Efficient Algorithms for Integer Programs with Two Variables per Constraint*

Reuven Bar-Yehuda and Dror Rawitz
Computer Science Department
Technion - IIT, Haifa 32000 Israel
e-mail: {`reuven,rawitz`}`@cs.technion.ac.il`
`http://www.cs.technion.ac.il/`$\sim$ {`reuven,rawitz`}

July 10, 1999

Abstract

Given a bounded integer program with n variables and m constraints, each with two variables, we present an $O(mU)$ time and $O(m)$ space feasibility algorithm for such integer programs, where U is the maximal variable range size. We show that with the same complexity we can find an optimal solution for the positively-weighted minimization problem for *monotone* systems. Using the local-ratio technique we develop an $O(nmU)$ time and $O(m)$ space 2-approximation algorithm for the positively-weighted minimization problem for the general case. We further generalize all results to nonlinear constraints (called *axis-convex constraints*) and to nonlinear (but monotone) weight functions.

Our algorithms are not only better in complexity than other known algorithms, but also considerably simpler, and they contribute to the understanding of these very fundamental problems.

Key words: Combinatorial Optimization, Integer Programming, Approximation Algorithm, Local Ratio Technique, 2SAT, Vertex Cover.

*A preliminary version of this paper appeared in the *Proceedings of the 7th Annual European Symposium on Algorithms* (1999).

1 Introduction

This paper is motivated by a paper of Hochbaum, Megiddo, Naor and Tamir [11] which discusses integer programs with two variables per constraint. The problem is defined as follows:

$$\begin{array}{lll} \text{(2VIP)} & \min \sum_{i=1}^{n} w_i x_i & \\ & \text{s.t.} \quad a_k x_{i_k} + b_k x_{j_k} \geq c_k & \forall k \in \{1,\ldots,m\} \\ & \qquad \ell_i \leq x_i \leq u_i & \forall i \in \{1,\ldots,n\} \\ & \qquad x_i \in I\!N & \forall i \in \{1,\ldots,n\} \end{array}$$

where $1 \leq i_k, j_k \leq n$, $w_i \geq 0$, $a, b, c \in \mathbb{Z}^m$ and $\ell, u \in I\!N^n$.

Obviously, this problem is a generalization of the well known *minimum weight vertex cover problem* (VC) and the *minimum weight* 2 *satisfiability problem* (2SAT). Both problems are known to be NP-hard [7], and the best known approximation ratio for VC [3, 10, 14] and 2SAT [9] is assimptotically 2. Both results are best viewed via the local ratio technique (see [2, 4]).

A 2VIP system is called *monotone* if each constraint is an inequality on two variables with coefficients of opposite signs. The problem of checking whether an integer monotone system has a feasible solution was shown to be NP-complete by Lagarias [13]. Therefore, even for the 2VIP feasibility problem, it is natural to consider pseudo-polynomial algorithms, i.e., algorithms with running time which is polynomial in the size of the input and in U, where $U = \max_i \{u_i - \ell_i\}$.

Hochbaum and Naor [12] were the first to consider efficient algorithms for integer programs with two variables per inequality. They presented an $O(mn^2 \log m + nmU)$ time feasibility algorithm for monotone 2VIP systems, and an optimization algorithm for monotone systems with general (possibly negative) weights. This optimization algorithm constructs a graph representing the monotone system in question and then uses a maximum flow algorithm. Consequently the time complexity of their algorithm is relatively high, i.e. when using Goldberg and Tarjan's maximum-flow algorithm [8] it is $O(nmU^2 \log(\frac{n^2 U}{m}))$. Using this optimization algorithm, Hochbaum, Megiddo, Naor and Tamir [11] were able to construct an $O(nmU^2 \log(\frac{n^2 U}{m}))$ time 2-approximation algorithm for the 2VIP problem. Their algorithm also uses an $O(mU)$ time and space 2VIP feasibility algorithm based on a transformation to 2SAT which they attribute to Feder.

By using the local-ratio technique, we present an $O(nmU)$ time and $O(m)$ space 2-approximation algorithm. This algorithm is not only more efficient, but also more natural and simpler. In order to develop an approximation algorithm, it seems natural to first study the feasibility problem[1]. Indeed, our 2-approximation algorithm is in fact a specific implementation of a feasibility algorithm presented here.

