INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center Street • Suite 600 • Berkeley, California 94704 • 1-510-642-4274 • FAX 1-510-643-7684

Dynamic maintenance of approximated solutions of Min–Weighted Node Cover and Min–Weighted Set Cover problems *

Giorgio Gambosi[†] **Marco Protasi**[‡]

Maurizio Talamo[§]

TR-?????

August 1993

Abstract

In this paper, we introduce new algorithms for the dynamic maintenance of approximated solutions of Min–Weighted Node Cover and Min–Weighted Set Cover. For what concerns Min–Weighted Node Cover, for any sequence of edge insertions and deletions, the algorithms maintain an solution whose approximation ratio (that is, the ratio between the approximate and the optimum value) is equal to the best asymptotic one for the static case. The algorithms require $O(1)$ *time for edge insertion, while an* $O(1)$ *amortized time is required for edge deletion.*

For what regards Min–Weighted Set Cover, we present dynamic algorithms whose approximation ratio matches one of the two different and incomparable best approximate bounds for the static case. The time complexity for element insertion and its amortized complexity for element deletion are proportional to the maximum redundancy of an element in the approximate solution.

*The work of the first two authors was partially done while visiting the International Computer Science Institute, Berkeley, Ca. Work partially performed in the framework of Esprit BRA projects "ALCOM" and "ALCOM 2", and of Italian MURST 40% project "Algoritmi, Modelli di Calcolo e Strutture Informative" and partially supported by the National Research Council (CNR) Project "Sistemi Informatici e Calcolo Parallelo"

[†]Dipartimento di Matematica Pura ed Applicata, University of L'Aquila, Vetoio di Coppito, I-68100 L'Aquila, Italy. E-mail: gambosi@smaq20.univaq.it

[‡]Dipartimento di Matematica, University of Rome II "Tor Vergata", via della Ricerca Scientifica, I-00133 Roma, Italy. E-mail: protasi@mat.utovrm.it

[§]Dipartimento di Informatica e Sistemistica, University of Rome "La Sapienza", via Salaria 113, I-00198 Roma, Italy. E-mail: talamo@disparcs.ing.uniroma1.it

1 Introduction

The topic of the dynamic maintenance of problem solutions, while problem instances are evolving in the time, currently represents an extensively studied research area in the field of the theory of algorithms. In contrast with classical (off–line or static) algorithms, which, after finding a solution for a problem instance, have to compute "from scratch" a solution for a different problem instance, dynamic algorithms try to gain efficiency by computing a new solution starting from an old one. In fact, dynamic algorithms are especially designed to manage (possibly unbounded) sequences $I_1, I_2, I_3 \ldots$ of problem instances (where I_k differs only "slightly" from I_{k-1}), with the aim of deriving the solution for a new instance I_k by updating the one previously derived for I_{k-1}.

The importance of the study of dynamic algorithms is testified by the large amount of combinatorial and geometric problems such as graph connectivity, minimum spanning trees, planarity, transitive closure, convex hull, etc. that have been considered in this framework. It is worthwhile to underline that dynamic algorithms have been applied, in the large majority of cases, to problems solvable in polynomial time. On the other hand, an interesting research topic is the design of dynamic algorithms for the maintenance of approximate solutions for optimization problems whose decision version is NP–complete. Generally speaking, such a topic has not been sufficiently studied, in spite of the fact that approximation algorithms are acquiring more and more relevance [19], [18], [1], [2].

In contrast to this general situation, the Min–Bin–Packing problem is an example of an NP–complete problem previously considered in terms of the maintenance of approximate solutions [10], [11], [9], [24], [25], [16].

Another important computationally intractable problem that has been studied from a dynamical point of view is the Min-Node Cover problem. In [15] a dynamic algorithm that maintains an approximate solution whose value is at most 2 times the optimal value is presented. The amortized running time is arbitrarily close to $O((v+e)^{\frac{\sqrt{2}}{2}})$, where v is the number of nodes and e denotes the number of edges at the time that the operation is made.

