems We Have Seen Before

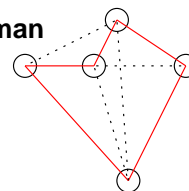
- Shortest Path
- Minimum Spanning Tree
- Maximum Flow
- Maximum Cardinality Matching
- Sorting (If we insist)

What do these Problems have in common?

- Abstractness
- Ubiquitousness
- We know how to solve them (very) efficiently

## 2 Less Easy Problems

### 2.1 Traveling Salesman



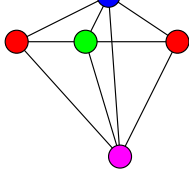
Given  $G = (V, E)$ , find simple cycle  $C = (v_1, v_2, \dots, v_n, v_1)$  such that  $n = |V|$  and  $\sum_{(u,v) \in C} d(u, v)$  is minimized.

$\approx$  Find the shortest path visiting **all** nodes.

The problem is NP-hard, even if  $G$  is complete, undirected and obeys the **triangle inequality**:

$$\forall u, v, w \in V : d(u, w) \leq d(u, v) + d(v, w)$$

## 2.2 Graph Coloring

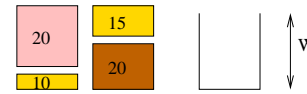


Given an undirected graph  $G = (V, E)$  find a mapping  $c : V \rightarrow \{1, \dots, k\}$  such that  $k$  is minimized subject to the condition

$$\forall \{u, v\} \in E : c(u) \neq c(v)$$

For this problem we do not even know efficient approximation algorithms with reasonable performance guarantees.

## 2.3 The Knapsack Problem



- $n$  items with weight  $w_i$  and profit  $p_i$
- Choose a subset  $x$  of items
- Capacity constraint  $\sum_{i \in x} w_i \leq W$
- Maximize profit  $\sum_{i \in x} p_i$

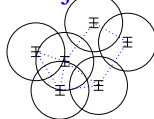
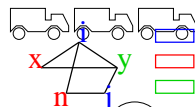
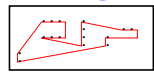
Peter Sanders: Optimization



7

## 3 Applications – Somewhat Different

- Shortest/Fastest train/car connections in Europe. Source destination shortest path problems, precomputation allowed, special structure, geometric information
- Cheapest connection. **not** a shortest path problem.
- Drilling holes into printed circuit boards. TSP with special structure
- Vehicle routing. Generalization of TSP. Many side constraints in practice.
- Register allocation in a compiler. Small graph coloring problems with special structure
- Frequency assignment to radio transmitters. Generalization of graph coloring



Peter Sanders: Optimization



8

- Portfolio planning. Generalization of knapsack. Also plan for **uncertainty**
- Communication network design. Generalization of MST (NP-hard). Also plan for **bandwidth** and **fault tolerance**.
- Reconstruct altitude information from interferometric radar images. Nobody knows the correct formalization. Two reasonable models lead to a weighted matching problem and a generalization of MST respectively.

### Good News and Bad News

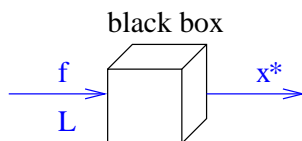
- Applications have complex, fuzzy, and variable models
- + Application instances have special structure and are often far from worst case.
- + Sometimes (close) approximations are good enough

## 4 Declarative Problem Solving

Idea, describe the problem but not how to solve it.

### Goals

- An easy to use and powerful modeling language
- Fast and reliable solution methods that can be implemented once and for all



- Continue to study simple problems. Since they show up in many places and are of independent theoretical interest.
- Develop specific algorithms for **important** more complex problems (e.g., human genome sequence assembly). But beware of ending up with a complicated algorithm for a complex but equally unrealistic model.
- Look at approximation algorithms.
- Study flexible **meta algorithms** that can be adapted to many situations

## 5 Linear Programming

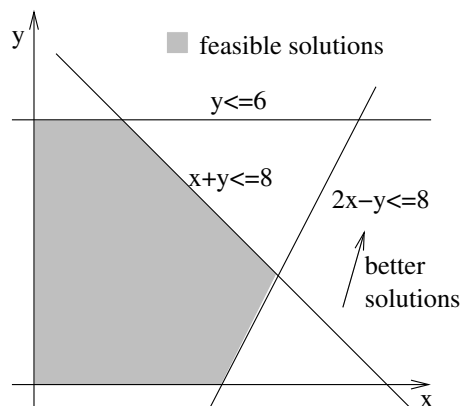
**Definition 2.** A **linear program** with  $n$  variables and  $m$  constraints is specified by the following minimization problem

- Cost function  $f(\mathbf{x}) = \mathbf{c} \cdot \mathbf{x}$   
 $\mathbf{c}$  is called the **cost vector**
- $m$  constraints of the form  $\mathbf{a}_i \cdot \mathbf{x} \bowtie_i b_i$  where  $\bowtie_i \in \{\leq, \geq, =\}$ ,  $\mathbf{a}_i \in \mathbb{R}^n$  We have

$$\mathcal{L} = \{\mathbf{x} \in \mathbb{R}^n : \forall 1 \leq i \leq m : x_i \geq 0 \wedge \mathbf{a}_i \cdot \mathbf{x} \bowtie_i b_i\} .$$

Let  $a_{ij}$  denote the  $j$ -th component of vector  $\mathbf{a}_i$ .

## 5.1 A Simple Example



## 5.2 An Application of LP

### Mixing Animal Food

- $n$  different kinds of food. Kind  $i$  costs  $c_i$  Euro/kg.
- $m$  requirements for a healthy nutrition. (calories, proteins, Vitamin C, . . .  $i$  contains  $a_{ji}$  percent of the daily requirement per kg of an animal with respect to requirement  $j$ .)
- Define  $x_i$  as needed quantity of kind  $i$  per animal and day.
- LP solution gives a cost optimal feasible diet.