The remainder of this paper is organized as follows: In Section 2 we present an $O(mU)$ time and $O(m)$ space feasibility algorithm for 2VIP systems. The 2-approximation algorithm for 2VIP systems is presented in Section 3. In Section 4 we show that the feasibility algorithm and the approximation algorithm presented in this paper can be generalized to some nonlinear systems with the same time and space complexity. We define a generalization of linear inequalities, called *axis-convex* constraints, and show that the algorithms can be generalized to work with such constraints. We also generalize the 2-approximation algorithm to objective functions of the form $\sum_{i=1}^{n} w_i(x_i)$, where all the w_i's are *monotone weight*

[1] This was done by Hochbaum and Naor [12] for the monotone 2VIP problem, by Hochbaum et al. [11] for the 2VIP problem, and by Gusfield and Pitt [9] for the 2SAT problem.

Table 1: Summary of Results

Problem	Previous results (time, space)	Our results (time, space)
2SAT Feasibility	$O(m), O(m)$ Even, Itai and Shamir [6]	
2VIP Feasibility	$O(mU), O(mU)$ by using reduction to 2SAT [11]	$O(mU), O(m)$
2SAT 2-approximation	$O(nm), O(n^2+m)$ Gusfield and Pitt [9]	$O(nm), O(m)$
2VIP 2-approximation	$O(nmU^2 \log(n^2U/m)), O(mU)$ Hochbaum, Megiddo, Naor and Tamir [11]	$O(nmU), O(m)$
Monotone 2VIP Optimization	$O(mU), O(mU)$ by using reduction to 2SAT	$O(mU), O(m)$

functions. An optimality algorithm for *monotone* linear systems appears in Section 5. We show that this algorithm can work with some non-linear constraints, and we generalize the algorithm to monotone weight functions as well.

Table 1 summarizes the results for 2VIP systems.

2 Feasibility Algorithm

Given a 2VIP system, we are interested in developing an algorithm which finds a feasible solution, if such a solution exists. Since the special case where $\ell = 0^n$ and $u = 1^n$ is the known 2SAT feasibility problem, it is natural to try to extend the well known $O(m)$ time and space algorithm of Even et al. [6]. It is possible to transform the given 2VIP system into an equivalent 2SAT instance with nU variables and $(m+n)U$ constraints (this transformation due to Feder appears in [11]). By combining this transformation with the linear time and space algorithm of Even et al. we get an $O(mU)$ time and space feasibility algorithm. In this section we present an $O(mU)$ time and $O(m)$ space feasibility algorithm which generalizes the algorithm by Even et al.

The main idea of the algorithm from [6] is as follows: choose a variable x_t, first discover the forced values of other variables by assigning $x_t = 0$, and then do the same for the assignment $x_t = 1$. If one of these assignments does not lead to a contradiction, assign this value to x_t and make the corresponding forced assignments. The correctness of the approach of Even et al. is shown by proving that a non-contradictory assignment preserves the feasibility property. The efficiency of their algorithm is achieved by discovering the forced assignments of $x_t = 0$ and those of $x_t = 1$ in parallel.

Our 2VIP feasibility algorithm works with bounds, as opposed to assignments. We choose a variable x_t and an integer $\alpha \in [\ell_t, u_t]$ and discover the forced bounds of other variables by considering in turn the bound $x_t \leq \alpha$ and the bound $x_t > \alpha$. For this purpose we use constraint propagation[2]. Much like Even et al. [6], we prove that a non-contradictory bound preserves the feasibility property, and use this to show the correctness of our algo-

[2] Constraint propagation was previously used for the LP version of the problem (e.g., see [1, 15]), and in [12] for integer feasibility over monotone inequalities.

