Answers to some of the exercises.

Chapter 2.

Ex.2.1 (a) There are several ways to do this. Here is one possibility. The idea is to apply the k-center algorithm first to D and then for each center in D to choose its nearest supplier.

First some notation. Let OPT be the optimal value. For any $d_i \in D$ let $f_i \in F$ be its nearest supplier. (If there are more nearest suppliers then select any of these as f_i.) Clearly, we must have $dist(d_i, f_i) \leq OPT$ for any i.

If $|D| \leq k$ then taking $S = \{f_i \mid d_i \in D\}$ gives an optimal solution. So from now assume that $|D| > k$.

Algorithm : Apply the k-center algorithm to the set D of customers. This gives a set $D' = \{d_1, d_2, \ldots, d_k\} \subseteq D$ of k customers. Take $S = \{f_i \mid d_i \in D'\}$ as solution.

We prove that this gives a 3-approximation. Consider an arbitrary customer $v \in D$. If there is a customer $d_i \in D'$ at distance at most 2Opt from v, then by the triangle inequality $dist(v, f_i) \leqslant dist(v, d_i) + dist(d_i, f_i) \leqslant 2\textsc{Opt} + \textsc{Opt} = 3\textsc{Opt}$. Now assume there is no customer $d_i \in D'$ at distance at most 2Opt from v. Then, by construction of the algorithm, for any two customers in $d_i, d_j \in D'$ we have $dist(d_i, d_j) > 2\textsc{Opt}$. But then the set $D' \cup \{v\}$ consists of $k+1$ customers such that any pair is at distance strictly more then 2Opt. This is impossible since in that case at least $k+1$ suppliers are needed for a solution with value Opt.

(b) The idea is to adjust the proof that was given for the k-center problem (see book Theorem 2.4). We reduce from the Dominating Set problem. First we define the reduction and then we prove that it is correct. Given an instance $G = (V, E)$ of the dominating set problem we define an instance of the k-suppliers problem as follows. For each $v \in V$ there is a customer d_v and supplier f_v. (See Figure 1.) That means, we have $|V|$ customers and $|V|$ suppliers in total. Now we define all distances. Let $dist(f_v, f_w) = dist(d_v, d_w) = 2$ for each pair $v, w \in V$. Let $dist(d_v, f_w) = 1$ if $v = w$ or if $(v, w) \in E$ and let it be 3 otherwise. It is easy to see that the triangle inequality is satisfied.

Note that any solution for this instance of the k-suppliers problem has value either 1 or 3. Assume that there is a dominating set S of size k. Now take as solution for the k suppliers instance exactly the suppliers that correspond with the vertices in S. Then, the value of this solution is 1. For the other direction, assume that there is a solution for the k suppliers instance of value 1. Then take as dominating set S the vertices that correspond with the k suppliers. This is a dominating set. We showed that there is a dominating set of size k if and only if the optimal value of the corresponding k suppliers instance is 1. Now, since the optimal value is either 1 or 3 this also gives us the lower bound of 3 on the approximation ratio. This works as follows.

Assume we have an α-approximation algorithm Alg for the k-suppliers problem with $\alpha < 3$. We shall prove that we can use such an algorithm to solve the dominating set problem. Assume we are given an instance G, k of the dominat-

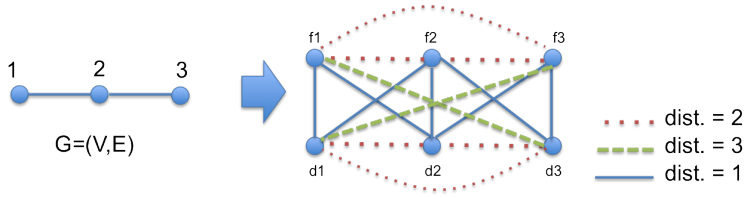


Figure 1: The reduction of Exercise 2.1(b).

ing set problem. That means, we want to compute if there exists a dominating set with at most k vertices. Now we reduce it to the instance of the k-suppliers problem as defined above and we apply the α-approximation algorithm. If there is a dominating set of value at most k then, as argued above, the optimal value of the k-suppliers instance is 1 ($\textsc{Opt}_k$=1) and the answer given by the algorithm will be at most $\alpha \cdot 1 < 3$. If, on the other hand, there is no dominating set of value at most k then the optimal value of the k-suppliers instance is 3 and the answer given by the algorithm will be (at least) 3. Therefore, by looking at the output of the algorithm we know if there is a dominating set of size at most k. However, we know that there is no polynomial time algorithm that solves the dominating set problem, assuming that $\mathcal{P} \neq \mathcal{NP}$. We conclude that no α-approximation algorithm with $\alpha < 3$ exists, assuming that $\mathcal{P} \neq \mathcal{NP}$.

