Appendix A

Algorithms for Set Covering. Documentation

A.1 Problem Definition

Implementation of three different algorithms to solve set covering problem: An exact algorithm and two approximation algorithms. Design of software to prove these algorithms.

A.1.1 Background Theory Explanation

Definition. Let X be a base set. Let $F = \{S_1, S_2, \ldots, S_n\}$ be a collection of subsets of X, this is, $S_i \subseteq X$. F covers X. This is, $S_1 \cup S_2 \cup \ldots \cup S_n = X$.

Problem. Find a family of subsets C such that $C \subseteq F$, C covers X and $|C|$ is minimum.

Now we present the algorithms:

Exact Algorithm. Gets an exact solution using a Branch and Bound strategy; this is, exhaustive search eliminating only the branches if the optimal solution can not be reached.

Simple Greedy. Gets a solution taking the set with maximum cardinality on each

iteration until set X is covered.

Greedy-Exact Algorithm. It takes the subset of maximum cardinality until some limit is reached. Then, it uses the exact algorithm to find the rest of the solution. Limit is a logarithmif function of the input; therefore, the algorithm is still polynomial.

Greedy-by-pairs Algorithm. Selects the pair of subsets of maximum cardinality in each iteration.

Hybrid Greedy Algorithm. It combines strategies of the two previous algorithms. First, it selects a pair of subsets in each iteration. Then, it uses a limit like Greedy-Exact algorithm to find the best possible solution at a lower level.

Dynamic Programming. It uses a strategy like knapsack problem to find the best solution in each iteration. For n possible subsets and some maximum m, $m < n$, the objective is to find a collection of m subsets, whose union has maximum cardinality over base set. It uses this function to reach the objective:

$$dsc(n,m) = \begin{cases} 0 & \text{if } m = 0, \text{ or } n = 0, \\ \max(\text{dsc}(n-1,m)), & \\ \text{card } (n \cup dsc(n-1,m-1)) & \text{Otherwise} \end{cases}$$

It fills a table for all n and m possible and finds a solution for set covering problem. This is also an approximation algorithm.

A.1.2 Software Design

Next we can find Data Flow Diagrams (DFD) for the program. Figure A.1 presents a general DFD, detailed in A.2. Then a DFD for each algorithm is presented: Exact, A.3; Simple Greedy is A.4; Greedy-Exact is A.5; Greedy-by-pairs is A.6; Hybrid Greedy algorithm is A.7 and Dynamic Programming algorithm is A.8. A diagram of methods

or functions is presented in A.9 and A.10. Finally, class diagram is presented to know how the main program uses the others as libraries in A.11.

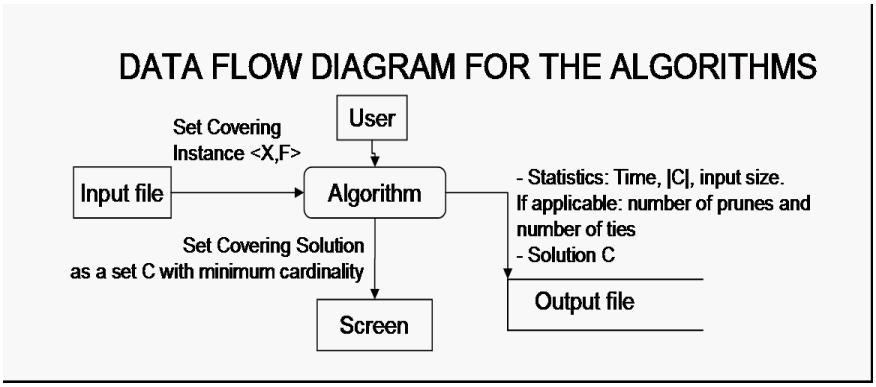

Figure A.1: General Data Flow Chart

A.2 Software Implementation

SOURCE FILE setCoverMain.cpp. Main program

mainMenu(). It presents the main set of options. Finds out what kind of input file will be given by the user (an instance of set covering or a set of instances).

algorithmMenu(). Finds out which algorithm will be used.

read_set(). Process a list of files with instances.

algorithmMenu(). Process a single file with an instance of set cover.

SOURCE FILE algorithms.cpp. Represent an instance of set covering.

