Algorithmics II (H)

Tutorial Exercises on Algorithms for "Hard" Problems

Outline Solutions

1. (a) Represent each course by a vertex, and join two vertices by an edge if the corresponding two courses overlap in time. A valid colouring of the graph represents a valid allocation of the courses to rooms, so the minimum number of rooms required is the smallest number of colours required to colour the vertices of the graph.

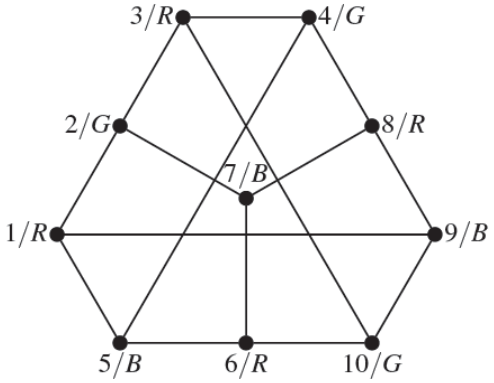
   (b) For an efficient algorithm, represent each course by a horizontal line on a timescale, with left hand end-point the start time and right hand end-point the finish time for the course. Sort the list L of end-points on their x-coordinates. In this sorted list, if two end-points (representing distinct lines) have the same x-coordinate, ties should be broken in favour of right hand, rather than left-hand, end-points. Form the room allocation as follows (assuming rooms are labelled 1, 2, . . .):

```
for each point p in L loop
   if p is a right-hand end-point, say of line x then
      release room allocated to x;
   else
      allocate first unoccupied room to x;
   end if;
end loop;
```

   It is clear that this algorithm gives an optimal solution, since the maximum number of rooms used is equal to the maximum number of simultaneously occurring classes.

   (c) This has no implication that P = NP; for this to follow, we would need to reduce graph colouring to the given problem, rather than the other way round.

2. Choosing a vertex s with a minimum number of available colours C_s keeps the branching factor low. Choosing a colour that reduces the available colours for the remaining vertices as little as possible may help to identify a colouring more quickly. Applying these rules yields a 3-colouring of the example graph without or with very little backtracking, a particular sequence of vertices and associated colours is shown to the right. The graph contains a cycle of length 5 and is therefore not 2-colourable, so the search can terminate once a 3-colouring has been found.



3. **3-Dimensional Matching**

Instance: 3 disjoint sets X, Y and Z of equal size n and a set S of triples of the form (x,y,z) where $x \in X$, $y \in Y$, $z \in Z$.
Question: does there exist a perfect matching, i.e., a subset M of S in which every element of X, Y and Z appears exactly once?

Example: $X = \{a,b,c,d,e\}$ $Y = \{1,2,3,4,5\}$ $Z = \{p,q,r,s,t\}$
$S = \{(a,2,r), (a,4,t), (a,5,q), (b,4,p), (b,5,s), (c,1,p), (c,2,t), (c,4,q), (d,2,q), (d,3,r), (d,5,q), (e,2,q), (e,2,t), (e,4,p), (e,5,q)\}$

This is a 'yes' instance – take $M = \{(a,4,t), (b,5,s), (c,1,p), (d,3,r), (e,2,q)\}$ (index values 1, 4, 5, 9, 11 chosen from S).

Assume S is represented as a global array of triples from $X \times Y \times Z$, indexed by $\{0,\ldots,t\text{-}1\}$, and recall that $|X| = |Y| = |Z| = n$. Assume `chosen` is a global `arrayList` of integers, representing the index values in S of the elements in the currently chosen subset of S.

For example: the set of triples M above would be represented by `chosen` $= \langle 1,4,5,9,11 \rangle$ (assuming indexing starts at zero).

If we represent a set of triples as a sequence of values in *increasing* order then it is not possible to extend a chosen set of r triples to a solution if the r^{th} index value of `chosen` (where r counts from 0) is $> r + t - n$. (We would need $n - r$ more triples, so the r^{th} value must be $\leq t - (n - r) = r + t - n$.)

