

SWAPS, DIVERSIFICATION, AND THE COMBINATORICS OF PIVOTING FOR THE MAXIMUM WEIGHT CLIQUE

MARCO LOCATELLI*, IMMANUEL M. BOMZE†, AND MARCELLO PELILLO‡

Abstract. Recently a powerful pivoting-based heuristic (PBH) has been introduced for attacking the maximum weight clique problem, based on the linear complementarity formulation of an equivalent (standard) quadratic program. In this paper, we show that PBH is equivalent to a combinatorial greedy heuristic. This interpretation allows us to modify PBH, by introducing swap and diversification strategies, particularly focussing on the unweighted case. Results over standard DIMACS graphs show that the resulting heuristic compares well with the most powerful algorithms available in the literature: results are considerably improved while computation time is kept low.

Key words. maximum clique, linear complementarity, pivoting methods, greedy heuristics, combinatorial optimization

AMS subject classifications. 90C27, 90C33, 90C49, 90C59, 05C69

1. Introduction. Given an undirected graph, the maximum clique problem (MCP) consists of finding a subset of pairwise adjacent vertices (i.e., a *clique*) having largest cardinality. The problem is known to be NP -hard for arbitrary graphs and so is the problem of approximating it within a constant factor. An important generalization of the MCP arises when positive weights are associated to the vertices of the graph. In this case the problem is known as the maximum weight clique problem (MWCP) and consists of finding a clique in the graph which has largest total weight (note that the maximum weight clique does not necessarily have largest cardinality in general, but that MCP coincides with MWCP in the special case when the weights assigned to the vertices are all equal). The MWCP has important applications in such fields as computer vision, pattern recognition and robotics, where weighted graphs are employed as a convenient means of representing high-level pictorial information. We refer to [3] for a recent review concerning algorithms, applications and complexity issues of this important problem.

Motivated by a recent quadratic programming formulation, which generalizes an earlier remarkable result by Motzkin and Straus, in a recent paper [11] a new pivoting-based heuristic (PBH) for the MWCP has been proposed which is based on the corresponding linear complementarity problem (LCP). The algorithm is essentially a variant of Lemke's classical algorithm that incorporates an effective look-ahead pivot rule, and it proved to be among the most powerful MWCP heuristics available in the literature.

The objective of this paper is twofold. First, after briefly describing PBH we provide some theoretical results which prove a few conjectures that have been put forward in [11] based on empirical observations. Interestingly, these results allow for an interesting combinatorial interpretation of the algorithm: PBH is essentially equivalent to a greedy combinatorial heuristic. Motivated by these results, we then focus on the unweighted case and modify the combinatorial counterpart of PBH to

*Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, I-10149 Torino, Italy (locatelli@di.unito.it)

†Institut für Statistik und Decision Support Systems, Universität Wien, Universitätsstraße 5, A-1010 Wien, Austria (immanuel.bomze@univie.ac.at).

‡Dipartimento di Informatica, Università Ca' Foscari di Venezia, Via Torino 155, I-30172 Venezia Mestre, Italy (pelillo@dsi.unive.it).

improve the quality of the results obtained, at the cost of slightly increasing the computation time. Basically, the major modification incorporates vertex swaps during the clique-construction process, but we also equip the algorithm with a simple and effective diversification strategy. The resulting algorithms have been tested extensively on various instances of DIMACS benchmark graphs and the results obtained clearly show the effectiveness of the approach.

2. The pivoting-based heuristic and its combinatorial interpretation.

2.1. Notations and definitions. Let $G = (V, E, w)$ be an arbitrary undirected and weighted graph, where $V = \{1, \dots, n\}$ is the *vertex set*, $\binom{V}{2}$ denotes the system of all two-element subsets of V , $E \subseteq \binom{V}{2}$ is the *edge set* and $w \in \mathbb{R}^n$ is the *weight vector*, the i -th component of which corresponds to the weight assigned to vertex i . It is assumed that $w_i > 0$ for all $i \in V$. Two distinct vertices $i, j \in V$ are said to be *adjacent* if they are connected by an edge, i.e., if $\{i, j\} \in E$. The *neighborhood* of a vertex i will be indicated with $N_i = \{j \in V : \{i, j\} \in E\}$, and its degree will be $d(i) = |N_i|$, the cardinality of N_i . Given a subset of vertices S , the weight assigned to S will be denoted by

$$W(S) = \sum_{i \in S} w_i .$$

As usual, the sum over the empty index set is defined to be zero.

A *clique* is a subset of V in which all vertices are pairwise adjacent. A clique S is called *maximal* if no strict superset of S is a clique. A maximal weight clique S is a clique which is not contained in any other clique having weight larger than $W(S)$. Since we are assuming that all weights are positive, it is clear that the concepts of maximal and maximal weight clique coincide, hence we shall not make any distinction throughout the paper. A maximum cardinality clique (or, simply, a *maximum clique*) is a clique whose cardinality is the largest possible. The maximum size of a clique in G is called the *clique number* (of G) and is denoted by $\omega(G)$. A *maximum weight clique* is a clique having largest total weight, and the maximum weight clique problem (MWCP) is the problem of finding such a clique. The *weighted clique number* of G , denoted by $\omega(G, w)$, is the maximum weight of a clique in G .

2.2. The PBH heuristic. Given a weighted graph $G = (V, E, w)$, consider the following standard quadratic program (StQP):

$$(1) \quad \begin{array}{ll} \text{minimize} & x^T Q_G x \\ \text{subject to} & x \in \Delta \end{array}$$

where the matrix $Q_G = (q_{ij})_{(i,j) \in V \times V}$ is defined as:

$$(2) \quad q_{ij} = \begin{cases} \frac{1}{2w_i} , & \text{if } i = j , \\ 0 , & \text{if } \{i, j\} \in E , \\ \frac{1}{2w_i} + \frac{1}{2w_j} , & \text{otherwise ,} \end{cases}$$

and Δ denotes the standard simplex in the n -dimensional Euclidean space \mathbb{R}^n :

$$\Delta = \left\{ x \in \mathbb{R}^n : \sum_{i \in V} x_i = 1 \text{ and } x_i \geq 0 \text{ for all } i \in V \right\} .$$

Given a subset of vertices $S \subseteq V$, the *weighted characteristic vector* of S , denoted by $x^{S,w}$, is the vector in Δ whose coordinates are given by:

$$x_i^{S,w} = \begin{cases} \frac{w_i}{W(S)} & \text{if } i \in S, \\ 0 & \text{otherwise.} \end{cases}$$

The PBH heuristic is based upon the following result [4].

THEOREM 2.1. *Let $G = (V, E, w)$ be an arbitrary graph with positive weight vector $w \in \mathbb{R}^n$. Then the following assertions hold.*

- *A vector $x \in \Delta$ is a local solution of (1) if and only if $x = x^{S,w}$, where S is a maximal clique of G .*
- *A vector $x \in \Delta$ is a global solution of (1) if and only if $x = x^{S,w}$, where S is a maximum weight clique of G .*

Moreover, all solutions of (1) are strict and are characteristic vectors of maximal cliques of G .