Nofsets. Number of subsets for the set cover instance; this is, cardinality of F.

s[ ]. Base set.

subs[ ][ ]. Collection F of subsets.

greedy(). Greedy algorithm. It gets a parameter to know if it is going to find the whole solution or only until some limit.

select_pair(). Get two subsets with maximum cardinality.

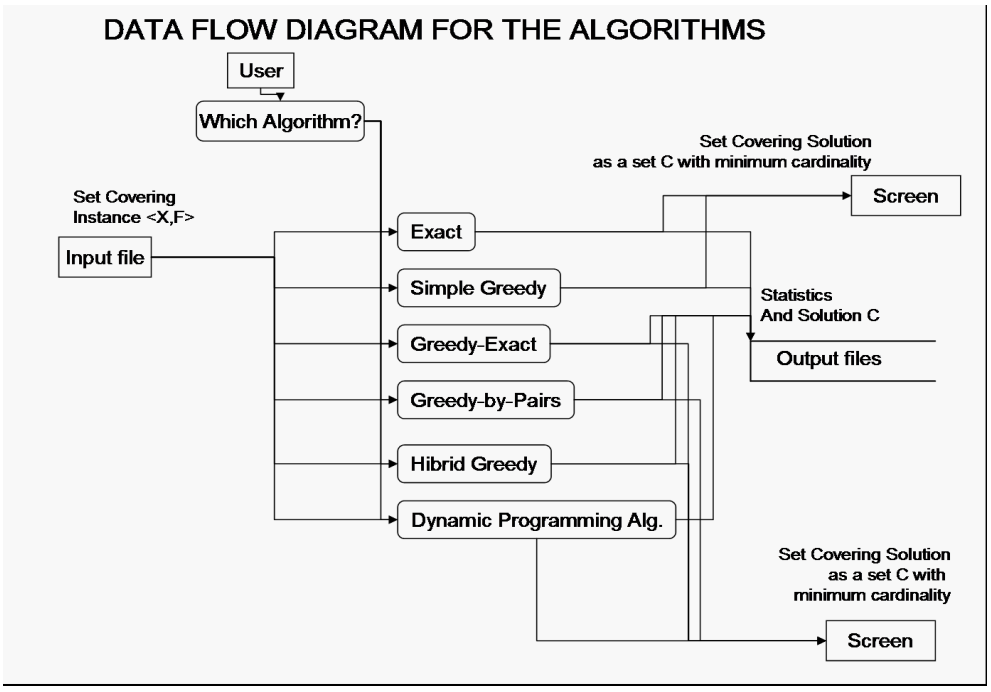

Figure A.2: Detailed General DFD

greedy_pair(). Greedy algorithm, using pairs of subsets instead of only one subset.

unique_el_prune(). Make a pruning over a set covering instance, adding to the solution subsets with elements which appear in only one subset.

subset_prune(). Make a pruning over set covering, removing from de solution sets S_i such that $S_i \cup S_j$, $S_i, S_j \in F$.

prove(). Recursive function. It checks all the possibilities to find the optimal solution.

fill_table(). Fill a table of subsets and its sizes. Uses the function $dsc()$ defined for Dynamic Programming algorithm.

backtracking(). It uses the table gotten in fill_table() to recover the set C with the best solution.

getMinEx(). It uses greedy algorithm to find a bound number, then it calls prove() to find the optimal solution.

