A Tabu Search Heuristic Based on k-Diamonds for the Weighted Feedback Vertex Set Problem

Francesco Carrabs[1], Raffaele Cerulli[1], Monica Gentili[2], and Gennaro Parlato[3]

[1] University of Salerno, Department of Mathematics
raffaele@unisa.it
[2] University of Salerno, Department of Computer Science
mgentili@unisa.it
[3] Liafa, CNRS and University Paris Diderot
gennaro@liafa.jussieu.fr

Abstract. Given an undirected and vertex weighted graph $G = (V,E,w)$, the Weighted Feedback Vertex Problem (WFVP) consists of finding a subset $F \subseteq V$ of vertices of minimum weight such that each cycle in G contains at least one vertex in F. The WFVP on general graphs is known to be NP-hard and to be polynomially solvable on some special classes of graphs (e.g., interval graphs, co-comparability graphs, diamond graphs). In this paper we introduce an extension of diamond graphs, namely the *k-diamond* graphs, and give a dynamic programming algorithm to solve WFVP in linear time on this class of graphs. Other than solving an open question, this algorithm allows an efficient exploration of a neighborhood structure that can be defined by using such a class of graphs. We used this neighborhood structure inside our Iterated Tabu Search heuristic. Our extensive experimental show the effectiveness of this heuristic in improving the solution provided by a 2-approximate algorithm for the WFVPon general graphs.

1 Introduction

Given an undirected graph $G = (V,E)$, a *Feedback Vertex Set* (fvs) of G is a subset $F \subseteq V$ of vertices such that each cycle in G contains at least one vertex in F, i.e. the residual graph induced by the set of vertices $V \setminus F$ is acyclic. The *Feedback Vertex Problem* (FVP) consists of finding an fvs of minimum cardinality. When a weight $w(v)$ is associated with each vertex v of G then we have a *vertex weighted graph*. The Weighted Feedback Vertex Problem (WFVP) on a weighted graph G consists of finding an fvs of minimum weight, where the weight of the set is the sum of the weights of its elements. Both FVP and WFVP are NP-complete problems and have application in several areas of computer science such as circuit testing, deadlock resolution, placement of converters in optical networks, combinatorial cut design. This problem becomes polynomial when addressed on diamond graphs [5], co-comparability graphs [6], convex bipartite graphs [6], permutation graphs [14], interval graphs [15]. The best known approximation algorithm for WFVP has approximation ratio 2. The MGA algorithm introduced in [3] was the first one having such an approximation ratio. Other approximation algorithms for the WFVS are proposed in [1,18] for general graphs and in [2,8,13] for special graph classes. There are also exact algorithms finding a minimum FVS in a graph on n vertices in time $O(1.9053^n)$ [17] and in time $O(1.7548^n)$ [9].

J. Pahl, T. Reiners, and S. Voß (Eds.): INOC 2011, LNCS 6701, pp. 589–602, 2011.

In this paper we focus on the weighted feedback vertex set problem (WFVP). In particular, we introduce an extension of diamond graphs, namely the *k-diamond* graphs, and give a linear time algorithm to solve WFVP on it based on a dynamic programming approach. Moreover, we show how this new class of graphs can be used to define a neighborhood structure (namely, the *k-diamond Neighborhood*) of a given feasible solution and, successively, we show how to solve the problem on general graphs by means of a tabu search technique using the k-diamond neighborhood. Such a class of neighborhood was already introduced in [4], where, however, the computational complexity of finding an optimum WFVP on a k-diamond graph was left open and a heuristic approach was used to solve the problem. We solve such an open problem (by giving a linear time algorithm) and also show the effectiveness of the chosen neighborhood in improving a given initial feasible solution when explored by means of our exploration strategy. In order to do this, experimental results are given to show how our Iterative Tabu Search can improve the initial feasible solution, returned by the 2-approximate MGA algorithm [3], when the k-diamond neighborhood is defined and efficiently
explored.

The sequel of the paper is organized as follows. Section 2 introduces the basic notation. Section 3 describes the class of k-diamond graphs and contains the main properties to solve WFVP in linear time on this class. The role of k-diamonds to define a neighborhood structure is described in Section 4, together with the proposed Iterated Tabu Search heuristic. Computational results are reported in Section 5. Finally, concluding remarks are discussed in Section 6.

2 Definitions and Notation

Let $G = (V,E,w)$ be an undirected and vertex weighted graph, where V is the set of n vertices, E is the set of m edges, and, $w(v)$ is a positive weight associated with each vertex $v \in V$. Given a subset $X \subseteq V$ of vertices, let us define its weight $W(X)$ as the sum of the weights of its elements, i.e. $W(X) = \sum_{v \in X} w(v)$ and $\bar{X} = V \setminus X$ its complementary set. If $X = \emptyset$ then $W(X) = 0$. We denote by $G[X]$ the subgraph of G induced by the set of vertices $X \subseteq V$. Formally, $G[X] = (X, E_{[X]}, w)$ where $E_{[X]} = \{(x,y) \in E : x,y \in X\}$. A *tree* T_r rooted in r is an acyclic and connected graph. We define a *forest* $\mathscr{F}$ as a graph where any connected component is a tree. A subset of vertices X is a feedback vertex set of G if and only if $G[\bar{X}]$ is a forest. From now on we denote by $F(G)$ and $F^*(G)$, any feedback vertex set and the minimum weight feedback vertex set of G, respectively. When no confusion may arise we simply denote these sets by F and F^* respectively. Moreover, we define $F_{\bar{v}}$ a feedback vertex set of G not containing vertex v. A vertex $v \in F$ is *redundant* if and only if $F \setminus \{v\}$ is a feedback vertex set of G. Any vertex $v \in V$ is said to be *appended* if it is not included in any cycle of G. Obviously, a set of vertices is an fvs of G if and only if it is an fvs of the graph G' obtained from G after deleting all the appended vertices. We say a graph is *reduced* if it does not contain any appended vertex. The reduction operation of a graph can be performed in linear time. W.l.o.g., from now on we suppose graph G to be a reduced graph. For any additional definition and notation we refer to [7].