- $\textsc{Opt}_{DS} \leq k \Rightarrow \textsc{Opt}_k = 1 \Rightarrow \textsc{Alg} \leq \alpha \textsc{Opt}_k < 3 \Rightarrow \textsc{Alg} = 1$.
- $\textsc{Opt}_{DS} > k \Rightarrow \textsc{Opt}_k = 3 \Rightarrow \textsc{Alg} = 3$.

Ex. 2.2 Let n be the number of jobs. Since $p_j \geq \textsc{Opt}/3$ for each job, there are at most two jobs on each machine in the optimal schedule.

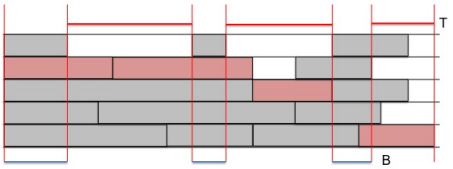
First we prove that in the LPT-schedule there are also at most two jobs per machine. Assume that LPT schedules three or more jobs on some machine i. Then the start time of the last job on machine i is larger than $2\textsc{Opt}/3$. But then, any machine must have a load strictly larger than $2\textsc{Opt}/3$ since otherwise LPT would have scheduled the last job of machine i on a different machine. Let k be the number machines in the LPT schedule with exactly one job. Then the number of jobs is $n \geq k + (m - k - 1)2 + 3 = 2m - k + 1$. On the other hand, these k jobs have a processing time more than $2\textsc{Opt}/3$ as argued, and are therefore also each scheduled alone on some machine in the optimal schedule. From this we conclude that the number of jobs is $n \leq k + 2(m - k) = 2m - k$. A contradiction! We conclude that LPT schedules at most two jobs per machine.

Let $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_n$. If $n \leqslant m$ then LPT schedules at most one job per machine and this is optimal. Now assume that $n = 2m - k$ and $0 \leqslant k \leqslant m - 1$. In an optimal schedule exactly k machines process a single job and $m - k$ machines receive two jobs. Clearly, it is optimal to schedule the longest k jobs each on its own machine. For the other jobs it is optimal to make pairs $(k + 1, n), (k + 2, n - 1), \ldots, (m, m + 1)$ and this is precisely what LPT does given that it schedules no more than two jobs per machine.

Ex.2.3 The lower bound (2.4) from section 2.3 still applies:

$$\textsc{Opt} \geqslant \sum_{j=1}^{m} p_j/m.$$

We say that a set of jobs $j_1, \ldots, j_k$ forms a *chain* if $j_1 \prec j_2 \prec \ldots, \prec j_k$. Then, another lower bound on the optimal value is the maximum sum of processing times in any chain. Let Q be this maximum: $\textsc{Opt} \geqslant Q$.



Consider a schedule produced by the list scheduling algorithm and let $C_{\max}$ be its length. Assume that at moment $t_1 < C_{\max}$, not all m machines are processing jobs, that means, some machines are idle at time t_1. Then any job that is available at time t_1 is being processed at that moment. Assume that also at t_2 with $t_1 < t_2 < C_{max}$, not all machines are busy. Consider an arbitrary job j_2 that is processed at time t_2. Then at time t_1, either job j_2 is processed or some other job j_1 is processed with $j_1 \prec j_2$. Now let T be the total time that at least one (and at most $m-1$) machine is idle. Then, by the reasoning above, we can find a chain of total processing time at least T. Hence, $\textsc{Opt} \geqslant Q \geqslant T$. Let B be the total time that all machines are busy. Then $B \leq \sum_j p_j/m \leq \textsc{Opt}$.

$$C_{\max} = T + B \leq 2\textsc{Opt}.$$

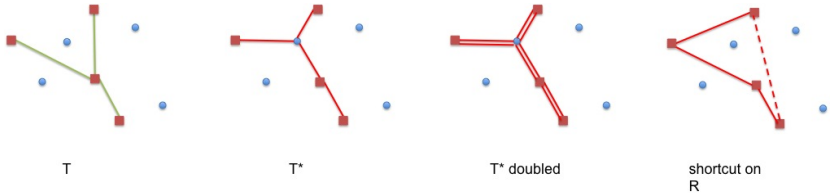
Ex. 2.4 (Note that this exercise gives a 2-approximation for the problem on *related* machines which is a more general model than the identical machine model for which we already know that list scheduling gives a 2-approximation.)

