A Hybrid Constraint Programming and Semidefinite Programming Approach for the Stable Set Problem*

W.J. van Hoeve

CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
W.J.van.Hoeve@cwi.nl
http://homepages.cwi.nl/~wjvh/

Abstract. This work presents a hybrid approach to solve the maximum stable set problem, using constraint and semidefinite programming. The approach consists of two steps: subproblem generation and subproblem solution. First we rank the variable domain values, based on the solution of a semidefinite relaxation. Using this ranking, we generate the most promising subproblems first, by exploring a search tree using a limited discrepancy strategy. Then the subproblems are being solved using a constraint programming solver. To strengthen the semidefinite relaxation, we propose to infer additional constraints from the discrepancy structure. Computational results show that the semidefinite relaxation is very informative, since solutions of good quality are found in the first subproblems, or optimality is proven immediately.

1 Introduction

This paper describes a hybrid method to solve a classical combinatorial optimization problem, the maximum weighted stable set problem, or *stable set problem*[1] in short. Given a graph with weighted vertices, the stable set problem is to find a subset of vertices of maximum weight, such that no two vertices in this subset are joined by an edge of the graph. In the unweighted case (when all weights are equal to 1), this problem amounts to the maximum cardinality stable set problem, which has been shown to be already NP-hard [24]. Practical applications of the stable set problem are plentiful, they appear in coding theory, computer vision, pattern recognition, and many other areas [4].

We propose a two-phase approach to solve the stable set problem, either with or without proving optimality. The first phase generates subproblems based upon a semidefinite relaxation, the second phase solves the subproblems using constraint programming. Concerning the first phase, given a model for the stable set problem, we solve its semidefinite relaxation. The solution provides us fractional values for the variables of the model. These fractional values are a good

* An earlier version of this paper appeared as [18].

[1] Alternative names for the stable set problem are vertex packing, coclique or independent set problem.

indication for the optimal (discrete) values of the variables. Hence we divide selected variable domains in two parts: a 'good' subdomain and a 'bad' subdomain. By branching on these subdomains using a limited discrepancy strategy [15], we obtain first a very promising subproblem, and subsequently less promising subproblems.

The second phase consists of the solution of the subproblems. Since they are much smaller than the original problem, we can easily solve them using a constraint programming solver.

As computational results will show, the semidefinite relaxation is quite informative. In several cases we can simply round the solution of the relaxation and obtain a provable optimal solution already. Otherwise, we are likely to find a good solution in one of the first subproblems. Using a limited number of subproblems to investigate, we yield an *incomplete* method producing good solutions. In order to obtain a *complete* search strategy, we need, in principle, to generate and solve all possible subproblems. A good upper bound is necessary to prove optimality earlier. For this reason we investigated the use of 'discrepancy cuts' that can be added to the semidefinite program to strengthen the relaxation and thus prune large parts of the search tree. However, computational results will show that they can not be applied efficiently on the instances we considered.

The outline of the paper is as follows. The next section gives a motivation for the approach proposed in this work. Then, in Section 3 some preliminaries on semidefinite programming are given. In Section 4 we introduce the stable set problem, integer optimization formulations and a semidefinite relaxation. A description of our solution framework is given in Section 5. Section 6 presents the computational results. This is followed by an overview of related literature in Section 7. Finally, in Section 8 we conclude and discuss future directions.

2 Motivation

Combinatorial optimization problems that are NP-hard are often solved with the use of a polynomially solvable relaxation. Let us first motivate why in this paper a semidefinite relaxation is used rather than a (more common) linear relaxation. Indeed, one could argue that linear programs are being solved much faster in general. However, for the stable set problem, linear relaxations are not very tight. Therefore one has to identify and add inequalities that strengthen the relaxation. But it is time-consuming to identify such inequalities, and by enlarging the model the solution process may slow down.

Several papers on approximation theory following [11] have shown the tightness of semidefinite relaxations. However, being tighter, semidefinite programs are more time-consuming to solve than linear programs in practice. Hence one has to trade strength for computation time. For large scale applications, semidefinite relaxations can often be preferred as the relaxation of choice to be used in a branch and bound framework. Moreover, our intention is not to solve a relaxation at every node of the search tree. Instead, we propose to only solve a relaxation at the root node of the first phase (its solution is used to identify

the subdomains), and optionally at the root node of a subproblem (in order to strengthen the upper bound). Therefore, we are willing to make the trade-off in favour of the semidefinite relaxation.

Another point of view is the following. Although semidefinite programming has been developing for many years now in the operations research community, no efforts of integration or cooperation with constraint programming have been made to our knowledge. Application of semidefinite programming to problems typical to constraint programming, as was done in the papers on approximation algorithms mentioned in Section 7, is not yet hybrid problem solving. In this paper, however, a first step is being made. The solution of the semidefinite relaxation is used to identify promising subdomains, and also produces a tight upper bound for the constraint programming solver. On the other hand, the solutions found by the constraint programming solver serve as a lower bound inside the semidefinite programming solver.