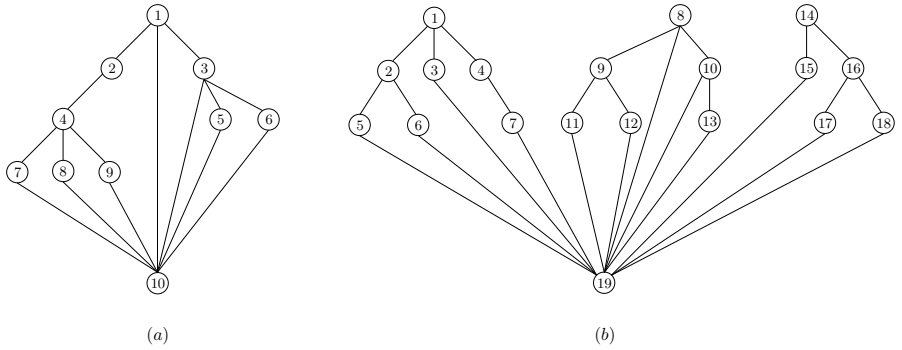

Fig. 1. *(a) A diamond with upper apex $r = 1$ and lower apex $z = 10$. (b) A 3-diamond with upper apices $R = \{1, 8, 14\}$ and lower apex $z = 19$. Note that, as stated by property 1, it is composed by the three diamonds D_1, D_8 and D_{14}.*

3 The Class of k-Diamond Graphs

In this section we first recall the definition of the class of diamond graphs introduced in [5], and successively we formally describe the extended class of k-diamond graphs. Then, we prove the basic properties that are useful to optimally solve WFVP on this new class in linear time.

A weighted diamond $D_{r,z} = (V_r, E_r, w)$ is an undirected and vertex weighted graph where (i) each vertex $v \in V_r$ is included in at least one simple path between r and z and (ii) $D_{r,z}[\bar{z}]$ is a tree. The two vertices r and z are called the *upper* and *lower* apex of a diamond $D_{r,z}$, respectively, and, the subgraph $D_{r,z}[\bar{z}]$ is referred to as the tree T_r rooted in r associated with $D_{r,z}$. In Figure 1(a) the diamond $D_{1,10}$ with upper apex $r = 1$ and lower apex $z = 10$ is shown. Note that by deleting vertex z we obtain the tree $T_1 = D_{1,10}[\bar{1}\bar{0}]$.

As shown in [5], WFVP can be solved in linear time on a diamond graph by a dynamic programming algorithm. Let us refer to such an algorithm as DP. In the sequel we show how to use DP to solve WFVP in linear time on a k-diamond. A k-diamond is a generalization of a diamond where multiple upper apices are allowed, formally:

Definition 1. *A weighted k-diamond $D_{R,z} = (V_R, E_R, w)$, where $k \geq 1$, $R = \{r_1, r_2, \ldots, r_k\} \subseteq V_R$ and $z \in V_R$, is an undirected and vertex weighted graph where (i) each vertex $v \in V_R$ is included in a simple path between exactly one of the k apices $r_i \in R$ and z and (ii) $D_{R,z}[\bar{z}]$ is a forest with k connected components.*

Following the definition introduced for diamond graphs, we refer to the set of vertices R and to vertex z of $D_{R,z}$ as the set of *upper apices* and the *lower apex* of $D_{R,z}$, respectively. The subgraph $D_{R,z}[\bar{z}]$ is referred to as the forest $\mathscr{F}_R$, associated with $D_{R,z}$, whose connected components are the k trees T_{r_i} rooted in $r_i \in R$. Figure 1(b) shows a 3-diamond. The set of upper apices is composed of the three vertices $R = \{1, 8, 14\}$, while the lower apex is vertex $z = 19$. The graph obtained from $D_{R,z}$, after deleting the lower apex, is a forest with the three connected components T_1, T_8 and T_{14} . To keep notation simple, in the sequel of the paper and when no confusion may arise, we denote a k-diamond $D_{R,z}$,

with $R = \{r_1, \ldots, r_k\}$, just by D_R and a diamond graph $D_{r,z}$ by D_r. Note that for $k = 1$ a k-diamond is a diamond. Moreover, it is easy to see that the following property holds:

Property 1 (Decomposition). Any k-diamond D_R is composed by k distinct diamonds D_{r_i}, with $r_i \in R$, having all the same lower apex z.

For instance, the 3-diamond depicted in Figure 1(b) is composed by three diamonds D_1, D_8 and D_{14}. We will see in the following how to use the decomposition property to solve WFVP on D_R. By definition of k-diamond, the following properties obviously hold.

Property 2. The singleton $\{z\}$ is an *fvs* of D_R.

Property 3. Every cycle of D_R contains vertex z and vertices belonging to the same connected component of $\mathscr{F}_R$.

Observe that, by property 2, a minimum weight feedback vertex set $F^*(D_R)$ of D_R either contains vertex z or not. Therefore, to find $F^*(D_R)$ we can proceed as follows: (i) compute the minimum weight feedback vertex set $F^*_{\bar{z}}(D_R)$ that does not contain z; (ii) if $W(F^*_{\bar{z}}(D_R)) < w(z)$ then set $F^*(D_R) = F^*_{\bar{z}}(D_R)$ otherwise set $F^*(D_R) = \{z\}$. The computation of $F^*_{\bar{z}}(D_R)$ can be carried out by finding the fvs $F^*(D_{R_i})$ of minimum weight on each of the k diamonds D_{r_i} that compose D_R as proven by the following lemma:

Lemma 1. *Given the k-diamond D_R, let $F^*_{\bar{z}}(D_{r_i})$, $\forall r_i \in R$, be a minimum weight feedback vertex set of diamond D_{r_i} not containing vertex z. Then: $F^*_{\bar{z}}(D_R) = \bigcup_{r_i \in R} F^*_{\bar{z}}(D_{r_i})$.*

Proof. Let $X = \bigcup_{r_i \in R} F^*_{\bar{z}}(D_{r_i})$. We need to prove that X is a minimum fvs of D_R not containing z, i.e. $X = F^*_{\bar{z}}(D_R)$. From property 3 and by definition of $F^*_{\bar{z}}(D_{r_i})$, it is evident that X is an fvs of D_R therefore we have only to prove that X is minimum. Let us suppose, by contradiction, there exists another fvs, say Y, such that $z \notin Y$ and $W(Y) < W(X)$. Let $Y_i = Y \cap D_{r_i}$. By property 3, each set Y_i is an fvs of D_{r_i} that does not contain vertex z. Therefore, since $Y = \bigcup_{r_i \in R} Y_i$ and $W(Y) < W(X)$, there must exist at least a set, say Y_h such that $W(Y_h) < W(F^*_{\bar{z}}(D_{r_h}))$: a contradiction. □

Corollary 1. *Given the k-diamond D_R, a minimum weight feedback vertex set $F^*(D_R)$ is either the set $F^*_{\bar{z}}(D_R)$ or the singleton $\{z\}$.*

From Corollary 1, the problem of finding an optimum WFVS on a k-diamond is reduced to compute $F^*_{\bar{z}}(D_{r_i})$ on each of the k diamonds that compose D_R. These fvs can be computed using the *DP* algorithm given in [5]. Fig. 2 reports the pseudo-code of our algorithm DP_{multi} that solves WFVP on k-diamonds. Theorem 1 to follow proves that this algorithm runs in linear time.