Routine OneOnOneImpact(ℓ, u, i, j, k)

Let constraint k be $ax_i + bx_j \geq c$.
If $b > 0$ then
  If $a > 0$ then $\ell'_j \leftarrow \lceil \frac{c-au_i}{b} \rceil$
  Else $\ell'_j \leftarrow \left\lceil \frac{c-a\ell_i}{b} \right\rceil$
  $\ell_j \leftarrow \max\left\{\ell_j, \ell'_j\right\}$
Else
  If $a > 0$ then $u'_j \leftarrow \left\lfloor \frac{c-a\ell_i}{b} \right\rfloor$
  Else $u'_j \leftarrow \lfloor \frac{c-au_i}{b} \rfloor$
  $u_j \leftarrow \min\left\{u_j, u'_j\right\}$

Figure 1: Routine *OneOnOneImpact*.

rithm. Also, the efficiency of our algorithm is achieved by discovering the forced bounds by both new bounds of x_t in parallel.

The purpose of this section is not only to show the factor $\Omega(U)$ improvement in space complexity, but also to lay the foundations for the 2-approximation algorithm presented in the next section.

Definition 1: For a given 2VIP instance

$$\mathrm{sat}(\ell, u) = \{x : \ell \leq x \leq u \text{ and } x \text{ satisfies all 2VIP constraints}\} \quad .$$

Definition 2: For the k'th constraint (on the variables x_{i_k}, x_{j_k}) of a given 2VIP instance

$$\mathrm{constraint}(k) = \{(\alpha, \beta) : x_{i_k} = \alpha, x_{j_k} = \beta \text{ satisfy constraint } k\} \quad .$$

Definition 3: Given $\alpha, \beta \in \mathbb{Z}$ we define $[\alpha, \beta] = \{z \in \mathbb{Z} : \alpha \leq z \leq \beta\}$.

A linear constraint is convex, thus for a constraint k on the variables x_i and x_j:

Observation 1 : If $(\alpha, \beta), (\alpha, \gamma) \in \mathrm{constraint}(k)$ then $(\alpha, \delta) \in \mathrm{constraint}(k)$ for all $\delta \in [\beta, \gamma]$.

Observation 2 : If $(\alpha_1, \alpha_2), (\beta_1, \beta_2), (\gamma_1, \gamma_2) \in \mathrm{constraint}(k)$ then all points inside the triangle induced by $(\alpha_1, \alpha_2), (\beta_1, \beta_2)$ and (γ_1, γ_2) satisfy constraint k.

We present a routine in Figure 1 which will be repeatedly used for constraint propagation. It receives as input two arrays ℓ and u of size n (passed by reference), two variables indices i, j and a constraint index k on these two variables. The objective of this routine is to find the impact of constraint k and the bounds ℓ_i, u_i on the bounds ℓ_j, u_j.

We denote by ℓ^{after} and u^{after} the values of the bounds ℓ and u after calling *OneOnOneImpact*(ℓ, u, i, j, k).

Observation 3 : $\mathrm{sat}(\ell^{\text{after}}, u^{\text{after}}) = \mathrm{sat}(\ell, u)$.

Routine OneOnAllImpact(ℓ, u, t)

```
Stack ← {t}
While Stack ≠ ∅ do
   i ← POP(Stack)
   For each constraint k involving x_i and another variable x_j
      OneOnOneImpact(ℓ, u, i, j, k)
      If u_j < ℓ_j then return "fail"
      If ℓ_j or u_j changed then PUSH j into Stack
```

Figure 2: Routine *OneOnAllImpact*.

Observation 4 : If $\beta \in [\ell_j^{\text{after}}, u_j^{\text{after}}]$ there exists $\alpha \in [\ell_i^{\text{after}}, u_i^{\text{after}}]$ such that $(\alpha, \beta) \in$ constraint(k).

The routine in Figure 2, which is called *OneOnAllImpact*, receives as input two arrays ℓ and u of size n (passed by reference) and a variable index t, and change ℓ and u according to the impact of ℓ_t and u_t on all the intervals.

We now prove that we do not lose feasible solutions after activating *OneOnAllImpact*.

Lemma 1: If ℓ^{after} and u^{after} are the values of ℓ and u after calling *OneOnAllImpact* then $\text{sat}(\ell^{\text{after}}, u^{\text{after}}) = \text{sat}(\ell, u)$.