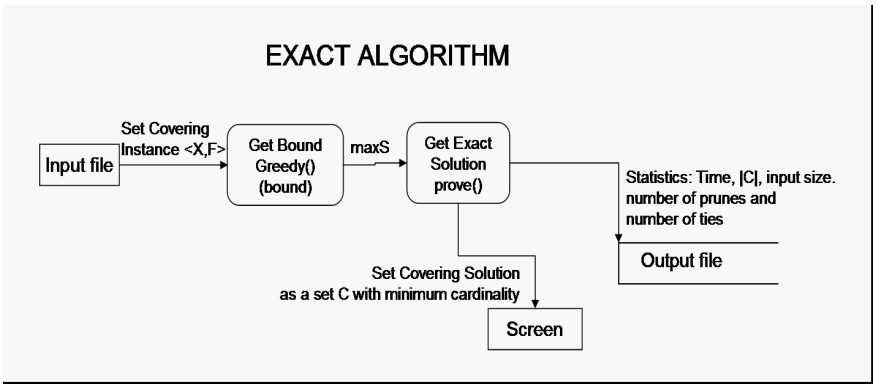

Figure A.3: DFD for the Exact algorithm

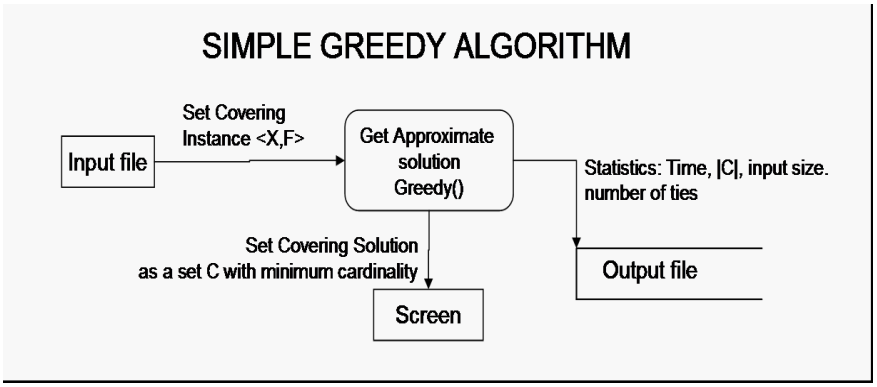

Figure A.4: DFD for the Simple Greedy algorithm

getMinSG(). It calls the Simple Greedy algorithm.

getMinGE(). It calls the Simple Greedy algorithm until some limit and then it calls prove().

getMinGP(). It uses the the function greedy_pair() with the approach of Greedy-by-Pairs algorithm.

getMinHG(). Hybrid Greedy implementation. Uses greedy() to find some bound, greedy_pair() to add some subsets to the solution and calls prove() at the lower level.

getMinDP(). Dynamic Programming approach implementation.

TimeBegin(). It initializes a time counter.

initialize(). Clean the subsets and statistic counters

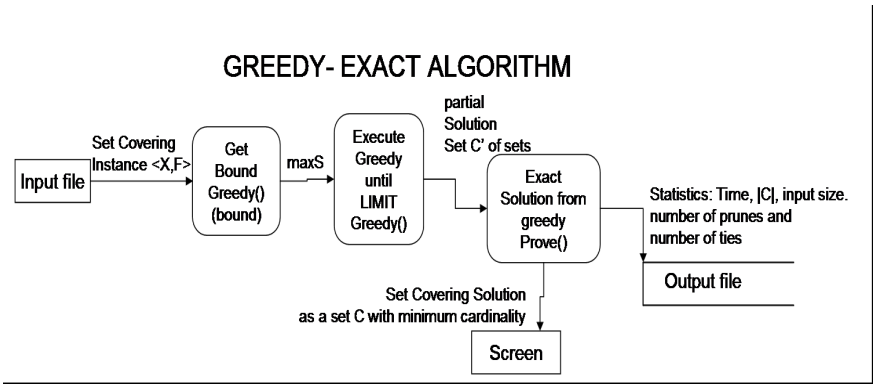

Figure A.5: DFD for the Greedy-Exact algorithm

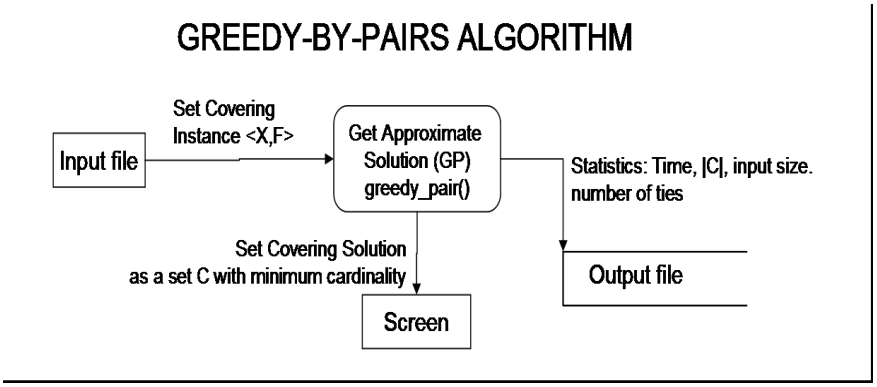

Figure A.6: DFD for the Greedy-by-Pairs algorithm

read(). Read a given input file.

printAll(). Print the set covering instance.

solve(). Solve an instance given a specified algorithm.

print_header(). Print statistic header over output stream.

printAll(). Print statistics given an algorithm in a given output stream.

