

## An example of an approximation algorithm

This is a simplified version of the problem in 11.1 in the course book.

Let us assume that we have  $n$  tasks with times  $t_1, t_2, \dots, t_n$  to complete. Let us say that we have two workers  $W_1$  and  $W_2$  and we want to distribute the task on them. Let  $T_i$  be the sum of the times of the tasks given to  $W_i$ . We want to distribute the workload evenly. The best would be if  $T_1 = T_2$ , but that might not be possible. So what we do is that we try to minimize  $T^* = \max(T_1, T_2)$ .

The crux is that if we could solve this problem efficiently, we can solve the NP-complete problem PARTITIONING efficiently, and we believe this is impossible. What we can do is that we can try to find an efficient way of getting an *approximation* of  $T^*$ .

We try to solve the problem by using a simple, greedy strategy. We give  $t_1$  to  $W_1$  and  $t_2$  to  $W_2$  and then continue giving each  $t_i$  to the  $W_j$  with least workload at that stage. Let us assume that  $T_{app}$  is the largest workload when all  $t_i$ :s have been distributed. This value is our approximation.

Obviously,  $T_{app} \geq T^*$ . Can we estimate how much larger  $T_{app}$  can be?

We can easily prove:

a.  $\frac{1}{2} \sum_i t_i \leq T^*$ .

b.  $t_i \leq T^*$  for all  $i$ .

Let us assume that  $T_1$  is the largest load when the algorithm terminates and that  $t_m$  is the last load added to  $W_1$ . Then we know that  $T_1 - t_m \leq \frac{1}{2}(T_1 + T_2) \leq T^*$  and  $t_m \leq T^*$ . This gives us  $T_{app} = (T_1 - t_m) + t_m \leq 2T^*$ .

$$T^* \leq T_{app} \leq 2T^*.$$

This estimate seems a bit pessimistic. If we first sort the  $t_i$ :s in decreasing order, we can get a tighter bound. If  $n \geq 3$  and  $m \geq 3$  it is then easy to prove that  $t_m \leq \frac{1}{2}T^*$ . If we use this in our previous estimate we get  $(T_1 - t_m) + t_m \leq T^* + \frac{1}{2}T^*$ . This gives us

$$T^* \leq T_{app} \leq \frac{3}{2}T^*.$$

## Approximation Algorithms

Many of the NP-Complete problems are most naturally expressed as optimization problems: TSP, Graph Coloring, Vertex Cover etc.

It is widely believed That  $\mathbf{P} \neq \mathbf{NP}$  so that it is impossible to solve the problems in polynomial time.

An *approximation algorithm* for solving an optimization problem corresponding to a decision problem in NP is an algorithm which in polynomial time finds an approximative solution which is guaranteed to be close to the optimal solution.

## Approximation of Vertex Cover

ApproxVertexCover( $G = (V, E)$ )

- (1)  $C \leftarrow \emptyset$
- (2) **while**  $E \neq \emptyset$
- (3)     Choose an arbitrary edge  
           $(u, v) \in E$
- (4)      $C \leftarrow C \cup \{u\} \cup \{v\}$
- (5)     Remove all edges in  $E$  which  
          contains  $u$  or  $v$
- (6) **return**  $C$

The algorithm always returns a vertex cover. When an edge is removed both of its vertices are added to  $C$ .

Now consider the edge  $(u, v)$ . At least one of the vertices  $u$  and  $v$  must be in an optimal vertex cover.

$\Rightarrow$  The vertex cover returned by the algorithm cannot be more than twice the size of an optimal vertex cover.

Time-complexity:  $O(|E|)$

## To measure approximability

*The Approximation Quotient* for an algorithm is

$\max \frac{approx}{opt}$  for minimization problems

$\max \frac{opt}{approx}$  for maximization problems

This means that the quotient is always  $\geq 1$  with equality if the algorithm always returns the optimal solution.

In all other cases the quotient is a measure of how far from the optimal solution we can get in the worst case.

The algorithm for finding minimal vertex covers has approximation quotient 2 since it returns a vertex cover at most twice as large as the minimal one.

## Degrees of approximability

There is a difference between the NP-Complete problems regarding how hard they are to approximate:

- For some problems you can, for every  $\epsilon > 0$ , find a polynomial algorithm with approximation quotient  $1 + \epsilon$ .

Ex.: The Knapsack Problem

- Other problems can be approximated within a constant  $> 1$  but not arbitrarily close to 1 **P  $\neq$  NP**.

Ex.: Vertex Cover

- Then there are problems that cannot be approximated within any constant if **P  $\neq$  NP**.

Ex.: Maximal Clique

## Approximation of TSP

We show that  $\text{TSP} \notin \text{APX}$ , i.e. TSP cannot be approximated. Assume, to reach a contradiction, that TSP can be approximated within a factor  $B$ .

Reduction from Hamiltonian Cycle:

Hamiltoncykel( $G$ )

- (1)  $n \leftarrow |V|$
- (2) **foreach**  $(v_i, v_j) \in E$
- (3)      $w(p_i, p_j) \leftarrow 1$
- (4) **foreach**  $(v_i, v_j) \notin E$
- (5)      $w(p_i, p_j) \leftarrow |V|B$
- (6) **if**  $\text{TSAPPROX}(p_i, t) \leq |V|B$
- (7)     **return** TRUE
- (8) **return** FALSE

If TSAPPROX can approximate TSP within factor  $B$ , then the algorithm decides in polynomial time if there is a Hamiltonian Cycle in  $G$  or not. That is impossible!



## Approximation of TSP with the triangle inequality

This is a special case of TSP which can be approximated.

**The triangle inequality:**  $w(i, j) \leq w(i, k) + w(k, j)$  for all nodes  $i, j, k$ .

The triangle inequality shows that if  $i, j, k_1, k_2, \dots, k_s$  form a cycle in the graph, we have  $w(i, j) \leq w(i, k_s) + w(k_s, k_{s-1}) + \dots + w(k_1, j)$ .

TSP with the triangle inequality is called  $\Delta$  TSP.

**Theorem:**  $\Delta$  TSP is NP- Complete.

Assume that we have a minimal spanning tree  $T$  in the graph. If we go back and forth along the edges in  $T$  we get a *walk* of length  $2w(T)$  where  $w(T)$  is the weight sum of the edges in  $T$ . This walk of course is no solution to the TSP-problem since it is not a cycle. Now, let  $C$  be an optimal cycle.

$w(C) = OPT$ . Since  $C$  is a spanning tree + an edge, we get  $w(T) \leq w(C)$ .

$$2 \cdot w(T) \leq 2 \cdot w(C) \leq 2 \cdot OPT$$

We can rearrange the walk along the tree  $T$  to a cycle  $C_1$  by visiting the nodes in the order that is given by the *inorder* ordering of the nodes in the tree.

**Claim:**  $w(C_1) \leq 2 \cdot w(T)$

This can be shown by repeated use of the triangle inequality.

We now get:

$$w(C) \leq w(C_1) \leq 2 \cdot w(T) \leq 2 \cdot w(C)$$

we set  $APP = w(C_1)$ . We then get:

$$OPT \leq APP \leq 2 \cdot OPT$$

We can compute  $APP$  in polynomial time. The approximation quotient is  $B = 2$ .

There are more advanced algorithms for approximation of  $\Delta$  TSP. One is *Christofides algorithm*. It uses the same ideas as our algorithm but has an approximation quotient  $\frac{3}{2}$ .