Acta Cybernetica, Vol. 10, Nr. 4, Szeged, 1992

Computing Maximum Valued Regions

G. J. Woeginger*†

Abstract

We consider the problem of finding optimum connected configurations in the plane and in undirected graphs. First, we show that a special case concerning rectilinear grids in the plane and arising in oil business is NP-complete, and we present a fast approximation algorithm for it. Secondly, we identify a number of polynomial time solvable special cases for the corresponding problem in graphs. The special cases include trees, interval graphs, cographs and split graphs.

1 Introduction

Problem statement and applications. In this paper, we deal with the MAXIMUM VALUED REGION problem (MVR, for short) which is defined as follows. We are given a subdivision of a rectangle into equisized squares. Every single square has some (known) positive value. The problem is to find for a given number k a connected subregion of the rectangle that consists of exactly k squares and that has the maximum overall value under these conditions.

Practical applications of MVR arise e.g. in the context of oil business, cf. Hamacher, Joernsten and Maffioli [6]. Suppose a company is searching for oil at many places of some large area and assigns *values* to the pieces of land according to the results of these trial prospects. The places form some regular (rectilinear) pattern as described above. Afterwards, the company will buy the 'best' k landpieces; assuming unit prices for the land we exactly arrive at MVR.

A related graph problem. The corresponding problem in vertex-valued graphs is to find a connected subgraph on k vertices with maximum overall value. We call this graph problem the *Maximum Valued Subtree* problem, MVS for short. Problem MVS is known to be NP-complete for arbitrary graphs (see [6]). It is easy to see that MVS restricted to *gridgraphs* becomes MVR.

Known results. Hamacher et al. [6] introduced the problem MVS and proved it to be NP-complete for arbitrary graphs. They also developped a branch-and-bound scheme for MVS, and gave an integer program formulation. As a main open problem they asked whether the restriction of MVS to gridgraphs can be solved in polynomial time. Maffioli [8] derived a polynomial time algorithm for solving MVS in trees.

*TU Graz, Institut für Theoretische Informatik, Klosterwiesgasse 32/II, A-8010 Graz, Austria. Electronic mail: gwoegi@igi.tu-graz.ac.at

†This research was supported by the Christian Doppler Laboratorium für Diskrete Optimierung.

Our results. We prove that MVR (and hence the restriction of MVS to gridgraphs) is NP-complete and we give a polynomial time approximation algorithm with worst case guarantee $O(\sqrt{k})$ (i.e. the approximation algorithm always outputs a solution with value at least the optimum value divided by $c\sqrt{k}$).

For the graph problem MVS, we will identify several polynomial time solvable subcases, e.g. MVS in trees, interval graphs and cographs. It turns out that MVS and the famous STEINER TREE problem are closely related in the following sense: The investigated restrictions to the various 'famous' graph classes (as described in Johnson [7]) are either NP-complete for *both* problems or polynomial time solvable for *both* problems.

Organization of the paper. In Section 2, we give the NP-completeness proof for MVR, and in Section 3 our approximation algorithm is described and analyzed. Section 4 deals with treelike graph classes for which MVS is polynomial time solvable by a dynamic programming approach. Section 5 summarizes some other results on special graph classes (interval graphs, cographs and split graphs). Section 6 contains the discussion.

2 NP-completeness of the Region Problem

To give a precise presentation of the problem and our method, we will need the following definitions. A *gridpoint* in the Euclidean plane is a point with both coordinates integer. Two gridpoints are called *adjacent* iff they are at distance one from each other. This adjacency relation induces an infinite graph on the gridpoints. A *region* is a set of gridpoints that is connected in this infinite graph.

MAXIMUM VALUED REGION PROBLEM (MVR)
Input. A rectangular region $R = [1, .., n] \times [1, .., n]$; with sidelength n; a value function $c : R \rightarrow \mathbb{Z}^+$; a positive integer k; an integer bound C.
Question. Does there exist a region $R' \subseteq R$ of exactly k points with total value $c(R') \geq C$?

We will show that the NP-complete *planar Steiner Tree problem* (cf. Garey and Johnson [3]) is polynomial time reducible to our problem MVR.

STEINER TREE IN PLANAR GRAPHS (PST)
Input. A planar graph $G = (V, E)$; a weight $w : E \rightarrow \mathbb{Z}$; a subset $X \subseteq V$; an integer bound $W \in \mathbb{Z}$.
Question. Does there exist a Steiner Tree $T = (V_T, E_T)$ of G for X (i.e. does there exist a subtree T of G with $X \subseteq V_S$) such that $w(E_T) \leq W$?

We start with an instance of PST and construct from this an instance of MVR that is solvable if and only if PST is solvable. To simplify the presentation, we will also use negative values for points in problem MVR. Since exactly k points have to be chosen, adding a large positive constant to all values yields an equivalent problem with positive values.