### Refinements

- Upper bounds (radioactivity, Cadmium, cow brain, . . . )
- Limited supplies (hay from the farm)
- Piecewise linear convex cost functions (transport cost grows with distance)

### Limitations

- Minimum quantities for buying
- Most nonlinear cost functions
- Integral quantities (small farms)
- Garbage in Garbage out

### 5.3 Maximum Flow as an LP

Given a directed graph  $G = (V, E)$  find a maximum flow  $f : E \rightarrow \mathbb{R}_+$  from  $s$  to  $t$  obeying edge capacities  $\text{cap}(e)$ .

$$\text{maximize } \sum_{(v,t) \in E} f(v,t)$$

subject to

$$\forall v \in V \setminus \{s, t\} : \sum_{(u,v) \in E} f(u,v) = \sum_{(v,w) \in E} f(v,w) \quad (1)$$

$$\forall e \in E : 0 \leq f(e) \leq \text{cap}(e) \quad (2)$$

(3)

There are several useful generalizations which are easily expressed as LPs yet break specialized maximum flow algorithms.

## 5.4 Complexity

**Theorem 1.** A linear program can be solved in polynomial time.

- Typical execution time is  $\mathcal{O}((m+n)N)$  where  $N = |\{a_{ij} \neq 0\}|$
- Worst case bounds are rather high
- The algorithm used in practice might take exponential worst case time
- Reuse is not only possible but almost necessary because a robust, efficient implementation is quite COMPLEX.

Peter Sanders: Optimization



19

## 6.1 How to Cope with (M)ILPs

- Solving (M)ILPs is NP-hard
- + Powerful modeling language
- + There are generic methods that sometimes work well
- + Many ways to get approximate solutions.
- + The solution of the integer relaxation helps. For example sometimes we can simply round.

Example: In the knapsack problem there is only **one fractional variable** in the linear relaxation. Rounding it **down** to 0 yields a feasible solution.

If weights and profits are small compared to  $M$  then this solution is close to optimal.

## 6 Integer Linear Programming

**ILP:** Integer Linear Program, A linear program with the additional constraint that **all the**  $x_i \in \mathbb{N}$

**MILP:** Mixed Integer Linear Program, A linear program with the additional constraint that **some** of the  $x_i$  must be integer.

**Linear Relaxation:** Remove the integrality constraints from an (M)ILP

### 6.0.1 Example: The Knapsack Problem

$$\text{maximize } \mathbf{p} \cdot \mathbf{x}$$

subject to

$$\mathbf{w} \cdot \mathbf{x} \leq M, x_i \in \{0, 1\} \text{ for } 1 \leq i \leq n .$$

$x_i = 1$  iff item  $i$  is put into the knapsack.

0/1 variables are typical for ILPs

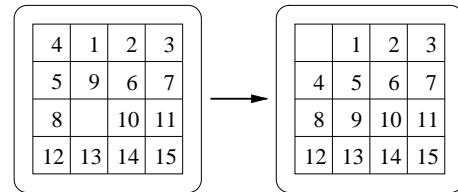
Peter Sanders: Optimization



20

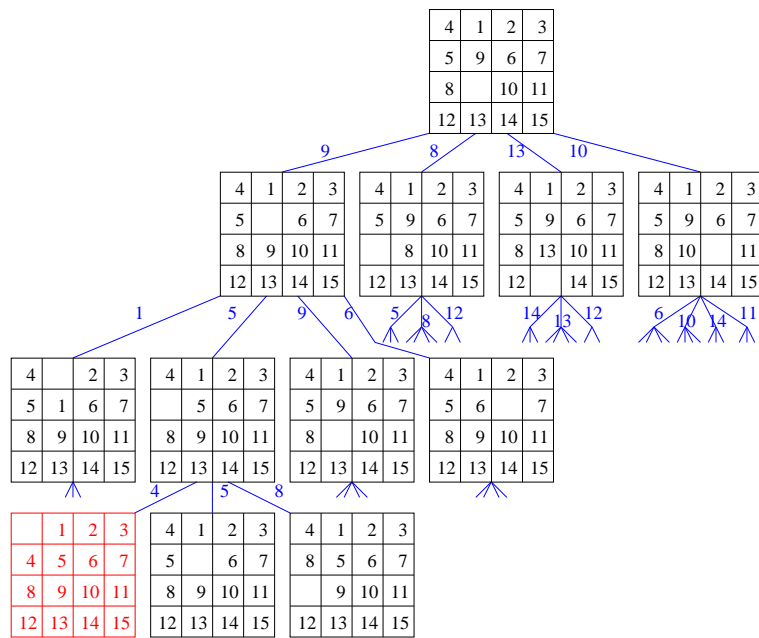
## 7 Systematic Search — If In Doubt Use Brute Force (?)

### 7.1 A Concrete Example



Move = move one piece into the hole

Solution with minimum number of moves?



Moving a tile back and forth won't help

**Bounding**

Already spent  $d$  moves?

$d'$  more needed even if all moves bring tiles towards target?

$\Rightarrow$  we need depth at least  $d + d'$ .

$d + d' > d_{max}$ ? Then do not go on.

**7.2 A Generic Routine**

```

cand := {U}
(* invariant:  $\exists M \in \text{cand}$ , optimal solution  $\mathbf{x}^* : \mathbf{x}^* \in M$  *)
 $\hat{\mathbf{x}} := \perp$  (* best solution found so far *)
while cand  $\neq \emptyset$  do
  delete some candidate curr from cand
  prune obviously useless elements from curr
  if |curr|  $\geq 1$  then
    if curr = {x} then (* singleton *)
      if  $f(\mathbf{x}) < f(\hat{\mathbf{x}})$  then  $\hat{\mathbf{x}} := \mathbf{x}$ 
    else
      split curr such that  $\text{curr} = \text{curr}_1 \cup \dots \cup \text{curr}_k$  and
         $\forall \text{curr}_i : \text{curr}_i \subseteq \text{curr}$  but  $\text{curr}_i \neq \text{curr}$ 
      cand := cand  $\cup \{\text{curr}_1, \dots, \text{curr}_k\}$  (* branch *)
  
```

**7.3 Instantiation for the 15 Puzzle**

**cand:** Stack of system states annotated with the number of moves already made. A state at level  $i$  represents the set of states reachable by applying  $d_{max} - i$  moves.