Theorem 1. *Given a k-diamonds $D_R = (V_R, E_R, w)$, the DP_{multi} algorithm computes $F^*(D_R)$ in $O(|V_R|)$ time.*

Proof. The computation of $F^*_{\bar{z}}(D_{r_i})$ carried out in step 1 of DP_{multi} algorithm takes $O(|V_{r_i}|)$ time (see [5]). Since this computation is repeated for each root $r_i \in R$, then the

Procedure: DP_{multi}

Step 1. **for all** D_{r_i} **compute** $F^*_{\bar{z}}(D_{r_i})$;
Step 2. **Set** $F^*_{\bar{z}}(D_R) \leftarrow \bigcup_{r_i \in R} F^*_{\bar{z}}(D_{r_i})$;
Step 3. **if** $W(F^*_{\bar{z}}(D_R)) < w(z)$ **then** $F^*(D_R) \leftarrow F^*_{\bar{z}}(D_R)$ **otherwise** $F^*(D_R) \leftarrow \{z\}$;
Step 4. **Return** $F^*(D_R)$;

Fig. 2. Pseudo code of algorithm DP_{multi}

total cost of step 1 is equal to $O(|V_R|)$ time. The joining operation carried out at step 2 requires $O(k)$ time, while step 3 and step 4 require constant time. Consequently, DP_{multi} runs in $O(|V_R|)$ time. □

The next sections contain a description of a general neighborhood structure based on the class of k-diamonds and introduce an operator that, using DP_{multi}, efficiently explores such a neighborhood. This operator will be later embedded into our Iterated Tabu Search.

4 The Neighborhood Structures and the Iterative Tabu Search

The basic paradigm of tabu search is to use information about the search history to guide local search approaches to overcome local optimality. Based on some sort of memory certain moves may be forbidden, we say they are set tabu (and appropriate move attributes are put into a list, the so-called tabu list). The search may imply acceptance deteriorating moves when no improving moves exist or all improving moves of the current neighborhood are set tabu. We implemented an extension of the standard Tabu Search [10,11,12] (TS), namely the *Iterated Tabu Search* [16] (ITS), whose central idea is based on the concept of *intensification* and *diversification*. The intensification phase is focused in finding a better (locally optimal) solution in "surroundings", i.e. neighborhood, of the current solution. The ITS method uses the classical TS to achieve such an improvement. The Diversification phase is used whenever the tabu memory indicates that one is trapped in a certain basin of attraction and then allows to escape from the current local optimum and to move towards new regions in the solution space.

In the following subsections the main components of the algorithm are described: (i) the neighborhood structures (namely, the k-diamond neighborhood and the 2-exchange neighborhood), (ii) the corresponding exploration strategies (namely, the $Single-Insert$ and the $Double-Insert$ operators, respectively); (iii) the tabu list and (iv) the diversification phase. The pseudo-code of our Iterative Tabu Search (ITS) is given in Fig. 5.

4.1 The k-Diamond Neighborhood

Given a graph G, let F be any not redundant fvs of G and $\mathscr{F} = G[\bar{F}]$ the forest induced by vertices not in F. By inserting a vertex $z \in F$ in $\mathscr{F}$ a k-diamond D_R is obtained. Let I_z

be an fvs of D_R not containing z, then the set $F' = I_z \cup \{F \setminus \{z\}\}$ is a new fvs of G. Note that, F' could contain redundant vertices. Let O_z be the set of the redundant vertices of F'. Note that, by construction of I_z, we have $O_z \subseteq F \setminus \{z\}$. Add z to O_z and consider the vertex set $F_{new} = I_z \cup \{F \setminus O_z\}$. F_{new} is a not redundant fvs of G and: if $W(I_z) < W(O_z)$, its weight is lower than the weight of F. Given a vertex $z \in F$, we define the couple (I_z, O_z) an *exchange set* of z, formally:

Definition 2. *Given a vertex $z \in F$, the couple (I_z, O_z), where $I_z \subseteq V \setminus F$, $O_z \subseteq F$ and $z \in O_z$, is a exchange set of z if the set $I_z \cup \{F \setminus O_z\}$ is a not redundant* fvs *of G.*

Let us denote by $\mathscr{E}(F,z)$ the collection of all the exchange sets associated with $z \in F$, i.e. $\mathscr{E}(F,z) = \big\{(I_z, O_z) : I_z \cup \{F \setminus O_z\}$ *is a not reduntant fvs of* $G\big\}$. The k-diamond neighborhood is defined as follows:

Definition 3. *Given a graph G and an fvs F, the k-diamond neighborhood $\mathscr{N}(F)$ is the set of all not redundant fvs of G that can be obtained from F through the exchange sets associated with each vertex $z \in F$:*

$$\mathscr{N}(F) = \Big\{ I_z \cup \{F \setminus O_z\} : (I_z, O_z) \in \mathscr{E}(F,z), \forall z \in F \Big\}$$

Note that, given a vertex $z \in F$, finding the minimum cost set I_z associated with it corresponds to find the minimum weight feedback vertex set on the k-diamond associated with z. Hence, by applying the DP_{multi} algorithm we can perform an implicit exhaustive exploration on the neighborhood to find a local optimum in polynomial time. This exploration is carried out by our first operator $Single-Insert$ that is described next.