In a first step, we compute a *rectilinear planar layout* of the graph G. Such a layout maps the vertices of G to (pairwise disjoint) horizontal line segments and maps the edges of G to (pairwise disjoint) vertical line segments, with all endpoints

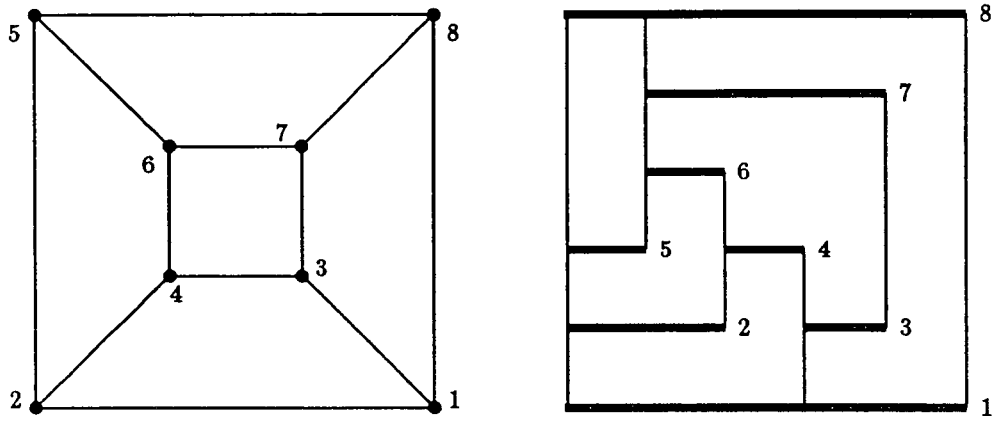

Figure 1: A planar graph and its rectilinear planar layout.

of segments at positive integer coordinates. Two horizontal vertex-segments are connected by a vertical edge-segment, if and only if the corresponding vertices are adjacent in the graph. Figure 1 shows a drawing of a planar graph together with its rectilinear planar layout.

Rosenstiehl and Tarjan [9] show how to compute a rectilinear planar layout for planar graphs with n vertices in $O(n)$ time. The height of their layout is at most $|V|$, and the width is at most $2|V|-4$. Most important, one can choose an arbitrary vertex to become the bottom horizontal vertex-segment of the layout. We choose some vertex x in X to become the bottom segment.

In the second step, we stretch the rectilinear planar layout in horizontal and vertical directions by a factor of two, i.e. we multiply the coordinates of all endpoints by 2. This ensures that points on distinct segments are at distance at least two, unless they correspond to a vertex-edge incidence.

In the third and last step, we finally transform the layout into a weighted region for problem MVR. We distinguish five types of gridpoints: vertex-points, edge-points, link-points, dummy-points and fill-points. The vertex-points, edge-points and link-points together cover exactly all gridpoints on the line segments of the stretched rectilinear planar layout.

- For each vertex v in X (the set that has to be spanned by the Steiner Tree), we choose an arbitrary gridpoint on the horizontal line segment corresponding to v and make it a *vertex-point* $G(v)$. The value c of every point $G(v)$ is set to $M := \sum_{e \in E} w(e) + 1$.

- For each edge e in E, we choose an arbitrary gridpoint on the vertical line segment corresponding to e and make it an *edge-point* $G(e)$. The value $c(G(e))$ equals the weight $-w(e)$.

- All points lying on line segments of the stretched rectilinear planar layout that are neither vertex-points nor edge-points become link-points and receive a value of 0.

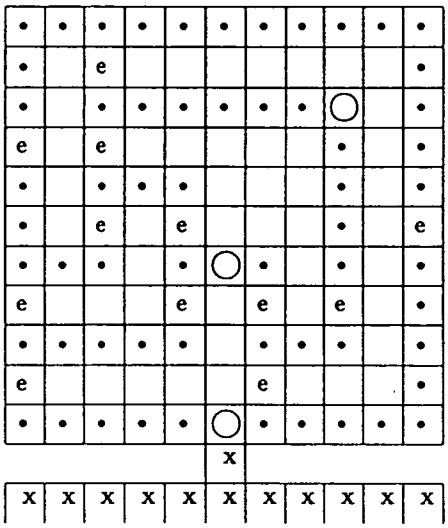

Figure 2: The upper part of the constructed region instance.

Now let n_v, n_e and n_ℓ denote the number of vertex-, edge- and link-points, respectively. Set $k = n_v + n_e + n_\ell$ and observe that $k \in O(|V|^2)$.

- We create a connected region of k points *just below* the layout, separated from the layout by a single row of unused gridpoints (in other words, the topmost points in this new region are at distance two from the lower border of the layout). Moreover, we connect this region by a single gridpoint in this unused row to the vertex-point corresponding to vertex $x \in X$. The gridpoints in this new region and this single gridpoint constitute the set of dummy-points. All of them have value 0.