3 Preliminaries on Semidefinite Programming

In this section we introduce semidefinite programming [28] as an extension of the more common linear programming. Both paradigms can be used to model polynomially solvable relaxations of NP-hard optimization problems.

In linear programming, combinatorial optimization problems are modeled in the following way:

$$\begin{aligned} &\max\ c^{\mathsf{T}} x \\ &\text{s.t. } a_j^{\mathsf{T}} x \leq b_j \ (j = 1, \ldots, m) \\ &\qquad x \geq 0. \end{aligned} \tag{1}$$

Here $x \in \mathbb{R}^n$ is an n-dimensional vector of decision variables and $c \in \mathbb{R}^n$ a cost vector of dimension n. The m vectors $a_j \in \mathbb{R}^n$ $(j = 1, \ldots, m)$ and the m-dimensional vector $b \in \mathbb{R}^m$ define m linear constraints on x. In other words, this approach models problems using *nonnegative vectors* of variables.

Semidefinite programming makes use of *positive semidefinite matrices* of variables instead of nonnegative vectors. A matrix $X \in \mathbb{R}^{n \times n}$ is said to be positive semidefinite (denoted by $X \succeq 0$) when $y^{\mathsf{T}} X y \geq 0$ for all vectors $y \in \mathbb{R}^n$. Semidefinite programs have the form

$$\begin{aligned} &\max\ \mathrm{tr}(CX) \\ &\text{s.t. } \mathrm{tr}(A_j X) \leq b_j \ (j = 1, \ldots, m) \\ &\qquad X \succeq 0. \end{aligned} \tag{2}$$

Here $\mathrm{tr}(X)$ denotes the *trace* of X, which is the sum of its diagonal elements, i.e. $\mathrm{tr}(X) = \sum_{i=1}^{n} X_{ii}$. The cost matrix $C \in \mathbb{R}^{n \times n}$ and the constraint matrices $A_j \in \mathbb{R}^{n \times n}$ are supposed to be symmetric. The m reals b_j and the m matrices A_j define again m constraints.

We can view semidefinite programming as an extension of linear programming. Namely, when the matrices C and A_j $(j = 1, \ldots, m)$ are all supposed to be

diagonal matrices[2], the resulting semidefinite program is equal to a linear program. In particular, then a semidefinite programming constraint $\mathrm{tr}(A_j X) \leq b_j$ corresponds to the linear programming constraint $a_j^\mathsf{T} x \leq b_j$, where a_j represents the diagonal of A_j.

Applied as a continuous relaxation (i.e. the integrality constraint on the variables is relaxed), semidefinite programming in general produces solutions that are much closer to the integral optimum than linear programming. Intuitively, this can be explained as follows. Demanding positive semidefiniteness of a matrix automatically implies nonnegativity of its diagonal. If this diagonal corresponds (as in the general case described above) to the nonnegative vector of the linear relaxation, the semidefinite relaxation is stronger than a linear relaxation. Unfortunately, it is not a trivial task to otain a good (i.e. efficient) semidefinite program for a given problem.

Theoretically, semidefinite programs have been proved to be polynomially solvable using the so-called ellipsoid method (see for instance [12]). In practice, nowadays fast 'interior point' methods are being used for this purpose (see [3] for an overview). Being a special case of semidefinite programming, linear programs are also polynomially solvable using an ellipsoid or interior point method. However, they are often solved with a special linear programming solver, the simplex method. Although this method can have an exponential running time in theory, in practice it is often faster than an interior point algorithm.

4 The Stable Set Problem

In this section, the stable set problem is formally defined, and formulated in two different ways as an integer optimization problem. From this, a semidefinite programming relaxation is inferred.

4.1 Definition

Consider an undirected weighted graph $G = (V, E)$, where $V = \{1, \ldots, n\}$ is the set of vertices and E a subset of edges $\{(i, j) | i, j \in V, i \neq j\}$ of G, with $|E| = m$. To each vertex $i \in V$ a weight $w_i \in \mathbb{R}$ is assigned (without loss of generality, we can assume all weights to be nonnegative in this case). A *stable set* is a set $S \subseteq V$ such that no two vertices in S are joined by an edge in E. The *stable set problem* is the problem of finding a stable set of maximum total weight in G. This value is called the *stable set number* of G and is denoted by $\alpha(G)$[3]. The maximum cardinality (or unweighted) stable set problem can be obtained by taking all weights equal to 1.

[2] A diagonal matrix is a matrix with nonnegative values on its diagonal entries only.