**U:** Start state.

**delete curr:** Pop from stack

**Pruning:** Bounding

**Split:** Apply all moves that do not go back. Push the new states on the stack with incremented move counter.

### 7.4 Branch-and-Cut for ILPs

**curr** A set of constraints defining the set of feasible solutions

**cand**: Stack or priority queue of ILPs. Store only root and the differences to children.

**U**: The original ILP.

**Pruning**: Bounding and Cutting

**Bounding**: Let  $\check{x}$  denote an optimal solution of the linear relaxation of curr. Stop if  $f(\check{x}) \geq f(\hat{x})$

**Cutting**: (Optional) Add a constraint to curr such that  $\check{x}$  is no longer a solution of curr yet curr still represents all solutions in  $\mathcal{L}$ .

**Split**: Consider a variable  $x_i$  from  $\check{x}$  such that  $x_i \notin \mathbb{N}$ . produce two new problems

$$\text{curr}_1 = \text{curr} + \{c_i \leq \lfloor x_i \rfloor\}$$

$$\text{curr}_2 = \text{curr} + \{c_i \geq \lceil x_i \rceil\}.$$

### Example: A Knapsack Instance

$$\text{maximize } (10, 15, 20, 20) \cdot \mathbf{x} \quad \text{subject to}$$

$$C = \{(1, 2, 3, 4) \cdot \mathbf{x} \leq 5, 0 \leq x_i \leq 1\}$$

$$\check{x} = (1, 1, 2/3, 0) \Rightarrow \text{branch on } x_3$$

$$\text{cand} = \underbrace{\{C \cup \{x_3 = 0\}\}}_{C_1}, \underbrace{\{C \cup \{x_3 = 1\}\}}_{C_2}$$

$$\text{delete curr} = C_1 \Rightarrow \check{x} = (1, 1, 0, 1/2) \Rightarrow \text{branch on } x_4$$

$$\text{cand} = \underbrace{\{C \cup \{x_3 = 0, x_4 = 0\}\}}_{C_{11}},$$

$$\underbrace{\{C \cup \{x_3 = 0, x_4 = 1\}\}}_{C_{12}}, \underbrace{C_2}_{C_2}$$

$$\text{delete curr} = C_{11} \Rightarrow \check{x} = (1, 1, 0, 0) \text{ feasible remember in } \hat{x}$$

$$\text{delete curr} = C_{12} \Rightarrow \check{x} = (1, 0, 0, 1) \Rightarrow \text{improved solution}$$

$$\text{delete curr} = C_2 \Rightarrow \check{x} = (1, 1/2, 1, 0) \Rightarrow \text{branch on } x_2$$

$$\text{cand} = \{C_{21} = C \cup \{x_2 = 0, x_3 = 1\},$$

$$C_{22} = C \cup \{x_2 = 1, x_3 = 1\}\}$$

$$\text{delete curr} = C_{22} \Rightarrow \check{x} = (0, 1, 1, 0) \Rightarrow \text{improved solution}$$

$$\text{delete curr} = C_{21} \Rightarrow \check{x} = (1, 0, 1, 1/4) \Rightarrow$$

cannot get better than 35. Bound.

cand is empty — done

## 8 Dynamic Programming

### — Building it Piece By Piece

Principle of Optimality

- An optimal solution can be viewed as **constructed** of **optimal** solutions for **subproblems**
- Optimal solutions for a given subproblem size are **interchangeable**

### Example: Shortest Paths

- Any subpath of a shortest path is a shortest path
- Shortest subpaths are interchangeable

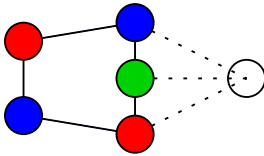


## 8.1.1 Constrained Shortest Paths

Assume the **fastest** way to get around the Hockenheim-Ring **20** times with a Formula-1 car **exhausts the fuel**. Then this is not a good substrategy for getting around **21** times because this would require a **pit-stop** after round 20.

## 8.1.2 Graph Coloring

Optimal solutions for subgraphs are not interchangeable.



## Proof

$P(i, C) \geq P(i-1, C)$ : Set  $x_i = 0$ , use optimal subsolution.

$P(i, C) \geq P(i-1, C - c_i) + p_i$ : Set  $x_i = 1 \dots$

$P(i, C) \leq \max(P(i-1, C), P(i-1, C - c_i) + p_i)$

Assume the **contrary**  $\Rightarrow$

$\exists \mathbf{x}$  that is **optimal** for the subproblem such that

$$P(i-1, C) < \mathbf{p} \cdot \mathbf{x} \wedge$$

$$P(i-1, C - c_i) + p_i < \mathbf{p} \cdot \mathbf{x}$$

**Case  $x_i = 0$ :**  $\mathbf{x}$  is also feasible for  $P(i-1, C)$ . Hence,

$$P(i-1, C) \geq \mathbf{p} \cdot \mathbf{x}. \text{ Contradiction}$$

**Case  $x_i = 1$ :** Setting  $x_i = 0$  we get a feasible solution  $\mathbf{x}'$  for

$$P(i-1, C - c_i) \text{ with profit } \mathbf{p} \cdot \mathbf{x}' = \mathbf{p} \cdot \mathbf{x} - p_i. \text{ Hence,}$$

$$P(i-1, C - c_i) + p_i \geq \mathbf{p} \cdot \mathbf{x}. \text{ Contradiction}$$