- Finally, we enclose the vertex-, edge-, link- and dummy-points by a rectangle (obviously, the sidelength of this rectangle is polynomial in $|V|$). All points in this rectangle to which we did not assign a value till this moment are the fill points; they have value $-|X| \cdot M - 1$.

An illustration for this construction is given in Figure 2, where the graph depicted in Figure 1 is transformed in its corresponding region. The vertices in X are 1, 4 and 7 and the corresponding vertex-points in the region are marked by a "○". The edge-points are marked by an "e", and the link-points by a "•". Empty space corresponds to fill points. For reasons of readability and space, we did not show all dummy-points (marked by an "x") that lie below the bottom "○".

We claim that the constructed instance of MVR has a solution with value at least $C = |X| \cdot M - W$ if and only if there exists a Steiner Tree of G for X with weight at most W. W.l.o.g. we assume $W \leq \sum_{e \in E} w(e) = M - 1$, as otherwise the PST is trivially solvable.

(only if) Assume there exists a region R' with value at least $|X| \cdot M - W$. Since the only points with positive value are exactly the $|X|$ vertex-points with value M

and since $W \leq M - 1$ holds, R' must contain all the vertex-points. Since all fill-points have value $-|X| \cdot M - 1$, no fill-point appears in R'.

Obviously, the dummy-points (outside of the layout!) cannot help in connecting the vertex-points. Hence, the connections must result from the edge- and from the link-points. Vertex-points on two distinct horizontal segments can be connected to each other only via points on the vertical edge-segments. Using link-points (with value 0) is no problem, but in the middle of each edge there sits an edge-point, substracting $w(e)$ from the value of region R'. In order to connect all vertex-points while substracting at most W from the total value $|X| \cdot M$ of all vertex-points, we must find a connected configuration that has edge-weight at most W and that contains X. This exactly yields the claimed Steiner Tree.

(if) Now assume we are given a Steiner Tree with edge-weight at most W. We start with putting into the region R' all gridpoints on line segments corresponding to edges and vertices used in the Steiner Tree. Hence, it contains all vertex-points (with total value $|X| \cdot M$), some edge-points (with total value at most W) and a number of link-points with zero value. By the definition of k, the overall number of these points is at most k. To get a region with exactly k points, we add an appropriate number of dummy-points to R' such that R' remains connected. As all dummy points have zero value, the total value of the constructed R' is at least $|X| \cdot M - W$.

Summarizing, we have proven the following theorem.

Theorem 2.1 *Problem* MVR *is NP-complete.* □

3 A Heuristic for MVR

In this section we analyze the following fast and simple heuristic for the region problem MVR (for convenience we assume throughout this section that $k = \alpha^2$ is a square number).

> Take the highest value axes-parallel quadratic region Q^* with sidelength $\sqrt{k}$.

Clearly, this quadratic region can be found in $O(kn^2)$ time and a slightly more sophisticated implementation runs in $O(\sqrt{k}n^2)$ time. Our main interest is to determine the worst case quality of Q^* compared to the optimum region R^*.

Let $k = \alpha^2$. Consider a *staircase*, consisting of $\alpha/2$ vertical and $\alpha/2$ horizontal line segments where each line segment contains exactly α gridpoints. All gridpoints on the staircase receive value one, all other gridpoints receive value zero. Then the optimum region R^* has value k, whereas no square with sidelength α can cover more than 2α gridpoints on the staircase. Hence, for this configuration our heuristic is a factor of $\Omega(\sqrt{k})$ away from the optimum. It is easy to see that the staircase is also a bad configuration for the more general heuristic where we do not only consider axes-parallel squares but also arbitrary (not necessarily axes-parallel) rectangles.

Surprisingly, $O(\sqrt{k})$ is also the worst that can happen as will be shown in the remaining part of this section. We cover the optimum region R^* by an orthogonal grid with gridlength α and vertices that are integer points shifted by the vector $(1/2, 1/2)$. This grid is called the A-grid and it partitions the plane into A-cells.

Lemma 3.1 *At most 10α of the A-cells contain a point of R^*.*

Proof. Denote by V_R the set of all A-cells that contain at least one point of R^*. Two A-cells A_1 and A_2 in V_R are called adjacent iff A_i contains a gridpoint $p_i \in R^*$, $1 \leq i \leq 2$, such that p_1 and p_2 are at distance one. Let $T = (V_R, E_R)$ be an arbitrary spanning tree of the graph induced by this adjacency relation. We root T at an arbitrary leaf of T; this defines fathers and sons, and by the choice of the root no vertex has more than three sons.