[3] In the literature $\alpha(G)$ usually denotes the unweighted stable set number. The weighted stable set number is then denoted as $\alpha_w(G)$. In this work, it is not necessary to make this distinction.

4.2 Integer Optimization Formulation

Let us first consider an integer linear programming formulation. We introduce binary variables to indicate whether or not a vertex belongs to the stable set S. So, for n vertices, we have n integer variables x_i indexed by $i \in V$, with initial domains $\{0,1\}$. In this way, $x_i = 1$ if vertex i is in the stable set S, and $x_i = 0$ otherwise. We can now state the objective function, being the sum of the weights of vertices that are in the stable set S, as $\sum_{i=1}^{n} w_i x_i$. Finally, we define the constraints that restrict two adjacent vertices to be both inside S as $x_i + x_j \leq 1$, for all edges $(i,j) \in E$. Hence the integer linear programming model becomes:

$$\begin{aligned} \alpha(G) = \max \ & \textstyle\sum_{i=1}^{n} w_i x_i \\ \text{s.t. } & x_i + x_j \leq 1 \ \forall (i,j) \in E \\ & x_i \in \{0,1\} \ \forall i \in V. \end{aligned} \tag{3}$$

Another way of describing the same solution set is presented by the following integer quadratic program

$$\begin{aligned} \alpha(G) = \max & \textstyle\sum_{i=1}^{n} w_i x_i \\ \text{s.t. } & x_i x_j = 0 \ \forall (i,j) \in E \\ & x_i^2 = x_i \ \forall i \in V. \end{aligned} \tag{4}$$

Note that here the constraint $x_i \in \{0,1\}$ is replaced by $x_i^2 = x_i$. This quadratic formulation will be used below to infer a semidefinite programming relaxation of the stable set problem.

In fact, both model (3) and model (4) can be used as a constraint programming model. We have chosen the first model, since the quadratic constraints take more time to propagate than the linear constraints, while having the same pruning power.

4.3 Semidefinite Programming Relaxation

The integer quadratic program (4) gives rise to a semidefinite relaxation introduced by Lovász [22] (see Grötschel et al. [12] for a comprehensive treatment). The value of the objective function of this relaxation has been named the *theta number* of a graph G, indicated by $\vartheta(G)$. Let us start again from model (4). As was indicated in Section 3, we want to transform the current model that uses a nonzero vector into a model that uses a positive semidefinite matrix to represent our variables. In the current case, we can construct a matrix $X \in \mathbb{R}^{n \times n}$ by defining $X_{ij} = x_i x_j$. Let us also construct a $n \times n$ cost matrix W with $W_{ii} = w_i$ for $i \in V$ and $W_{ij} = 0$ for all $i \neq j$. Since $X_{ii} = x_i^2 = x_i$, the objective function becomes $\mathrm{tr}(WX)$. The edge constraints are easily transformed as $x_i x_j = 0 \Leftrightarrow X_{ij} = 0$. The first step in the transformation of model (4) can now be made:

$$\begin{aligned} & \max \mathrm{tr}(WX) \\ & \text{s.t. } X_{ij} = 0 \quad \forall (i,j) \in E \\ & \qquad X_{ij} = x_i x_j \ \forall i,j \in V \\ & \qquad x_i^2 = x_i \quad \forall i \in V. \end{aligned} \tag{5}$$

This model is still a quadratic program, although reformulated. The problem remains how to model the last, very important, constraint. We need a mapping of the diagonal entries $X_{ii} = x_i^2$ to the vector entries x_i. For this reason, we extend X with another row and column (both indexed by 0) that contain vector x, and define the $(n+1) \times (n+1)$ matrix Y as

$$Y = \begin{pmatrix} 1 & x_1 & \cdots & x_n \\ x_1 & & & \\ \vdots & & X & \\ x_n & & & \end{pmatrix}$$

where the 1 in the leftmost corner of Y is needed to obtain positive semidefiniteness. In this case we can express the required mapping as $Y_{ii} = \frac{1}{2}Y_{i0} + \frac{1}{2}Y_{0i}$ (note that X and Y are symmetric), since then $x_i^2 = Y_{ii} = \frac{1}{2}Y_{i0} + \frac{1}{2}Y_{0i} = x_i$. The final step in the transformation consists of replacing the constraints on X by constraints on Y. In particular, instead of demanding X to be a product of nonnegative vectors, we restrict Y to be a positive semidefinite matrix. Namely, if the vector x represents a stable set, then the matrix Y is positive semidefinite. However, not all positive semidefinite Y matrices represent a stable set, in particular its values can take fractional values.