(a) This can be done by binary search. Let $P = \sum_j p_j$. The exercise does not mention that all processing times and speeds are integer but lets assume that this is the case. How many different values $C_{\max}$ can we have? The length of the schedule on a machine i is K/s_i for some $K \in \{0, 1, \ldots, P\}$. That means, there are at most $P+1$ different values for the length on machine i. Apply binary search as in Section 3.2. To make it easier we may first guess the machine that determines the optimum makespan (that means, we do the binary search for all $i = 1, \ldots, m$.) If we assume that machine i determines the optimal makespan then $\textsc{Opt}$ must be a multiple of $1/s_i$. Maintain a lower bound L and an upper bound U such that $L \leqslant \textsc{Opt}$ and such that we have a schedule of length at most ρU. Start with $L = 1/s_i$ and $U = P/s_i$. Now take $T = (L+U)/2$ and round it down to the nearest multiple of $1/s_i$. Apply the ρ-relaxed decision procedure. If it produces a schedule then set $U \leftarrow T$, otherwise let $L = T + 1/s_i$. This continues until $L = U$. The number of steps for each i is $O(\log P)$ which is polynomial in the input size.

(b) If all jobs are scheduled by this procedure then the length of the schedule is no more than $2D$ since no job starts after time D and the algorithm will only place a job j on machine i if $p_j/s_i \leqslant D$. (Note that if a job has label i than it may be processed by a machine $i' < i$ but not by a machine $i' > i$.)

Assume now that not all jobs are scheduled by this algorithm. We prove that $\textsc{Opt} > D$ in that case. Assume that job j is not scheduled by the algorithm and let it be labeled i. Then, any machine $i' \leqslant i$ must be fully loaded, that means it has no idle time between time 0 and D. Moreover, any job scheduled on these machines must have a label $i' \leqslant i$ (since the algorithm gives preference to job j above some job of label $i' > i$). Let S be the set of jobs of label at most i. In any schedule with makespan D all these jobs are scheduled on the first i machines. But this is impossible since, as argued above, the total processing time of all jobs in S is at least $iD + p_j > iD$.

Ex. 2.5 (a) Let T be the tree found by the algorithm and let T^* be the optimal Steiner tree. We need to show that $Cost(T) \leqslant 2Cost(T^*)$. Consider



the optimal tree T^*. Double its edges. We obtain a TSP-tour on R by making an Euler tour in this double tree and then shortcut the edges. (Just like we did in the double tree algorithm for TSP). The length of the resulting tour is most $2Cost(T^*)$. By removing one edge from the TSP-tour we get a tree (actually even a TSP-path) on R. The cost of this tree is at least the cost of the computed optimal tree T since T is a minimal tree. We conclude that $Cost(T) \leqslant 2Cost(T^*)$.

(b) Let $\hat{T}^*$ be the optimal Steiner tree using the costs c'_{ij}. Then, from (a) we know that $Cost(T') \leqslant 2Cost(\hat{T}^*)$. We will next show that

$$Cost(T) \leq Cost(T') \text{ and } Cost(\hat{T}^*) \leq Cost(T^*).$$

In the transformation from T' to T, each time an edge (i, j) is replaced by a path from i to j whose total length is exactly the cost of edge (i, j). This results in a connected graph of the same cost. This graph contains a spanning tree T of cost at most $Cost(T')$.

To see that $Cost(\hat{T}^*) \leq Cost(T^*)$ note that $c'_{ij} \leq c_{ij}$ for any edge $(i, j) \in E$. Hence,

$$Cost(T) \leqslant Cost(T') \leqslant 2Cost(\hat{T}^*) \leqslant 2Cost(T^*).$$

Ex.2.6 In Section 2.6 (Theorem 2.18) it was shown that the Minimum Degree Spanning Tree problem (MDST)is NP-complete. The proof was done by a

reduction from the Hamiltonian Path problem (HP). The HP problem is defined as follows: Given a graph $G = (V, E)$, is there a path that goes through every vertex exactly once? It is easy to see that the question if a graph has a spanning tree with maximum degree at most two is the same as the Hamiltonian path problem.

Assume that we have an α-approximation algorithm with $\alpha < 3/2$. Let graph G be an instance of the HP problem. We apply the algorithm to G. If G has a Hamiltonian path then the algorithm will output a value at most $\alpha \cdot 2 < 3$. If there is no Hamiltonian path then the algorithm will output a value at least 3. Therefore, such an algorithm can be used to solve the HP problem in polynomial time, which is impossible assuming that $\mathcal{P} \neq \mathcal{NP}$.