TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Fachbereich Informatik
AG Algorithmen und Komplexität

Issue Date: 01.12.2014
Version: 2014-12-01 19:54

5th Exercise Sheet for Kombinatorische Algorithmen, WS 14/15

Hand In: *Until Monday, 08.12.2014, 12:00,*
deliver or email to Raphael (`reitzig@cs.uni-kl.de`*).*

Problem 7

2 + 2 points

Let $G = (V, E)$ be a simple graph with integral edge capacities[1] $c : E \to \mathbb{N}$.

Prove or disprove:

a) If all capacities are *even* numbers, i. e. $c(E) \subseteq 2\mathbb{N} = \{2n \mid n \in \mathbb{N}_0\}$, then there is a *maximal* (s, t)-flow f^* with even flow values only, i. e. $f^*(E) \subseteq 2\mathbb{N}$.

b) If all capacities are *odd* numbers, i. e. $c(E) \subseteq 2\mathbb{N} + 1 = \{2n + 1 \mid n \in \mathbb{N}_0\}$, then there is a *maximal* (s, t)-flow f^* with odd flow values only, i. e. $f^*(E) \subseteq 2\mathbb{N} + 1$.

Problem 8

3 points

In applications, we are often not interested in *maximal* flows but rather if a given network admits a certain (additional) amount of flow from certain sources to certain sinks. For example, consider a wastewater system to which we add certain amounts of water (per time) at storm drains and have to move it to treatment facilities.

Formally, we model this as a decision problem:

Feasible Flow

Input: Simple graph $G = (V, E)$ with capacities $c : E \to \mathbb{R}_{\geq 0}$ and *excess* $b : V \to \mathbb{R}$.

Question: Is there a *feasible* flow $f : E \to \mathbb{R}$ with

$$\forall\, v \in V \qquad b(v) \; + \sum_{\substack{e \in E \\ e=(u,v)}} f(e) \;=\; \sum_{\substack{e \in E \\ e=(v,u)}} f(e) \quad \text{and} \tag{1}$$

$$\forall\, e \in E \qquad 0 \leq f(e) \leq c(e) \quad ? \tag{2}$$

We call a node $v \in V$ with positive excess $b(v) > 0$ a *source* and one with negative excess $b(v) < 0$ – i. e. a node with *demand* – a *sink*.

Show that the Feasible Flow problem reduces to the Max-Flow problem. That is, describe an algorithm that solves Feasible Flow by calling an algorithm for Max-Flow as subroutine.

[1] Unless otherwise stated, we use *capacity* synonymous to upper capacity bound and assume that no lower capacity bounds are given, i. e. $l(e) = 0$ for all $e \in E$.

Problem 9

4 points

We consider a certain (class of) *scheduling* problem(s), i. e. the task of assigning "jobs" to "machines" on a discrete time scale so that all jobs finish on time, subject to certain constraints.

Multi-Machine Scheduling with Preemption (MMSP)

Input: Number $m \in \mathbb{N}$ and $T_j = (r_j, p_j, d_j) \in \mathbb{N}^3$ with $d_j \geq r_j + p_j$ for $j \in [1..n]$.

We call r_j the *release time*, p_j the *processing time* and d_j the *deadline* of *job* T_j.

Question: Is there a scheduling of jobs $T_1, \ldots, T_n$ on m identical machines $M_1, \ldots, M_m$, i. e. a mapping $S : \mathbb{N} \times [1..m] \to [0..n]$, which fulfills the following constraints?

i) No job starts early, i. e.

$$\forall\, j \in [1..n], k \in [1..m].\ t < r_j \implies S(t,k) \neq j\ .$$

ii) All jobs finish on time (and are not "overprocessed"), i. e.

$$\forall\, j \in [1..n].\ \sum_{t=r_j}^{d_j} \sum_{k=1}^{m} [S(t,k) = j] = p_j\ .$$

iii) At any given time, at most one machine can process the same job, i. e.

$$\forall\, j \in [1..n], t \in \mathbb{N}.\ \sum_{k=1}^{m} [S(t,k) = j] \leq 1\ .$$

iv) At any given time, every machine can process only one job, i. e. S is indeed a well-defined function.

Output: A schedule that is feasible in the above sense, if there are any.

Note that we have implicitly that

- the machines work synchronously in parallel,
- any machine can process any step of any job and
- jobs may be *preempted* without cost, i. e. processing of any (unfinished) job can be paused at any time and continued on any other machine;

these two properties in particular distinguish MMSP from other, harder scheduling problems.

a) Reduce MMSP to Max-Flow using (at most) logarithmic space (in addition to input and output). Less efficient solutions may yield partial credit.

Note: This is possible because Max-Flow is log-space complete in $\mathcal{P}$ [1].

b) Use the reduction from a) to develop an algorithm for MMSP. What is the (asymptotic) runtime of your algorithm?

References

[1] Leslie M. Goldschlager, Ralph A. Shaw, and John Staples. "The maximum flow problem is log space complete for P." In: *Theoretical Computer Science* 21.1 (1982), pp. 105–111. ISSN: 0304-3975. DOI: 10.1016/0304-3975(82)90092-5.