In order to maintain equal dimension to Y, a row and a column (both indexed by 0) should be added to W, all entries of which containing value 0. Denote the resulting matrix by $\tilde{W}$. The theta number of a graph G can now be described as

$$\begin{aligned} \vartheta(G) = \max \quad & \operatorname{tr}(\tilde{\mathrm{W}}\mathrm{Y}) \\ \text{s.t. } Y_{ii} &= \tfrac{1}{2}Y_{i0} + \tfrac{1}{2}Y_{0i} \quad \forall i \in V \\ Y_{ij} &= 0 \qquad\quad \forall (i,j) \in E \\ Y &\succeq 0. \end{aligned} \tag{6}$$

By construction, the diagonal value Y_{ii} serves as an indication for the value of variable x_i ($i \in V$) in a maximum stable set. In particular, this program is a relaxation for the stable set problem, i.e. $\vartheta(G) \geq \alpha(G)$. Note that program (6) can easily be rewritten into the general form of program (2). Namely, $Y_{ii} = \frac{1}{2}Y_{i0} + \frac{1}{2}Y_{0i}$ is equal to $\operatorname{tr}(AY)$ where the $(n+1) \times (n+1)$ matrix A consists of all zeroes, except for $A_{ii} = 1$, $A_{i0} = -\frac{1}{2}$ and $A_{0i} = -\frac{1}{2}$, which makes the corresponding b entry equal to 0. Similarly for the edge constraints.

The theta number also arises from other formulations, different from the above, see [12]. In our implementation we have used the formulation that has been shown to be computationally most efficient among those alternatives [13]. Let us introduce that particular formulation (called ϑ_3 in [12]). Again, let $x \in \{0,1\}^n$ be a vector of binary variables representing a stable set. Define the $n \times n$ matrix $X = \xi\xi^{\mathsf{T}}$ where $\xi_i = \frac{\sqrt{w_i}}{\sqrt{\sum_{j=1}^{n} w_j x_j}} x_i$. Furthermore, let the $n \times n$ cost matrix U be defined as $U_{ij} = \sqrt{w_i w_j}$ for $i, j \in V$. Observe that in these definitions we

exploit the fact that $w_i \geq 0$ for all $i \in V$. The following semidefinite program

$$\begin{aligned} \vartheta(G) = \max \quad & \operatorname{tr}(UX) \\ \text{s.t. } & \operatorname{tr}(X) = 1 \\ & X_{ij} = 0 \; \forall (i,j) \in E \\ & X \succeq 0 \end{aligned} \tag{7}$$

gives exactly the theta number of G. When (7) is solved to optimality, the scaled diagonal element $\vartheta(G)X_{ii}$ (a fractional value between 0 and 1) serves as an indication for the value of x_i ($i \in V$) in a maximum stable set (see for instance [13]). Again, it is not difficult to rewrite program (7) into the general form of program (2).

Program (7) uses matrices of dimension n and $m+1$ constraints, while program (6) uses matrices of dimension $n+1$ and $m+n$ constraints. This gives an indication why program (7) is computationally more efficient.

5 Solution Framework

5.1 Overview

The two-phase solution approach proposed here is similar to the one described in [23]. In the first phase subproblems are generated, which are being solved in the second phase. A subproblem consists of a constraint programming model (program (3)) on restricted variable domains. The restricted domain values are selected by a heuristic, in our case the solution to the semidefinite programming relaxation. Each subproblem is solved to optimality using a constraint programming solver. A general overview of the method is presented in Algorithm 1 and explained hereafter.

Let us first explain how we use the solution of the semidefinite program (7) to partition the domain D_i of a variable x_i into D_i^{good} and D_i^{bad} (for $i \in V$). As was stated before, the solution of program (7) assigns fractional values between 0 and 1 to its variables. Naturally, if for a variable x_i the corresponding fractional value $\vartheta(G)X_{ii}$ is close to 1, we regard 1 to be a good value for variable x_i. More specifically, we select the variable x_i with the highest corresponding fractional value $\vartheta(G)X_{ii}$, set $D_i^{\text{good}} = \{1\}$ and $D_i^{\text{bad}} = \{0\}$, and mark it as handled. Then we mark all its neighbours j (with $(i,j) \in E$) as being handled, keeping their original domain $D_j = \{0,1\}$. This procedure is repeated until all variables are handled. For later convenience, we partition V into two distinct sets $V_0 = \{i \in V \mid D_i^{\text{good}} = \{1\}\}$ and $V_1 = V \setminus V_0$. Here V_1 represents the set of neighbours j of V_0, with $D_j = \{0,1\}$.