The $Single-Insert$ **Operator** Given a not redundant fvs F of G and the incumbent solution F^*, the $Single-Insert$ operator builds, for each $z \in F$, the k-diamond D_R by introducing z in $\mathscr{F} = G[\bar{F}]$. Successively, it computes an exchange set (I_z, O_z) where $I_z = F^*_{\bar{z}}(D_R)$, i.e I_z is the minimum feedback vertex set of D_R not containing z. The operator selects the best exchange set (I^*_z, O^*_z) such that $W(I^*_z) - W(O^*_z) = min_{(I_z,O_z):z\in F}\{W(I_z) - W(O_z)\}$. More in detail (see Fig. 3), the operator builds the k-diamond D_R (step 1), finds the fvs $F^*_{\bar{z}}(D_R)$ by applying algorithm DP_{multi} and sets $I_z \leftarrow F^*_{\bar{z}}(D_R)$ (step 2). The operator (step 3) finds redundant vertices (if any) of the new fvs $F_{new} = F \setminus \{z\} \cup \{I_z\}$ to be inserted in O_z (initially $O_z = \{z\}$). To this end, $Single-Insert$ builds the forest $\mathscr{F}' = G[\bar{F}_{new}]$ and reintroduces, one by one, each vertex $z' \in F \setminus \{z\}$ to check whether z' is redundant or not. If z' is redundant then it is moved from F_{new} to O_z. The final fvs $F_{new} = I_z \cup \{F \setminus O_z\}$ is then obtained after all the vertices in $F \setminus \{z\}$ are checked for redundancy. Note that the pair (I_z, O_z) is the move corresponding to the transition from solution F to its neighbor F_{new}.

The weight of the new set F_{new} is then compared with the weight of the incumbent solution F^* found so far. If $W(F_{new}) < W(F^*)$, then the operator sets the best move (I^*_z, O^*_z) equal to (I_z, O_z) even if this move is tabu (this represents the application of an aspiration criterion [12]). Otherwise, if (I_z, O_z) is not tabu and the corresponding solution is better than the solution associated with (I^*_z, O^*_z), the algorithm sets (I^*_z, O^*_z) equal to (I_z, O_z). Finally, if both previous cases do not hold, (I_z, O_z) is neglected.

Procedure: *Single – Insert(G,F, F*)*
Set $W(I_z^*) \leftarrow \infty$, $W(O_z^*) \leftarrow 0$
for all $z \in F$ **do**

Step 1. Insert z in $G[\bar{F}]$ and reduce the obtained graph to produce the k-diamond D_R;
Step 2. Set $I_z \leftarrow F_z^*(D_R)$;
Step 3. Find the set of redundant nodes O_z, add z to O_z, and set $F_{new} \leftarrow I_z \cup \{F \setminus O_z\}$;
Step 4. **if** $W(F_{new}) < W(F^*)$ **do** *// aspiration criterion //*
$I_z^* \leftarrow I_z$,$O_z^* \leftarrow O_z$;
else if $W(I_z) - W(O_z) < W(I_z^*) - W(O_z^*)$ **and** (I_z, O_z) *is not tabu* **do**
$I_z^* \leftarrow I_z$,$O_z^* \leftarrow O_z$;

end for
return $(I_z^* \cup \{F \setminus O_z^*\})$;

Fig. 3. *Pseudo-code of operator Single – Insert*

4.2 The 2-Exchange Neighborhood and the *Double – Insert* Operator

Additional neighborhoods similar to the k-diamond neighborhood above described can be considered if more than one vertex of F is selected to be introduced in $\mathscr{F} = G[\bar{F}]$. Indeed, a drawback of the *Single – Insert* operator concerns the diversification of the explored solutions. In fact, when there are not redundant vertices, only one vertex (the lower apex z) is moved from F to $\mathscr{F} = G[\bar{F}]$. Hence, in the worst case, several applications of the operator *Single – Insert* are necessary to remove more than one vertex from F. In order to overcome this issue, we consider a new neighborhood, namely the 2-exchange neighborhood, to diversify the explored solutions, that is, we considered the case when two vertices $\{z_i, z_j\}$ are selected to be inserted in $\mathscr{F}$. This neighborhood is explored by the operator *Double – Insert* (see Fig. 4) that differs from *Single – Insert* since it inserts two lower apices $\{z_i, z_j\}$ into $\mathscr{F}$, and finds the fvs I_{z_i,z_j} by applying algorithm MGA. MGA is a greedy algorithm that selects at each iteration the vertex v such that the ratio $w(v)/d(v)$ is minimum, where $d(v)$ is the degree of the vertex. When a vertex is selected, it is removed from G and G is then reduced to obtain the subgraph G'. The degree of each vertex v in G' is updated and for each edge (u,v) that was removed during the reduction process, the weight of its endpoints is decreased by the quantity $w(v)/d(v)$. The selection of a new vertex is then carried out on G' until it is not empty. For more details on MGA the reader can refer to [3].

The *Single – Insert* operator and the *Double – Insert* operator will be used during the intensification phase of Iterative Tabu Search metaheuristic.

4.3 The Tabu List

At iteration t, after a relocation of vertices is carried out according to a resulting exchange set (I_z, O_z), the inverse move (O_z, I_z) cannot be carried out for the next Δ iterations, where Δ is the tabu list size. To implement a fast way for storing each move (O_z, I_z) we use a bit mask and an hash-table. Since both I_z and O_z are vertex sets and each vertex has a distinct ID, we allocate two bit-mask bl and br whose size is $|V|$. We set the bits in bl and br corresponding to the vertices in O_z and I_z, respectively, equal to

Procedure: *Double – Insert(G,F,F*)*
Set $W(I_C^*) \leftarrow \infty$, $W(O_C^*) \leftarrow 0$
for all pair $C = (z_i, z_j)$ with $z_i, z_j \in F$ **do**

Step 1. Insert z_i and z_j in $G[\bar{F}]$ and reduce it to obtain G';
Step 2. Apply MGA to find an fvs I_C of G';
Step 3. Find the set of redundant nodes O_C, add z_i and z_j to O_C and set $F_{new} \leftarrow I_C \cup \{F \setminus O_C\}$;
Step 4. **if** $W(F_{new}) < W(F^*)$ **do** *// aspiration criterion //*
$I_C^* \leftarrow I_C$, $O_C^* \leftarrow O_C$;
else if $W(I_C) - W(O_C) < W(I_C^*) - W(O_C^*)$ **and** (I_C, O_C) is not tabu **do**
$I_C^* \leftarrow I_C$, $O_C^* \leftarrow O_C$;

end for
return $I_C^* \cup \{F \setminus O_C^*\}$;

