Solutions for Assignment 2

November 30, 2003

1. Algorithm DNF-SAT (ϕ)

```
1. For each clause C_i of φ do
2.     SAT ← TRUE
3.         For each literal l_j in C_i do
4.             if ¬l_j appears in C_i then
5.                 SAT ← FALSE
6.         end for
7.     if (SAT) then return 1
8. end for
9. return 0;
```

<u>RUNNING TIME</u>

Let m be the number of clauses and n be the number of variables in ϕ. The loop 1-8 runs at most m times. The loop 3-6 runs at most in n times. The rest in line 4 can be done in O(n). The running time of the algorithm is O(mn^2)

<u>CORRECTNESS OF THE ALGORITHM</u>

Recall that ϕ is formed as an "OR" of clauses that are "ANDs" of literals. Therefore ϕ is satisfiable if at least one of the clauses is satisfiable. For a clause to be satisfiable it is sufficient that it does not contain a literal and its negation, since their "AND" would never be satisfiable. The algorithm above just checks these properties.

2.

<u>part</u> 1) : IntProg = $\{< A, b >$: A is an $n \times m$ integer matrix, b is an m-vector of integers and there exists a vector $x \in \{0,1\}^n$ such that $Ax \geq b$ $\}$

Step 2: Idea for the Reduction : 3-CNF-SAT$\leq_p$ IntProg

Ex: $\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_3 \vee \neg x_4)$

Transform each clause into an inequality, transforming each literal into expressions x_i or $(1 - x_i)$ depending whether it is a variable or its negation that appears in the clause:

$(x_1 \vee x_2 \vee x_3)$: $\quad x_1 + x_2 + x_3 \geq 1$

$(x_1 \vee \neg x_3 \vee \neg x_4)$: $\quad x_1 + (1 - x_3) + (1 - x_{4)} \geq 1$ which is equivalent to $x_1 - x_3 - x_4 \geq -1$

So, the corresponding instance of IntProg is:

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & -1 & -1 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Step 3: Reduction Algorithm

```
Algorithm F(< φ >)
    Check whether φ is in 3-CNF format.
    If it is not, then return (< A = [1], b = [2] >);
    -let m be the number of clauses and n be the
    number of variables in φ;

    FOR i = 1 to m DO
        NUMNEGATED ← 0;
        FOR j = 1 TO m DO A[i, j] = 0;
        FOR EACH LITERAL l in C_i DO
            IF l = x_j THEN A[i, j] = A[i, j] + 1;
            ELSE IF l = ¬x_j THEN
                A[i, j] = A[i, j] − 1;
                NUMNEGATED++;
            END IF
        END FOR
        b[i] = 1 − NUMNEGATED;
    END FOR
    RETURN (< A, b >)
```

Step 4:

$< \phi > \in$ 3-CNF-SAT $\iff < A, b > \in$ IntProg:

if ϕ is not in 3-CNF format then $A = [1]$ and $b = [2]$, and the system $1x_1 \geq 2$ has no solution for $x_1 \in 0, 1$. It remains to look at the

case that ϕ is in 3-CNF format.
In this case, ϕ is satisfiable if and only if there exists a truth assignment to $x_1, x_2, ..., x_n$ such that each clause is satifiable.
It remains to show that this is true if and only if thre exists values $0, 1$ assigned to variables $x_1, x_2, ..., x_n$ such that $Ax \geq b$.
Let us analyze each clause C_i. C_i is satisfied by $x_1, x_2, ..., x_n$ if and only if at least one of its literals is assigned the value TRUE.
Let y_1, y_2, y_3 correspond to the truth values of each literal in C_i
Thus C_i is satisfied if and only if at least one of y_1, y_2, y_3 is equal to 1, which is equivalent to saying that $y_1 + y_2 + y_3 \geq 1$.
Now we relate the values of y_1, y_2, y_3 with the values of $x_1, x_2, ..., x_n$
Let $l_1, l_2, andl_3$ be the literals in C_i. Then if $l_j = x_{kj}$ then $y_i = x_{kj}$, but if $l_j = \neg x_{kj}$ then $y_i = 1 - x_{kj}$ since $y_1 = 1$ if $x_{kj} = 0$ and $y_1 = 1$ if $x_{kj} = 1$.
Substituting this into the equation $y_1 + y_2 + y_3 \geq 1$ we get $a_{k1}x_{k1} + a_{k2}x_{k2} + a_{k3}x_{k3} \geq 1 - n_i$ where