SOURCE FILE arreglos.cpp. Array Manipulation

imprimir_arr(). It print all the elements of a bitmap. This is, it prints the positions of the array where the value is 1.

clean_bm(). Initialize an array to values of 2.

copy_arr_to(). Make an exact copy of an array.

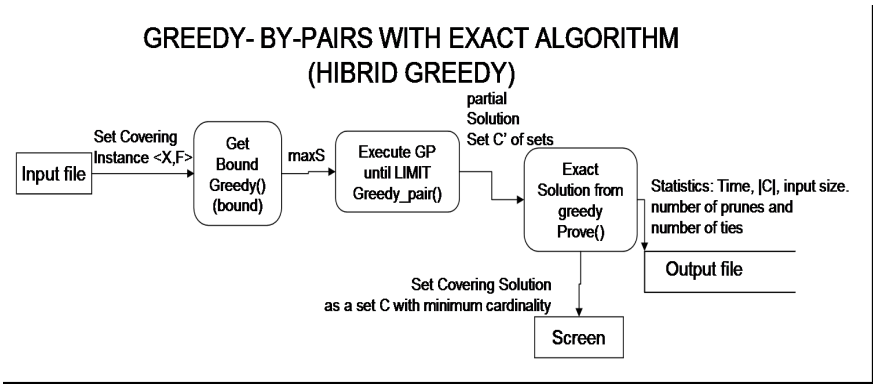

Figure A.7: DFD for the Hybrid Greedy algorithm

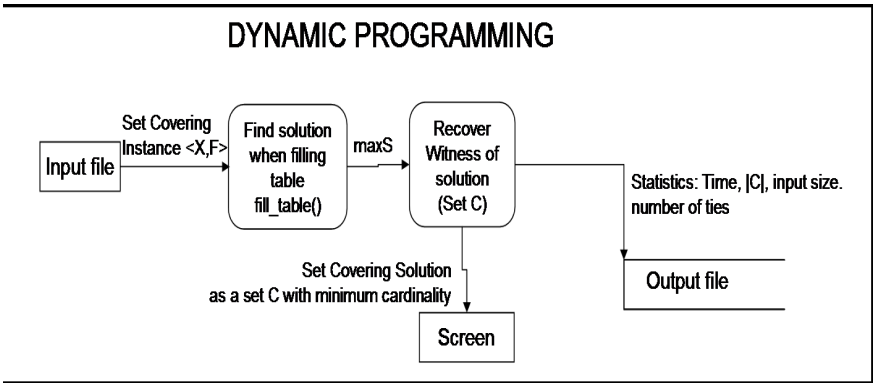

Figure A.8: DFD for the Dynamic Programming algorithm

SOURCE FILE set.cpp. Set Manipulation

arraylength. Maximum size of the array. Each byte or position can keep 8 elements (1 char = 8 bits).

setsize. Size of base set X.

add_element(). Add an element to a given subset.

clear_set(). Return the given set as an empty set.

unite(). Return the union operation result of two sets.

difference(). Return the union operation result of two sets.

remove_element(). Remove an element from a given set.

intersect(). Return the intersection operation result of two sets.