Fig. 4. *Pseudo-code of operator Double – Insert*

1. The two strings are then concatenated to generate the string of bits *bl-br*, that is the *key* associated with the move. This key, that is unique for each move, is given to the hash function to save the move. To verify if a move is tabu it is sufficient to generate its key and check whether it is inside the hash table. The key generation, the insertion into the hash table and the checking operations require $O(|I_z| + |O_z|)$ time. The keys are saved inside a FIFO queue whose size is Δ, hence when the queue is full and a new key has to be inserted, the key on the head is removed from the queue and from the hash table. This operation requires constant time. We used a reactive tabu list, that uses a list whose size is dynamically updated during the computation according to the evolution of the search. The value of Δ ranges between a lower bound β^- and an upper bound β^+ that are fixed at the beginning of the computation and never change. Given an initial fvs F, we set $\beta^- = 5$, $\beta^+ = \max\left\{3\beta^-, \frac{|F|}{3}, \frac{|\bar{F}|}{3}\right\}$ and $\Delta = \beta^- + \frac{(\beta^+ - \beta^-)}{2}$. After each iteration t, if the new solution F' found during the intensification phase (steps 4-17 in Fig. 5) is better than F^*, then Δ is increased by one. Otherwise, if F' is worse than F^* but better than the solution found at the previous iteration then Δ is not changed. Finally, if F' is worse than the previous one then Δ is decreased by one.

4.4 The Diversification Phase

The diversification phase is implemented using a modified version of the *Double – Insert* operator (namely the *Multi – Insert* operator). Given a solution F, *Multi – Insert* differs from *Double – Insert* since a subset of vertices $P \subset F$ with $|P| > 2$ is inserted into the forest $\mathscr{F} = G[\bar{F}]$ to obtain a new graph G'. There are three main aspects to take into account in the diversification phase: (i) when to apply the diversification and on which solution, (ii) the cardinality of the set P, and, (iii) which vertices to introduce in P. We apply the diversification either to the best solution F^* found so far (step 23 in Fig. 5) or to the solution F' (step 25 in Fig. 5) computed during the intensification phase. We keep a counter q that ranges from 1 to θ (that is the maximum number of diversification operations performed by the algorithm) and, as soon as this bound is reached, the ITS stops. The cardinality of P is computed according to the following

Procedure: $ITS(G,\theta,\sigma)$

```
1: F ← F* ← MGA(G);
2: for q = 1 to θ do
3:    // Intensification Phase
4:    F' ← V;
5:    for h = 1 to σ do
6:       F1 ←Single-Insert(G,F,F*);
7:       F2 ←Double-Insert(G,F,F*);
8:       if W(F1) < W(F2) then
9:          F ← F1;
10:      else
11:         F ← F2;
12:      end if
13:      Save the inverse move into the tabu list.
14:      if W(F) < W(F') then
15:         F' ← F; h ← 1;
16:      end if
17:   end for
18:   if W(F') < W(F*) then
19:      F* ← F';
20:   end if
21:   // Diversification Phase
22:   if q is even then
23:      F ← Diversification(F*);
24:   else
25:      F ← Diversification(F');
26:   end if
27: end for
28: return F*;
```

Fig. 5. *Pseudo-code of Iterated Tabu Search*

formula: $\max\left\{5, \frac{|F|\times(20+5q)}{100}\right\}$. Finally, to remove vertices from F we consider the last iteration $it^+(v)$ when v has been inserted in F: the vertices of F are sorted in increasing order according to $it^+(v)$ and the first $|P|$ vertices of F are selected.

5 Computational Results

The ITS algorithm was coded in C and run on a 2.33 GHz Intel Core2 Q8200 processor. Since there are no available benchmark instances for the WFVP, we generated instances for the following class of graphs: random graphs, squared and not squared grids, taurus and hypercube. Each instance is characterized by the number of vertices, the number of edges, a seed and a range of values for the weight of the vertices. The weight ranges are: 10-25, 10-50 and 10-75. For each combination of parameters we generated five instances with the same characteristics except for the seed. The results reported in the tables are average values over these five instances. Small instances have 25, 50 and 75 vertices. Large instances have 100, 200, 300, 400 and 500 vertices.

Tables 1-2-3 report the results of the MGA algorithm and of our ITS algorithm. The first and second columns in each table report the id and characteristics of each instance, respectively. For random graphs (Table 1): number of vertices (n), number of edges (m), the lower (low) and upper (up) bounds for the weight of each vertex. For the hypercube graphs (Table 2a and 3a) the number of edges is omitted because it depends on the number of vertices. For the remaining graph classes: x is the number of rows and y is the number of columns. The third column in each table reports the solution value returned

Table 1. *(a) Test results on random graphs: (a) small instances and (b) large instances*

(a)

RANDOM GRAPHS: Small Instances									
ID	**Instance**				**MGA**	**ITS**		**GAP**
	n	**m**	**low**	**up**	**Value**	**Value**	**Time**	
1	25	33	10	25	70.8	63.8	0.00	-9.89%
2	25	33	10	50	105.4	99.8	0.00	-5.31%
3	25	33	10	75	133.6	125.2	0.00	-6.29%
4	25	69	10	25	166.8	157.6	0.00	-5.52%
5	25	69	10	50	294.8	272.2	0.00	-7.67%
6	25	69	10	75	455	409.4	0.00	-10.02%
7	25	204	10	25	286.4	273.4	0.02	-4.54%
8	25	204	10	50	527	507	0.01	-3.80%
9	25	204	10	75	829.8	785.8	0.01	-5.30%
10	50	85	10	25	191.4	175.4	0.03	-8.36%
11	50	85	10	50	298.2	280.8	0.03	-5.84%
12	50	85	10	75	377.2	348	0.02	-7.74%
13	50	232	10	25	409	389.4	0.07	-4.79%
14	50	232	10	50	746.8	708.6	0.06	-5.12%
15	50	232	10	75	1018.4	951.6	0.04	-6.56%
16	50	784	10	25	612.6	602.2	0.11	-1.70%
17	50	784	10	50	1204.2	1172.2	0.15	-2.66%
18	50	784	10	75	1685.2	1649.4	0.14	-2.12%
19	75	157	10	25	347.2	321	0.13	-7.55%
20	75	157	10	50	571.2	526.2	0.14	-7.88%
21	75	157	10	75	815	757.2	0.11	-7.09%
22	75	490	10	25	654.2	638.6	0.16	-2.38%
23	75	490	10	50	1286.6	1230.6	0.27	-4.35%
24	75	490	10	75	1870.8	1793.6	0.13	-4.13%
25	75	1739	10	25	903.2	891	0.40	-1.35%
26	75	1739	10	50	1681	1664.8	0.35	-0.96%
27	75	1739	10	75	2479.8	2452.8	0.33	-1.09%
AVG								**-5.18%**