Then we repeat the following two steps over and over again until T contains less than 10 vertices. (The tree is processed in a bottom-up fashion from the leaves towards the root. Step 1 removes subtrees that are paths on four vertices, and Step 2 removes 'branching' subtrees).

(Step 1). Assume, T contains a vertex v whose only descendants form a path on three vertices v_1, v_2 and v_3 (such a subgraph is called a *type-1 subgraph*). Then the corresponding four A-cells contain at least α points of R^*. The only interesting case occurs when the A-cells corresponding to v, v_1, v_2 and v_3 form a 2×2 configuration. W.l.o.g. let v be the lower left A-cell in this configuration, and let v be connected to its father via its left side. Then this left side must be linked to the diametric A-cell v_2. A-cell v_2 is at distance α from the left side of v, and there are at least α vertices necessary to link them.

We iteratively remove all type-1 subgraphs from T.

(Step 2). Next, we consider some vertex w of degree at least two with no descandants of degree at least two (w must exist, unless Step 1 deleted all but three vertices; in this case we terminate).

Vertex w has at most three sons, and since T does not contain any type-1 configuration any more, each of its sons has at most two descendants. Summarizing, the maximal subgraph of T rooted at w contains at most 10 A-cells. On the other hand, the A-cell corresponding to w has outside connections on at least three of its four sides (one to its father, and at least two to its sons). Two of these outside connections must lie on opposite sides of the cell, and in order to link them to each other, the A-cell must contain at least α points of R^*.

We remove the maximal subgraph of T rooted at w from T and return to Step 1.

To finish the proof of the lemma, we observe that each removal operation in Step 1 and 2 removes (i) at most 10 A-cells and (ii) at least α points of R^*. Because of (ii), at most k/α removal operations are performed and because of (i), T contains at most $10k/\alpha = 10\alpha$ A-cells. □

Theorem 3.2 *There exist constants $c_1 > c_2 > 0$ such that the heuristic detects for all instances a region Q^* whose value is at least the value of R^* divided by $c_1\sqrt{k}$, and there exist instances for which the value of Q^* is at most the value of R^* divided by $c_2\sqrt{k}$.*

Proof. We prove the statement for $c_1 = 10$ and $c_2 = 1/2$.

Applying Lemma 3.1 and an averaging argument, we see that there exists an A-cell A_0 such that the points in $R^* \cap A_0$ have overall value at least the optimum value divided by $10\sqrt{k}$. Since all other gridpoints in A_0 have nonnegative value, the value of Q^* is at least the value of R^* divided by $10\sqrt{k}$.

The lower bound follows from the staircase configuration described at the beginning of this section. □

Remark. The factor 10 in the statement of Lemma 3.1 is not the smallest possible. A more elaborate argument decreases the factor down to 4. For our purposes, any constant factor suffices.

4 Results for Trees

In this section, we consider the following problem corresponding to MVR in graphs.

> MAXIMUM VALUED SUBTREE PROBLEM (MVS)
> **Input.** A graph $G = (V, E)$; a value function $c : V \to \mathbb{N}$; a positive integer k and an integer bound C.
> **Problem.** Does there exist a k-subtree T of G with total value at least C ?

For general graphs, problem MVS is NP-complete since MVR is a special case of MVS. We will present a polynomial time result for trees. As usual, we assume that the input graph G is given by its *adjacency list*, i.e. for each vertex $v \in V$ we have a list of its neighbors in G. The number of vertices in G will always be denoted by n. A tree on k vertices will also be called a *k-tree* or a *k-subtree*.

Subsection 4.1 analyzes a related matrix problem and Subsection 4.2 gives an $O(nk^2)$ algorithm for MVS on trees, The algorithms are based on Dynamic Programming approaches. Maffioli [8] derived another (more complicated) polynomial time algorithm for MVS in trees with the same running time as our solution.

4.1 A Matrix Problem

In this subsection we will analyze a matrix problem that is closely related to the MVS-problem. Let M be a matrix with nonnegative integer entries that consists of d rows and k columns. We define that the entry in the i-th row and j-th column has *value* $M_{i,j}$ and *weight* j. For $0 \leq j \leq k$, we denote by $\text{MAXVAL}(M, j)$ the maximum value for which there exists a subset S_j of the entries in M fulfilling the following conditions.

- S_j contains at most one entry from every single row in M,
- the overall weight of S_j equals j, and
- the overall value of S_j is $\text{MAXVAL}_M(j)$.

Lemma 4.1 *For a $d \times k$ matrix M, all numbers* $\text{MAXVAL}(M, 0), \ldots, \text{MAXVAL}(M, k)$ *can be computed in overall time $O(k^2 d)$.*

Proof. We apply the *Divide and Conquer* paradigm to solve the problem by the following recursive procedure.