It is well known that stationary points of quadratic optimization problems with linear constraints can be characterized as the solutions of a linear complementarity problem (LCP), a class of inequality systems for which a rich theory and a large number of algorithms have been developed [8]. Hence, once that the MWCP is formulated in terms of an StQP, the use of LCP algorithms naturally suggests itself, and this is precisely the main idea proposed in [11].

Specifically, computing the stationary points of (1) can be done by solving the LCP (q_G, M_G) , which is the problem of finding a vector x satisfying the system

$$(3) \quad y = q_G + M_G x \geq 0, \quad x \geq 0, \quad x^T y = 0$$

where

$$(4) \quad q_G = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ -1 \\ 1 \end{bmatrix} \quad M_G = \begin{bmatrix} Q_G & -e & e \\ e^T & 0 & 0 \\ -e^T & 0 & 0 \end{bmatrix}$$

with Q_G as in (2) and e is the vector with all coordinates equal to 1. With the above definitions, it is well known that if $x \in \mathbb{R}^n$ solves (3) – we say then that $z = [x^T, y^T]^T$ is a complementary solution of LCP (q_G, M_G) – then x is a stationary point of (1). Note that Q_G is strictly \mathbb{R}_+^n -copositive, hence so is M_G and this is sufficient to ensure that LCP (q_G, M_G) , or (3), always has a solution [8].

Among the many LCP methods presented in the literature, pivoting procedures are widely used and within this class Lemke's method is certainly the best known largely for its ability to provide a solution for several matrix classes. Given the generic LCP (q, M) , it deals with the augmented problem (q, d, M) defined by

$$(5) \quad y = q + [M, d] \begin{bmatrix} x \\ \theta \end{bmatrix} \geq 0, \quad \theta \geq 0, \quad x \geq 0, \quad x^T y = 0.$$

The vector d is called the *covering vector*, and must satisfy $d_i > 0$ whenever $q_i < 0$. A solution of (q, d, M) with $\theta = 0$ promptly yields a solution for LCP (q, M) and Lemke's method intends to compute precisely such a solution. We refer to [8] for a detailed description of Lemke's algorithm. In [11], $d = e$ was chosen, and our problem does not expose peculiarities that would justify a deviation from this common practice.

Unfortunately, like other pivoting schemes, the convergence of Lemke's algorithm is guaranteed only for non-degenerate problems, and ours is indeed degenerate. In [11], standard degeneracy resolution strategies were tested over a number of benchmark graphs, but the computational results obtained were rather discouraging. The inherent degeneracy of the problem, however, is beneficial as it leaves freedom in choosing the blocking variable, and this property is exploited to develop a variant of Lemke's algorithm: the look-ahead pivot rule which we will now shortly describe, for the readers' convenience.

As customary, we will use a superscript for the problem data and, to simplify notation, subscripts indicating the dependence on graph G will be omitted. Hence, q^ν and M^ν will identify the situation after ν pivots and Q^ν will indicate the leading principal $n \times n$ submatrix of M^ν . Consistently, y^ν and x^ν will indicate the vectors of basic and non-basic variables, respectively, each made up of a combination of the original x_i and y_i variables. The notation $\langle x_i^\nu, y_j^\nu \rangle$ will be used to indicate pivoting transformations. The index set of the basic variables that satisfy the min-ratio test at iteration ν will be denoted with Ω^ν , i.e.

$$\Omega^\nu = \arg \min_i \left\{ \frac{-q_i^\nu}{m_{is}^\nu} : m_{is}^\nu < 0 \right\}$$

where s is the index of the driving column. Also, in the sequel the auxiliary column that contains the covering vector d in (5) will be referred to as the column $n + 3$ of matrix $M = M_G$. The non-degeneracy assumption basically amounts to having $|\Omega^\nu| = 1$ for all ν , thereby excluding any cycling behavior.

PBH uses the least-index rule, which amounts to blocking the driving variable with a basic one that has minimum index within a certain subset of Ω^ν , i.e. $r = \min \Phi^\nu$ for some $\Phi^\nu \subseteq \Omega^\nu$. The set Φ^ν is chosen in order to make the number of degenerate variables decrease as slowly as possible, i.e. among the index set

$$\Phi^\nu = \arg \min_i \{ |\Omega^\nu| - |\Omega_i^{\nu+1}| > 0 : i \in \Omega^\nu \} \subseteq \Omega^\nu$$

where $\Omega_i^{\nu+1}$ is the index set of those variables that would satisfy the min-ratio test at iteration $\nu + 1$ if the driving variable at iteration ν were blocked with y_i^ν as $i \in \Omega^\nu$. The previous conditional implies that a pivot step is taken and then reset in a sort of look-ahead fashion, hence we refer to this rule as the *look-ahead (pivot) rule*. The resulting procedure is specified as Algorithm 2.1 below.

Empirical evidence indicated h as a key parameter for the quality of the final result of Algorithm 2.1. Unfortunately no effective means could be identified to restrict the choice of values in V that can guarantee a good sub-optimal solution, so one has to consider iterating for most, if not all, vertices of V as outlined in Algorithm 2.2. A simple criterion avoids considering those nodes that cannot drive to larger cliques than the current one, because their weight and that of their neighborhood is too small. Unfortunately, this criterion is effective only for very sparse graphs. It has been observed that the schema is sensitive to the ordering of nodes. Since the best figures were obtained by reordering G by decreasing weight of each node and its neighborhood, this feature is formalized in Algorithm 2.2, which is the Pivoting Based Heuristic (PBH).

Algorithm 2.1 A reduced version of Lemke’s Scheme I with the look-ahead rule, applied to the MWCP.

Input: A graph $G = (V, E, w)$ and $h \in V$.

Let $Q_h = (q_{ij})$. $\nu \leftarrow 0$. $K \leftarrow \emptyset$.

The driving variable is x_h .

Infinite loop

Let x_s^ν denote the driving variable.

$\Omega^\nu = \{i : q_{is}^\nu < 0\}$.

If $\Omega^\nu \subseteq \{h\}$ stop: the result is K .

$\Phi^\nu = \arg \min_i \{|\Omega^\nu| - |\Omega_i^{\nu+1}| > 0 : i \in \Omega^\nu\}$.

$r = \min \Phi^\nu$

If $y_r^\nu \equiv x_i$ for some i , then $K \leftarrow K \setminus \{i\}$

Perform $\langle y_r^\nu, x_s^\nu \rangle$

The new driving variable is the variable complementary to y_r^ν

$\nu \leftarrow \nu + 1$

If $y_r^\nu \equiv x_i$ for some i , then $K \leftarrow K \cup \{i\}$

Algorithm 2.2 The pivoting-based heuristic (PBH) for the MWCP.

Input: A graph $G = (V, E, w)$.

Let $G' = (V', E', w')$ be a permutation of G

with $W(\{u'\} \cup N_{u'}) \geq W(\{v'\} \cup N_{v'})$ for all $u', v' \in V'$ with $u' < v'$.

$K^* \leftarrow \emptyset$.