(b)

RANDOM GRAPHS: Large Instances									
ID	**Instance**				**MGA**	**ITS**		**GAP**
	n	**m**	**low**	**up**	**Value**	**Value**	**Time**	
1	100	247	10	25	536.4	501.4	0.33	-6.52%
2	100	247	10	50	910.4	845.8	0.37	-7.10%
3	100	247	10	75	1279.2	1223.8	0.28	-4.33%
4	100	841	10	25	846	828.2	0.27	-2.10%
5	100	841	10	50	1793.2	1729.6	0.60	-3.55%
6	100	841	10	75	2512.2	2425.6	0.35	-3.45%
7	100	3069	10	25	1151.2	1134	0.59	-1.49%
8	100	3069	10	50	2218	2179	0.69	-1.76%
9	100	3069	10	75	3284	3228.8	0.77	-1.68%
10	200	796	10	25	1547.8	1488.4	3.48	-3.84%
11	200	796	10	50	2544.2	2442.6	2.50	-3.99%
12	200	796	10	75	3277.4	3157	2.78	-3.67%
13	200	3184	10	25	2035.6	2003.6	2.78	-1.57%
14	200	3184	10	50	3775.2	3683.6	2.67	-2.43%
15	200	3184	10	75	5259	5158.6	2.76	-1.91%
16	200	12139	10	25	2467.4	2450	11.31	-0.71%
17	200	12139	10	50	4182.2	4149.4	8.91	-0.78%
18	200	12139	10	75	5568.8	5531.4	6.98	-0.67%
19	300	1644	10	25	2136.6	2072.6	10.19	-3.00%
20	300	1644	10	50	4384.6	4239.4	9.12	-3.31%
21	300	1644	10	75	6411.2	6154.4	11.09	-4.01%
22	300	7026	10	25	3267.6	3231	19.59	-1.12%
23	300	7026	10	50	6368.4	6261.4	21.12	-1.68%
24	300	7026	10	75	8825.2	8660.6	17.21	-1.87%
25	300	27209	10	25	3749.2	3729.2	44.74	-0.53%
26	300	27209	10	50	5774.2	5738	29.26	-0.63%
27	300	27209	10	75	10514	10469.6	50.88	-0.42%
28	400	2793	10	25	3097	3015.2	29.99	-2.64%
29	400	2793	10	50	6726.8	6528	35.82	-2.96%
30	400	2793	10	75	9006.8	8730	35.36	-3.07%
31	400	12369	10	25	4514.4	4451.8	55.14	-1.39%
32	400	12369	10	50	6896	6837.4	35.88	-0.85%
33	400	12369	10	75	10788.8	10661.8	48.12	-1.18%
34	400	48279	10	25	5090	5060.8	123.27	-0.57%
35	400	48279	10	50	7142.6	7109.2	85.15	-0.47%
36	400	48279	10	75	15202.4	15114.6	127.31	-0.58%
37	500	4241	10	25	4197.4	4102.8	68.35	-2.25%
38	500	4241	10	50	7447.6	7285	70.14	-2.18%
39	500	4241	10	75	11619.6	11285.6	63.93	-2.87%
40	500	19211	10	25	5817.2	5745.8	99.12	-1.23%
41	500	19211	10	50	7819.2	7725	89.63	-1.20%
42	500	19211	10	75	14335.4	14167.8	80.09	-1.17%
43	500	75349	10	25	6388.6	6366.4	181.71	-0.35%
44	500	75349	10	50	8709	8671.2	155.18	-0.43%
45	500	75349	10	75	16994.6	16939.2	201.96	-0.33%
AVG								**-2.09%**

Table 2. *(a) Test results on small instances:(a) hypercube graphs, (b) taurus graphs, (c) squared grid graphs and (d) not squared grid graphs*

(a)

HYPERCUBE GRAPHS: Small Instances							
ID	Instance			MGA	ITS		GAP
	n	low	up	Value	Value	Time	
1	16	10	25	77.4	72.2	0.00	-6.72%
2	16	10	50	99.8	93.8	0.00	-6.01%
3	16	10	75	99.8	97.4	0.00	-2.40%
4	32	10	25	177.2	170	0.01	-4.06%
5	32	10	50	249.4	241	0.00	-3.37%
6	32	10	75	286.2	277.6	0.00	-3.00%
7	64	10	25	377.6	354.6	0.13	-6.09%
8	64	10	50	486.2	476	0.05	-2.10%
9	64	10	75	514	503.8	0.05	-1.98%
AVG							**-3.97%**

(b)

TAURUS GRAPHS: Small Instances								
ID	Instance				MGA	ITS		GAP
	x	y	low	up	Value	Value	Time	
1	5	5	10	25	113.2	101.4	0.00	-10.42%
2	5	5	10	50	135.2	124.4	0.00	-7.99%
3	5	5	10	75	167.4	157.8	0.00	-5.73%
4	7	7	10	25	206	197.4	0.03	-4.17%
5	7	7	10	50	243.4	234.2	0.02	-3.78%
6	7	7	10	75	282.6	269.6	0.02	-4.60%
7	9	9	10	25	324.8	310.4	0.20	-4.43%
8	9	9	10	50	388.4	370	0.17	-4.74%
9	9	9	10	75	448.4	432.2	0.16	-3.61%
AVG								**-5.50%**

(c)