## Knapsack Problem

maximize  $\mathbf{p} \cdot \mathbf{x}$

subject to  $\mathbf{c} \cdot \mathbf{x} \leq W, x_i \in \{0, 1\}$

Define

$P(i, C) = \text{optimal profit from items } 1, \dots, i \text{ using capacity } \leq C.$

**Lemma 1.**

$$\forall 1 \leq i \leq n : P(i, C) = \max(P(i-1, C),$$

$$P(i-1, C - c_i) + p_i)$$

8.2.1 Computing  $P(i, C)$  bottom up:

**Procedure** knapsack( $\mathbf{p}, \mathbf{c}, n, W$ )

array  $P[0 \dots W] = [0, \dots, 0]$

bitarray decision $[1 \dots n, 0 \dots W] = [(0, \dots, 0), \dots, (0, \dots, 0)]$

**for**  $i := 1$  **to**  $n$  **do**

**(\* invariant:  $\forall C \in \{1, \dots, W\} : P[C] = P(i-1, C)$  \*)**

**for**  $C := W$  **downto**  $w_i$  **do**

**if**  $P[C - c_i] + p_i > P[C]$  **then**

$P[C] := P[C - c_i] + p_i$

decision $[i, C] := 1$

### 8.2.2 Recovering a Solution

```

C := W
array x[1 . . . n]
for i := n downto 1 do
  x[i] := decision[i, C]
  if x[i] = 1 then C := C - w_i
endfor
return x

```

Analysis:

Time:  $\mathcal{O}(nW)$  pseudo-polynomial

Space:  $W + \mathcal{O}(n)$  words plus  $Wn$  bits.

### Example: A Knapsack Instance

```

maximize (10, 20, 15, 20) · x
subject to (1, 3, 2, 4) · x ≤ 5

```

$P(i, C), (\text{decision}[i, C])$

$i \setminus C$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0, (0)	10, (1)	10, (1)	10, (1)	10, (1)	10, (1)
2						
3						
4						

### Example: A Knapsack Instance

```

maximize (10, 20, 15, 20) · x
subject to (1, 3, 2, 4) · x ≤ 5

```

$P(i, C), (\text{decision}[i, C])$

$i \setminus C$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0, (0)	10, (1)	10, (1)	10, (1)	10, (1)	10, (1)
2	0, (0)	10, (0)	10, (0)	20, (1)	30, (1)	30, (1)
3						
4						

### Example: A Knapsack Instance

```

maximize (10, 20, 15, 20) · x
subject to (1, 3, 2, 4) · x ≤ 5

```

$P(i, C), (\text{decision}[i, C])$

$i \setminus C$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0, (0)	10, (1)	10, (1)	10, (1)	10, (1)	10, (1)
2	0, (0)	10, (0)	10, (0)	20, (1)	30, (1)	30, (1)
3	0, (0)	10, (0)	15, (1)	25, (1)	30, (0)	35, (1)
4						

### Example: A Knapsack Instance

maximize  $(10, 20, 15, 20) \cdot \mathbf{x}$

subject to  $(1, 3, 2, 4) \cdot \mathbf{x} \leq 5$

$P(i, C), (\text{decision}[i, C])$

$i \setminus C$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0, (0)	10, (1)	10, (1)	10, (1)	10, (1)	10, (1)
2	0, (0)	10, (0)	10, (0)	20, (1)	30, (1)	30, (1)
3	0, (0)	10, (0)	15, (1)	25, (1)	30, (0)	35, (1)
4	0, (0)	10, (0)	15, (0)	25, (0)	30, (0)	35, (0)

### 8.3 Designing a Dynamic Programming Algorithm

- Find structure of optimal subproblems
- Find a recurrence
- Design a bottom up schedule for computing table entries
- How to save memory?
- How to recover the solution?

## 9 Sometimes Being Greedy Helps

Greedy Strategies

- View problem solving as making a **sequence of decision**
- Always pick a decision that look **locally** optimal
- That is all. Never undo a decision.

### 9.1 Success Stories

- Kruskal's** Algorithm for **Minimum Spanning Trees**.  
Grow a forest.
- Jarnik/Prim's Algorithm for **Minimum Spanning Trees**.  
Grow a tree
- Dijkstra's** Algorithm for **shortest paths**.  
Grow a shortest path tree
- Fractional Knapsacks**  
Pack and cut.
- Heapsort**

All these algorithms yield an optimal solution.

Problem	Decisions	Criterion
Kruskal MST	edges	smallest
Jarnik MST	edges	smallest connection
Dijkstra SSSP	nodes	closest
fract. Knapsack	items	best profit density
Heapsort	elements	smallest

**Counterexample: Knapsack**



Greedy gives objective value 25  
optimal would be 35

**9.2 Scheduling Independent Weighted Jobs on Parallel Machines**

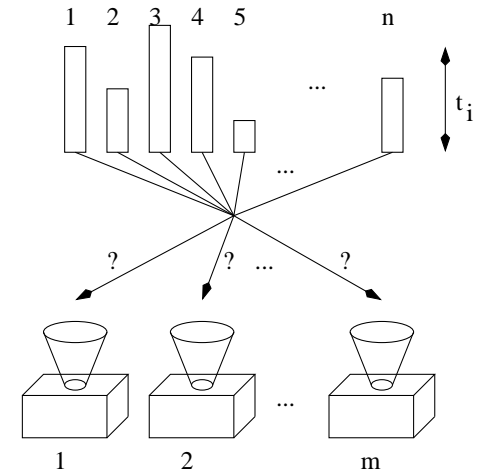
$x(j)$ : Machine where job  $j$  is executed

$L_i$ :  $\sum_{x(j)=i} t_j$ , load of machine  $i$

Objective: Minimize

$$L_{\max} = \max_i L_i$$

(Makespan)



Details: Identical machines, independent jobs, known processing times, offline

NP-hard

**9.2.1 Least Processing Time First Scheduling**

LPT( $n, m, t$ )