For instance, in our example, if `chosen` $= \langle 4, 12, \ldots \rangle$ we would need 3 more index values but only two (13 and 14) are available. Here $n = 5$, $t = 15$ and $r = 1$.

```
/** recursive backtracking for 3D_Matching;
    Triple is a class representing triple, s is a global array of Triples,
    chosen is a global arrayList of int, matchingFound a global boolean */
private void choose(int r, int n) { // choose the r_th triple
  int next = 0;
  if (r > 0)
    next = chosen.get(r-1) + 1; // triples must form increasing sequence
  while (next <= r+s.length-n && !matchingFound){
    chosen.add(r,next); // choose next triple from set S
    if (isOkay(r))
      if (r == n-1)
        matchingFound = true;  // matching complete
      else
        choose(r+1,n);
    if (!matchingFound)
      chosen.remove(r);   // remove most recent triple
    next++;               // move on to next triple
  }
}

/** checks whether the last entry in the triples ArrayList is compatible
    with the earlier entries */
private boolean isOkay(int r) {
  for (int i=0; i < r; i++)
    if (s[chosen.get(r)].overlaps(s[chosen.get(i)]))
```

```
        return false;
    return true;
}
```

In the `Triple` class:

```
/** checks whether this triple overlaps triple t */
private boolean overlaps(Triple t) {
  return (isEqual(this.getFirst(), t.getFirst()) ||
          isEqual(this.getSecond(), t.getSecond()) ||
          isEqual(this.getThird(), t.getThird());
}
```

4. (a) KP-D may be specified as follows:

Instance: n items, where item i has weight w_i and profit p_i ($1 \le i \le n$); knapsack capacity C and target integer P.
Question: does there exist a subset S of $\{1,\ldots, n\}$ such that $\Sigma_{r \in S}\ w_r \le C$ and $\Sigma_{r \in S}\ p_r \ge P$?

(b) We prove that KP-D is NP-complete. Firstly, it is clear that KP-D belongs to NP. Now let $x_1, \ldots, x_n$ (a sequence of positive integers) and K (a target positive integer) be an instance I of SS. Let $w_i = p_i = x_i$ ($1 \le i \le n$) and $C=P=K$. This defines an instance J of KP-D, which may be constructed in time polynomial in the size of I. Clearly I has a "yes" answer if and only if J has a "yes" answer.

5. The following algorithm will output the items corresponding to the optimal profit value of a knapsack packing.

Let $i=n$ and let $j=C$. We then iterate the following process as long as $i>0$ and $j>0$. If $S(i,j)=S(i-1,j)$ then decrement i. Otherwise $S(i,j)=S(i-1,j-w_i)+p_i$, so decrease j by w_i units, output item i as being taken and decrement i.

Now suppose we have 4 items, with weights 2, 1, 3, 2 and profits 12, 10, 20, 15 respectively, and knapsack capacity 5. The dynamic programming table is as follows:

			Capacity j					
		i	0	1	2	3	4	5
weights	profits	0	0	0	0	0	0	0
2	12	1	0	0	12	12	12	12
1	10	2	0	10	12	22	22	22
3	20	3	0	10	12	22	30	32
2	15	4	0	10	15	25	30	37

$S(4,5) \ne S(3,5)$ so we take item 4 of weight 2 and profit 15. $S(3,3)=S(2,3)$ so we do not take item 3. $S(2,3) \ne S(1,3)$ so we take item 2 of weight 1 and profit 10. $S(1,2) \ne S(0,2)$ so we take item 1 of weight 2 and profit 12. Hence items 1, 2 and 4 are taken, of weights 2, 1, 2 (totalling 5) and profits 12, 10, 15 (totalling 37) respectively.

6. (a) Firstly assume that GOAL=max. The decision version Π_d of Π may be defined as follows:

Instance: any instance x of Π and a positive integer k
Question: is there a feasible solution $y \in \mathrm{SOL}(x)$ such that $m(x,y) \geq k$?