For $v' = 1, \dots, n : W(\{v'\} \cup N_{v'}) > W(K^*)$ do

Run Algorithm 2.1 with G' and v' as input.

Let K be the obtained result.

If $W(K) > W(K^*)$, then $K^* \leftarrow K$.

The result is the mapping of K^* in G .

2.3. Combinatorial interpretation of PBH. Given a set $S \subseteq V$ of vertices, abbreviate by $S_i = S \setminus N_i$ the vertices in S that are not adjacent to i , and denote by

$$\tilde{d}_S(i) = \sum_{j \in S_i} \left(1 + \frac{w_j}{w_i}\right), \quad i \in V \setminus S$$

(twice) the average of weighted and unweighted co-degree of i w.r.t. S . Note that if $S_i \neq \emptyset$, then $\tilde{d}_S(i) \geq 1$, and that additivity holds: two disjoint subsets S and T satisfy $\tilde{d}_{S \cup T}(i) = \tilde{d}_S(i) + \tilde{d}_T(i)$ for any $i \in V \setminus (S \cup T)$. Further, for $i \neq j$ we have $\frac{1}{2w_j} \tilde{d}_{\{j\}}(i) = q_{ij}$ as defined in (2).

Let us represent the tableau at iteration ν of Algorithm 2.1 in Table 1, where x_p denotes the driving variable. Notice that, without loss of generality, the vertices of the graph have been numbered in such a way that the variables which entered the basis up to iteration ν are those corresponding to vertices 1 up to $p-1$; these variables entered the basis in the same order of the vertices (i.e. x_1 first, then x_2 , and so on); the driving variable x_p is the one corresponding to vertex p . Note that each entry $a_{i,j}^\nu$

TABLE 1
Tableau at iteration ν of Algorithm 2.1

	q	y_2	\dots	y_p	x_p	\dots	x_n	y_1	x_{n+2}	y_{n+1}
x_{n+1}	1	$a'_{1,1}$		$a'_{1,p-1}$	$a'_{1,p}$		$a'_{1,n}$			
x_1	0	$a'_{2,1}$		$a'_{2,p-1}$	$a'_{2,p}$		$a'_{2,n}$			
\vdots	\vdots									
x_{p-1}	0	$a'_{p,1}$		$a'_{p,p-1}$	$a'_{p,p}$		$a'_{p,n}$			
y_{p+1}	0	$a'_{p+1,1}$		$a'_{p+1,p-1}$	$a'_{p+1,p}$		$a'_{p+1,n}$			
\vdots	\vdots									
y_n	0	$a'_{n,1}$		$a'_{n,p-1}$	$a'_{n,p}$		$a'_{n,n}$			
θ										
y_{n+2}										

of the tableau corresponds to the following couple of variables

$$\begin{aligned}
(x_{i-1}, x_j) & \quad \text{for } i = 1, \dots, p, \quad \text{and } j = p, \dots, n; \\
(y_i, x_j) & \quad \text{for } i = p+1, \dots, n, \quad \text{and } j = p, \dots, n; \\
(x_{i-1}, y_{j+1}) & \quad \text{for } i = 1, \dots, p, \quad \text{and } j = 1, \dots, p-1; \\
(y_i, y_{j+1}) & \quad \text{for } i = p+1, \dots, n, \quad \text{and } j = 1, \dots, p-1.
\end{aligned}$$

where $x_0 \equiv x_{n+1}$.

Given the situation displayed in the tableau we now prove the following results, which allow us to give a nice combinatorial interpretation of PBH and to prove a few conjectures stated in [11].

PROPOSITION 2.2. *At iteration ν of the PBH algorithm, let x_p be the driving variable and $K = \{i \in V : x_i \text{ is basic}\} = \{1, \dots, p-1\}$. Then, the following statements are true:*

1. $\forall i \notin K \cup \{p\}$

$$a'_{i,p} = \frac{1}{2w_p} \left[\tilde{d}_{K \cup \{p\}}(i) - 1 \right].$$

which can be negative only if $a'_{i,p} = -\frac{1}{2w_p}$, i.e. when $K \cup \{p\} \setminus N_i = \emptyset$ or, equivalently, vertex i is adjacent to any vertex in $K \cup \{p\}$.

2. $\forall i \notin K \cup \{p\}$ such that $a'_{i,p} < 0$ (or equivalently, in view of point 1., $a'_{i,p} = -\frac{1}{2w_p}$), it holds that

$$a'_{i,i} = \frac{1}{2w_i}.$$

3. $\forall i \notin K \cup \{p\}$ such that $a'_{i,p} < 0$ it holds that $\forall s \notin K \cup \{i, p\}$

$$a'_{s,i} = \frac{1}{2w_i} \tilde{d}_{\{i\}}(s) = q_{is}.$$

Proof. The proof is by induction on ν . We first prove that the result is true at iteration $\nu = 1$ and then we assume that it is true at iteration $\nu \geq 1$ and we prove it is true at iteration $\nu + 1$.

$\nu = 1$

Point 1 For any $i \neq p$ it holds that, according to (3.6) in [11],

$$(6) \quad a_{i,p}^1 = q_{ip} - q_{pp} = q_{ip} - \frac{1}{2w_p}$$

Since

$$(7) \quad q_{ip} = \left\{ \begin{array}{ll} 0 & \text{if } \{i, p\} \in E \\ \frac{1}{2w_i} + \frac{1}{2w_p} & \text{otherwise} \end{array} \right\} = \frac{1}{2w_p} \tilde{d}_{K \cup \{p\}}(i)$$

for $K = \emptyset$, point 1 holds.

Point 2 It follows from (6) and (7) that $a_{i,p}^1 < 0$ implies $q_{ip} = q_{pi} = 0$. Since $a_{i,i}^1 = q_{ii} - q_{pi} = \frac{1}{2w_i} - 0 = \frac{1}{2w_i}$, point 2 follows.

Point 3 Now let $a_{i,p}^1 < 0$. For any $s \notin \{i, p\}$

$$a_{s,i}^1 = q_{si} - q_{pi} = q_{si} = \frac{1}{2w_i} \tilde{d}_{\{i\}}(s)$$

so that point 3 is satisfied.