(1) We divide matrix M by a horizontal line into an upper and into a lower submatrix of equal size. We call these two submatrices U and L.

(2) We recursively calculate all numbers $\text{MAXVAL}(U, *)$ and $\text{MAXVAL}(L, *)$.

(3) We determine MAXVAL$(M, *)$ from MAXVAL$(U, *)$ and MAXVAL$(L, *)$ according to the formula

$$\text{MAXVAL}(M, j) = \max_{0 \le i \le j} \text{MAXVAL}(U, j-i) + \text{MAXVAL}(L, i)$$

for all j, $0 \le j \le k$.

The correctness of the algorithm is obvious. Since the Divide-Step (1) takes only constant time and the Merge-Step (3) is done with at most k^2 operations, the time complexity $T(d, k)$ fulfills the inequality

$$T(d, k) \le 2T(d/2, k) + k^2.$$

Standard calculations yield $T(d, k) \le dT(1, k) + dk^2$, and consequently the time complexity is at most $O(dk^2)$. □

4.2 Trees

Now let the tree $G = (V, E)$ constitute an instance of MVS with $n = |V| = |E| + 1$. We root G at an arbitrary vertex r. This assigns to every vertex v (with exception of the root r) a unique father $f(v)$. With every vertex $v \in V$, we associate the maximal subtree rooted at v. Let $v_1, v_2, \ldots, v_n$ be an enumeration of the vertices in V such that each v comes before its father $f(v)$. Such an enumeration can easily be found in $O(n)$ time.

We introduce a two-dimensional integer array AR$[i, j]$ with $n(k+1)$ entries. The rows are indexed by the vertices v_i in the above enumeration, and the columns are indexed by the numbers from 0 to k.

> The meaning of "AR$[i, j] = w$" is that "the maximum value j-subtree of $T(v_i)$ that *also contains its root* v_i, has value w".

Lemma 4.2 *The values of all entries in* AR *can be calculated in* $O(k^2 n)$ *time.*

Proof. We consecutively calculate all rows of AR, starting with the row corresponding to v_1 and ending with the row corresponding to $v_n = r$.

If $T(v_i)$ consists of the single vertex v_i, we set AR$[i, 0] = 0$, AR$[i, 1] = c(v_i)$ and all other entries in AR$[i, *]$ to $-\infty$.

If $T(v_i)$ consists of at least two vertices, we consider the sons $v_{m_1}, v_{m_2}, \ldots, v_{m_d}$ of v_i, where $d = \deg(v_i) - 1$. In order to compute AR$[i, j]$, we must find the optimum partitioning of the number $j - 1$ into d nonnegative numbers $j_1, \ldots, j_d$ that maximizes $\sum_{i=1}^{d}$ AR$[v_{m_i}, j_i]$. But this exactly amounts to solving the matrix problem treated in Section 1 on the submatrix M of AR$[*, *]$ generated by the rows corresponding to the vertices $v_{m_1}, v_{m_2}, \ldots, v_{m_d}$. According to Lemma 4.1, this problem can be solved in $O(k^2 d)$ time. Finally, we add to each of the k resulting numbers the value $c(v_i)$ of the root of this subtree.

To get the overall time complexity for computing AR$[*, *]$, we have to sum up the $k^2(\deg(v_i) - 1)$ steps for every v_i with at least one son plus the k steps for every v_i without a son. This is clearly dominated by $k^2 \sum_i \deg(v_i) \in O(k^2 n)$. □

Theorem 4.3 *For trees, the problem* MVS *can be solved in* $O(k^2n)$ *time and* $O(kn)$ *space.*

Proof. Assume that the maximum value k-subtree T is spanned by vertices $W = \{w_1, \ldots, w_k\}$ and let w denote the unique vertex in W whose father is not in W. Then the value of T equals $\text{AR}[w, k]$. Conversely, each entry in $\text{AR}[*, j]$ corresponds to a j-subtree.

Hence, the maximum number in the k-th column of $\text{AR}[*, *]$ gives the value of the maximum value k-subtree. The time complexity follows from the preceding lemma, the space complexity is determined by the size of AR. □

Remark. We only showed how to find the *value* of the maximum value k-subtree. If we also want to find the corresponding *k-subtree*, we have to store for each entry in $\text{AR}[*, *]$ its 'history' consisting of at most $k - 1$ predecessor entries as used in the dynamic program. This increases the time and the space complexity both by a factor of k.

5 Other Graph Families

This section deals with interval graphs, cographs and split graphs. We derive polynomial time results for the former two graph families and an NP-completeness proof for the latter family.

5.1 Interval Graphs

The vertices of an *interval graph* $G = (V, E)$ can be represented by intervals on the real line in such a way that two intervals intersect if and only if the corresponding vertices are adjacent. Most NP-complete graph problems become polynomial time solvable when restricted to interval graphs, cf. [5,7].