SQUARED GRID GRAPHS: Small Instances								
ID	Instance				MGA	ITS		GAP
	x	y	low	up	Value	Value	Time	
1	5	5	10	25	122.4	114	0.00	-6.86%
2	5	5	10	50	208.4	199.8	0.00	-4.13%
3	5	5	10	75	335.2	312.6	0.00	-6.74%
4	7	7	10	25	270.8	252.4	0.03	-6.79%
5	7	7	10	50	464.6	439.8	0.03	-5.34%
6	7	7	10	75	749.4	718.4	0.03	-4.14%
7	9	9	10	25	466	444.2	0.22	-4.68%
8	9	9	10	50	805.8	754.6	0.29	-6.35%
9	9	9	10	75	1209.6	1138	0.13	-5.92%
AVG								**-5.66%**

(d)

NOT SQUARED GRID GRAPHS: Small Instances								
ID	Instance				MGA	ITS		GAP
	x	y	low	up	Value	Value	Time	
1	8	3	10	25	104.8	96.8	0.00	-7.63%
2	8	3	10	50	174.8	157.4	0.00	-9.95%
3	8	3	10	75	246.6	220	0.00	-10.79%
4	9	6	10	25	326.4	295.8	0.07	-9.37%
5	9	6	10	50	512	489.4	0.04	-4.41%
6	9	6	10	75	801	755	0.04	-5.74%
7	12	6	10	25	431.6	399.8	0.15	-7.37%
8	12	6	10	50	717.2	673.4	0.12	-6.11%
9	12	6	10	75	1092.8	1017.4	0.10	-6.90%
AVG								**-7.59%**

by MGA. We do not report the computational time of MGA since it is always negligible. Fourth and fifth columns in the tables report the solution value and the computational time (in seconds) of our ITS algorithm. Finally, last column reports the percentage gap between the solution values returned by the two algorithms. This gap is positive if MGA finds a better solution than ITS and negative otherwise. The last line of the tables reports the average value of this gap computed on all the instances of the table. On small instances of random graphs (Table 1) we can see from the gap column that ITS always finds a better solution than MGA and the CPU time is less than half of a second. On the 27 instances of Table 1a, this gap is greater than 5% for 15 instances and in one case (instance 6) it is greater than 10%. On average, the improvement obtained by ITS is around 5%. It is interesting to observe that as the density of graph increases the gap decreases. This reveals that the selection criterion applied by MGA (the ratio between weight and degree of node) is less effective on sparse graphs. This trend is evident on large instances (Table 1b) where (see for example instances with 500 vertices) the gap for sparse instances is more that 2% and it is less that 0.4% on more dense instances. The computational time of ITS is less than 1 minute for the first 33 instances and is less that 4 minutes for the remaining large instances.

Consider Table 2 (for small instances) and Table 3 (for large instances) to compare the algorithms on the other types of graph. Let us analyze the small instances for the hypercube graphs. From the gap column, we can see that in three cases (instances 1,

Table 3. *(a) Test results on large instances: (a) hypercube, (b) taurus, (c) squared grid and (d) not squared grid graphs*

(a)

HYPERCUBE GRAPHS: Large Instances							
ID	Instance			MGA	ITS		GAP
	n	low	up	Value	Value	Time	
1	128	10	25	784.8	740	1.09	-5.71%
2	128	10	50	1125.4	1071	0.40	-4.83%
3	128	10	75	1196.4	1163.6	0.34	-2.74%
4	256	10	25	1641.2	1542.6	9.41	-6.01%
5	256	10	50	2429.4	2311.4	6.45	-4.86%
6	256	10	75	2673.4	2590.8	3.94	-3.09%
7	512	10	25	3416.4	3240.8	73.51	-5.14%
8	512	10	50	5147.4	4921.8	67.58	-4.38%
9	512	10	75	5789.2	5588.6	51.74	-3.47%
AVG							**-4.47%**

(b)

TAURUS GRAPHS: Large Instances								
ID	Instance				MGA	ITS		GAP
	x	y	low	up	Value	Value	Time	
1	10	10	10	25	413	388.8	0.38	-5.86%
2	10	10	10	50	476.4	458.6	0.37	-3.74%
3	10	10	10	75	523	504.8	0.25	-3.48%
4	14	14	10	25	793.8	750.8	5.96	-5.42%
5	14	14	10	50	908.2	875.6	3.68	-3.59%
6	14	14	10	75	1062.4	1017.2	3.59	-4.25%
7	17	17	10	25	1167.4	1110.2	21.98	-4.90%
8	17	17	10	50	1364.8	1307.6	20.93	-4.19%
9	17	17	10	75	1551.4	1502.4	23.18	-3.16%
10	20	20	10	25	1621.2	1548.6	88.75	-4.48%
11	20	20	10	50	1867.2	1803.4	81.03	-3.42%
12	20	20	10	75	2109.6	2042.6	55.19	-3.18%
13	23	23	10	25	2136.4	2043.4	278.08	-4.35%
14	23	23	10	50	2520	2412.2	177.53	-4.28%
15	23	23	10	75	2818.8	2705.4	184.99	-4.02%
AVG								**-4.15%**

(c)

SQUARED GRID GRAPHS: Large Instances								
ID	Instance				MGA	ITS		GAP
	x	y	low	up	Value	Value	Time	
1	10	10	10	25	613	570.6	0.54	-6.92%
2	10	10	10	50	1002	948.8	0.41	-5.31%
3	10	10	10	75	1657.4	1566	0.51	-5.51%
4	14	14	10	25	1273.6	1209.4	8.07	-5.04%
5	14	14	10	50	2103	2008.6	8.06	-4.49%
6	14	14	10	75	3618.6	3401.2	7.23	-6.01%
7	17	17	10	25	1917	1834.2	42.63	-4.32%
8	17	17	10	50	3231	3070.6	29.71	-4.96%
9	17	17	10	75	5380.8	5089.8	29.68	-5.41%
10	20	20	10	25	2781	2619.8	85.42	-5.80%
11	20	20	10	50	4516.8	4321.2	103.84	-4.33%
12	20	20	10	75	7650.4	7272.6	127.81	-4.94%
13	23	23	10	25	3626.8	3462.8	371.23	-4.52%
14	23	23	10	50	6171.4	5865.4	291.52	-4.96%
15	23	23	10	75	10195.6	9723.4	240.50	-4.63%
AVG								**-5.14%**

(d)