$J := \{1, \dots, n\}$

array  $L[1..m] = [0, \dots, 0]$

**while**  $J \neq \emptyset$  **do**

pick  $j \in J$  such that  $t_j$  is maximized

$J := J \setminus \{j\}$

(\* Shortest Queue: \*)

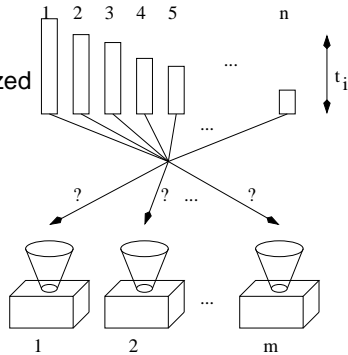
pick  $i$  such that  $L[i]$  is minimized

$x(j) := i$

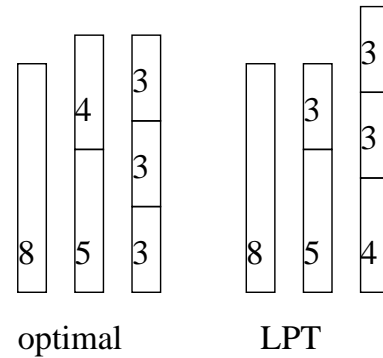
$L[i] := L[i] + t_j$

**endwhile**

**return**  $x$



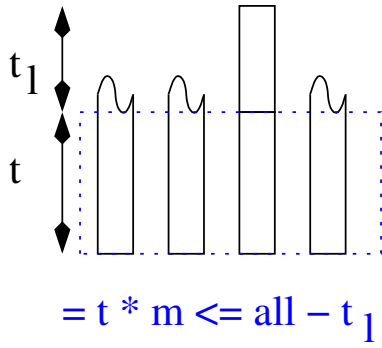
**LPT Scheduling is Not Optimal**



**Lemma 2.** If  $\ell$  is the index of the job finishing last then

$$L_{\max} \leq \sum_j \frac{t_j}{m} + \frac{m-1}{m} t_\ell$$

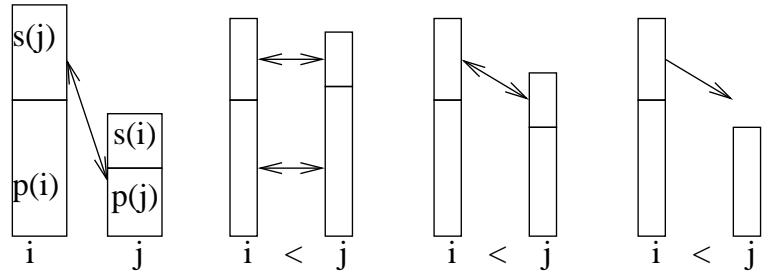
Proof



$$L_{\max} = t + t_\ell \leq \sum_{j \neq \ell} \frac{t_j}{m} + t_\ell = \sum_j \frac{t_j}{m} + \frac{m-1}{m} t_\ell .$$

**Lemma 3.** When an optimal schedule assigns at most *two* jobs to each machine, LPT scheduling computes an *optimal* schedule.

- Wlog  $i < j \Rightarrow t_i < t_j$ , LPT puts job  $n - i + 1$  on machine  $i$  for  $1 \leq i \leq m$ .
- Transform optimal schedule into “organ pipe” form:



- Move jobs further right to finalize transformation to LPT schedule

### 9.2.2 Approximation Ratios

**Definition 3.** A minimization algorithm achieves *approximation ratio*  $\rho$  if for all inputs  $I$ , it produces a solution  $\mathbf{x}(I)$  such that

$$\frac{f(\mathbf{x}(I))}{f(\mathbf{x}^*(I))} \leq \rho$$

where  $\mathbf{x}^*(I)$  denotes the optimum solution for input  $I$ .

**Theorem 2.** *leastProcessingTimeFirst* achieves an *approximation ratio* of

$$\frac{4}{3} - \frac{1}{3m}$$

**Proof**

Assume  $\exists$  instance such that  $\frac{4}{3} - \frac{1}{3m} < \frac{f(\mathbf{x})}{f(\mathbf{x}^*)}$ .

$$\frac{4}{3} - \frac{1}{3m} < \frac{f(\mathbf{x})}{f(\mathbf{x}^*)}$$

By Lemma 2  $\leq \frac{\sum_j t_j/m}{f(\mathbf{x}^*)} + \frac{m-1}{m} \cdot \frac{t_\ell}{f(\mathbf{x}^*)}$

$$\leq 1 + \frac{m-1}{m} \cdot \frac{t_\ell}{f(\mathbf{x}^*)}$$

Using  $f(\mathbf{x}^*) \geq \sum_j t_j/m$ .

Bringing  $f(\mathbf{x}^*)$  on one side yields  $f(\mathbf{x}^*) < 3t_\ell$ .

We can even arrange that  $t_\ell$  is the smallest job (drop later jobs).

$\Rightarrow$  at most two jobs per machine

$\Rightarrow$  optimal by Lemma 3. Contradiction.

