Ain Shams University
Cairo-Egypt

Modification of algorithms for NP-problems and implementing them using Functional programming languages

A Dissertation
Submitted to the Mathematical Department-
Faculty of Science – Ain Shams University
In Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy
In
Computer Science
Presented By

Niveen Samy Morckos Soliman

Supervised by

Prof. Dr. Sameh S. Daoud	**Prof. Dr. Fayed F.Ghaleb**
Department of Mathematics	Department of Mathematics
Faculty of Science	Faculty of Science
Ain Shams University	Ain Shams University

20١٠

Acknowledgment

First of all, Thanks are to God, The most beneficent and merciful.
I wish to express my deepest thanks and appreciation to my supervisors, Prof. Dr. Sameh Sami Daoud and Prof. Dr. Fayed Fayek Mohamed Ghalip for helping and guiding me during the preparation of this thesis. Finally, I am grateful to my family for their encouragement and their help.

Summary

The thesis contains four chapters and four appendices. Chapter 1, titled by "An introduction to NP-problems and functional programming languages", this chapter consists of two sections. Section 1, contains the basic definitions of P, NP and NP-complete classes and how we can prove that a problem is NP-complete problem. Section 2, introduces what is the functional programming languages paradigm and we introduce in precisely the functional programming language Miranda, which we use in our implementations. Chapter 2, has the title "Approximation algorithms for NP-hard optimization problems"; and consists of six sections. These sections discuss the approximation algorithm concept and the classes of the NP-optimization problems according to their approximation ratios. Also They discuss that there are some problems hard to approximate with any constant approximation ratio using the inapproximation concept. Chapter 3, has the title "Set Cover problem"; and consists of two sections. Section 1, discusses the Set Cover problem (SC), its definition and some approximation algorithms for it. Section 2, contains a modification to an algorithm for unweighted SC problem that uses a new technique (flow algorithm), we generalize this algorithm for the weighted SC problem. Chapter 4, has the title " Minimum Membership Set Cover problem" and consists of two sections. Section 1 discusses the Minimum Membership Set Cover problem (MMSC), its definition, an approximation algorithm for it.

Section 2, contains a new greedy algorithm for MMSC problem and we prove that this algorithm get a solution with approximation ratio in order O(log n).
Finally, Appendix A contains the Miranda implementation of flow algorithm for WSC problem. Appendix B contains the Miranda implementation of the greedy algorithm for WSC problem. Appendix C contains the Miranda implementation of our new algorithm for MMSC problem. Appendix D, contains Java classes which create the Random instances for WSC and MMSC problems.

The conclusion

We conclude that NP-hard problems have many applications in the world and it is important to study it. The approximation algorithm concept is useful to give a solution close to the optimal in time polynomial instead of trying to get the optimal solution, since there is no algorithm gives the optimal solution in time polynomial. We modify an approximation algorithm for unweighted Set Cover problem that uses a new technique (flow graph algorithm), we generalize this algorithm for the weighted Set Cover problem. We also make new greedy algorithm for Minimum Membership Set Cover problem and prove that this algorithm gives a solution with approximation ratio in order O(log n).
In future we will study and try to improve approximation algorithms for NP-hard problems and study flow graph technique more.

Contents

Chapter 1

An introduction to NP-problems and functional programming languages

1.1 NP- hard problems

A problem is a general question to be answered, and usually has several parameters. A problem is described by giving a general description of all its parameters and a statement of what properties the answer is required to satisfy.

In this thesis we shall mainly discuss two types of problems:

1- decision problems
2- Optimization problems
Their definition according to [11] is as follows:

Definition 1.1 A decision problem is a triple (L, U, $\sum$) where $\sum$ is an alphabet and $L \leq U \leq \sum^*$. An algorithm A solves (decides) it, if for every $x \in U$, (i) A (x) = 1, if $x \in L$, and (ii) A (x) = 0, if $x \in U- L$ ($x \notin L$).

Example 1.1

K-Clique Problem; decide for a given graph G and a positive integer k, whether G contains a complete graph K_k of k vertices as a sub-graph of G
Input: A positive integer k and a graph G
Output: "yes" if G contains a clique of size k, "no" otherwise.

Definition 1.2. An optimization problem is a 7-tuple U=($\sum_I$, $\sum_O$, L , L_I , M, Value , Goal), where
(i) $\sum_I$ is an alphabet, called the input alphabet of U,
(ii) $\sum_O$ is an alphabet, called the output alphabet of U,

(iii) $L \subseteq \sum_I^*$ **is the language of feasible problem instances,**

(iv) $L_I \subseteq L$ **is the language of the (actual) problem instances of U,**

(v) **M is a function from L to Pot(**$\sum_O^*$**),** $\forall\, x \in L$**, M(x) is the set of feasible solutions for x,**

(vi) **Value is a function, such that** $\forall$ **(u,x), where** $u \in M(x)$ **for some** $x \in L$**, it assigns a positive real number Value(u,x),**

(vii) **Goal** $\in$ **{ minimum , maximum }**

Example 1.2: Minimum vertex cover problem

Input: A graph G=(V,E), and
Constraint: M(G)= { $S \subseteq V$ **| that every edge of E is adjacent to at least one vertex of S },**
Value : $\forall\, S \in M(G)$**, Value(S,G)=|S|,**
Goal: Minimum.