Secondly assume that GOAL=min. The decision version Π_d of Π may be defined as follows:

Instance: any instance x of Π and a positive integer k
Question: is there a feasible solution $y \in \mathrm{SOL}(x)$ such that $m(x,y) \leq k$?

(b) Suppose that $\Pi \in \mathrm{PO}$ and Π is NP-hard. The fact that $\Pi \in \mathrm{PO}$ implies that $\Pi_d \in \mathrm{P}$. For, suppose that A is a polynomial time algorithm which finds an optimal solution $y^* \in \mathrm{SOL}(x)$, given an instance x of Π. Then by comparing $m^*(x)=m(x,y^*)$ with k, we can decide in polynomial time whether there is a feasible solution $y \in \mathrm{SOL}(x)$ such that $m(x,y) \geq k$ (if GOAL=max) or such that $m(x,y) \leq k$ (if GOAL=min). Also, since Π is NP-hard, Π_d is NP-complete by definition. Thus P=NP, as required.

7. (a) Let $G = (V, E)$ be a bipartite graph. First we note that a minimum vertex cover cannot have size $< m$, where m is the size of a maximum matching. For let M be such a matching. Then any vertex cover must contain at least one of the end vertices of each edge in M.
We now show how to use a maximum matching M to give a minimum vertex cover of size $|M|$.
Let $V = U \cup W$ be a bipartition of the vertex set V. Relative to a maximum matching M we say that an edge is *reachable* if it lies on an alternating path that starts at an exposed vertex of U. We define the set S of vertices as follows: S contains one end vertex of each edge in M; if the edge is reachable its W-endpoint is in S, otherwise its U-endpoint is in S. Clearly $|S| = m$, so if we can show that S is a vertex cover it must be a minimum vertex cover.
Suppose not, i.e., suppose there is an edge $e = (u,w)$ such that neither u nor w is in S. Then edge e is not in M, but at least one of its endpoints must be matched (otherwise e can be added to the matching).
Case (i) Suppose w is matched in M, with $(x,w) \in M$. Then we must have $x \in S$, so edge (x,w) is not reachable. But (x,w) would be reachable (from u) if u were exposed, so u must be matched in M, say $(u,y) \in M$. But then (u,y) is not reachable (because (x,w) isn't) so that, by definition of S, $u \in S$ – a contradiction.
Case (ii) Suppose u is matched in M but w is not, and $(u,y) \in M$. Then we must have $y \in S$, so edge (u,y) is reachable. But then the alternating path from an exposed vertex in U that reaches edge (u,y), appended with edge e, is an augmenting path for M – a contradiction.
Finally, M can be computed in polynomial time, and a simple depth-first or breadth-first traversal will reveal which edges are reachable. Hence S can be found in polynomial time.

(b) Consider the following polynomial-time algorithm which constructs a set of vertices S in G:

```
S = ∅;
F = E;
while ( F != ∅ )
{   choose e={u,v}∈F;
    S = S ∪ {u, v};
    F = F \ {e};
    for ( each e' adjacent to e in F )
        F = F \ {e'};
}
```

Clearly S is a vertex cover in G. To show that the performance guarantee of 2 holds, let C be a minimum vertex cover in G. For every pair of vertices $\{v,w\}$ added to S at an iteration of the while loop, C must contain either v or w, since some vertex of C must cover the edge $\{v,w\}$. Thus $2|C| \geq |S|$ as required.

(c) A worst-case example for the above approximation algorithm is as follows. For any $n \geq 1$, let G_n be a path on $2n$ vertices (i.e. the vertices of G_n are $\{1,2,\ldots,2n\}$ and the edges of G_n are $\{\{i,i+1\} : 1 \leq i \leq 2n-1\}$). Clearly G_n is connected. The approximation algorithm might consider edges of G_n in the order $\{1,2\}$, $\{3,4\}$, $\{5,6\}$, …, $\{2n-1,2n\}$ and it could therefore arrive at the vertex cover $S_n=\{1,2,\ldots,2n\}$ of size $2n$. On the other hand, a minimum vertex cover in G_n is $\{2,4,\ldots,2n\}$ of size n.