W.l.o.g. we may assume that intervals corresponding to distinct vertices have distinct endpoints. To find the MVS of a vertex-valued interval graph, we use the following decomposition of a *connected* interval graph G: The interval with the rightmost right endpoint is called the *head* of G. In general, there will be several intervals covering the left endpoint of the head. Among those intervals we choose that one with leftmost left endpoint, and we call it the *neck* of G; its endpoints are denoted by n_l and n_r. The remaining intervals either belong to the *body* of G (if their right endpoint lies to the left of n_r) or to the *hairs* of G (if their right endpoint lies to the right of n_r). Intuitively speaking, the body intervals are connected to the head via the neck. The hair intervals are directly connected to the head (their left endpoints are to the right of n_l, and their right endpoints are covered by the head).

Now sort the intervals from left to right according to their right endpoint and call the resulting sequence $I_1, \ldots, I_n$. We construct a twodimensional array $\text{AR}[1 \ldots |V|, 1 \ldots k]$ such that the maximum valued subtree with head I_j and consisting of k' vertices $(1 \leq k' \leq k)$ has value $\text{AR}[j, k']$. We compute all values in $\text{AR}[*, *]$, starting with level $\text{AR}[1, *]$ and going up to level $\text{AR}[n, *]$. The initialization steps are trivial, hence we only show how to compute $\text{AR}[j, k']$ for some fixed $j > k'$.

By definition, I_j constitutes the head of the optimum subgraph $G'(j, k')$ we are looking for. There are at most $n-1$ possibilities for the neck of $G'(j, k')$. There are at most $O(k^2)$ pairs (k_1, k_2) with sum $k_1 + k_2 + 2 = k'$, where k_1 denotes the number

of hairs and k_2 denotes the number of body–vertices. For fixed head and neck and for fixed numbers k_1 and k_2, the value of the optimum $G'(j,k')$ can be found in the following way. The body is the maximum valued connected subgraph on k_2+1 vertices and with our neck as new head; its value has already been computed and we find it in constant time. The hairs are the k_1 most precious intervals with left endpoints to the right of the left endpoint of the neck, and with right endpoints covered by the head. We claim that the optimum value for the k_1 hairs can be calculated in constant time with $O(nk)$ preprocessing for every head.

For a fixed head h, we enumerate all intersecting intervals sorted by their left endpoints from left to right in $O(n)$ time (in a preprocessing step, we sort all intervals by their left endpoints; if we deal with a fixed head, we run thru this list and select all intersecting intervals). We run through this enumeration from right to left and always store the k most precious values in a balanced tree: if the current interval has a value larger than the minimum in the tree, we remove the minimum and insert the value of the current interval (in case the tree has less than k vertices, we just insert the new value). Hence, we know in every single step the $1 \leq k' \leq k$ largest values and can compute their sum in $O(k)$ time.

Theorem 5.1 *For interval graphs, the problem* MVS *can be solved in* $O(k^2n^2)$ *time and* $O(kn)$ *space.*

Proof. The approach described above takes $O(k_1k_2n+kn)$ time for each of the n possible heads. Hence, the overall time is in $O(k^2n^2)$. The space requirements are dominated by the space of array AR. □

5.2 Cographs

In this section, we give a polynomial time algorithm for MVS in cographs.

Definition 5.2 *For* $r \geq 2$ *disjoint graphs* $G_i = (V_i, E_i)$ *with* $V_i \cap V_j = \emptyset$ *for* $i \neq j$, *the* union $\bigcup_{i=1}^{r} G_i$ *is defined as the graph* $(\bigcup_{i=1}^{r} V_i, \bigcup_{i=1}^{r} E_i)$. *Their* product $\times_{i=1}^{r} G_i$ *is obtained by first taking the union of the* r *graphs and then adding all edges* (v_i, v_j) *with* $v_i \in V_i$, $v_j \in V_j$ *and* $i \neq j$. □

Definition 5.3 *The class of* cographs *is the smallest set of graphs fulfilling the following rules.*

1. *The graph with one vertex and no edges is a cograph.*
2. *If* G_i, $1 \leq i \leq r$ *are cographs with pairwise disjoint vertex sets, then their union is a cograph.*
3. *If* G_i, $1 \leq i \leq r$ *are cographs with pairwise disjoint vertex sets, then their product is a cograph.* □

To each cograph $G = (V, E)$, we associate a corresponding rooted tree $T = (I, F)$, called the *cotree* of G and reflecting the above definition in the following way. Each non-leaf vertex in the tree is labeled either with $\cup$ (union-vertex) or $\times$ (product-vertex) and has two or more children. If two non-leaf vertices are connected by an edge, then they have different labels. Each vertex $x \in I$ of the cotree corresponds to a cograph $G_x = (V_x, E_x)$, and a leaf corresponds to a single-vertex graph. A union-vertex (product-vertex) corresponds to the union (product) of the cographs associated with the children of the vertex. Finally, the entire cograph is given by the cograph associated with the root $r \in I$ of the cotree. Corneil, Perl and Stewart [2] have shown that one can decide in linear time $O(|V| + |E|)$, whether a graph is a cograph, and build the corresponding cotree.