1.1.1 NP-class

To discuss the NP-class we need the notion of Deterministic Turing Machine DTM and Nondeterministic Turing Machine NDTM. For a good review of this notion see [19]. Now we define the class P as follows:

$$P = \{L \mid \text{there is a polynomial time DTM program M for which } L = L_M\}.$$

Therefore P is the class of all problems which can be solved by a DTM program in a time polynomial on the input size, (i.e), if the input size is n, then the worst-case running time is $O(n^c)$ **for a constant c. Any problem in P is considered "tractable".**

Any DTM can be figured as follows:

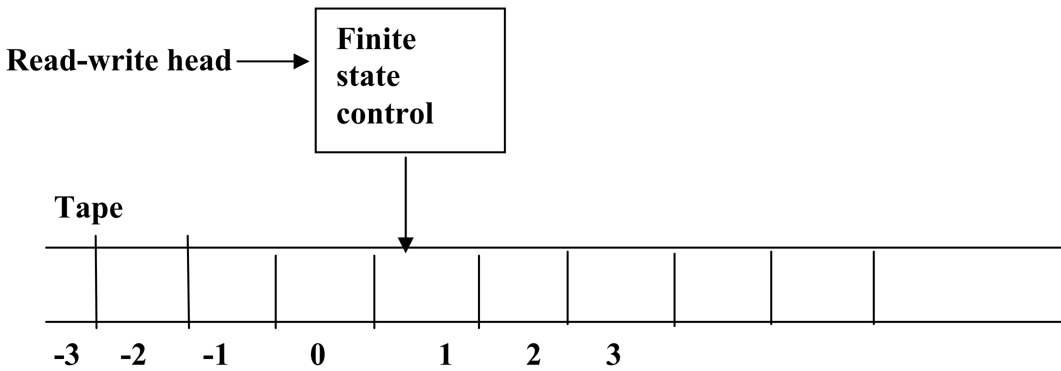

Figure 1.1 Schematic representation of a deterministic one-tape Turing machine

The class NP is defined as follows:

NP = { L | there is a polynomial time Nondeterministic (one-tape) Turing Machine program M for which L_M = L } or,

It is the class of problems whose solutions can be verified in time polynomial in the size of the input.
The Nondeterministic Turing machine (NDTM) model has exactly the same structure as a DTM, except that it is augmented with a guessing module having its own write – only head as illustrated schematically in figure 1.2.

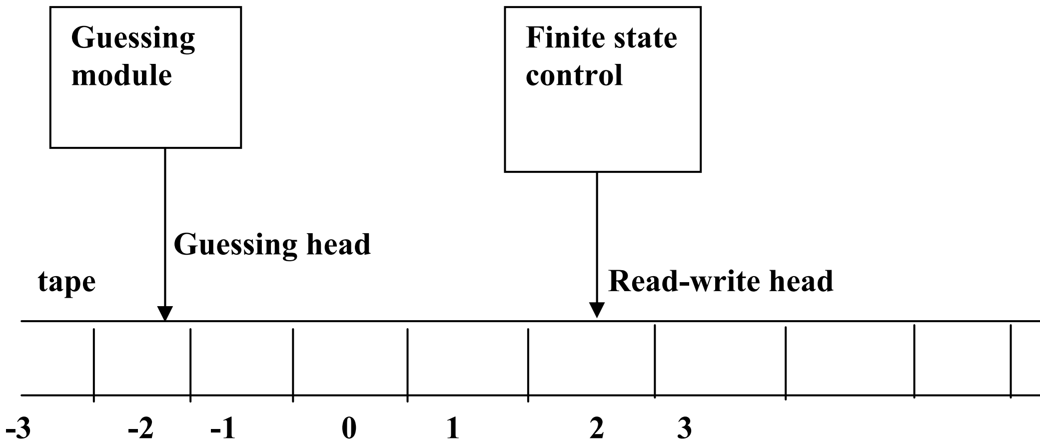

Figure 1.2 Schematic representation of a nondeterministic one-tape Turing machine

We will define now the transformations (Reductions) between Decision problems polynomial time transformation f from L_1 to L_2 ($L_1 \leq_P L_2$).