# 10 Local Search — Think Globally, Act Locally

Find some feasible solution  $x \in \mathcal{L}$

$\hat{x} := x$  (\* best solution found so far \*)

**while** not satisfied with  $\hat{x}$  **do**

curr :=  $\mathcal{N}(x) \cap \mathcal{L}$

$x :=$  some heuristically chosen element from curr

**if**  $f(x) < f(\hat{x})$  **then**  $\hat{x} := x$

Assumptions and Heuristics:

- Neighborhood
- We always work with feasible solutions
  - If in doubt move constraints into the objective function
- Selection criterion
- Stopping criterion

Peter Sanders: Optimization



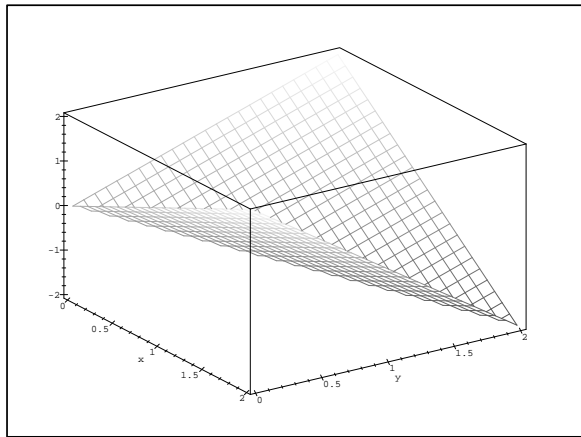
51

## 10.1.1 Why the Neighborhood is Important

Example:  $\mathcal{L} = \{0, \dots, k\}^2$ . A natural choice seems to be

$\mathcal{N}((x, y)) = \{(x + 1, y), (x - 1, y), (x, y - 1), (x, y + 1)\}$ .

This fails even for  $f(x, y) = \begin{cases} x - 2y & \text{for } x > y \\ y - 2x & \text{else} \end{cases}$



## 10.1 Hill Climbing

Find some feasible solution  $x \in \mathcal{L}$

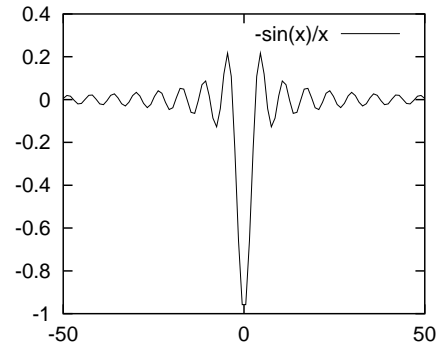
$\hat{x} := x$  (\* best solution found so far \*)

**loop**

**if**  $\exists x \in \mathcal{N}(x) \cap \mathcal{L} : f(x) < f(\hat{x})$  **then**  $\hat{x} := x$

**else return**  $\hat{x}$  (\* local optimum found \*)

Main Problem: Local Optima are not global optima



Peter Sanders: Optimization

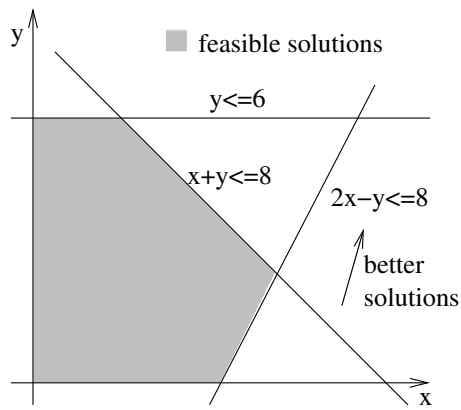


52

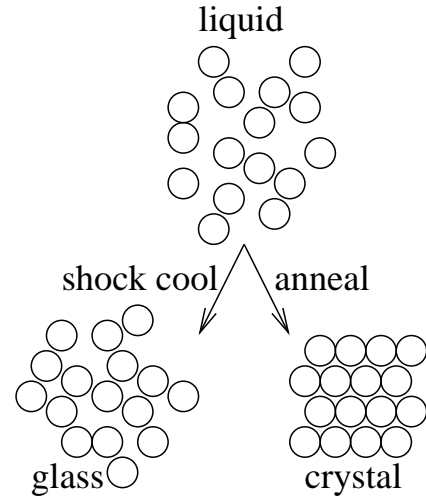
## 10.1.2 The Simplex Algorithm

For Linear Programming

- $\mathcal{L}$  is a **convex polytope**
  - Corner points** are points in  $\mathcal{L}$  where  $n$  constraints are satisfied
  - Theorem:  $\exists$  corner  $x^*$  that is **globally optimal**. Continuous problem  $\leadsto$  discrete problem
  - 2 Corners are neighbors if they differ by one constraint
  - Theorem: If  $x$  is nonoptimal and  $x$  is at a corner then there is a better neighboring corner
- $\Rightarrow$  Hillclimbing on the set of corners and the above notion of neighborhood works (almost)



A physical analogy



Qualitative explanation:

Locally optimal substructures have time to dissolve

10.2.1 The Boltzmann Distribution

Equilibrium energy distribution

$$P[E_x] = \frac{\exp(-\frac{E_x}{T})}{\sum_{y \in \mathcal{L}} \exp(-\frac{E_y}{T})}$$

$T$  = temperature of the system in Kelvin multiplied by the Boltzmann constant  $k_B \approx 1.4 \cdot 10^{-23} J/K$ .

Theorem 3. If there is a finite set of possible energies

$$\lim_{T \rightarrow 0} P[E_x] = 1$$

if and only if  $E_x$  is globally minimal.

⇒ If we cool infinitely slowly we get a perfect crystal.

10.2.2 Translation into a Minimization Problem

energy	$E_x \longleftrightarrow f(\mathbf{x})$	objective function
system state	$\mathbf{x} \longleftrightarrow \mathbf{x}$	feasible solution
Boltzmann distribution	$\longleftrightarrow$	Boltzmann distribution (!?)
temperature	$T \longleftrightarrow T$	a parameter in the select

Annealing Selection Rule

pick  $\mathbf{x}'$  from  $\mathcal{N}(\mathbf{x}) \cap \mathcal{L}$  uniformly at random

Annealing Acceptance Rule

with probability  $\min(1, \exp(\frac{f(\mathbf{x}) - f(\mathbf{x}')}{T}))$  do  $\mathbf{x} := \mathbf{x}'$

Theorem 4. Consider the graph

$G = (\mathcal{L}, E)$  where  $E = \{(\mathbf{x}, \mathbf{y}) : \mathbf{y} \in \mathcal{N}(\mathbf{x})\}$ .