In this paper we continue the investigation of on-line algorithms for this kind of problems studying the the Min-Weighted Node Cover and the Min-Weighted Set Cover problems. As a particular case the new algorithms we are going to present can be therefore applied to the Min-Node Cover problem. Many research efforts have been aimed to the design of efficient off–line approximation algorithms for such problems ([4], [5], [7], [8], [13], [14], [17], [20], [22], [23]) and in order to design efficient on-line algorithms we will start from the approach introduced in [13] and then exploited in [3]. We will use the algorithm presented in [3] because, while obtaining the same approximation ratio found in [13], it achieves a better time complexity.

In this paper, we first study Min–Weighted Node Cover and present a dynamic algorithm which, for any sequence of edge insertions and edge deletions, maintains an approximate solution whose value is at most 2 times the optimal value. Note that this bound is the best (asymptotic) bound till now achievable even in the case of static algorithms [3], [5], [22]. The algorithm requires $O(1)$ time for edge insertion, while an $O(1)$ amortized time is required for edge deletion. We note that this algorithm obtains better complexity bounds with respect to the approach given in [15].

The approach introduced for the design of the previous algorithm can be extended to the the Min–Weighted Set Cover problem. In such a case, the algorithm achieves an approximate solution whose value is k times the optimal value, where k is the maximum redundancy of an element in the approximate solution, that is the maximum number of sets in the approximate solution containing a same element. In the static case, there are two different and incomparable best approximation ratios [13], [7]. The approximation ratio of our algorithm is equal to the one given in [13] and in [3], while its complexity for element insertion and its amortized complexity for element deletion are proportional to the maximum redundancy of an element in the approximate solution.

In Section 2 some definitions and previous results about static algorithms for the the Min–Weighted Node Cover and the Min–Weighted Set Cover problems are presented. In Section 3 the dynamic algorithm for the Min–Weighted Node Cover is introduced and an evaluation of its behavior in terms of complexity and of approximation bound is provided. Finally, we extend our approach to study the Min–Weighted Set Cover problem .

2 Definitions and previous results

In this section, after a formal definition of the Min–Weighted Node Cover and Min–Weighted Set Cover problems, we present some approximation results for these problems, which we will exploit for the design of dynamic algorithms.

Definition 2.1 *Let $\mathcal{SU} = \{S_1, S_2, \ldots, S_n\}$ be a finite family of finite sets, where each set S_i has a non negative weight $w_i \in N$. Let $U = \{e_1, e_2, \ldots, e_t\}$ be $\cup_{i=1}^{n} S_i$. The Min–Weighted Set Cover is a subfamily $SC \subseteq \mathcal{SU}$, such that $\cup_{S_i \in SC} S_i = U$ and $\sum_{S_i \in SC} w_i$ is minimum.*

Let us now define for each $e_j, j = 1, \ldots, t$, $F(j) = \{S_i \mid e_j \in S_i\}$.

Note that, in the case $\mid F(j) \mid= 2$ for all $j = 1, \ldots, t$, the Min–Weighted Set Cover problem reduces to Min–Weighted Node Cover as follows:

A problem instance becomes a weighted graph $G = (V, E)$, where the set of nodes $V = \{v_1, v_2, \ldots, v_n\}$ with associated weights $\{w_1, w_2, \ldots, w_n\}$, corresponds to the set family $S_1, S_2, \ldots, S_n$, while $E = U$ and each element e_j with $F(j) = \{S_i, S_k\}$ is represented by an edge (v_i, v_k).

In this case, a Min–Weighted Node Cover can be interpreted as a subset $NC \subseteq V$ such that:

1. for each edge (v_i, v_k) at least one among v_i and v_k belongs to NC.
2. $\sum_{v_i \in NC} w_i$ is minimum.

As usual, we will denote as Min Node Cover (Min Set Cover), the case of Min–Weighted Node Cover (Min–Weighted Set Cover) where $w_i = w_j \forall i, j, 1 \leq i, j \leq n$.

Several efficient (polynomial–time) approximation algorithms for Min–Weighted Node Cover have been introduced in the literature. In particular, in [13] and in [3] it was shown how to derive an approximated solution whose value is at most two times the optimum value. This result has been slightly improved in [14], in [22], [5], where the ratio between the approximated value and the optimum is bounded by $2 - o(1)$.