From ν to $\nu + 1$

Again by renumbering of the vertices if necessary we can assume that the new driving variable at iteration $\nu + 1$ is x_{p+1} . Then, in particular, it holds that $a_{p+1,p}^\nu < 0$ or, equivalently, in view of point 1 of the inductive assumption (putting $i = p + 1$), $a_{p+1,p}^\nu = -\frac{1}{2w_p}$. For any s, t with $s \neq p + 1$ and $t \neq p$ it therefore holds, in view of the pivoting update, that

$$(8) \quad a_{s,t}^{\nu+1} = a_{s,t}^\nu - \frac{a_{p+1,t}^\nu}{a_{p+1,p}^\nu} a_{s,p}^\nu = a_{s,t}^\nu + 2w_p a_{p+1,t}^\nu a_{s,p}^\nu.$$

Similarly, for $t \neq p$ we have the update of the pivot-row

$$(9) \quad a_{p+1,t}^{\nu+1} = \frac{a_{p+1,t}^\nu}{-a_{p+1,p}^\nu} = 2w_p a_{p+1,t}^\nu.$$

Finally,

$$a_{p+1,p}^{\nu+1} = \frac{1}{a_{p+1,p}^\nu} = -2w_p.$$

Point 1 For any $s \notin K \cup \{p, p + 1\}$ it follows from (8) for $t = p + 1$ that

$$(10) \quad a_{s,p+1}^{\nu+1} = a_{s,p+1}^\nu + 2w_p a_{p+1,p+1}^\nu a_{s,p}^\nu = a_{s,p+1}^\nu + \frac{1}{2w_{p+1}} \left[\tilde{d}_{K \cup \{p\}}(s) - 1 \right],$$

where the last equality follows from point 1 (putting $i = s$) and point 2 (putting $i = p + 1$) of the inductive assumption. In view of point 3 of the inductive assumption (putting again $i = p + 1$), we have

$$a_{s,p+1}^\nu = \frac{1}{2w_{p+1}} \tilde{d}_{\{p+1\}}(s),$$

and therefore, using the additivity property for \tilde{d} addressed above,

$$a_{s,p+1}^{\nu+1} = \frac{1}{2w_{p+1}} \left[\tilde{d}_{K \cup \{p,p+1\}}(s) - 1 \right],$$

which means that point 1 holds also at iteration $\nu + 1$.

Point 2 Now let us assume that $a_{i,p+1}^{\nu+1} < 0$. We want to prove that $a_{i,i}^{\nu+1} = \frac{1}{2w_i}$. First we notice that in view of the proof of point 1 it holds that $a_{i,p+1}^{\nu+1} < 0$ if and only if

$$\begin{aligned} (11) \quad & a_{i,p+1}^\nu = 0; \\ (12) \quad & a_{i,p}^\nu < 0; \text{ and} \\ (13) \quad & \{p+1, i\} \in E. \end{aligned}$$

Moreover, it follows from $a_{p+1,p}^\nu < 0$ (since x_{p+1} is now the driving variable), and from point 3 of the inductive assumption for $s = p+1$

$$(14) \quad a_{p+1,i}^\nu = \frac{1}{2w_i} \tilde{d}_{\{i\}}(p+1) = 0,$$

because of (13). It follows from (8) with $s = t = i$ and (14) that

$$a_{i,i}^{\nu+1} = a_{i,i}^\nu + 2w_p a_{p+1,i}^\nu a_{i,p}^\nu = a_{i,i}^\nu = \frac{1}{2w_i},$$

where the last equality follows from (14) and point 2 of the inductive assumption. Then point 2 is satisfied also for $\nu + 1$.

Point 3 Again let us assume that $a_{i,p+1}^{\nu+1} < 0$. As in the proof for the previous point, we obtain (12) and (14). Now for any $t \notin K \cup \{s, p, p+1\}$

$$a_{s,i}^{\nu+1} = a_{s,i}^\nu + 2w_p a_{p+1,i}^\nu a_{s,p}^\nu = a_{s,i}^\nu$$

where the last equality follows from (14). But since (12) holds, point 3 of the inductive assumption implies that

$$a_{s,i}^\nu = \frac{1}{2w_i} \tilde{d}_{\{i\}}(s),$$

and point 3 also holds for $\nu + 1$. \square

PROPOSITION 2.3. *At iteration ν of the PBH algorithm, let x_p be the driving variable and $K = \{i \in V : x_i \text{ is basic}\} = \{1, \dots, p-1\}$. Then, the following statements are true:*

1. For any $j \in K$

$$a_{j+1,p}^\nu = \frac{w_j}{w_p}.$$

2. If $a_{i,p}^\nu < 0$, $t \neq 1$, then

$$\forall j \in K : a_{j+1,t}^\nu = 0.$$

3. If $a_{i,p}^\nu < 0$, $t \neq 1$, then

$$a_{1,t}^\nu = a_{1,t}^1 = -1.$$

4.

$$a_{1,p}^\nu = \frac{1}{2w_p} [1 - 2W(K \cup \{p\})] .$$

Proof. The proof is again by induction. We first prove that the result is true at iteration $\nu = 1$ and then we assume that it is true at iteration $\nu \geq 1$ and we prove it is true at iteration $\nu + 1$.

$\nu = 1$

Points 1 and 2 Points 1 and 2 are trivially satisfied because at iteration $\nu = 1$ it holds that $K = \emptyset$.

Point 3 Since, by assumption, the first driving variable is x_1 (i.e., $p = 1$) and $t \neq 1$, then $a_{t,1}^1 = q_{t1} - q_{11} = \frac{1}{2w_1} [\tilde{a}_{\{1\}}(t) - 1]$. If $a_{t,1}^1 < 0$, then $\{t, 1\} \in E$. Thus

$$a_{1,t}^1 = q_{1t} - 1 = q_{t1} - 1 = -1$$

and point 3 is satisfied.

Point 4 Finally, since $K = \emptyset$,

$$a_{1,1}^1 = q_{11} - 1 = \frac{1}{2w_1} [1 - 2W(K \cup \{1\})] .$$

From ν to $\nu + 1$

Again we assume that the new driving variable at iteration $\nu + 1$ is x_{p+1} .

Point 1 For any $j \in K$ it holds that

$$a_{j+1,p+1}^{\nu+1} = a_{j+1,p+1}^\nu + 2w_p a_{p+1,p+1}^\nu a_{j+1,p}^\nu .$$

Now since x_{p+1} is the driving variable, we have $a_{p+1,p}^\nu < 0$ and from point 2 of Proposition 2.2 with $i = p + 1 \notin K \cup \{p\}$, we have $a_{p+1,p+1}^\nu = \frac{1}{2w_{p+1}}$. Moreover, point 2 of the inductive assumption implies $a_{j+1,p+1}^\nu = 0$. Therefore

$$a_{j+1,p+1}^{\nu+1} = a_{j+1,p+1}^\nu + \frac{w_p}{w_{p+1}} a_{j+1,p}^\nu = \frac{w_p}{w_{p+1}} a_{j+1,p}^\nu .$$

In view of point 1 of the inductive assumption, it holds that $a_{j+1,p}^\nu = \frac{w_j}{w_p}$. Thus, point 1 is satisfied also for $\nu + 1$ for any $j \in K$. We still need to prove it for $j = p$. But in this case, (9) for $t = p + 1$ implies

$$a_{p+1,p+1}^{\nu+1} = 2w_p a_{p+1,p+1}^\nu = \frac{2w_p}{2w_{p+1}} = \frac{w_p}{w_{p+1}} ,$$

and point 1 is true also for $j = p$.