In a similar way, we can use the solution of the semidefinite relaxation to compute a first feasible integer solution. Namely, follow the same procedure, but now instantiate the selected variable $x_i = 1$ and set its neighbours $x_j = 0$. The objective value of this feasible integer solution is in many cases already equal to the (in case of integer weights downward rounded) solution of the semidefinite relaxation. In that case, we have found an optimal solution and finish. In other

Algorithm 1 Solution framework

```
read problem
set maximum discrepancy
solve semidefinite program (7) → upper bound
round solution of (7) → lower bound
for i ∈ V_0 do
  define D_i^good and D_i^bad using solution of (7)
end for
set discrepancy = 0
while lower bound < upper bound and discrepancy ≤ maximum discrepancy do
  generate subproblem using LDS branching strategy on D_i^good and D_i^bad
  solve subproblem → lower bound
  discrepancy = discrepancy + 1
end while
```

cases, we can still use this first solution as a lower bound to be applied during the solution of the subproblem.

Next, we explain how to generate subproblems using these subdomains. The generation of subproblems makes use of a tree structure of depth $|V_0|$ in which we branch on D_i^{good} versus D_i^{bad}. The tree is traversed using a limited discrepancy strategy (LDS) [15]. LDS visits the nodes of a search tree differently from depth-first search. It tries to follow a given suggestion as good as it can. Branches opposite to the suggestion are regarded as discrepancies and are gradually allowed to be traversed. The first 'run' of LDS doesn't allow any discrepancies, the second allows only one, and so on. This means that for a particular discrepancy k, a path from the root to a leaf is allowed to consist of maximally k right branches. Typically this method is applied until a limited number of discrepancies is reached (say 2 or 3), which yields an incomplete search strategy. In order to be complete, one has to visit all nodes, up to discrepancy d for a binary tree of depth d.

In our case, the suggestion that should be followed are branches of the kind D_i^{good}, while branches D_i^{bad} are regarded as discrepancies. Hence, our first subproblem is the subproblem defined by program (3), with $x_i \in D_i^{\text{good}}$ for all $i \in V_0$. The next $|V_0|$ subproblems have all $x_i \in D_i^{\text{good}}$, except for one $x_k \in D_k^{\text{bad}}$ $(i, k \in V_0)$. The next discrepancy generates $\frac{1}{2}(|V_0|^2 + |V_0|)$ subproblems, each of which contains two variables $x_{k_1} \in D_{k_1}^{\text{bad}}$ and $x_{k_2} \in D_{k_2}^{\text{bad}}$ $(k_1, k_2 \in V_0)$, and so on. Since we expect to obtain a very good solution already in the first subproblems, we will only generate subproblems up to a certain maximum discrepancy. In our experiments the maximum discrepancy is chosen 2 and 4 respectively. Finally, we solve all subproblems to optimality using a constraint programming solver.

Note that the first subproblem, corresponding to discrepancy 0, only contains one solution, namely the one that we obtain in our rounding procedure. By propagation of the edge constraints, all variables $x_j \in D_j = \{0, 1\}$ $(j \in V_1)$

are instantiated automatically to 0. Hence, the subproblem corresponding to discrepancy 0 is obsolete in our current implementation.

5.2 Adding Discrepancy Cuts

In the case one needs to generate and solve subproblems up to a large discrepancy, it is preferable to prove possible suboptimality of a subproblem before entering it, especially when the subproblems are still relatively large. This can be done in several ways.

First, before entering a subproblem, we can identify variables which have a subdomain of size 1, namely those with $i \in V_0$. For those variables, one can add an additional constraint to the semidefinite program (7), enforcing either $x_i = 1$ or $x_i = 0$. Then the semidefinite program can be solved again, and will in general provide a tighter bound, hopefully lower than the current lowerbound, in which case we have proven suboptimality. However, solving the semidefinite program each time before entering a subproblem is very time-consuming and this method will not be very practical.

A better alternative would be to add a specific constraint, a *discrepancy cut*, that is valid for all subproblems of a given discrepancy. Recall that $V_0 = \{i \in V \mid D_i^{\text{good}} = \{1\}\}$. Hence, all subproblems of discrepancy k consist of k variables x_i with $x_i = 0$, and $|V_0| - k$ variables x_i with $x_i = 1$ ($i \in V_0$). This gives rise to two discrepancy cuts, given discrepancy k:

$$\sum_{i \in V_0} x_i = |V_0| - k \tag{8}$$

$$\sum_{i \in V_0} 1 - x_i = k \tag{9}$$

We implemented both of them, and cut (9) gives the best results. Stated in terms of semidefinite program (6), the discrepancy cut looks like $\text{tr}(AY) = |V_0| - k$, with $A_{ii} = 1$ if $i \in V_0$ and $A_{ij} = 0$ otherwise ($i, j \in \{0, \ldots, n\}$). As mentioned before, solving a semidefinite relaxation is relatively expensive, and one should make a tradeoff between its computation time and the gain in time of not solving the subproblems. For the instances we considered, the time needed to solve a semidefinite program is always larger than the time needed to solve all subproblems we would like to proof suboptimal. However, these cuts might be helpful for larger instances.