For what concerns Min–Weighted Set Cover, the two best approximated algorithms are due to Chvatal [7] and Hochbaum [13].

```
begin
    for i = 1,...,n do
        w̄_i := w_i;
    Sol := ∅;
    J := U;
    while J ≠ ∅ do
        begin
        Choose e_j ∈ J;
        min_weight := Min{w̄_i | S_i ∈ F_j};
        Let S_k ∈ F_j such that w̄_k = min_weight;
        for each S_i ∈ F_j do
            w̄_i := w̄_i − min_weight;
        Sol := Sol ∪ {S_k};
        J := J − {e_i | e_i ∈ S_k}
        end
end.
```

Figure 1: Bar-Yehuda, Even algorithm

The approach introduced in [13] was exploited in [3] to get a more efficient algorithm. Because of this reason we will start from the Bar-Yehuda, Even algorithm.

Roughly speaking, the algorithm in figure 1 at each step considers an uncovered element e, selects, among all sets containing e, the one with the minimum current weight $\overline{w}$ and adds such a set to the current Set Cover, while decreasing the current weights of all the other sets containing e.

The following theorem has been proved in [13] and in [3].

Theorem 2.1 *The above algorithm provides an approximate solution Sol of value W for the Min–Weighted Set Cover problem, which satisfies the following inequality:*

$$W \leq W_{opt} \cdot Max_{1 \leq j \leq t} \mid F(j) \cap Sol \mid .$$

In particular, for the Min–Weighted Node Cover problem:

$$W \leq 2 \cdot W_{opt}$$

3 Dynamic algorithms for Min–Weighted Node Cover and Min–Weighted Set Cover Problems

In this section, we introduce a dynamic algorithm for Min–Weighted Node Cover and study its performance with respect to time complexity and approximation ratio. The approach introduced for this problem will be then extended to deal with the Min-Weighted Set Cover problem.

Of course the algorithms that will present for the Min-Weighted Set Cover problem apply to the Min-Weighted Node Cover as a particular case. However, in order to have a simpler presentation and to stress the techniques we use, we prefer to present our results starting from the node cover case.

Given a graph $G = (V, E)$, we are going to introduce two algorithms for supporting two different update operations:

- *insert_edge*(v_i, v_j). That is, $E := E \cup \{(v_i, v_j)\}$. The current Node Cover NC has to be updated accordingly.

- *delete_edge*(v_i, v_j). That is, $E := E - \{(v_i, v_j)\}$. Again, the current Node Cover NC has to be updated accordingly.

Algorithms for both *insert_edge* and *delete_edge* use the following information associated to nodes and edges:

- for each node $v_i \in NC$ an edge, denoted as $mark(v_i)$, such that, if v_i has been introduced in NC in correspondence to an *insert_edge*(e) operation, then $mark(v_i) = e$. Note that $mark(v_i)$ is incident to v_i.

- for each node $v_i \in NC$, a set $list(v_i)$ of incident edges such that $(v_i, v_k) \in list(v_i)$ iff $v_k \notin NC$. We assume that $list(v_i) = \emptyset$ in case $v_i \notin NC$.

- for each node $v_i \in V$ a current weight $\overline{w}_i$, initially set to w_i.

- for each edge $(v_i, v_j) \in E$, a weight p_{ij}.

The *insert_edge*(v_i, v_j) algorithm behaves as follows: if at least one node between v_i and v_j is in NC, the current Node Cover is not updated. Otherwise (neither v_i nor v_j belongs to NC), edge (v_i, v_j) is covered by the node with minimal current weight. W.l.o.g, let v_i be such node, then $\overline{w}_j$ is decreased by $\overline{w}_i$.

Moreover, in order to allow an efficient implementation of the *delete_edge* operation, node v_i is marked by edge (v_i, v_j) (that is, $mark(v_i) = (v_i, v_j)$) and weight p_{ij} is set to $\overline{w}_i$. Finally, we observe that also sets $list(v_i), list(v_j)$ are maintained for the same reason.

In Figure 2, we formally present the algorithm for *insert_edge*(v_i, v_j).

Let $E_k \subseteq E$ be the set of edges incident to node v_k: the following Lemma can be stated for what regards the *insert_edge* operation.