Theorem 5.4 *For a cograph $G = (V, E)$, the problem* MVS *can be solved in $O(k^2 n)$ time and $O(kn)$ space.*

Proof. We will compute two twodimensional arrays $\text{ARC}[x, k']$ and $\text{ARA}[x, k']$, where the rows correspond to the vertices x of the cotree and where $0 \leq k' \leq k$ holds. Once more, we start the computation of the array values at the leaves and go up to the root. $\text{ARC}[x, k']$ stores the largest possible value of any *connected* subgraph on k' vertices of the cograph G_x associated with x, and $\text{ARA}[x, k']$ stores the corresponding value for *arbitrary* (not necessarily connected) subgraphs.

The initialization is straight forward and we only show how to compute $\text{ARC}[x, *]$ and $\text{ARA}[x, *]$ for a non-leaf vertex x. The computation of $\text{ARA}[x, k']$ is easy: We simply take the k' most valuable vertices in the corresponding cograph. Applying e.g. the matrix algorithm from Subsection 4.1 this can be performed in $O(k^2 n)$ overall time for all vertices in the cotree. The computation of $\text{ARC}[x, k']$ is more envolved; we have to distinguish between union- and product-vertices x. For a union-vertex x, $\text{ARC}[x, k']$ equals the maximum of $\text{ARC}[s, k']$ over all sons s of x in the cotree (as a union operation cannot change connectivity properties of the graph). For a product-vertex x with sons $s_1, \ldots, s_p$, we perform two computations to find $\text{ARC}[x, k']$:

(i) We compute the maximum value of $\text{ARA}[s_1, k_1] + \ldots + \text{ARA}[s_p, k_p]$ over all p-tuples $(k_1, \ldots, k_p)$ with sum k' and at least two non-zero k_i (by applying the matrix algorithm). Since this maximum value results from at least two distinct sons of x, the corresponding graph is connected.

(ii) We compute $\max_i \text{ARC}[s_i, k']$. By the definition of $\text{ARC}[s_i, *]$, the corresponding graph is again connected.

Obviously, the maximum of the two values computed in (i) and (ii) yields $\text{ARC}[x, k']$.

The entry $\text{ARC}[r, k]$ for the root r of the cotree gives the desired value of the MVS. Since the cotree has $O(n)$ vertices, the claimed time and space complexity follows from the discussion in Subsection 4.1. □

5.3 Split Graphs

A graph $G = (V, E)$ is a *split graph*, if there is a partition of its vertices into an independent set I and in a clique C (and arbitrary edges between I and C), see Golumbic [4].

Theorem 5.5 *Problem* MVS *restricted to split graphs remains NP-complete.*

Proof. By reduction from the NP-complete SET-COVERING PROBLEM: Given a set $S = \{1, \ldots, p\}$ and subsets $A_1, \ldots, A_q \subset S$, the SET-COVERING PROBLEM consists in finding r subsets $A_{i_1}, \ldots, A_{i_r}$ with $\cup_{j=1}^{r} A_{i_j} = \{1, \ldots, p\}$. This problem is known to be NP-complete [3].

We construct a split graph on $p + q$ vertices that are labeled by some label in $\{1, \ldots, p, A_1, \ldots, A_q\}$. The vertices $A_1, \ldots, A_q$ form a clique, the vertices $1, \ldots, p$ form an independent set. We introduce an additional edge from ℓ to A_{i_j} iff $\ell \in A_{i_j}$ holds. All vertices $1, \ldots, p$ receive value 1, all other vertices receive value 0. Finally, we set $k = p + r$ and ask whether there exists a subtree with value at least p.