6 Computational Results

Our experiments are being done on a Pentium 1GHz processor, with 256 Mb RAM. As constraint programming solver we use the ILOG solver library, version 5.1 [19]. As semidefinite programming solver, we use CSDP version 4.1 [5]. The reason for our choices is that both solvers are among the fastest in their field,

and because ILOG solver is written in C++, and CSDP is written in C, they can be easily hooked together.

The first instances we consider are randomly generated weighted graphs with n vertices and m edges. The vertex weights are randomly chosen integers from a range of 1 up to n. The edge density is chosen such that the constraint programming solver has difficulties solving them. Namely, the more edge constraints we have, the more propagation can be performed, and the easier the instance is solved by constraint programming. On the other hand, more edge constraints will slow down the semidefinite programming solver, because it is highly sensitive to the size of the semidefinite program to solve.

The name of the instances represent the number of vertices and the edge density, i.e. `g75d015` is a graph on 75 vertices with an approximate edge density of 0.15. For these graphs, we have chosen to generate subproblems up to a maximum discrepancy of 4, based upon earlier experience.

We also considered structured instances (`1tc.64` up to `1et.256`), obtained from problems arising in coding theory [27]. These are unweighted graphs, therefore we have set all weights equal to 1. For these graphs, we generate subproblems up to a discrepancy of 2.

The results of our experiments are given in Table 1. It consists of three parts: the first part describes the instances, the next part gives the results of our approach (*sdp and cp*), the last part concerns the results of a sole constraint programming approach (*cp alone*) applied to program (3).

The columns in this table represent the following. An instance *name* has n vertices and m edges. For the part on our approach, the value of the semidefinite relaxation is ϑ, the rounded solution of the semidefinite relaxation has value *round*, and *best* is the value of the best solution found. This best solution is found in a subproblem generated during discrepancy *best discr*. Note that we generate subproblems up to discrepancy 4 in all cases, as was mentioned in Section 5. The time spent on solving the semidefinite relaxation is denoted by *time sdp*. The time spent on solving all generated subproblems is denoted by *time subp*. These values together form the *total time*. All times are measured in seconds. The number of all backtracking steps made during the search in our approach is collected in *backtracks*. Concerning the sole constraint programming approach, we report the best solution found (*best*), the *total time* spent during search, and the total number of *backtracks*. Note that we have set time limits for the constraint programming solver, to create a fair comparison with our approach. They are 100 seconds for `g50d005` up to `g150d010`, 190 seconds for `g150d015` and 324 seconds for the structured instances. Best found solutions that are proven to be optimal are indicated by an asterisk (*).

For the instances in Table 1 we only solved one semidefinite relaxation per problem, namely at the root node. The reason for this is that the time spent during the subproblem search is less than the time spent on computing another relaxation, as reported in the table. Therefore, we cannot gain time by adding discrepancy cuts and computing another semidefinite relaxation.

Table 1. Computational results on randomly generated weighted graphs and structured unweighted graphs. Best found solutions that are proven optimal are marked with an asterisk (*).

instance			sdp and cp								cp alone		
name	n	m	ϑ	round	best	best discr	time sdp	time subp	total time	back-tracks	best	total time	back-tracks
g50d005	50	69	746.00	746	746*	0	0.48	0.00	0.48	0	746*	6.38	160185
g50d010	50	130	568.00	568	568*	0	0.53	0.00	0.53	0	568*	1.54	40878
g50d015	50	191	512.00	512	512*	0	0.71	0.00	0.71	0	512*	0.54	13355
g75d005	75	139	1472.17	1455	1466	1	0.99	3.47	4.46	36492	1077	100.00	2555879
g75d010	75	280	1148.25	1122	1134	3	2.05	1.01	3.06	12964	1074	100.00	2299226
g75d015	75	414	966.76	924	946	4	3.50	0.88	4.38	11136	951*	46.63	1000409
g100d005	100	250	2903.00	2903	2903*	0	3.13	0.00	3.13	0	2415	100.01	1578719
g100d010	100	495	2058.41	1972	2029	4	6.92	7.17	14.09	74252	1850	100.02	1666892
g100d015	100	725	1704.61	1568	1608	4	17.87	4.28	22.15	47222	1644	100.01	1469221
g125d005	125	367	3454.00	3454	3454*	0	7.10	0.00	7.10	0	2656	100.02	1644993
g125d010	125	761	2448.94	2208	2271	4	22.00	13.98	35.98	107610	1668	100.01	1643183
g125d015	125	1110	2033.74	1839	1846	4	59.56	6.15	65.71	53103	1733	100.02	1454377
g150d005	150	549	5043.09	5035	5035	0	15.79	26.89	42.68	135051	3090	100.02	1426234
g150d010	150	1094	3651.38	3281	3423	2	64.83	13.10	77.93	78263	2294	100.02	1462125
g150d015	150	1641	2935.84	2572	2572	0	181.54	4.56	186.10	27306	2224	190.03	2119929
1tc.64	64	192	20.00	20	20*	0	1.05	0.00	1.05	0	19	324.01	10969580
1et.64	64	264	18.85	18	18*	0	0.97	0.00	0.97	0	18*	179.00	5763552
1tc.128	128	512	38.00	38	38*	0	12.03	0.00	12.03	0	19	324.02	9455475
1et.128	128	672	29.33	28	28	0	13.48	0.67	14.15	1929	19	324.02	8562959
1dc.128	128	1471	16.89	16	16*	0	107.71	0.00	107.71	0	13	324.04	6497027
1zc.128	128	2240	20.84	16	18	2	323.53	0.41	323.94	2094	18	324.04	7584769
1tc.256	256	1312	63.43	60	62	2	129.82	8.63	138.45	11588	13	324.03	7284451
1et.256	256	1664	55.14	50	50	0	230.39	5.02	235.41	7184	18	324.04	7656162