Lemma 3.1 *$\forall v_k \in V$, the following equality R_k is maintained by an insert_edge operation: $\overline{w}_k + \sum_{(v_i,v_j) \in E_k} p_{ij} = w_k$.*

Proof. Let us assume equality R_k is verified for $k = 1, \ldots, n$ just before an *insert_edge* (v_i, v_j) operation. Two cases are then possible:

1. At least one among v_i and v_j belongs to NC.

2. Both v_i and v_j do not belong to NC.

```
begin
    if $v_i \in NC \vee v_j \in NC$
       then $p_{ij} := 0$;
    if $v_i \in NC$
       then $list(v_i) := list(v_i) \cup \{(v_i, v_j)\}$
       else $list(v_j) := list(v_j) \cup \{(v_i, v_j)\}$;
    if $v_i \notin NC \wedge v_j \notin NC$
       then
            Let $v_i$ such that $\overline{w}_i < \overline{w}_j$
            then
                begin
                    $NC := NC \cup \{v_i\}$;
                    $p_{ij} := \overline{w}_i$;
                    $\overline{w}_j := \overline{w}_j - \overline{w}_i$;
                    $\overline{w}_i := 0$;
                    $\overline{w}_i := 0$;
                    $mark(v_i) := (v_i, v_j)$;
                    $list(v_i) := \{(v_i, v_j)\}$
                end
end.
```

Figure 2: Algorithm Insert_Edge

In case 1 edge (v_i, v_j) is covered by the current Node Cover. From the *insert_edge* algorithm, p_{ij} gets value 0, while neither current weights associated to nodes nor weights associated to edges are modified. This immediately implies that equality R_k remains verified for all $k = 1, \ldots, n$.

In case 2 one between v_i and v_j must enter NC. According to the *insert_edge* algorithm, the node with smaller weight is chosen to be added to NC. W.l.o.g., let v_i be the chosen node: according to the algorithm, $\overline{w}_i$, $\overline{w}_j$ and p_{ij} are the only weights modified. This implies that all values considered in equalities R_k, $k \neq i, j$ are not modified, thus leaving such equalities still verified.

For what concerns R_i and R_j, it is to note that in the left side of these equalities the same quantity is added and decreased while the right side does not change, thus leaving both equalities still verified. □

Let us present in Figure 3 an algorithm for the *delete_edge*(v_i, v_j) operation. Such an algorithm first checks whether one among v_i and v_j is marked by (v_i, v_j). W.l.o.g. let v_i be such a node: then, the current weights of both v_i and v_j are increased by p_{ij}, v_i is deleted from the Node Cover and all edges $(v_i,\, v_k) \in list(v_i)$ are considered as potential candidates to be inserted again, since they are now uncovered as a consequence of the deletion of node v_i from NC.

The following Lemma can now be stated for what regards the *delete_edge* operation.

Lemma 3.2 *$\forall v_k \in V$, equality R_k is maintained by a* delete_edge *operation.*

Proof. Let us assume equality R_k is verified for $k = 1, \ldots, n$ just before a *delete_edge*(v_i, v_j) operation. Two cases are then possible:

1. Exactly one between v_i and v_j belongs to NC. In this case, w.l.o.g. let $v_i \in NC$. Two subcases are possible:

   (a) $mark(v_i) = (v_i, v_j)$. Then, the weights of both nodes v_i and v_j are increased by the weight p_{ij}. Since edge (v_i, v_j) is then deleted from both E_i and E_j, the equality is verified for both v_i and v_j just before the execution of the loop at line 13 of the algorithm. The equality is also immediately verified for the other nodes. By lemma 3.1, after the execution of the same loop, the equality is still holding for all nodes.

   (b) $mark(v_i) \neq (v_i, v_j)$. Then, $p_{ij} = 0$ by the *insert_edge* algorithm and, since no node weight is modified by the *delete_edge* algorithm, the equality remains true.

2. Both v_i and v_j belong to NC. In this case, it is not possible that both $mark(v_i) = (v_i, v_j)$ and $mark(v_j) = (v_i, v_j)$. Two subcases are then possible (again, we assume w.l.o.g. that $v_i \in NC$.

   (a) $mark(v_i) = (v_i, v_j)$: then, we are in the same situation as in subcase 1a above and the lemma can be proved as in such subcase.