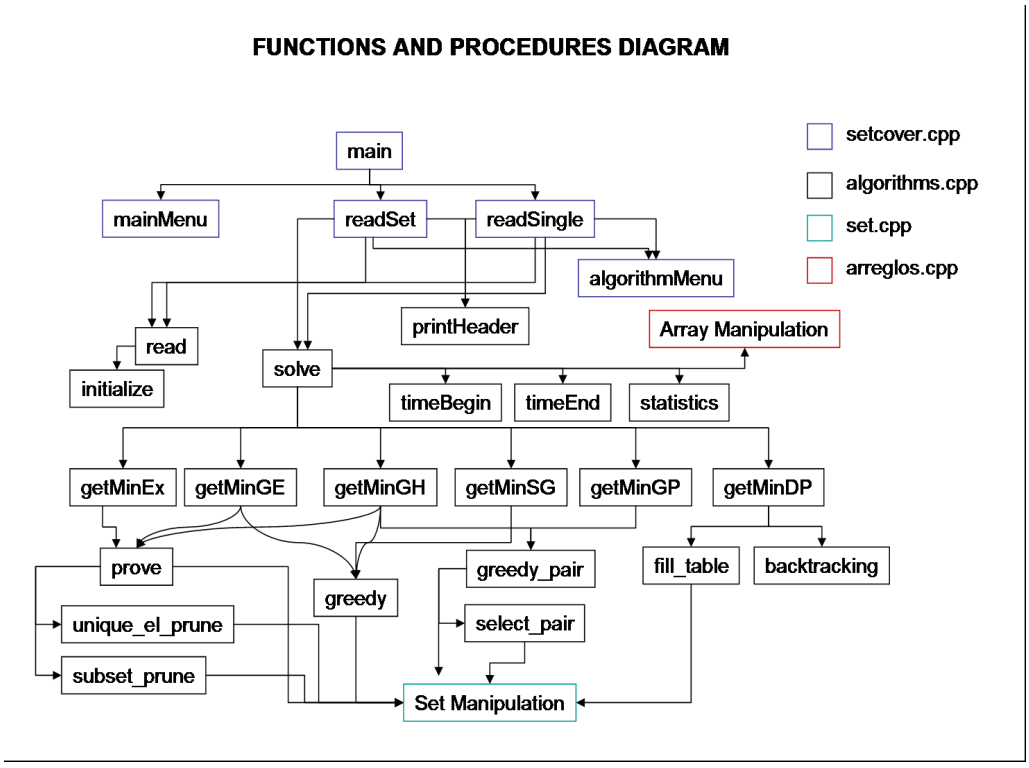

Figure A.9: Function calls (to be continued...)

exor(). Exclusive or operation over two sets.

print_set(). Print to standard output a given set.

isin(). Test if a given element is in some set.

cardinality(). Get the cardinality of a given subset.

is_empty(). Return 1 if the subset is empty, 0 otherwise.

copy_to(). Make an exact copy from a given set.

subset(). Find out if set $S_i \subseteq S_j$.