$$a_{kj} = \begin{cases} 1 & \text{if } l_i = x_{kj} \\ -1 & \text{if } l_j = \neg x_{kj} \end{cases}$$

and n_i is the number of variables in C_i that apear negated.
This ϕ is satisfiable if and only if there exists a 0-1 assignment to variables $x_1, ..., x_n$ such that $A_x \geq b$.

Step 5: F Runs in Polynomial Time:
Checking whether ϕ is in 3-CNF format can be done in linear time on the number of clauses.
The loop on i runs in m steps.
The loop on j runs in n steps.
The loop on the literals run in constant number of steps, sunce there are only 3 literals per clause. So the running time of F is $O(nm)$.

3.

Step 1: HAMPATH $\in NP$
CERTIFICATE: A sequence of vertices y
VERIFICATION: Check whether y is a hamiltonian path from u to v in G.

ALGORITHM $A(< G, u, v >, < y >)$

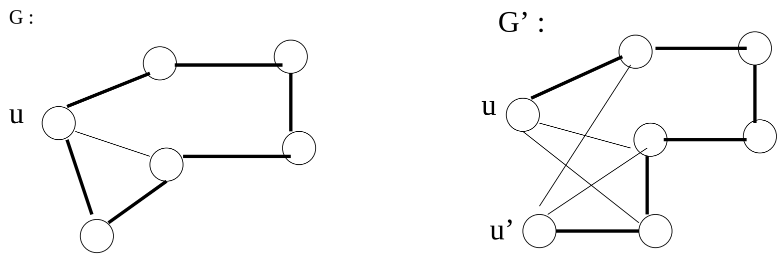
1. Check whether y has n vertices; if not return 0;
2. Check whether $y = (y_1, y_2, ..., y_n)$ has repeated vertices; if so return 0;
3. Check whether $\{y_i, y_{i+1}\} \in E$ for $i = 1, ..., n-1$ and whether $\{y_n, y_1\} \in E$. If some of the tests fail then return 0;
4. Check whether $y_1 = u$ and $y_n = v$. If not return 0; otherwise return 1.

A runs in polynomial time, since

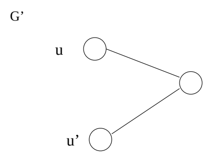
1. runs in $O(n)$ steps.
2. runs in $O(n)$ steps.
3. runs in $O(n)$ steps.
4. runs in $O(1)$ steps.

So A runs in $O(n)$ steps.
It is easy to see A is a correct verification algorithm for HAMPATH.
Step 2: HAMCYCLE $\leq_p$ HAMPATH

Idea for the reduction:
Given G, pick an arbitrary vertex u and create a new vertex u' connecting it to all the neighbours of u, in the new graph G'. A hamiltonian in G corresponds to a hamiltonian path from u to u' in G'.
Example:



The only case for which this reduction fails is G: $u \circ - \circ$
since G has no hamiltonian cycle but



has
a hamiltonian path between u and u'. Therefore we treat the case $\mid v \mid = 2$ separately in the algorithm.