   (b) neither $mark(v_i) = (v_i, v_j)$ nor $mark(v_j) = (v_i, v_j)$: then, the considerations given in subcase 1b above can still be applied for both nodes.

```
begin
    Erase edge (v_i, v_j);
    if mark(v_i) = (v_i, v_j) ∨ mark(v_j) = (v_i, v_j)
       then Let v_i be such that mark(v_i) = (v_i, v_j)
            begin
              \overline{w}_j := \overline{w}_j + p_{ij};
              \overline{w}_i := p_{ij};
              NC := NC − {v_i};
              Let list(v_i) = {(v_i, v_{k_1}), (v_i, v_{k_2}), ..., (v_i, v_{k_r})};
              q := 1;
              while v_i ∉ NC ∧ q ≤ r do
                    begin
                    insert_edge(v_i, v_{k_q});
                    if v_i ∉ NC
                       then
                           begin
                              list(v_i) := list(v_i) − {(v_i, v_{k_q})};
                              list(v_{k_q}) := {(v_i, v_{k_q})};
                           end;
                    q := q + 1;
                    end
              if v_i ∈ NC
                 then
                    for s := 1 toq − 1 do
                        list(v_{k_s}) := ∅
            end
end.
```

Figure 3: Algorithm Delete_edge

□

Lemma 3.3 *$\forall v_k \in V$, equality R_k holds under an arbitrary sequence of insert_edge and delete_edge operations, starting from $E = \emptyset$.*

Proof. The relation is trivially true for $E = \emptyset$, since $E_k = \emptyset$ for each node v_k and $\overline{w}_k$ is initialized to w_k.

The equality is proved to be maintained under any sequence of *insert_edge* and *delete_edge* operations by Lemma 3.1 and Lemma 3.2. □

In order to prove the next theorem, we also need the following Lemma.

Lemma 3.4 *Given any weighted graph $G = (V, E, w)$ and any Node Cover NC of G, the ratio between the value of NC and the value of minimum Node Cover is bounded by 2 if there exists three functions $\overline{w} : V \mapsto N$, $p : E \mapsto N$, $mark : E \mapsto NC$ (where* mark *is a partial function) such that the following conditions hold:*

1. *$\forall v \in NC : \overline{w}(v) = 0$.*
2. *$\forall v \in V : \overline{w}(v) + \sum_{e=(v,u)\in E} p(e) = w(v)$*
3. *$\forall e \in E$, if $p(e) > 0$ then $mark(e)$ is defined.*
4. *$\forall v \in NC$ there exists exactly one edge e incident to v such that $mark(e) = v$.*
5. *the subgraph $G' = (V, E')$ induced by the set of edges $E' = \{e \mid mark(e) \text{ is defined}\}$, is acyclic.*

Proof. Given the functions $\overline{w}$, p, $mark$, and a Node Cover NC, there must exist, in the case $G' = (V, E')$ is acyclic, a node $v_1 \in NC$ such that exactly one edge $e_1 = (v_1, v_i) \in E'$ is incident to v_1. Notice that, by condition 4, $mark(e_1) = v_1$. Notice moreover that, if e_1 is selected by deleting v_1 and all incident edges in E and by updating the value $\overline{w}(v_i)$ to $\overline{w}(v_i) + p(e_1)$, it is possible to iteratively select a sequence $e_1, e_2, \ldots, e_{|NC|}$ of $\mid NC \mid = \mid E' \mid$ edges (and nodes).

It is easy to see that such a sequence corresponds to a possible sequence of edges (and nodes) chosen during some application of the Bar-Yehuda and Even algorithm.

The Lemma derives by observing that any Node Cover returned by the algorithm in [3] has an approximation ratio bounded by 2. □

Theorem 3.5 *Algorithms* insert_edge *and* delete_edge *maintain an approximate solution of the Min–Weighted Node Cover problem with approximation ratio* 2.

Proof. Given any sequence $O_1, O_2, \ldots$ of *insert_edge* and *delete_edge* operations, let us denote as $G_i = (V, E_i)$ the graph resulting by the execution of operations $O_1, \ldots, O_i$ and as NC_i the current Node Cover.

Let us now consider the relations in Lemma 3.4, instantiated on weights $\overline{w}_i$ and p_{ij}, on marks $mark(v)$ and on Node Cover NC_i, as resulting from sequence $O_1, \ldots, O_i$.

