max planck institut informatik

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Antonios Antoniadis and Marvin Künnemann **Winter 2018/19**

Exercises for Randomized and Approximation Algorithms

www.mpi-inf.mpg.de/departments/algorithms-complexity/teaching/winter18/rand-apx-algo/

Exercise Sheet 3: Local Search Algorithms

To be handed in by **November 6th, 2018** via e-mail to André Nusser (CC to Antonios Antoniadis and Marvin Künnemann)

Exercise 1 *(10 Points)*

Prove the following lemma *(Lemma 2.8 from the book)*.

Lemma. *For any input to the problem of minimizing the makespan on identical parallel machines for which the processing requirement for each job is more than one-third the optimal makespan, the longest processing time rule computes an optimal schedule.*

Recall that the longest processing time (LPT) rule performs list scheduling, but instead of assigning the jobs in an arbitrary order it does so by always assigning the job with the longest processing requirement amongst the remaining jobs.

Exercise 2 *(10 Points)*

Consider the problem of scheduling jobs on identical machines, as seen in the lecture, but where the jobs are subject to *precedence constraints*. We say that $i \prec j$ if in any feasible schedule, job i must be completely processed before the processing of job j begins. We slightly modify the list scheduling algorithm so that instead of picking an arbitrary job to assign it picks an arbitrary *available* job, where a job j is said to be available if all jobs i such that $i \prec j$ have already been completely processed.

Prove that this algorithm is a 2-approximation algorithm for the problem of scheduling jobs on identical machines with precedence constraints.

Exercise 3 *(20 Points)* Consider the following local search algorithm for the Max Cut problem.

Algorithm: Local Search Max Cut

· Start with some arbitrary partition/cut (A, B) of the vertex set.
· Pick an arbitrary vertex v and move it from its current side of the partition to the other side if this increases the value of the cut.
· If no such vertex v exists, return the current (last) partition.

(i) Prove that the cut output by the algorithm has weight at least $1/2$ times the weight of an optimal cut. You may assume that the edge weight are positive integers. *(8 Points)*

(ii) Argue about the runtime of the algorithm. Why is it not polynomial in the input size? Would it be polynomial time if all edges had unit weight? *(4 Points)*

(iii) Can you adapt the algorithm, so that it runs in polynomial time (at perhaps a slight loss in the approximation guarantee)?

Hint: Only move a vertex from its current side if the increase in the cut value is "big". Can you define "big" so that the algorithm runs in polynomial time and achieves an approximation ratio of $1/2 - \epsilon$ *for any given* $0 < \epsilon < 1/2$*?*

(8 Points)