- **F transforms the input for L_1 into an input for L_2 such that the transformed input is a yes-input for L_2 if and only if the original input was a yes-input for L_1.**
- **F is computable in polynomial time.**

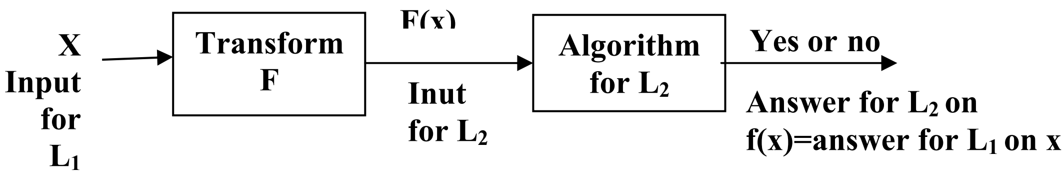

Figure 1.3

Theorem 1.1: if $L_1 \leq_P L_2$ and $L_2 \in$ P, then $L_1 \in$ P.

If there exist polynomial time algorithm for L_2 , then there is Polynomial time algorithm for L_1.
If there is not Polynomial time algorithm for L_1, then there can not be a Polynomial time algorithm for L_2 .
Fact: The relation $\leq_P$ is transitive, i.e. $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$ implies that $L_1 \leq_P L_3$.

We shall use this fact in chapter 2, 3, and 4.

Definition 1.3: A decision problem L is NP-complete (NPC) if:

1. **$L \in NP$, and**
2. **for every $L' \in NP$, $L' \leq_P L$.**

To prove that a problem L is NPC it is sufficitiont to get a known NPC problem L' and show $L' \leq_P L$.
The first problem which was proved to be an NP-complete is the Satisfiability problem (SAT) by [Cook's theorem], which is

Theorem 1.2 (Cook's theorem) Satisfiability is an NP-complete, see [19] for the proof.

By this theorem we can make sequence of reductions to prove that many problems are NP-complete. For example the following figure is a tree of transformations between NP-complete problems.

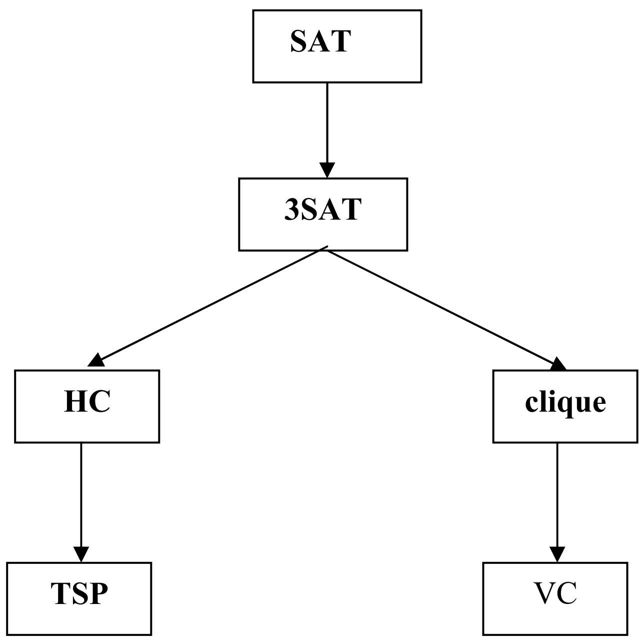

Figure 1.4

See [19] for details

If we want to show that a problem U is an NPC, then we (i) show that U $\in$ NP (ii) select a known NPC problem L' and show that L' $\leq_P$ L.

Example 1.3

In this example we prove that the vertex cover is an NP-complete problem and also we study the relation between it and the two NP-complete problems the clique and the independent set in a certain graph G. We first define the three problems.

The vertex cover problem (VC):

Instance: a graph G=(V,E) and a positive integer k $\leq$ |V|

Output: Decide whether there is a subset V' $\subseteq$ V of size $\leq$ k such that each edge in E has at least one endpoint in V'.

The Clique problem:

Instance: a graph G=(V,E) and a positive integer k $\leq$ |V|

Output: Decide whether G have a Clique of size $\geq$ k (i.e. a subset V' $\subseteq$ V of size at least k such that V' make a complete graph).

The independent set problem (IS):

Instance : a graph G=(V,E) and a positive k $\leq$ |V|.

Output: Decide whether G have an independent set of size $\geq$ k (i.e. a subset V' $\subseteq$ V of size at least k such that G has no edge between any pair).