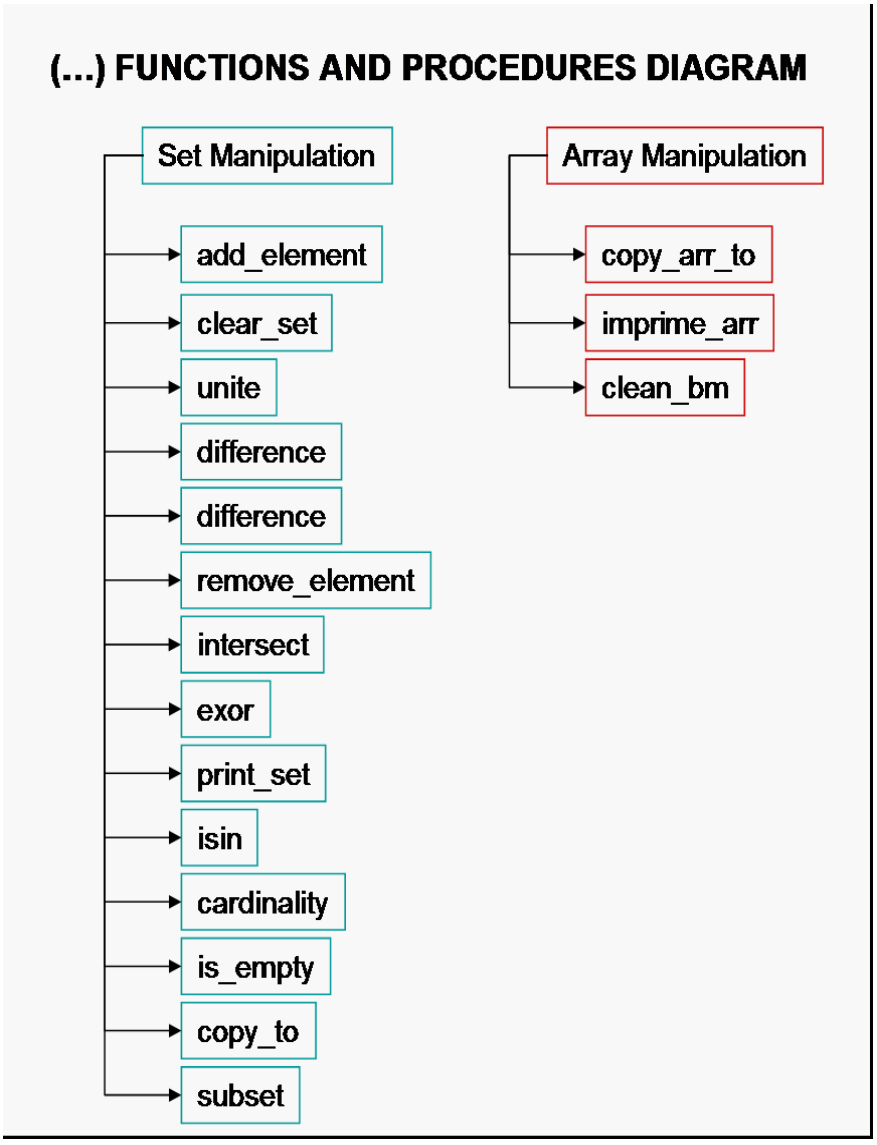

Figure A.10: ...Function calls

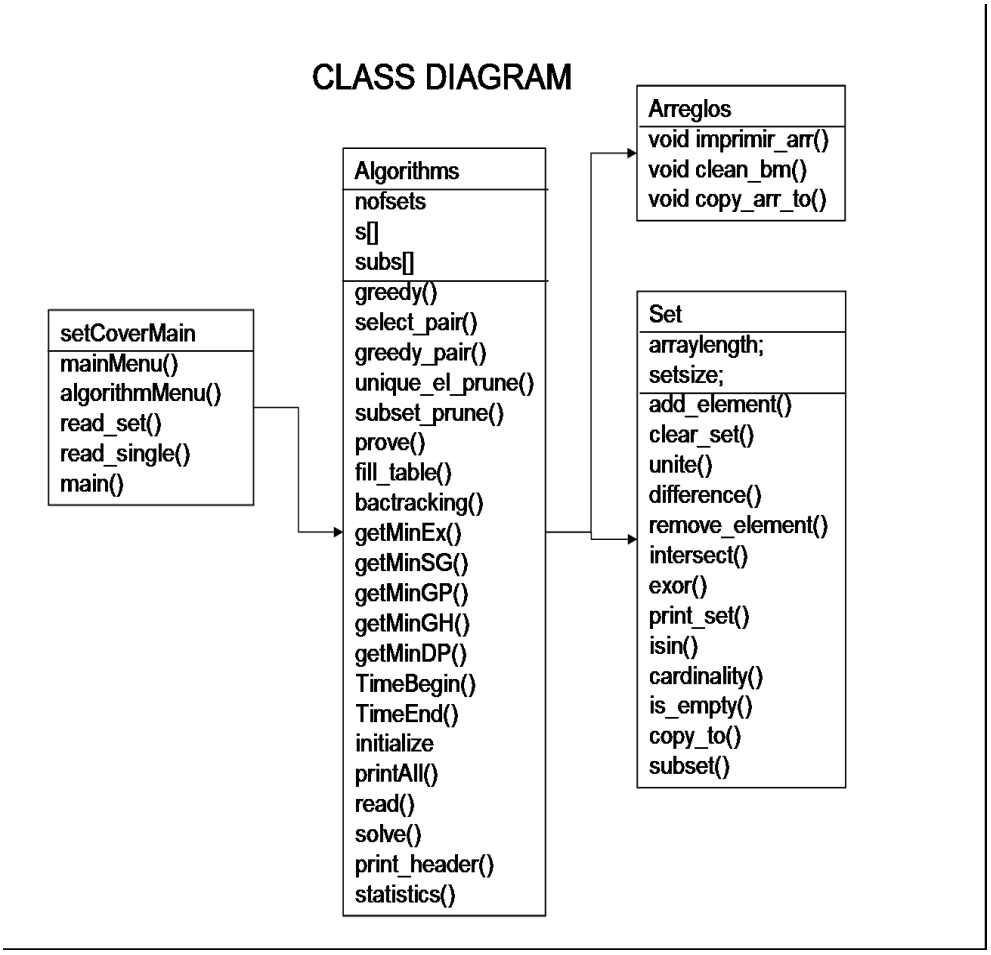

Figure A.11: Class Diagram