If  $G$  is regular symmetric and strongly connected

then for a fixed temperature  $T$ , local search with the above simulated annealing rule approaches a state with the Boltzmann distribution

- The proof of Theorem 2 uses the theory of **Markov Chains**.
- **Regularity** and **symmetry** are unrealistic. Lifting them destroys the Boltzmann distribution yet still

$$\lim_{T \rightarrow 0} \mathbf{P}[E_{\mathbf{x}}] = 1$$

- Two infinities (Equilibrium, cooling) can be removed. For example, Traveling Salesman in time  $\mathcal{O}(n^{n^{2n-1}})$ . Compare this to  $n!$  for brute force.

## 10.2.5 An Example:

### Graph Coloring by Simulated Annealing

#### Penalty Function Annealing

##### Neighborhood

- Choose a color  $j$  used somewhere uniformly at random
- Pick a node  $v$  colored  $j$  uniformly at random
- Pick another color  $j'$  uniformly from the used colors and one unused color

Problem: This is rarely feasible

Solution: Move coloring constraint into the objective function

**Neighborhood:** As in all local Search routines

**Selection from  $\mathcal{N}$ :** Possibly nonuniform

**Starting Temperature:** As **small** as possible such that **acceptance rate** is still **high**

**Equilibrium Detection:** #steps  $\gg |\mathcal{N}|, \dots$

**Cooling:** Multiply  $T$  with some constant typically 0.8 . . . 0.99.

**Termination:**  $T \approx \min\{|f(\mathbf{x}) - f(\mathbf{x}')| : \mathbf{x}, \mathbf{x}' \in \mathcal{L}\} \Rightarrow T := 0$   
basically local search

Setting the parameters is witchcraft.

**Dynamic Schedules:** Try to set make decisions adaptively.

## The Penalty Function

Color class  $i$ :  $C_i = \{v \in V : \mathbf{x}(v) = i\}$  Illegal edges:

$$E_i = \{(u, v) \in E \cup C_i \times C_i\}$$

$$f(\mathbf{x}) = 2 \sum_i |C_i| \cdot |E_i| - \sum_i |C_i|^2.$$

Observation: Every local minimum is a feasible coloring

Reason: Assume  $C$  contains an illegal edge  $(u, v)$  in  $\mathbf{x}$ . Color  $v$  with a **fresh color** in  $\mathbf{x}'$ .

$$\begin{aligned} f(\mathbf{x}) - f(\mathbf{x}') &= (2|C||E| - |C|^2) - \\ &\quad (2(|C| - 1)(|E| - 1) - (|C| - 1)^2) + 1 \\ &= 2|E| \end{aligned}$$

Continue recoloring until a legal coloring is reached

## Fixed $K$ Annealing

Guess a number of colors  $K$  (e.g., binary search)

$\mathcal{L}$  = Any  $K$  color assignment  $\mathbf{x} : V \rightarrow \{1, \dots, K\}$ .

Neighborhood: Change a color.

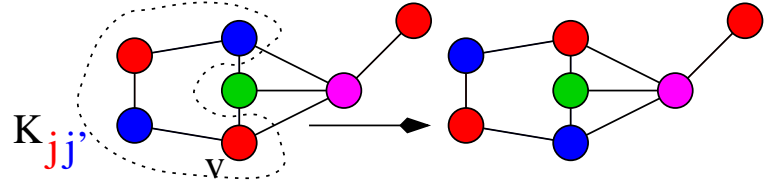
$$f(\mathbf{x}) = |\{(u, v) \in E : \mathbf{x}(u) = \mathbf{x}(v)\}|$$

## Kempe Chain Annealing

$\mathcal{L}$  = all legal colorings.

Neighborhood:

- Pick colors  $j, j'$  and node  $v$  as before.
- Let  $K_{jj'}$  denote the maximal connected component containing  $v$  and nodes colored  $j$  and  $j'$ .  $K_{jj'}$  is called a **Kempe Chain**.
- Swap colors  $j \leftrightarrow j'$  in  $K_{jj'}$ .
- $f(\mathbf{x}) = -\sum_i |C_i|^2$ .



## DSATUR: A Simple Greedy Heuristics

**Decisions:** Color a node

**Selection:** A node that maximizes the number of edges to nodes already colored. Break ties **randomly**

**Color Choice:** Smallest legal color

## XRLF: An Expensive Greedy Heuristics

**Definition 4.** Given a graph  $(V, E)$ ,  $U \subseteq V$  is an **independent set** if  $(U \times U) \cap E = \emptyset$ .

**Decisions:** Find a color class

**Selection:** Find a large independent set of uncolored nodes

Finding maximum independent sets is NP-hard.

**Definition 5.**  $\mathcal{G}(n, p)$  is the probability distribution over  $n$  node graphs where edge  $\{u, v\}$  is present with probability  $p$  independent of all the other node pairs.

$\mathcal{G}(1000, 0.5)$  XRLF  $\succ$  Kempe chain  $\succ$  penalty function

$\mathcal{G}(1000, 0.9)$  Kempe chain  $\succ$  XRLF  $\succ$  penalty function

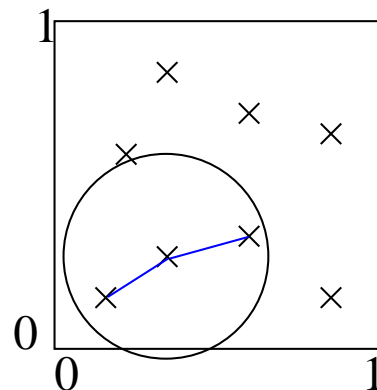
$\mathcal{G}(1000, 0.1)$  Fixed  $K$  annealing  $\succ$  XRLF  $\succ$  Kempe chain

DSATUR is almost always fastest and almost always worst.

Pick  $n$  nodes uniformly at random from  $[0, 1] \times [0, 1]$ .

$$E := \{(u, v) : \|u - v\|_2 \leq d\}$$

For large  $d$  (repeated) DSATUR is best.