In general, our method produces better solutions than the constraint programming approach alone. In many cases, the rounded solution of the semidefinite relaxation is already optimal. However, note that there are two instances for which the constraint programming approach gives better solutions. Note also that the structured instances are handled quite well by our approach, while the constraint programming approach produces very low quality solutions for the larger instances.

A final remark concerns instance `1et.256`. In [27], the maximum value of this instance is 48, while we find a solution with value 50. After notification, the author of [27] agreed with our solution.

7 Related Literature

Since the stable set problem is NP-hard, no complete (or exact) algorithm is known that solves the stable set problem in polynomial time. Many other techniques have been proposed, including approximation algorithms, heuristics, or branch and bound structured methods. A survey of different formulations, complete methods and heuristics for the maximum clique problem[4] is given by Pardalos and Xue [25] and, more recently, by Bomze et al. [4]. The maximum clique problem has also been succesfully attacked using constraint programming, by Fahle [7] and Régin [26]. Both papers make use of specialized propagation algorithms for the maximum clique problem.

Although semidefinite programs can be solved in polynomial time theoretically, it lasted until a few years ago until fast solvers for this purpose were implemented. Until then, application inside a branch and bound framework was unrealistic. Still, solving a semidefinite program takes relatively much time, compared to solving a linear program. However, since semidefinite programming solvers are getting faster, semidefinite relaxations become a serious candidate to be used within a branch and bound framework, see for instance the paper by Karisch et al. [20].

A large number of references to papers concerning semidefinite programming are on the web pages of Helmberg [16] and Alizadeh [2]. A general introduction on semidefinite programming applied to combinatorial optimization is given by Goemans and Rendl [10].

Another area that made semidefinite programming useful in practice is that of approximation algorithms. In this field one tries to give a performance guarantee for an algorithm on a particular problem. In particular, the paper [11] by Goemans and Williamson uses a semidefinite relaxation and randomized rounding to prove such a performance guarantee for the maximum cut problem of a graph and satisfiability problems. Following this result numerous papers appeared, also concerning the approximation of satisfiability problems, including [14] and [21].

[4] The maximum weighted stable set problem of a graph is equivalent to the maximum weighted clique problem of its complement graph.

The solution structure of the current work, namely problem decomposition by branching on promising subdomains, is similar to the method described in [23], which is also present in [8]. In [23] a linear relaxation is used to identify promising values. Moreover, by exploiting the discrepancy structure of the method combined with reduced costs, suboptimality of subproblems can be proved very fast.

Another hybrid approach, using linear programming and constraint programming, has been investigated by Ajili et al. [1] and El Sakkout et al. [6]. A subset of constraints is relaxed as a linear program in such a way that its solution is always integral. The solution to the relaxation serves as a suggestion (a 'probe') for solving the complete program using a constraint programming solver. A probe is used to detect infeasibility, to remove inconsistent domain values and to guide the search. During search, many probing steps are being made. This results in a tight cooperation of the linear programming and constraint programming solver.

8 Conclusions and Further Research

We introduced a method that combines semidefinite programming and constraint programming to solve the stable set problem. Our experiments show that constraint programming can indeed benefit greatly from semidefinite programming. On instances that were very difficult to handle for a constraint programming solver, our hybrid method obtained very good results.

The discrepancy cuts we proposed to strengthen the semidefinite relaxation could not be applied efficiently to the instances we considered. However, for larger instances they could be helpful.

Further research in this direction would for instance be to obtain a filtering mechanism similar to the cost-based domain filtering for linear relaxations [9]. In [17], Helmberg describes such a procedure, called variable fixing, for semidefinite relaxations. It would be interesting to see how his method can be applied in a constraint programming framework.