Relations 1, 3, 4 immediately hold by the algorithms' structure, relation 2 is verified by Lemma 3.3 and relation 5 can be easily proved by induction. Hence the approximation ratio is achieved by Lemma 3.4. □

For what concerns complexity issues, the following theorems can be proved:

Theorem 3.6 *Algorithm* insert_edge *has time complexity* $O(1)$.

Proof. Derives immediately by the algorithm structure. □

For what regards the *delete_edge* algorithm, it is possible to prove an $\Omega(n)$ lower bound on the number of operations to be performed in the worst case in correspondence to an edge deletion.

Fact 3.1 *A single delete_edge operation requires* $\Omega(n)$ *nodes to be inserted in the Node Cover in the worst case.*

Proof. Let us consider the graph in figure 4. Assume $NC = \{v_0\}$ and assume also that a sequence of operations *insert_edge*(v_0, v_1), *insert_edge*(v_0, v_2), ..., *insert_edge*(v_0, v_n) has been performed. Then, in correspondence to a *delete_edge*(v_0, v_1) operation, v_0 is extracted from NC and all edges (v_0, v_2), (v_0, v_3), ..., (v_0, v_n) have to be considered, since it is immediate to verify that, when edge (v_0, v_i) is considered, node v_i is inserted in NC. □

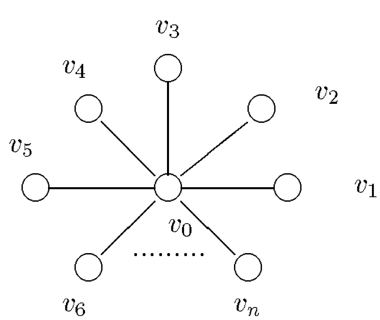

Figure 4: A lower bound case

It is however possible to show that a sequence of *delete_edge* and *insert_edge* operations presents a good amortized complexity. This is proved in the following Theorem.

Let us first consider edge (v_i, v_j) and let us assume that $v_i \in NC$. We denote (v_i, v_j) as *stabilized* if $mark(v_i) = (v_i, v_j)$. In the case that both $v_i \in NC$ and $v_j \in NC$, (v_i, v_j) is stabilized if either $mark(v_i) = (v_i, v_j)$ or $mark(v_j) = (v_i, v_j)$.

We denote the edge as stabilized since it cannot remain "uncovered" as a consequence of the deletion of another edge. This implies that we are sure that we will not have to find a cover for a stabilized edge again.

Theorem 3.7 *Algorithm* delete_edge *has* $O(1)$ *amortized time complexity.*

Proof. As noticed above, it may happen that, as a consequence of the deletion of an edge (v_i, v_j) with $v_i \in NC$, a certain set of edges $(v_i, v_{i_1}), (v_i, v_{i_2}), \ldots, (v_i, v_{i_k})$ have to be reconsidered, since they are not covered anymore. By definition of stabilized edge, all such edges are not stabilized.

The *delete_edge* algorithm covers all such edges by making a subset $(v_i, v_{i_1}), (v_i, v_{i_2}), \ldots, (v_i, v_{i_t})$, $t \leq k$ of them stabilized and performing $O(t)$ operations.

Since, by definition of stabilized edge, an edge can be stabilized only once during its lifetime, such $O(t)$ complexity is amortized by the complexity of the corresponding *insert_edge*(v_i, v_{i_1}), *insert_edge*(v_i, v_{i_2}), ..., *insert_edge*(v_i, v_{i_t}) operations already performed. This implies that the amortized complexity of a *delete_edge* operation is $O(1)$. □

Exploiting the technique in so far used, we may extend our approach to the Min–Weighted Set Cover problem. In this case, the operations we consider are *insert_elem*$(e, \mathcal{S})$ (where $\mathcal{S} \subseteq \mathcal{SU}$) and *delete_elem* (e). The first operation introduces a new element e both in the universe U and in all sets contained in $\mathcal{S}$, while the second one eliminates an element e from the universe U (and, as a consequence, from all sets in $\mathcal{SU}$ in which e is included).

The algorithms for *insert_elem* and *delete_elem* are extensions of the corresponding algorithms for Min–Weighted Node Cover.