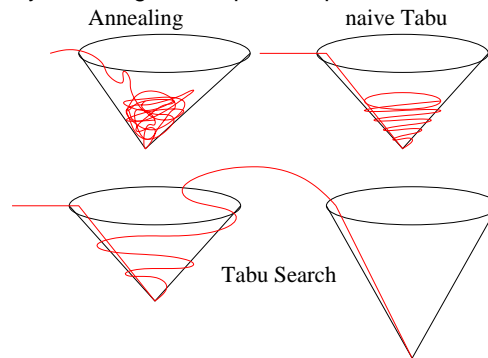
**Summary**

- Simulated Annealing has the potential to yield improved solutions at the expense of a lot of time without much problem specific insight
- Simulated Annealing usually better than repeated simple greedy algorithms
- There are many ways to spend a lot of time
- Results (at least) depend on
  - Problem instance family
  - Implementation details
  - Parameter tuning

**10.3 Tabu Search**

**The Idea**

Annealing may take long to escape from pits.



First try: Avoid cycles of length  $k$  by forbidding the last  $k$  states.

Second try: Forbid local properties.

Example: Tabu list entries in graph coloring could be pairs (Node, Color)

- How to choose **initial solution**
- Candidate generator  $\mathcal{N}^* = \mathcal{N}(x) \cap \mathcal{L}$ . Take best  $x \in \mathcal{N}^*$  that passes the rules below
- **tabu conditions** that rule out elements in  $\mathcal{N}^*$ .
- **Aspiration conditions** that allow exceptions.  
Example: Improved solutions
- Temporary modifications of the objective function for **intensification** and **diversification**
- Stopping criteria

- Very similar application areas
- + More flexible
- Too flexible?
- + Alternates automatically between hill climbing in unexplored areas and strategies of escaping local minima
- Less appealing?

## 10.4 More on Local Search

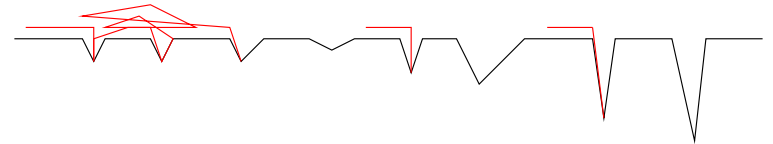
### 10.4.1 Threshold Acceptance

A simplified variant of Simulated Annealing

Accept new state iff  $f(x') < T$ .

### 10.4.2 Restarts

Run local search several times from random starting points.



## 11 A Preview: Evolutionary Algorithms

Translation of Biological terms into Optimization

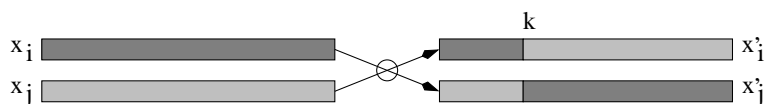
**Population:**  $N$  currently considered candidate solutions.  $N = 1 \Rightarrow$  local search

**Fitness:** objective function  $f$

**Mutation:** random local search operation.  $N > 1 \Rightarrow$  parallel local searches coupled by fitness

**Survival of the Fittest:** Removing solutions with low objective function value from the population.

**Reproduction by mating:** Producing a new candidate solutions with similarities to two ancestors



Peter Sanders: Optimization



75

### 12.1 A Scientific View

1. **Understand** the Problem
2. Is there an **immediate solution**?
3. Study the **literature** and related problems
4. Is there a promising **custom algorithm**?
5. Try to **rank** the **meta heuristics**
6. **Design experiments** carefully to **test hypotheses**
7. **Implement** and **compare** the most promising approaches. **Avoid Dogmas**
8. **Insight into the problem** before rote exhaustive experiments
9. Iterate

## 12 Summary

**How** should an Optimization Problem be **attacked**?

**Which** solution **technique** should be chosen?

Peter Sanders: Optimization



76

### 12.2 Economic Refinements

1. Really **understand** the problem
2. Try to use **existing Software**
3. Keep implementation **flexible**
4. **Prototype** using a very simple approach, e.g., greedy. Potentially reuse
5. Plan for a reasonable **solution** that will be **on time**
6. Plan some time for improvements
7. Further improvements in **later releases**
8. Leave risky approaches to **cooperations with academic institutions**
9. Smaller selection of alternative approaches than in science
10. Stop refining at **point of diminishing return**

## 12.3 Which Approach to Choose

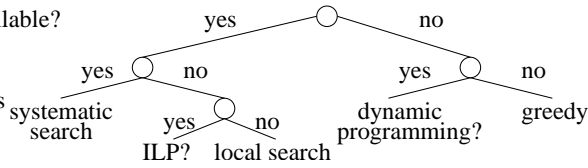
### 12.3.1 Summary of Pros and Cons

When custom algorithms and Literature Search have failed:

a lot of time available?

small instances?

optimal solutions needed?



**Greedy:** Simple and fast. Rarely optimal. Sometimes approximation guarantees

**Systematic Search:** Easy with tools e.g. constraint programming. Gives optimal solutions. Even clever implementations may only work for small inputs.

**Linear Programming:** Easy and reasonably fast. Optimal if the model fits. Rounding heuristics yield approximate solutions.

**Dynamic Programming:** Optimal solutions if subproblems are optimal and interchangeable. Space consuming.

**Integer Linear Programming:** Powerful tool for obtaining optimal solutions. Good formulations may require some know how.

Peter Sanders: Optimization



79

**Local Search:** Flexible and simple. Slow but gives good solutions to hard problems.

**Hill climbing:** Simple but haunted by local minima.

**Simulated Annealing and Tabu Search:** Powerful but slow and tuning can be messy.

**Evolutionary Algorithms:** Similar advantages and disadvantages as local search. Mating is more powerful but also slower and difficult to get right. Less target directed.