8. (a) Let $\{u,v\} \in E$ and suppose that $u \in S$. Then A removes each vertex adjacent to u from X, including v. Hence $v \notin S$, so that S is independent.

(b) Let $c>1$ be any constant and let $k=\lfloor c \rfloor+1$. Construct a graph G with vertices $\{1,2,\ldots,k+1\}$ and edges $\{1, i\}$ $(2 \leq i \leq k+1)$. Algorithm A might choose vertex 1 first, terminating with the independent set $S=\{1\}$. However $T=\{2,\ldots,k+1\}$ is a maximum independent set in G, of size k. Hence $|T| = k > c = c.|S|$, so that A is not a c-approximation algorithm for MIS.

9. (a) Let G be a cycle on 3 vertices. Then G admits no independent vertex cover.

(b) A graph G admits an independent vertex cover if and only if it is bipartite. To see this, suppose that G is bipartite. Then the vertex set of G can be partitioned into two disjoint sets, namely U and W, so that every edge in G joins a vertex in U to a vertex in W. It may be verified that U is an independent vertex cover of G. Conversely if G admits an odd cycle then it is straightforward to verify that G can have no independent vertex cover. Depth-first search can be used to determine whether a graph is bipartite.

(c) Let G be a path on 3 vertices, namely u, v and w. Then $S_1=\{v\}$ is an independent vertex cover, and so is $S_2=\{u,w\}$.

(d) As G is bipartite, the vertex set of G can be partitioned into two disjoint sets, namely U and W, so that every edge in G joins a vertex in U to a vertex in W. We claim that no independent vertex cover of G can have vertices in both U and W. For, suppose $\{u,w\} \subseteq S$ where $u \in U$ and $w \in W$. As G is connected, there is a path in G from u to w. But P must be of odd length, as G is bipartite. Then P comprises vertices $v_1,\ldots,v_k$, where $v_1=u$ and $v_k=w$, for some even value of k. But then as S is an independent vertex cover, every vertex of P with odd index must belong to S and hence $v_{k-1} \in S$, a contradiction. Thus U and W are the only possible independent vertex covers of G, and we output whichever is the larger.

10. For any $v \in V$, define the binary variable x_v, where $x_v=1$ if v belongs to the vertex cover, and 0 otherwise. The following is an IP model for MVC:

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in V} x_v \\ \text{subject to} \quad & x_u + x_v \geq 1 \forall \{u,v\} \in E \\ & x_v \in \{0,1\} \forall\ v \in V. \end{aligned}$$

11. Define the following variables:

- $x_{v,c} \in \{0,1\}$ for each vertex $v \in V$ and colour $c \in \{1,\ldots,K\}$.
  $x_{v,c} = 1$ if vertex v is assigned colour c.
- $y_c \in \{0,1\}$ for each colour $c \in \{1,\ldots,K\}$.
  $y_c = 1$ if colour c is used by at least one vertex.

The following is an IP model for MGC:

$$\text{minimize} \quad \sum_{c=1}^{K} y_c$$

$$\text{subject to} \quad \sum_{c=1}^{K} x_{v,c} = 1 \quad \forall v \in V \quad \text{(each vertex receives exactly one colour)}$$

$$x_{u,c} + x_{v,c} \leq 1 \quad \forall \{u,v\} \in E,\ \forall c = 1,\ldots,K \quad \text{(adjacent vertices cannot share a colour)}$$

$$x_{v,c} \leq y_c \quad \forall v \in V,\ c = 1,\ldots,K \quad \text{(if a vertex uses colour } c\text{, that colour is "used")}$$

$$x_{v,c} \in \{0,1\},\ y_c \in \{0,1\}$$

12. We define the decision version of SSO as follows:

Subset Sum Decision (SSD)
Instance: a collection of positive integers $a_1, a_2, \ldots, a_n$, and target values t, k
Question: is there a subset S of $\{1,\ldots,n\}$ such that $k \leq \sum_{i \in S} a_i \leq t$?