Step 3 Reduction Algorithm:

Algorithm $F(< G >)$
  Let $G = (V, E)$.
  If $\mid V \mid = 2$ then return $< G' = (V' = \{u, v\}, E' = \phi), u, v >$
  select a vertex u in G.
  $V' \leftarrow V \cup \{u'\}$
  $E' \leftarrow E$
  For each v in V do
    if $\{u, v\} \in E$ then $E' \leftarrow E' \cup \{u', v\}$
  return $< G' = (V', E'), u, u' >$;

Step 4 The reduction works, that is:
G has a hamiltonian cycle if and only if G' has a hamiltonian path from u to u'.
($\Rightarrow$) Let $C = (v_1 = u, v_2, ..., v_n)$ be a hamiltonian cycle in G (we can assume $v_1 = u$ without loss of generality). It is easy to see that $(v_1 = u, v_2, v_3, ..., v_n, v_{n+1} = u')$ is a hamiltonian path between u and u' in G', since $(v_1, v_2, ..., v_n)$ are distinct vertices in G so $(v_1 = u, v_2, ..., v_n, v_{n+1} = u')$ are distinct vertice in G'; moreover if $\{v_i, v_{i+1}\} \in E, i = 1, ..., n,$ and $\{v_n, u\} \in E$ then $\{v_i, v_{i+1}\} \in E', i = 1, ..., n,$ and $\{v_n, u'\} \in E'$.
($\Leftarrow$) Let $P = (u = v_1, v_2, v_3, ..., v_n, v_{n+1} = u')$ be a hamiltonian path in G'. Then, $(v_1, v_2, ..., v_n, v_{n+1} = u')$ are distinct vertices in G', so $(v_1, v_2, ..., v_n)$ are distinct in G. Moreover, since $\{v_i, v_{i+1}\} \in E', i = 1, ..., n,$ and $\{u', v_1\} \in E'$
then we conclude $\{v_i, v_{i+1}\} \in E, i = 1, ..., n,$ and $\{u, v_1\} \in E'$.
Moreover, since $n > 2$, then $\{v_1, v_2\} \neq \{v_n, v_1\}$, so $(v_1, v_2, ..., v_n)$ is a hamiltonian cycle in G.

Step 5 F runs in Polynomial Time:
Copying G into G' takes time $O(n^2)$ where $n =\mid v \mid$. Creating u' and its incidence edges takes time in $O(n)$. Therefore, F runs in $O(n^2)$.

4.

Refer to Bellman Ford Algorithm in page 588 of the textbook.
The following is a decider for HAMPATHDIRACYCL:
ALGORITHM $A(< G, u, v >)$
    Let $G = (V, E)$, let $n =\mid V \mid$
    For each edge $(u, v) \in E$ do $w(u, v) = -1$;
    Result←Bellman-Ford(G, w, u);
    If (Result = false) then $//G$ is not directed acyclic
        Return 0;
    If $d[v] = -(n-1)$ then return 1;
    else return 0;
END ALGORITHM.

Note that Bellman-Ford returns true only if G does not contain directed negative cycles, which in our case of all weights being negative is equivalent to G being directed acyclic graph. If this algorithm return true, then $d[\cdot]$ contains the shortest distance between every vertex and the source vertex u.

The correctness of our algorithm is based on the following fact:

$\ll$ Let G be a directed acyclic graph. Then, there exists a hamiltonian path from u to v if and only if the shortest path between u and v, putting all edge weights equal to -1, has wight $-(n-1)$. $\gg$

Proof of the fact:

($\Rightarrow$) If G has a hamiltonian path from u to v then the weight of this path is $-(n-1)$, since a hamiltonian path has $n-1$ edges and each edge has weight -1.

Because there are no directed cycle in G, any other path must have distinct vertices, so the number of vertices is smaller than or equal to n, so its weight is $\geq -(n-1)$. So the shortest path in G has weight $-(n-1)$.

$\Leftarrow$ If G has no hamiltonian path from u to v then the number of vertices in a path from u to v is at most $n-2$. Thus its weight is at least $-(n-2) > -(n-1)$. So the shortest path from u to v has weight $> -(n-1)$.

Algorithm A runs in polynomial time:

Let $n = \mid V \mid$ and $m = \mid E \mid$.
Step 2 runs in $O(m)$.
Step 3 runs in $O(m \cdot n)$ (see textbook).
All other steps run in $O(1)$. So A runs in $O(m \cdot n)$.