Point 2 First we notice that in point 1 we have just shown that for $j \in K \cup \{p\} = \{1, \dots, p\}$ it holds that $a_{j+1,p+1}^{\nu+1} = \frac{w_j}{w_{p+1}} > 0$. This also means that for $t = j + 1$, the relation $a_{t,p+1}^{\nu+1} < 0$ is impossible for $t \in \{2, \dots, p + 1\}$. Hence $a_{t,p+1}^{\nu+1} < 0$ and $t \neq 1$ implies $t \notin K \cup \{p\}$. Now, we recall that in the proof of point 2 in Proposition 2.2

we showed that (putting $t = i$) $a_{i,p+1}^{\nu+1} < 0$ implies $a_{p+1,t}^{\nu} = 0$ due to (14) and $a_{t,p}^{\nu} < 0$ due to (12). Next we notice that for any $j \in K$ it follows from (8) that

$$a_{j+1,t}^{\nu+1} = a_{j+1,t}^{\nu} + 2w_p a_{p+1,t}^{\nu} a_{j+1,p}^{\nu} = a_{j+1,t}^{\nu}.$$

But since $a_{t,p}^{\nu} < 0$, point 2 of the inductive assumption implies that $a_{j+1,t}^{\nu} = 0$ and point 2 is satisfied also for $\nu + 1$ for any $j \in K$. We still need to prove that it is true for $j = p$. But in such case it follows from (9) that

$$a_{p+1,t}^{\nu+1} = 2w_p a_{p+1,t}^{\nu} = 0,$$

and point 2 is satisfied also for $j = p$.

Point 3 For any $t \neq 1$ such that $a_{t,p+1}^{\nu+1} < 0$ we have $a_{p+1,t}^{\nu} = 0$ and thus it holds that

$$a_{1,t}^{\nu+1} = a_{1,t}^{\nu} + 2w_p a_{p+1,t}^{\nu} a_{1,p}^{\nu} = a_{1,t}^{\nu},$$

and since $a_{t,p}^{\nu} < 0$ (refer, again, to the proof of Proposition 2.2), point 3 of the inductive assumption implies $a_{1,t}^{\nu} = -1$, so that point 3 is satisfied also for $\nu + 1$.

Point 4 Note that $a_{p+1,p}^{\nu} < 0$ implies $a_{1,p+1}^{\nu} = -1$ in view of point 3 of the inductive assumption, while $a_{p+1,p+1}^{\nu} = \frac{1}{2w_{p+1}}$ (see Proposition 2.2). Then,

$$a_{1,p+1}^{\nu+1} = a_{1,p+1}^{\nu} + 2w_p a_{p+1,p+1}^{\nu} a_{1,p}^{\nu} = -1 + \frac{w_p}{w_{p+1}} a_{1,p}^{\nu}.$$

But by point 4 of the inductive assumption, $a_{1,p}^{\nu} = \frac{1}{2w_p} [1 - 2W(K \cup \{p\})]$. Thus

$$a_{1,p+1}^{\nu+1} = -1 + \frac{1}{2w_{p+1}} [1 - 2W(K \cup \{p\})] = \frac{1}{2w_{p+1}} [1 - 2W(K \cup \{p, p+1\})],$$

and point 4 is satisfied also for $\nu + 1$. \square

2.4. Algorithmic consequences. Now we are ready to summarize the consequences of the results in Propositions 2.2 and 2.3.

1. As conjectured in [11], once a x_i variable with $i \in V$ enters the basis, it never exits. This follows from point 1 of Proposition 2.3 where each entry $a_{j+1,p}^{\nu}$ in the column of the driving variable and related to a variable x_j in the basis is equal to $\frac{w_j}{w_p}$ so that x_j cannot exit the basis.
2. The candidates to become the new driving variable, i.e. with $a_{i,p}^{\nu} < 0$ are only the variables related to vertices i which form a clique with the vertices in $K \cup \{p\}$. This follows from point 1 of Proposition 2.2 from which $a_{i,p}^{\nu} < 0$ if and only if $[K \cup \{p\}]_i = \emptyset$ which exactly means that i is adjacent to every vertex in $K \cup \{p\}$.
3. The rule to choose the next driving variable in [11] is the following greedy rule. Let H the set of candidates to become the new driving variable, i.e.,

$$H = \{j : a_{j,p}^{\nu+1} < 0\} = \bigcap_{i \in K \cup \{p\}} N_i.$$

For each $j \in H$ let

$$U(j) = \{s \in H \setminus \{j\} : \{s, j\} \in E\} = H \cap N_j.$$

Then, the new driving variable is selected in the set

$$\arg \max\{|U(j)| : j \in H\},$$

and, more precisely, as the variable with the lowest index in this set.

Note that the algorithm always stops if and only if $H = \emptyset$ or, equivalently, when K is a maximal clique. This proves the conjecture stated in [11] that saddle points detected by the algorithm are always maximal cliques, and thus local solutions, even if program (1) has stationary points which are not local minimizers.

3. Experience with new heuristic approaches.

3.1. Description of the heuristics. In the previous section we proved that the continuous approach PBH for the solution of the MWCP is equivalent to a combinatorial heuristic. In the unweighted case, to which we shall now restrict our attention, such a heuristic turns out to belong to the class of greedy heuristics denoted in [5] by SM^i ($i = 1, 2, \dots$), which are described in what follows.

Greedy heuristic SM^i

Step 1 Given a graph $G = (V, E)$, let \mathcal{Q} be the set of all cliques of graph G with cardinality i . Set $\mathcal{K} = \emptyset$, $K^* = \emptyset$ and $max = 0$.

Step 2 If $\mathcal{Q} \neq \emptyset$, select a clique $H \in \mathcal{Q}$ and update $\mathcal{Q} = \mathcal{Q} \setminus \{H\}$. Put $K = H$ and go to Step 3; otherwise, return max and K^* .

Step 3 If K is a maximal clique, then update $\mathcal{K} = \mathcal{K} \cup \{K\}$ and go to Step 4; otherwise go to Step 5.

Step 4 If $|K| > max$, set $max = |K|$ and $K^* = K$. Go back to Step 2.

Step 5 Select a vertex

$$(15) \quad \ell \in \arg \max_{j \in C_0(K)} |C_0(K) \cap N_j|$$

at random, where

$$(16) \quad C_0(K) = \{j \in V : \{j, k\} \in E \ \forall k \in K\} = \bigcap_{k \in K} N_k$$

is the set of all vertices in G adjacent to each vertex in the current clique K .

Step 6 Set $K = K \cup \{\ell\}$ and go back to Step 3.

We notice that the combinatorial version of PBH in the unweighted case is equivalent to SM^1 , except for the minimal difference of the random selection of ℓ in (15)—in PBH ℓ is selected as the node with the lowest index in $C_0(K)$. Of course, although the heuristic SM^1 frequently returns results of good quality, the heuristic SM^2 is often superior to SM^1 (see Table 2). On the other hand, as expected, the computation times for SM^2 are much larger than those for SM^1 (see Table 3). The aim of this section is that of proposing modifications of the heuristic SM^1 in such a way that, at the price of larger computation times, we will also be able to obtain better results. Basically, we would like to modify the heuristic SM^1 in such a way that the modification is closer to SM^2 than to SM^1 from the point of view of the quality of the results, but is much closer to SM^1 than to SM^2 from the point of view of the computation times.