As for the Min–Weighted Node Cover, we assume each set S_i has weight w_i. Moreover, our algorithms refer to a weight $p(e)$ for each element $e \in U$; we also introduce two functions $mark$ and $Sets$. The (partial) function $mark : \mathcal{SU} \mapsto U$, similarly to the Node Cover case, associates to each set $S_i \in SC$ the unique element $e \in S_i$ whose insertion caused S_i to be included in the Set Cover SC. The function $Sets : U \mapsto \mathcal{SU}$ associates to each element the collection of sets in which the element is contained.

Let us now introduce the algorithms for the *insert_elem* and *delete_elem* operations.

The approximation ratio mantained by this algorithms can be proved by some lemmata, whose proofs are omitted because are generalizations of analogous proofs presented for the Min–Weighted Node Cover problem.

Lemma 3.8 *$\forall S_k \in \mathcal{SU}$, the following equality T_k is maintained by an* insert_elem *operation:* $\overline{w}_k + \sum_{e \in S_k} p(e) = w_k$.

Lemma 3.9 *$\forall v_k \in V$, equality T_k is maintained by a delete_edge operation*

Lemma 3.10 *$\forall S_k \in \mathcal{SU}$, equality T_k holds under an arbitrary sequence of* insert_elem *and* delete_elem *operations, starting from $S = \emptyset$ for all $S \in \mathcal{SU}$.*

Theorem 3.11 *Algorithms* insert_elem *and* delete_elem *maintain an approximate solution* Sol *of Min–Weighted Set Cover with approximation ratio $Max_{1 \leq j \leq t} \mid F(j) \cap Sol \mid$.*

Proof. The Theorem is a generalization of Theorem 3.5, resulting from the application of the same approach used in the proof of such a theorem to equalities T_k. □

For what concerns complexity issues, the following theorems can be proved:

Theorem 3.12 *Algorithm* insert_elem *has time complexity $O(Max_{1 \leq j \leq t} \mid F(j) \mid)$.*

Proof. Derives immediately by the algorithm structure. □

```
begin
    if there exists S_i ∈ 𝒮 such that S_i ∈ SC
        then p(e) := 0;
    if there exists no S_i ∈ 𝒮 such that S_i ∈ SC
        then
            begin
             Let S_i be such that w_i = min{w_j | S_j ∈ 𝒮}
             then
                begin
                    SC := SC ∪ {S_i};
                    p(e) := w_i;
                    for each S_j ∈ 𝒮 do
                    begin
                            w_j := w_j − w_i;
                            mark(S_i) := e
                    end;
                    Sets(e) := ∅;
                    for each S_j ∈ 𝒮 do
                    Sets(e) := Sets(e) ∪ S_j
                end
            end
end.
```

Figure 5: Algorithm Insert_elem

```
begin
    Erase element e;
    if there exists S_i ∈ SC such that mark(S_i) = e
    then
        begin
            for each S_j ∈ SU such that e ∈ S_j do
                    w_j := w_j + p(e);
            w_i := p(e);
            SC := SC − {S_i};
            for each e' ∈ S_i do
                    insert_elem (e', Sets(e') − {S_i})
        end
end.
```

Figure 6: Algorithm Delete_elem

Theorem 3.13 *Algorithm* delete_elem *has an* $O(\mid U \mid \cdot Max_{1 \leq j \leq t} \mid F(j) \mid)$ *time complexity.*

Proof. Derives immediately by the algorithm structure. □

It is anyway possible to show that a suitable modification of the above algorithms makes it possible to manage a sequence of *delete_elem* and *insert_elem* operations with a good amortized complexity.

In fact, it is possible, for each set $S \in SC$, to maintain a set $List(S) \subseteq S$, where $list(S)$ is the set of elements in S which are not covered by other sets in SC. Given an element e and a set $S \in SC$ such that $mark(S) = e$, this will make it possible, in correspondence to a *delete_elem(e)* operation, to consider as candidates for insertion only those elements of S which are not covered by other sets. Note that this corresponds to the management of sets $List(v)$ in the Min–Weighted Node Cover case.

Let us consider element e: we denote e as *stabilized* if there exists $S \in SC$ such that $e \in S$ and $mark(S) = e$.