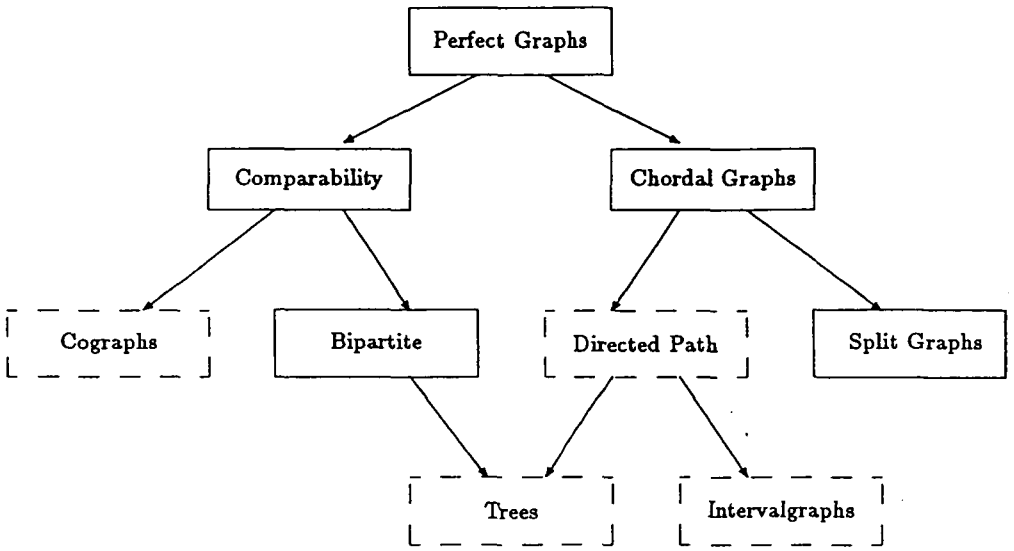

Figure 3: Containment relations for some of the treated graph classes.

In case such a tree exists, it uses all p vertices with value 1 and r vertices belonging to the clique must connect them; this yields the existence of a small set cover. In case a set cover with at most r subsets exists, we choose the corresponding r vertices in the clique and all p vertices not in the clique; clearly, the spanned graph is connected and of value p. □

Remark. We observe that there exists a simple approximation algorithm for MVS in split graphs with (tight) worst case guarantee 2: We simply take the $k/2$ most valuable vertices $v_1, \ldots, v_{k/2}$ and for every i some clique vertex c_i adjacent to v_i. Obviously, the resulting spanned graph is connected and its value is at least half of the optimum possible value.

6 Discussion

In this paper, we investigated the computational complexity of two closely related combinatorial problems, called MVR and MVS. The geometric problem MVR was shown to be NP-complete, and a polynomial time approximation algorithm was derived. The graph problem MVS is NP-complete for arbitrary graphs, but it can be solved efficiently on many well known special graph classes by applying Dynamic Programming techniques.

Figure 3 summarizes some of our results for MVS. Directed arcs represent containment of the lower graph class in the upper graph class. For classes with

a solid frame, MVS is NP-complete, and for classes with a dashed frame, MVS is polynomial time solvable (for exact definitions of all graph classes cf. Johnson [7]). Cographs, trees, interval graphs and split graphs were treated in this paper. The NP-completeness result for split graphs implies NP-completeness for chordal graphs and for perfect graphs. NP-completeness of MVS for bipartite graphs can be seen easily (by subdividing the edges of an arbitrary graph, assigning value zero to the new vertices and replacing k by $2k-1$), and this also yields the NP-completeness for comparability graphs. Finally, a polynomial time algorithm for directed path graphs can be derived by standard Dynamic Programming techniques (the method is analogous to that we applied to trees and cographs, and left to the ambitious reader as an exercise).

Moreover, for planar graphs we have proven the following results. MVS on grid-graphs (and consequently on arbitrary planar graphs) is NP-complete. However, the restriction to outerplanar and series-parallel graphs (these two classes are subsets of the partial 2-trees) can be solved in polynomial time. Bodlaender [1] derived an $O(k^2n)$ algorithm that solves MVS in partial K-trees, where K is not part of the input.

The most intriguing open problem is to construct polynomial time approximation algorithms for the geometric problem MVR with *constant* worst case guarantee (or prove that such algorithms do not exist).

References

[1] H.L.Bodlaender, private communication, 1992.

[2] D.G.Corneil, Y.Perl and L.K.Stewart, A linear recognition algorithm for cographs, *SIAM J. Comput.* **4**, 1985, 926–934.

[3] M.R.Garey and D.S.Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[4] M.C.Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[5] U.I.Gupta, D.T.Lee and J.Y.-T.Leung, Efficient algorithms for interval graphs and circular–arc graphs, *Networks* **12**, 1982, 459–467.

[6] H.Hamacher, K.Joernsten and F.Maffioli, Weighted k-cardinality trees, Report 91.023, Dipartimento di Elettronica, Politecnico di Milano, 1991.

[7] D.S.Johnson, The NP-Completeness Column: an Ongoing Guide, *J. Algorithms* **6**, 1985, 434–451.

[8] F.Maffioli, Finding a best subtree of a tree, Report 91.041, Dipartimento di Elettronica, Politecnico di Milano, 1991.

[9] P.Rosenstiehl and R.E.Tarjan, Rectilinear planar layouts of planar graphs and bipolar orientations, *Discr. Comp. Geometry* **1**, 343–353, 1986.

Received November 8, 1992