CS4055: Mini-Project
Exercises on Branching Algorithms

Taha Adeel Mohammed CS20BTECH11052

(Kratsch - Branching algorithms exercises)

1. The HAMILTONIAN CIRCUIT problem can be solved in time $O^*(2^n)$ via dynamic programming or inclusion-exclusion. Construct a $O^*(3^{m/3})$ branching algorithm deciding whether a graph has a hamiltonian circuit, where m is the number of edges.

Below we propose an algorithm for solving the equivalent problem of finding a Hamiltonian path in a graph in time $O^*(3^{m/3})$. To find a Hamiltonian circuit, repeat the algorithm for each vertex as a starting point and check if the last vertex is connected to the first vertex, adding only a polynomial factor to the running time.

Algorithm:

function checkHamiltonianPath($G(V, E),\ s$)
  if $|V| = 1$ **then return** true
  if $|adj(s)| = 0$ **then return** false
  for each vertex v in $adj(s)$ **do**
    $G' = G(V - s, E - \{(s, adj(s))\})$
    if checkHamiltonianPath($G',\ v$) **then**
      return true
  return false

In the algorithm, if we have only one vertex, then it is a Hamiltonian path. Else if the vertex s has no adjacent vertices, then we cannot have a Hamiltonian path. Then for each adjacent vertex v of s, we remove the vertex s and all edges connected to it from the graph and recursively call the function with the new graph and the starting vertex v. If the function returns true for any of the adjacent vertices, then the graph has a Hamiltonian path. If the function returns false for all adjacent vertices, then the graph does not have a Hamiltonian path.

Running Time Analysis:

For a graph G with m edges and starting vertex s with degree d, we recursively call the function d times with graph G' with $m - d$ edges. Hence the running time is given by the recurrence relation:

$$T(m) \leq d \cdot T(m - d)$$

which has the solution $T(m) = d^{m/d}$. Since this function has its maximum value for integral d when d = 3, we obtain $T(m) \leq 3^{m/3}$. Hence the running time of the algorithm is $O^*(3^{m/3})$.

□

4. Construct a branching algorithm for the DOMINATING SET problem on graphs of maximum degree 3.

Below we propose a set of reduction and branching rules for the Dominating Set problem on graphs of maximum degree 3, covering all possible cases.

Let each node initially be assigned a state $S[u] = 0$. Let $S[u] = 1$ denote that the node u has a neighbour in the dominating set (and it itself is not in the dominating set), while $S[u] = 2$ denotes that the node u is in the dominating set. The DOMINATING SET problem requires us to find an assignment for the state of each node such that the number of nodes with $S[u] = 2$ is minimized and there are no nodes with $S[u] = 0$.

Now we consider the following cases based on the nodes degrees and their neighbours' states.

I. deg[u] = 0, adj[u] = $\emptyset$:
Case 1: $S[u] = 0$ *(Reduction Rule)*
Set $S[u] = 2$

II. deg[u] = 1, adj[u] = {v}:
Case 2: $S[u] = 0$, $S[v] \in \{0, 1\}$ *(Reduction Rule)*
Set $S[v] = 2$, $S[adj[v]] = 1$ (if S[adj[v]] = 0)

III. deg[u] = 2, adj[u] = $\{\mathbf{v_1}, \mathbf{v_2}\}$:
Case 3: $S[u] = 0$, $S[v_1] = 1$, $S[v_2] = 1$ *(Reduction Rule)*
Set $S[u] = 2$.
Case 4: wlog $S[u] \in \{0, 1\}$, $S[v_1] \in \{1, 2\}$, $S[v_2] = 0$ *(Reduction Rule)*
Set $S[v_2] = 2$, $S[adj[v_2]] = 1$.
Case 5: $S[u] = 0$, $S[v_1] = 0$, $S[v_2] = 0$ *(Branching Rule)*
(a) Set $S[u] = 2$, $S[adj[u]] = 1$
(b) Set $S[v_1] = 2$, $S[v_2] = 2$, $S[adj[v_1]] = 1$, $S[adj[v_2]] = 1$.

IV. deg[u] = 3, adj[u] = $\{\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}\}$:
Case 6: $S[u] = 0$, $S[v_1] = 1$, $S[v_2] = 1$, $S[v_3] = 1$ *(Reduction Rule)*
Set $S[u] = 2$.
Case 7: wlog $S[u] \in \{0, 1\}$, $S[v_1] \in \{1, 2\}$, $S[v_2] \in \{1, 2\}$, $S[v_3] = 0$ *(Reduction Rule)*
Set $S[v_3] = 2$, $S[adj[v_3]] = 1$.
Case 8: wlog $S[u] \in \{0, 1\}$, $S[v_1] \in \{1, 2\}$, $S[v_2] = 0$, $S[v_3] = 0$ *(Branching Rule)*
(a) Set $S[u] = 2$, $S[adj[u]] = 1$
(b) Set $S[v_2] = 2$, $S[v_3] = 2$, $S[adj[v_2]] = 1$, $S[adj[v_3]] = 1$.
Case 9: $S[u] = 0$, $S[v_1] = 0$, $S[v_2] = 0$, $S[v_3] = 0$ *(Branching Rule)*
(a) Set $S[u] = 2$, $S[adj[u]] = 1$
(b) Set one of $S[v_1]$, $S[v_2]$, $S[v_3]$ to 2 (3 branches).

Correctness:
In the reduction rules, we can argue that our assignment is not worse than any other assignment, and each rule reduces the number of nodes with $S[u] = 0$ by atleast 1. Similarly in the branching rules, we check all possible good cases and branch accordingly.

7. Construct a $O^*(1.6181^n)$ branching algorithm to solve 3-SAT. (Harder version: $O^*(1.49^n)$)

Reduction Rules:

1. For all literals that are present only in one clause, set the truth value of the literal such that the clause is true and remove the clause.
2. For clauses with a single literal, set the truth value of the literal such that the clause is true and remove all true clauses and false literals.

Branching Rules:
Below we propose the branching rules for a clause c in a 3-SAT formula F. Note that when we are branching, we assign a truth value for some literals, then remove all clauses that are true and remove all literals that are false.

1. $c = (\emptyset)$: Not possible to satisfy the formula, return false.
2. $c = (x)$: Reduce
3. $c = (x \vee y)$: Branch into $x = T$ and $x = F, y = T$

$$T(n) \leq T(n-1) + T(n-2)$$
$$\text{Branching Vector} = (1, 2)$$
$$\text{Branching Factor} = \frac{1+\sqrt{5}}{2} = 1.6181$$

4. $c = (x \vee y \vee z)$: Branch into $x = T$ and $x = F, y = T$ and $x = F, y = F, z = T$

$$T(n) \leq T(n-1) + T(n-2) + T(n-3)$$
$$\text{Branching Vector} = (1, 2, 3)$$
$$\text{Branching Factor} = 1.8393$$

However note that after our reduction rules, besides possibly the first branching, there will always be a clause with atmost 2 literals, since each literal is present in atleast 2 clauses. Hence the branching factor will be atmost 1.6181. Therefore the running time of the algorithm is $O^*(1.6181^n)$.

□