The relationships between the above three problems are as follows:

For any graph G=(V,E) and a subset V' $\subseteq$ V, the following statements are equivalent:

(a) **V' is a vertex cover for G.**
(b) **V-V' is an independent set for G.**
(c) **V-V' is a clique in the complement G^c=(V,E^c) with E^c={{u,v}: u,v $\in$ V and {u,v} $\notin$ E}.**

These relationships between them make it a trivial matter to transform any one of these problems to each of the other two problems. This implies that the NP-completeness of all the three

problems will follow as an immediate consequence of proving that any one of them is NP-complete.
The following theorem proves that VC is NP-complete

Theorem 1.3. Vertex Cover is a NP-complete.
Proof:
It is easy to show that VC $\in$ NP, since a nondeterministic algorithm needs only to guess a subset of vertices and check in a polynomial time whether that this subset contains at least one endpoint of every edge and that its size is an appropriate number. We first transform 3SAT to VC. Let $U=\{u_1,u_2,\ldots,u_n\}$ and $C=\{c_1,c_2,\ldots\ldots,c_m\}$ be an instance of 3SAT. Now we should construct a graph G=(V,E) and a positive integer k, from this instance and prove that it has vertex cover of size k if and only if C is satisfiable.
The constructing graph will be as follows:
For each variable $u_i \in U$, there is truth-setting component $T_i = (V_i,E_i)$, with $V_i =\{u_i,u_i'\}$ (where u_i' is the complement of u_i) and $E_i = \{\{u_i,u_i'\}\}$, that is, each two vertices joined by a single edge.
For each clause $c_j \in C$, there is $S_j =(V_j',E_j')$, consisting of three vertices and three edges joining them to form a triangle:
$V_j' = \{a_1[j],a_2[j],a_3[j]\}$, $E_j' = \{\{a_1[j],a_2[j]\}, \{a_1[j],a_3[j]\} , \{a_2[j],a_3[j]\}\}$
Note that any vertex cover will have to contain at least two vertices from V_j' in order to cover the edges in E_j'.
For each clause $c_j \in C$, let the three literals in c_j be denoted by x_j, y_j, and z_j. Then the communication edges emanating from S_j are given by:
$E_j''=\{\{a_1[j],x_j\},\{a_2[j],y_j\},\{a_3[j],z_j\}\}$
The graph now is completed as follows, G=(V,E), where $V=(\bigcup_{i=1}^{n} V_i) \cup (\bigcup_{j=1}^{m} V_j')$ and $E=(\bigcup_{i=1}^{n} E_i) \cup (\bigcup_{j=1}^{m} E_j') \cup (\bigcup_{j=1}^{m} E_j'')$.
The following figure illustrates the construction of an instance of VC from the instance $U=\{u_1,\ldots,u_4\}$, $C=\{\{u_1',u_2,u_4'\},\{u_1,u_3',u_4'\}\}$

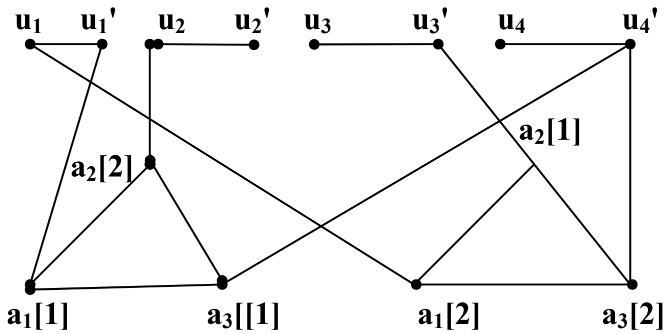

Figure 1.5. vertex cover instance constructed from instance 3SAT instance($U=\{u_1,\ldots,u_4\}$, $C=\{\{u_1',u_2,u_4'\},\{u_1,u_3',u_4'\}\}$)

Now we will show that C is satisfiable if and only if G has a vertex cover of size k or less by setting k=n + 2m. Firstly, suppose that $V' \subseteq V$ is a vertex cover for G with $|V'| \leq k$. V' must contain at least one vertex from T_i and at least two vertices from each S_j. since this gives a total of at least n+2m=k vertices, from this vertex cover we can get a truth assignment t: $U \rightarrow \{True,False\}$. We set $t(u_i)$=True if $u_i \in V'$ and $t(u_i)$=False if $u_i' \in V'$. To prove that this assignment satisfies the clauses in C, consider the three edges in E_j'', only two of those edges can be covered by vertices from $V_j' \cap V'$, so one of them must be covered by a vertex from some V_i that belongs to V'. But that implies the corresponding literal, either u_i or u_i', from clause c_j is true under the truth assignment t, and hence clause c_j is satisfied by t. Because this holds for every $c_j \in C$, it follows that t is a satisfying truth assignment for C. conversely, suppose that t : U $\rightarrow$ {True,False} is a satisfying truth assignment for C. The corresponding vertex cover V' includes one vertex from each T_i and two vertices from each S_j. The vertex from T_i in V' is u_i if $t(u_i)$=True and is u_i' if $t(u_i)$=False. This ensures that at least one of the three edges from each set E_j'' is covered, because t satisfies each clause c_j. Now we need only select endpoints from S_j of the other two edges in E_j'', which gives the required vertex cover.