In order to show that SSO is NP-hard, it suffices to show that SSD is NP-complete. But SS is NP-complete. If we restrict SSD to the case that $t=k$, then we obtain the problem SS. Hence by restriction, it follows that SSD is NP-complete.

13. Here n=6, so that for a solution with measure at least one third of optimal, we have $\varepsilon = 2$ and $\varepsilon/2n = 1/6$. Thus $\delta = 1/6$ and $1-\delta = 5/6$.

The set S at each iteration in the main loop is as follows (elements marked by * are trimmed, and values > 26 are automatically excluded.)

i=1: S={ 0, 3 }
i=2: S={ 0, 3, 7, 10 }
i=3: S={ 0, 3, 7, 10, 13, 16, 20, 23* }
i=4: S={ 0, 3, 7, 10, 13, 15*, 16, 18*, 20, 22*, 25 }
i=5: S={ 0, 3, 7, 10, 13, 16, 19*, 20, 22*, 25, 26* }
i=6: S={ 0, 3, 7, 10, 13, 16, 20, 21*, 24*, 25 }

The largest element remaining in S is 25(=3+7+15), and hence this is returned by the algorithm. An optimal solution has measure 26(=7+19).

14. (a) Suppose we are given an instance of MBP, in which C denotes the bin capacity and S denotes the total size of all the items. Suppose that the optimal bin packing uses T bins. Then $T \geq \lceil S / C \rceil \geq S / C$. Now suppose that the bin packing produced by Next Fit uses N bins, and that X_i is the sum of the sizes of the items in bin i. We show that $N \leq 2T$, which establishes that Next Fit is a 2-approximation algorithm. Any two consecutive bins B_i, B_{i+1} in the bin packing produced by Next Fit must contain items of total weight > C, for otherwise the items in B_{i+1} would have been placed in B_i, a contradiction. Hence

$$S = (X_1 + X_2) + (X_3 + X_4) + \ldots + (X_{N-1} + X_N) > \lfloor N / 2 \rfloor C$$

Hence $\lfloor N / 2 \rfloor C < S \leq CT$, and which implies that $\lfloor N / 2 \rfloor < T$ and thus $N \leq 2T$, proving that Next Fit is a 2-approximation algorithm.

(b) Given a positive integer n, define an instance of MBP with n items of size n and n items of size 1, and define the bin capacity to be n. Algorithm Next Fit might consider items of weight n and 1 alternately, using a total of $2n$ bins. On the other hand, the optimal packing uses n+1 bins: the n items of size n are placed in n bins, and the n items of size 1 are placed in a single bin. Therefore, asymptotically Next Fit uses twice as many bins as the optimal packing.

15. (a) The same analysis as in Question 14(a) can be used. However we can also arrive at a performance guarantee of 2 by making deductions based on the behaviour of FF, as follows. With notation as in Question 14(a) we again have $T \geq \lceil S / C \rceil \geq S / C$. It is immediate that the sum of the sizes of items in any two bins is greater than C, otherwise one of the bins would not have been used. Summing over all pairs of bins gives

$$(X_1 + X_2) + (X_1 + X_3) + \ldots + (X_{N-1} + X_N) > N(N-1)/2 \; . \; C$$

i.e., $(N\text{-}1)\ S > CN(N\text{-}1)/2$.

It follows that $N < 2S/C \leq 2T$.

(b) Suppose there are $6m$ items of each of weights 19, 43, and 64, for some parameter m, and that they are presented in that order. The FF algorithm allocates the items of weight 19 to exactly m bins (6 items per bin), each with a total weight of 114, then the items of weight 43 to $3m$ bins (2 items per bin), each with a total weight of 86, then the items of weight 64 to $6m$ bins (1 item per bin). This gives $10m$ bins in total.

However, the optimal solution uses just $6m$ bins, each with one item of weight 19, one item of weight 43, and one item of weight 64 (total weight 126).