Also, one could consider a different way of selecting promising values from the solution of the semidefinite relaxation. A strategy that incorporates randomized rounding possibly yields better results. This thought is motivated by the use of randomized rounding of semidefinite relaxations in approximation algorithms, as discussed in Section 7.

Finally, this work has much in common with our previous work [23]. The underlying general principle of decomposing a problem into promising subproblems according to a certain heuristic is currently under research.

Acknowledgements

Many thanks to Michela Milano for fruitful discussion and helpful comments while writing this paper. Also thanks to Monique Laurent for constructive remarks.

References

1. F. Ajili and H. El Sakkout. LP probing for piecewise linear optimization in scheduling. In *Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'01)*, pages 189–203, 2001.
2. F. Alizadeh. The Semidefinite Programming Page. `http://new-rutcor.rutgers.edu/~alizadeh/sdp.html`.
3. F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.
4. I.M. Bomze, M. Budinich, P.M. Pardalos, and M. Pelillo. The Maximum Clique Problem. In D.-Z. Du and P.M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 4. Kluwer Academic Publishers, Boston, MA, 1999.
5. B. Borchers. A C Library for Semidefinite Programming. *Optimization Methods and Software*, 11(1):613–623, 1999.
6. H. El Sakkout and M. Wallace. Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling. *Constraints*, 5(4):359–388, 2000.
7. T. Fahle. Simple and Fast: Improving a Branch-And-Bound Algorithm for Maximum Clique. In *10th Annual European Symposium on Algorithms (ESA 2002)*, volume 2461 of *LNCS*, pages 485–498. Springer Verlag, 2002.
8. F. Focacci. *Solving Combinatorial Optimization Problems in Constraint Programming*. PhD thesis, University of Ferrara, 2001.
9. F. Focacci, A. Lodi, and M. Milano. Cost-based domain filtering. In J. Jaffar, editor, *Fifth International Conference on the Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *LNCS*, pages 189–203. Springer Verlag, 1999.
10. M. Goemans and F. Rendl. Combinatorial Optimization. In H. Wolkowicz, R. Saigal, and L. Vandenberghe, editors, *Handbook of Semidefinite Programming*, pages 343–360. Kluwer, Dordrecht, 2000.
11. M.X. Goemans and D.P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *Journal of the ACM*, 42(6):1115–1145, 1995.
12. M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. John Wiley & Sons, New York, 1988.
13. G. Gruber and F. Rendl. Computational experience with stable set relaxations. *SIAM Journal on Optimization*, 13(4):1014–1028, 2003.
14. E. Halperin and U. Zwick. Approximation algorithms for MAX 4-SAT and rounding procedures for semidefinite programs. *Journal of Algorithms*, 40:184–211, 2001.
15. W. D. Harvey and M. L. Ginsberg. Limited Discrepancy Search. In C. S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95); Vol. 1*, pages 607–615, 1995.
16. C. Helmberg. Semidefinite Programming website. `http://www-user.tu-chemnitz.de/~helmberg/semidef.html`.
17. C. Helmberg. Fixing variables in semidefinite relaxations. *SIAM Journal on Matrix Analysis and Applications*, 21(3):952–969, 2000.
18. W.J. van Hoeve. A hybrid constraint programming and semidefinite programming approach for the stable set problem. In *Fifth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'03)*, pages 3–16, 2003.
19. ILOG. ILOG Solver 5.1, Reference Manual, 2001.

20. S. E. Karisch, F. Rendl, and J. Clausen. Solving graph bisection problems with semidefinite programming. *INFORMS Journal on Computing*, 12(3):177–191, 2000.
21. H. C. Lau. A new approach for weighted constraint satisfaction. *Constraints*, 7(2):151–165, 2002.
22. L. Lovász. On the Shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25:1–7, 1979.
23. M. Milano and W.J. van Hoeve. Reduced cost-based ranking for generating promising subproblems. In P. van Hentenryck, editor, *Eighth International Conference on the Principles and Practice of Constraint Programming (CP'02)*, volume 2470 of *LNCS*, pages 1–16. Springer Verlag, 2002.
24. C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, 1982.
25. P.M. Pardalos and J. Xue. The Maximum Clique Problem. *SIAM Journal of Global Optimization*, 4:301–328, 1994.
26. J.-C. Régin. Solving the Maximum Clique Problem with Constraint Programming. In *Fifth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'03)*, pages 166–179, 2003.
27. N.J.A. Sloane. Challenge Problems: Independent Sets in Graphs. `http://www.research.att.com/~njas/doc/graphs.html`.
28. H. Wolkowicz, R. Saigal, and L. Vandenberghe, editors. *Handbook of Semidefinite Programming*, volume 27 of *International series in operations research and management science*. Kluwer, Dordrecht, 2000.