The first modification proposed here is to allow vertex swaps while we build a maximal clique. The heuristics SM^i start with a clique of cardinality i and add a new vertex at each iteration until a maximal clique is reached. Some other approaches do the same but do not stop once a maximal clique is reached, allowing for the removal of vertices when this situation occurs (see e.g. the plateau search in [2]). The modification proposed here does not always add a new vertex to the current clique but also allows to swap vertices. If K is the current clique, let $C_0(K)$ be defined as in (16) and let

$$C_1(K) = \{j \in V \setminus K : |N_j \cap K| = |K| - 1\},$$

i.e. $C_1(K)$ is the set of vertices outside K which are connected to all vertices in K but exactly one. If we select a vertex $\ell \in C_0(K)$, then we can add it to K and update K as follows

$$K = K \cup \{\ell\}.$$

This is exactly what heuristics SM^i do with the selection (15). But if we select a vertex $\ell \in C_1(K)$ we can not simply add it to K ; if we still want to have a clique we need to swap vertex ℓ with the unique vertex

$$(17) \quad k_\ell \in K \quad \text{such that} \quad \{\ell, k_\ell\} \notin E,$$

i.e. the new clique will be

$$K = K \cup \{\ell\} \setminus \{k_\ell\}.$$

Therefore, if we select a vertex in $C_1(K)$ we change the current clique but without increasing the cardinality, i.e. the method is not strictly monotone. In the modification proposed here we decide whether to increase the cardinality, by selecting a vertex in $C_0(K)$, or to swap vertices, by selecting a vertex in $C_1(K)$. Note that $C_0(K) = \emptyset$ if K is a maximal clique. Similarly, some authors call a maximal clique K *strictly maximal*, if $C_1(K) = \emptyset$, i.e., if every swap destroys the clique property.

Now, selection of a vertex in $C_0(K) \cup C_1(K)$ is done according to a rule similar to (15) but with the computation of the maximum extended also to vertices in $C_1(K)$, i.e.

$$(18) \quad \ell \in \arg \max_{j \in C_0(K) \cup C_1(K)} |C_0(K) \cap N_j|.$$

We underline that in some cases we force the heuristic to select the next vertex according to (15), namely:

- during a fixed number ($START_SWAP$) of initial iterations;
- when the number of swaps performed is a fixed number T times the cardinality of the current clique $|K|$;
- when the vertex selected in $C_1(K)$ is the same as the last one removed by the last swap. Without this condition, which can be interpreted as a very limited application of taboo rules, it has been observed that the heuristic is much less efficient (worse results and larger computation times), apparently because it happens that the same two vertices are swapped in and out until the maximum allowed number of swaps is reached and then the heuristic behaves exactly as the standard greedy one.

The heuristic with the proposed modification, denoted with SM^1_SWAP is described in what follows.

Heuristic SM^1_SWAP

Step 1 Given a graph $G = (V, E)$, let $W = V$, $K^* = \emptyset$ and $max = 0$. Select a value for the parameters $START_SWAP$ and T .

Step 2 If $W = \emptyset$ return K^* and max . Otherwise select a vertex $h \in W$, set $W = W \setminus \{h\}$ and set $K = \{h\}$. Set $n_swaps = 0$ and $last_swap = \emptyset$.

Step 3 If K is a maximal clique, then set $\mathcal{K} = \mathcal{K} \cup \{K\}$ and go to Step 4; otherwise go to Step 5.

Step 4 If $|K| > max$, set $max = |K|$ and $K^* = K$. Go back to Step 2.

Step 5 If $n_swaps \leq T|K|$ and $|K| \geq START_SWAP$, then go to Step 6, otherwise go to Step 7.

Step 6 Select randomly $\ell \in V$ according to (18). If $\ell = last_swap$, then go to Step 7, otherwise go to Step 8.

Step 7 Select $\ell \in V$ according to (15).

Step 8 If $\ell \in C_1(K)$, then update

$$n_swaps = n_swaps + 1 \quad \text{and} \quad last_swap = k_\ell$$

where k_ℓ is defined in (17), and

$$K = K \cup \{\ell\} \setminus \{k_\ell\}.$$

Otherwise set

$$K = K \cup \{\ell\}.$$

Go back to Step 3.

3.2. Benchmark results and comparisons. We consider a selection of DIMACS benchmark graphs [10]¹ (some benchmarks, such as the *c-fat* and *johnson* graph classes and most of the *hamming* graphs, have been omitted because their solutions were easily detected by all the tested heuristics). In Table 2 we report the average, worst and best results for the heuristics SM^1 , SM^1_SWAP and SM^2 over 10 random runs, while in Table 3 we report the the average computation times. For the heuristic SM^1_SWAP we employed the following parameter values: $START_SWAP = 5$, $T = 2$.

We notice that, as required, the results for SM^1_SWAP are often competitive with those of SM^2 , while the computation times are much closer to those of SM^1 . If we compare the results of SM^1 with those of SM^1_SWAP , we notice that SM^1_SWAP often outperforms SM^1 . With the only exception of the *p-hat1500-1* graph, we notice that SM^1_SWAP largely outperforms SM^1 over most of the cliques in the *p-hat* class. The same is true for many cliques in the *san* class and for the *hamming10-4* graph. It is remarkable that, in spite of the much lower computational times, the heuristic SM^1_SWAP turns out to be superior even to SM^2 over three graphs, namely *p-hat1000-3*, *p-hat1500-3* and *hamming10-4*. We remark that the best known clique size (68) reported for the *p-hat1000-3* graph has, to the authors' knowledge, only been very recently detected in [6].

We also note that changing the parameter $START_SWAP$ from 5 to 2 turned out to be profitable in some cases. In particular the average result for *keller6* has been 55.7 with a minimum of 55 and a maximum of 57. Finally, note that on *keller6* even a single run of SM^2 did not terminate after a whole day and hence we stopped it prematurely (this has been indicated with a hyphen in Tables 2 and 3).

Now we compare our results to those reported in four other papers, some of them classics, some published recently, and one still unpublished. By mere coincidence, all first authors' names start with a "B" and their lexicographic order coincides with the historic one. The classic contribution by Balas and Niehaus [1] treats 20 cases occurring in the above list (e.g., all graphs in the *san* class are missing), and is considered to be one of the dominating results of the DIMACS challenge [10]. Their results are still, ten years ago, competitive, however, are dominated by the reactive local search procedure of Battiti and Protasi [2] who report on the same subset of 20 instances. The latter specify runtimes to achieve the best observed solution, with

¹<ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/cliue>

TABLE 2

Average (worst, best) results over 10 random tests for SM^1 , SM^1_SWAP and SM^2 . When only a single value is reported, all the random tests returned that single value.