NOT SQUARED GRID GRAPHS: Large Instances								
ID	Instance				MGA	ITS		GAP
	x	y	low	up	Value	Value	Time	
1	13	7	10	25	552	513	0.36	-7.07%
2	13	7	10	50	870	803.4	0.31	-7.66%
3	13	7	10	75	1471	1390.8	0.34	-5.45%
4	18	11	10	25	1284.6	1208	6.78	-5.96%
5	18	11	10	50	2149.2	2049.8	8.77	-4.62%
6	18	11	10	75	3643.6	3431	5.79	-5.83%
7	23	13	10	25	2049.4	1930.6	42.54	-5.80%
8	23	13	10	50	3366.2	3194.8	43.01	-5.09%
9	23	13	10	75	5653	5286.6	34.27	-6.48%
10	26	15	10	25	2690.6	2532.8	104.81	-5.86%
11	26	15	10	50	4387.4	4164.8	82.30	-5.07%
12	26	15	10	75	7427.6	7063.4	85.79	-4.90%
13	29	17	10	25	3443.2	3270	236.94	-5.03%
14	29	17	10	50	5716.6	5430.4	251.17	-5.01%
15	29	17	10	75	9451.8	8993.2	196.66	-4.85%
AVG								**-5.65%**

2 and 7) the gap is greater than 5% while on average it is around 4%. This difference becomes more significant on the other three classes of graphs: for taurus graph the average gap is around 5.5%, for the squared grid graphs the average gap is 5.66% and on the not squared grid graphs it is 7.59% (and, except for instance 5, it is always greater than 5%). The CPU time of ITS on these four classes of graphs is negligible being always less than half of a second. Note that, since in these graphs several vertices have the same degree, the selection criterion applied by MGA is essentially led by the weight of the vertices and this probably causes its poor results.

On large instances, there is a sensible reduction of the gap between ITS and MGA for taurus, squared grid and not squared grid graphs, while this gap increases on hypercube graphs. In detail, on the hypercube, the gap is greater than 3% for three instances (instances 1, 4 and 7) with an average value of 4.47%. The computational time of ITS

on this class of graphs is, in the worst case, slightly more that 1 minute. On taurus graphs the average gap is equal to 4.15% and on two instances (1 and 4) it is greater than 5%. ITS computational time increases to 5 minutes in the worst case. For half of the squared grid instances, the gap is greater than 5% while the average gap is equal to 5.14%. These graphs ended to be more expensive for ITS in terms of computational time. Finally, as already observed for small instances, the not squared grid graphs are the hardest instances for MGA. Indeed, only in 3 cases (instances 5, 12 and 15) the gap is less than 5% while the average gap is equal to 5.65%.

6 Conclusions

We addressed a well known NP-complete problem in the literature (the Weighted Feedback Vertex Set Problem) with application in several areas of computer science such as circuit testing, deadlock resolution, placement of converters in optical networks, combinatorial cut design. In this paper we presented a polynomial time exact algorithm (the DP_{multi} algorithm) to solve the problem on a special class of graphs, namely the k-diamond graphs. In addition, we proposed an Iterative Tabu Search algorithm considering two different neighborhood structures one of which is based on the k-diamond graphs where the DP_{multi} algorithm was hugely used for a better exploration. We carried out an extensive experimentation to show the effectiveness of our approach when compared with the well known 2-approximation algorithm MGA. Our approach shows a very good trade-off between solution quality and computational time: our ITS solves the problem in less than 1 second for instances up to 100 vertices with an improvement of the quality of the solution when compared to those returned by MGA. This makes ITS suitable to be embedded on an exact approach, that is object of our future research.

References

1. Bafna, V., Berman, P., Fujito, T.: A 2-Approximation Algorithm for the Undirected Feedback Vertex Set Problem. SIAM J. Discrete Math. 12(3), 289–297 (1999)
2. Bar-Yehuda, R., Geiger, D., Naor, J., Roth, R.M.: Approximation Algorithms for the Feedback Vertex Set Problem with Applications to Constraint Satisfaction and Bayesian Inference. SIAM J. Comput. 27(4), 942–959 (1998)
3. Becker, A., Geiger, D.: Approximation Algorithms for the Loop Cutset Problem. In: Proceedings of the 10th Conference on Uncertainty in Articial Intelligence, pp. 60–68 (1994)
4. Brunetta, L., Maffioli, F., Trubian, M.: Solving The Feedback Vertex Set Problem On Undirected Graphs. Discrete Applied Mathematics 101, 37–51 (2000)
5. Carrabs, F., Cerulli, R., Gentili, M., Parlato, G.: A Linear Time Algorithm for the Minimum Weighted Feedback Vertex Set on Diamonds. Information Processing Letters 94, 29–35 (2005)
6. Chang, M.S., Liang, Y.D.: Minimum feedback vertex sets in cocomparability graphs and convex bipartite graphs. Acta Informatica 34, 337–346 (1997)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge (2001)
8. Even, G., Naor, J., Schieber, B., Zosin, L.: Approximating Minimum Subset Feedback Sets in Undirected Graphs with Applications. SIAM J. Discrete Math. 13(2), 255–267 (2000)

9. Fomin, F.V., Gaspers, S., Pyatkin, A.V.: Finding a minimum feedback vertex set in time $\mathcal{O}(1.7548^n)$. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 184–191. Springer, Heidelberg (2006)
10. Glover, F.: Tabu Search. ORSA Journal on Computing 1, 190–206 (1989)
11. Glover, F., Laguna, M.: Tabu search. Kluwer, Dordrecht (1997)
12. Glover, F.: Tabu Search: A Tutorial. Interfaces 20, 74–94 (1990)
13. Kleinberg, J., Kumar, A.: Wavelength Conversion in Optical Networks. Journal of Algorithms, 566–575 (1999)
14. Liang, Y.D.: On the feedback vertex set problem in permutation graphs. Information Processing Letters 52, 123–129 (1994)
15. Lu, C.L., Tang, C.Y.: A Linear-Time Algorithm for the Weighted Feedback Vertex Problem on Interval Graphs. Information Processing Letters 61, 107–111 (1997)
16. Misevicius, A., Lenkevicius, A., Rubliauskas, D.: Iterated tabu search: an improvement to standard tabu search. Information Technology and Control 35(3), 187–197 (2006)
17. Razgon, I.: Exact computation of maximum induced forest. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 160–171. Springer, Heidelberg (2006)
18. Vazirani, V.V.: Approximation Algorithms. Springer, Heidelberg (2001)