Theorem 3.14 *It is possible to manage a sequence of* insert_elem *and* delete_elem *operations in* $O(Max_{1 \leq j \leq t} \mid F(j) \mid)$ *amortized time complexity.*

Proof. The proof is similar to the one of Theorem 3.7. It is in fact immediate to show that an element can be stabilized only once during its lifetime and that a non stabilized element becomes stabilized the second time it is considered. On the other hand, considering element e_j requires at most $O(\mid F(j) \mid)$ steps, from which the amortized bound derives. □

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy. Proof Verification and Hardness of Approximation Problems. *Proceedings 33rd IEEE Symposium on Foundations of Computer Science* (1992), 14–23.

[2] S. Arora, S. Safra. Probabilistic checking of proofs. *Proceedings 33rd IEEE Symposium on Foundations of Computer Science* (1992),2–132–13

[3] R. Bar-Yehuda, S. Even. A Linear–Time Approximation Algorithm for the Weighted Vertex Cover Problem. *Journal of Algorithms*, **2** (1981), 198–203.

[4] R. Bar-Yehuda, S. Even. On approximating a vertex cover for planar graphs. *Proceedings 14th ACM Symposium on Theory of Computing* (1982), 303-309.

[5] R. Bar-Yehuda, S. Even. A Local Ratio Theorem for Approximating the Weighted Vertex Cover Problem. *Annals of Discrete Mathematics*, **25** (1985), 27–45.

[6] A. Borodin, N. Linial, M. Saks. An Optimal on–line algorithm for metrical task systems. *Proceedings 19th ACM Symposium on Theory of Computing*, (1987), 373–382.

[7] V. Chvatal. A Greedy Heuristic for the Set–Covering Problem. *Mathematics of Operations Research*, **4** (1979), 233–235.

[8] K.L. Clarkson. A modification of the greedy algorithm for vertex cover. *Information Processing Letters*, **16** (1983), 23–25.

[9] D. Coppersmith, P. Raghavan. Multidimensional on-line bin packing: algorithms and worst-case analysis. *Operations Research Letters*, **8** (1989), 17–20.

[10] G. Gambosi, A. Postiglione, M. Talamo. *New Algorithms for on line bin packing. Proceedings 1st Italian Conference on Algorithms and Complexity*, (1990).

[11] G. Gambosi, A. Postiglione, M. Talamo. On–Line Maintenance of an Approximate Bin–Packing Solution. Submitted for publication.

[12] M.R. Garey, D.S. Johnson. *Computers and intractability: A Guide to the theory of NP–completeness.* W.H.Freeman and Company, (1979).

[13] D. Hochbaum. Approximation Algorithms for the Set Covering and Vertex Cover Problems. *SIAM Journal on Computing*, **11** (1982), 555–556.

[14] D. Hochbaum. Efficient Bounds for the Stable Set, Vertex Cover and Set Packing Problems. *Discrete Applied Mathematics*, **6** (1983), 243–254.

[15] Z. Ivkovic, E.L. Lloyd. Fully dynamic maintenance of vertex cover. *Proceedings International Workshop on Graph-Theoretic Concepts in Computer Science*, (1993).

[16] Z. Ivkovic, E.L. Lloyd. Fully dynamic algorithms for Bin Packing. CIS Tech.Rep., no.92-26, Univ. of Delaware, (1993).

[17] D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, **9**, (1974), 256–278.

[18] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardós, S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. *Proceedings 23rd ACM Symposium on Theory of Computing*, (1991), 101–111.

[19] J.H. Lin, J.S. Vitter. ϵ-approximations with minimum packing constraint violation. *Proceedings 24th ACM Symposium on Theory of Computing*, (1992), 771–782.

[20] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, **13**, (1975), 383–390.

[21] M.S. Manasse, L.A. McGeoch, D.D. Sleator. Competitive Algorithms for Server Problems. *Journal of Algorithms*, **11**, (1990), 208–230.

[22] B. Monien, E. Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica*, **22**, (1985), 115–123.

[23] G.L. Nemhauser, L.E. Trotter. Vertex packing: structural properties and algorithms. *Mathematical Programming*, **8**, (1975), 232–248.

[24] P. Ramanan, D.C. Brown, C.C. Lee, D.T. Lee. On-line bin packing in linear time. *Journal of Algorithms*, **10**, (1989), 305–326.

[25] A.C. Yao. New algorithms for bin packing. *Journal of the ACM*, **27**, (1980), 207-227.