Graph	Best(known) value	SM^1	SM^1_SWAP	SM^2
p_hat300-1	8	8	8	8
p_hat300-2	25	25	25	25
p_hat300-3	36	35	36	36
p_hat500-1	9	9	9	9
p_hat500-2	36	36	36	36
p_hat500-3	≥ 50	49	50	50
p_hat700-1	11	10.2 (9,11)	11	11
p_hat700-2	44	43.8 (43,44)	44	44
p_hat700-3	≥ 62	62	62	62
p_hat1000-1	≥ 10	10	10	10
p_hat1000-2	≥ 46	46	46	46
p_hat1000-3	≥ 68	64	68	66.8 (66,67)
p_hat1500-1	12	12	11.4 (11,12)	12
p_hat1500-2	≥ 65	64	65	65
p_hat1500-3	≥ 94	91 (90,92)	94	93
MANN_a27	126	126	126	126
MANN_a45	345	343.8 (343,344)	343.8 (343,344)	344
keller4	11	11	11	11
keller5	27	27	27	27
keller6	≥ 59	54.4 (54,55)	55.4 (55,56)	—
brock200_1	21	20.6 (20,21)	20.9 (20,21)	21
brock200_2	12	11	11	12
brock200_3	15	14	13.8 (13,14)	15
brock200_4	17	16	16	17
brock400_1	27	23.8 (23,24)	24.2 (24,25)	25
brock400_2	29	24	24.5 (24,25)	29
brock400_3	31	24	24.9 (24,25)	31
brock400_4	33	24	24.7 (24,25)	33
brock800_1	23	21	20.2 (20,21)	21
brock800_2	24	20	20.7 (20,21)	21
brock800_3	25	20.8 (20,21)	20.9 (20,22)	22
brock800_4	26	20.2 (20,21)	20.2 (20,21)	21
san_200_0.7.1	30	30	30	30
san_200_0.7.2	18	17.4 (17,18)	18	18
san_200_0.9.1	70	70	70	70
san_200_0.9.2	60	60	60	60
san_200_0.9.3	44	41.4 (40,44)	44	44
san_400_0.5.1	13	13	13	13
san_400_0.7.1	40	40	40	40
san_400_0.7.2	30	30	30	30
san_400_0.7.3	22	17.8 (17,19)	19.9 (17,22)	22
san_400_0.9.1	100	100	100	100
sanr_200_0.7	18	18	18	18
sanr_200_0.9	42	41	41.9 (41,42)	42
sanr_400_0.5	13	12.4 (12,13)	13	13
sanr_400_0.7	≥ 21	20	21	21
san1000	15	15	15	15
hamming10-4	≥ 40	36	40	36

TABLE 3

Average computation times (in seconds) over 10 random tests for SM^1 , SM^1_SWAP and SM^2 .

Graph	SM^1	SM^1_SWAP	SM^2
p-hat300-1	0.03	0.11	0.73
p-hat300-2	0.15	0.66	7.04
p-hat300-3	0.31	1.19	22.75
p-hat500-1	0.12	0.46	4.26
p-hat500-2	0.64	3.04	50.96
p-hat500-3	1.37	5.70	166.72
p-hat700-1	0.25	1.03	12.59
p-hat700-2	1.86	7.84	181.86
p-hat700-3	3.87	13.53	612.05
p-hat1000-1	0.61	2.39	39.52
p-hat1000-2	4.26	18.93	605.61
p-hat1000-3	10.34	36.33	2324.86
p-hat1500-1	1.95	9.84	163.85
p-hat1500-2	15.94	92.29	3418.38
p-hat1500-3	42.33	180.20	13481.74
MANN_a27	3.44	17.45	436.60
MANN_a45	102.16	659.05	46273.89
keller4	0.02	0.10	0.88
keller5	1.74	8.81	380.68
keller6	373.8	1651.2	—
brock200.1	0.07	0.24	3.63
brock200.2	0.03	0.09	0.80
brock200.3	0.05	0.14	1.51
brock200.4	0.05	0.16	2.04
brock400.1	0.41	1.25	37.57
brock400.2	0.45	1.25	37.69
brock400.3	0.43	1.25	39.08
brock400.4	0.43	1.28	37.87
brock800.1	1.71	4.90	217.42
brock800.2	1.73	5.02	219.61
brock800.3	1.69	4.88	221.50
brock800.4	1.73	4.76	213.87
san_200_0.7_1	0.07	0.43	3.28
san_200_0.7_2	0.06	0.35	2.66
san_200_0.9_1	0.23	1.29	16.56
san_200_0.9_2	0.20	0.97	12.92
san_200_0.9_3	0.15	0.58	10.63
san_400_0.5_1	0.12	0.55	7.30
san_400_0.7_1	0.44	2.54	39.49
san_400_0.7_2	0.36	2.11	30.53
san_400_0.7_3	0.29	1.35	25.03
san_400_0.9_1	1.30	6.45	180.41
sanr_200_0.7	0.06	0.19	2.59
sanr_200_0.9	0.18	0.61	11.68
sanr_400_0.5	0.14	0.49	8.15
sanr_400_0.7	0.33	1.02	27.10
san1000	1.35	7.85	192.38
hamming10-4	4.83	28.19	1606.12

averages and standard deviation of the same order of magnitude (sometimes with larger deviations), which indicates a highly variable runtime behaviour that is prone to relatively large outliers. Since we report, in Table 3, the *overall* average runtime, it is not surprising that the times reported in [2] are comparatively small. Still, there is an exception, the *brock* instances, where the times in [2] consistently exceed even those in SM^2 , save two 800-vertex cases where they are comparable to the SM^1_SWAP times. In a paper to appear [6], Burer and co-workers provide results for all cases covered here, which are slightly dominated by those in Table 1. However, comparing their runtimes with those of Table 2, one sees that their procedure is considerably slower even, with very few exceptions, compared to SM^2 . Finally, the unpublished results by Busygin [7] are also dominated by those presented here, with exception of his really impressive achievements on the large *brock* instances, which also beat all other approaches discussed here. Runtimes reported in [7] are roughly within the same order of magnitude as those in Table 2, only for the *brock*, some *phat* and the *hamming* instances his procedure is slower by a factor ranging between four and ten. Of course, comparing runtimes is always problematic due to differences in implementation, architecture and hardware platforms. Our experiments have been performed on a 1 GHz Pentium III under Linux 2.4 with 240 MB RAM.

3.3. Diversification. The only benchmarks over which we did not achieve very satisfying results were those belonging to the *brock* class. However, the less satisfying results over the *brock* graphs do not come as a surprise. The authors of [5], who proposed the *brock* class, explicitly say that it was their intention to create difficult problems for the class of greedy heuristics SM^i . We notice that SM^1 is unable to detect all except one of the maximum cliques for the twelve *brock* graphs. We can also notice that SM^2 is unable to detect the maximum clique for the *brock400_1* graph and for all the *brock800* graphs. Unfortunately the introduction of swaps does not significantly improve the results of SM^1 . Therefore, we would like to propose a further modification in order to be able to reach results at least comparable to that of SM^2 . While we run a heuristic such as SM^1_SWAP we collect in the set \mathcal{K} , updated in Step 3, all the maximal cliques generated. Some information, which for the moment we only denote by *Info* without specifying it in more detail, can be extracted from \mathcal{K} . Once we have extracted such information, we can use it to run the less time consuming heuristic SM^1 but by changing rule (15) for the selection of a new vertex ℓ . Namely, the rule to select the next vertex ℓ among vertices $j \in C_0(K)$ could be some function of $C_0(K)$, N_j but also of the information *Info*. Therefore we could modify heuristic SM^1 by modifying Step 5 as follows:

Step 5' Randomly select a vertex $\ell \in C_0(K)$ according to the following rule:

$$(19) \quad \ell \in \arg \max_{j \in C_0(K)} \nu(C_0(K), N_j, Info).$$

Of course we still need to specify how to extract the information *Info* from \mathcal{K} and how to use it in order to define the function ν . The kind of information we decide to extract depends on how we intend to use it. Here we propose to use it in order to diversify the search for cliques. In order to accomplish such a diversification of the search we extract from \mathcal{K} the following information

$$Info = \{S_j : j \in V\},$$

where, for any $j \in V$

$$(20) \quad S_j = \{i \in N_j : \{i, j\} \not\subseteq K \ \forall K \in \mathcal{K}\},$$

i.e. S_j is the set of vertices adjacent to vertex j which never appear together with j in any of the maximal cliques generated by heuristic SM^1_SWAP . The above information is used as follows to define the function ν employed in Step 5'

$$(21) \quad \nu(C_0(K), N_j, Info) = |C_0(K) \cap N_j| + |C_0(K) \cap S_j|$$

The first term in the sum is the usual one favouring, according to the greedy principle, vertices with a lot of neighbors in $C_0(K)$. But the second term favours a vertex j if many of its neighbors in $C_0(K)$ have never been together with j in the cliques generated by heuristic SM^1_SWAP (i.e. vertices belonging to S_j). Therefore, function (21) is a compromise between the standard greedy rule and a term which favours the generation of cliques different from those previously generated by heuristic SM^1_SWAP . This is basically equivalent to changing the weights of the vertices in the graph. While in SM^1 each vertex has weight 1 and $|C_0(K) \cap N_j|$ is equivalent to the sum of the weights of the vertices in $C_0(K)$ which are adjacent to vertex j , in the proposed modification a vertex $i \in C_0(K)$ has weight 1 if $i \notin S_j$ and weight 2 if $i \in S_j$. The modification of weights after each restart of a greedy algorithm has been already explored in the adaptive algorithms presented in [9]. However, we underline that while in [9] the weight of a vertex is the same with respect to any other vertex, in the modification proposed in (21) each vertex has not a fixed weight but its weight is relative to other vertices. The same vertex i may have weight 1 with respect to a vertex j_1 (if $i \notin S_{j_1}$) but it may have weight 2 with respect to a different vertex j_2 (if $i \in S_{j_2}$).

We remark that running again SM^1 with the modified Step 5' after having run heuristic SM^1_SWAP only slightly increases the computation times with respect to those of SM^1_SWAP . In Table 4 we report the results and the computation times obtained by running heuristic SM^1_SWAP followed by heuristic SM^1 with the modified Step 5' over the *brock* graphs. We notice that with a slight increase in the computation times with respect to SM^1_SWAP we are now able to detect the maximum clique for many *brock* graphs. The exceptions are the *brock400_1* graph and the *brock800* graphs, but on these graphs even the much more time consuming heuristic SM^2 fails. Again, changing the parameter *START_SWAP* from 5 to 2 turned out to be profitable in some cases. In particular for *brock400_4* the optimal value 33 has been detected in all the random runs and for *brock200_4* the average increased to 16.3.

After presenting the successes of such a modification we also need to underline its failures. On the other graphs for which SM^1_SWAP was not always able to return the maximum clique, the modification proposed above did not lead to any improvement (with the only exception of the *MANN_a45* graph for which an improvement from an average of 343.8 to 344 was accomplished). This is probably due to the fact that the information extracted as in (20) and its use in (21) are not always as appropriate as they are for the *brock* graphs. However, the successes obtained with such graphs justify the hope that ways other than (20) of extracting information and inserting it into the definition of a function ν , could lead to further good results. We believe that gaining insight into the reasons of the successes (and the failures) of the information extracted and the ways it is employed could be an interesting issue for future research.

REFERENCES

TABLE 4

Average (worst, best) results and average computation times over 10 random tests for SM^1_SWAP followed by the modified SM^1 . When only a single value is reported, all the random tests returned that single value.

Graph	average (worst, best) result	average time
brock200_1	20.9 (20,21)	0.36
brock200_2	11.7 (11,12)	0.14
brock200_3	15	0.21
brock200_4	16.1 (16,17)	0.25
brock400_1	24.2 (24,25)	1.98
brock400_2	25.8 (24,29)	2.04
brock400_3	26.7 (24,31)	2.02
brock400_4	30.6 (25,33)	2.03
brock800_1	20.2 (20,21)	8.24
brock800_2	20.7 (20,21)	8.52
brock800_3	20.9 (20,22)	8.26
brock800_4	20.2 (20,21)	8.25

- [1] E. BALAS AND W. NIEHAUS, *Finding large cliques in arbitrary graphs by bipartite matching*, in Cliques, Coloring and Satisfiability, D. Johnson and M. A. Trick, eds., vol. 26 of DIMACS Series, AMS, 1996, pp. 29–51.
- [2] R. BATTITI AND M. PROTASI, *Reactive local search for the maximum clique problem*, *Algorithmica*, 29 (2001), pp. 610–637.
- [3] I. M. BOMZE, M. BUDINICH, P. M. PARDALOS, AND M. PELILLO, *The maximum clique problem*, in Handbook of Combinatorial Optimization—Suppl. Vol. A, D.-Z. Du and P. M. Pardalos, eds., Kluwer Academic Publishers, 1999, pp. 1–74.
- [4] I. M. BOMZE, M. PELILLO, AND V. STIX, *Approximating the maximum weight clique using replicator dynamics*, *IEEE Trans. Neural Networks*, 11 (2000), pp. 1228–1241.
- [5] M. BROCKINGTON AND J. C. CULBERSON, *Camouflaging independent sets in quasi-random graphs*, in Cliques, Coloring and Satisfiability, D. Johnson and M. A. Trick, eds., vol. 26 of DIMACS Series, AMS, 1996, pp. 75–88.
- [6] S. BURER, R. D. C. MONTEIRO, AND Y. ZHANG, *Maximum stable set formulations and heuristics based on continuous formulations*, *Math. Programm.* (to appear).
- [7] S. BUSYGIN, personal homepage <http://www.busygin.dp.ua/>.
- [8] R. W. COTTLE, J. PANG, AND R. E. STONE, *The Linear Complementarity Problem*, Academic Press, Boston, MA, 1992.
- [9] A. JAGOTA AND L. A. SANCHIS, *Adaptive, restart, randomized greedy heuristics for maximum clique*, *J. Heurist.*, 7 (2001), pp. 565–585.
- [10] D. JOHNSON AND M. A. TRICK, eds., *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*, vol. 26 of DIMACS Series, AMS, 1996.
- [11] A. MASSARO, M. PELILLO, AND I. M. BOMZE, *A complementary pivoting approach to the maximum weight clique problem*, *SIAM J. Optim.*, 12 (2002), pp. 928–948.