Proof: All changes to ℓ and u are made by routine *OneOnOneImpact*. It is easy to prove the lemma by induction using Observation 3. □

Lemma 2: If *OneOnAllImpact*(ℓ, u, t) terminates without failure with the bounds ℓ^{after} and u^{after} and $\text{sat}((\ell_1, \ldots, \ell_{t-1}, -\infty, \ell_{t+1}, \ldots, \ell_n), (u_1, \ldots, u_{t-1}, \infty, u_{t+1}, \ldots, u_n)) \neq \emptyset$ then $\text{sat}(\ell^{\text{after}}, u^{\text{after}}) \neq \emptyset$.

Proof: Let $y \in \text{sat}((\ell_1, \ldots, \ell_{t-1}, -\infty, \ell_{t+1}, \ldots, \ell_n), (u_1, \ldots, u_{t-1}, \infty, u_{t+1}, \ldots, u_n))$. We define a vector y' as:

$$y'_t = \begin{cases} y_t, & y_t \in [\ell_t^{\text{after}}, u_t^{\text{after}}] \\ \ell_t^{\text{after}}, & y_t < \ell_t^{\text{after}} \\ u_t^{\text{after}}, & y_t > u_t^{\text{after}} \end{cases}$$

Consider constraint k on x_i and x_j. We need to show that $y'_i, y'_j \in$ constraint(k).

Case 1: $y_i \in [\ell_i^{\text{after}}, u_i^{\text{after}}]$ and $y_j \in [\ell_j^{\text{after}}, u_j^{\text{after}}]$.
$(y'_i, y'_j) = (y_i, y_j) \in$ constraint(k).

Case 2: $y_i < \ell_i^{\text{after}}$ and $y_j \in [\ell_j^{\text{after}}, u_j^{\text{after}}]$.
y is a feasible solution, thus $(y_i, y_j) \in$ constraint(k). When we changed the lower bound of x_i to ℓ_i^{after} we called *OneOnOneImpact* for all constraints involving x_i including constraint k. By Observation 4 there exists $\alpha \in [\ell_i^{\text{after}}, u_i^{\text{after}}]$ for which $(\alpha, y_j) \in$ constraint(k). Thus, by Observation 1 we get that $(\ell_i^{\text{after}}, y_j) \in$ constraint(k).

Algorithm ***Feasibility***$(\ell, u \in \mathbb{Z}^n)$

If $\ell = u$ then
  If $x = \ell$ is a feasible solution
    Then return ℓ
    Else return "fail"
Choose a variable x_t, for which $\ell_t < u_t$
$\alpha \leftarrow \left\lfloor \frac{1}{2}(\ell_t + u_t) \right\rfloor$ /* An arbitrary value $\alpha \in [\ell_i, u_i - 1]$ suffices as well */
$(\ell^{left}, u^{left}) \leftarrow (\ell, (u_1, \ldots, u_{t-1}, \alpha, u_{t+1}, \ldots, u_n))$
$(\ell^{right}, u^{right}) \leftarrow ((\ell_1, \ldots, \ell_{t-1}, \alpha + 1, \ell_{t+1}, \ldots, \ell_n), u)$
Call *OneOnAllImpact*$(\ell^{left}, u^{left}, t)$ and *OneOnAllImpact*$(\ell^{right}, u^{right}, t)$
If both calls fail then return "fail"
Choose a successful run of *OneOnAllImpact*
If call *left* was chosen
  Then return *Feasibility*(ℓ^{left}, u^{left})
  Else return *Feasibility*$(\ell^{right}, u^{right})$

Figure 3: Feasibility Algorithm

Case 3: $y_i < \ell_i^{\text{after}}$ and $y_j < \ell_j^{\text{after}}$.
y is a feasible solution, thus $(y_i, y_j) \in \text{constraint}(k)$. When we changed the lower bound of x_i to ℓ_i^{after} we called *OneOnOneImpact* for all constraint involving x_i including constraint k. By Observation 4 there exists $\alpha \in [\ell_i^{\text{after}}, u_i^{\text{after}}]$ for which $(\alpha, \ell_j^{\text{after}}) \in \text{constraint}(k)$. From the same arguments we get that there exists $\beta \in [\ell_j^{\text{after}}, u_j^{\text{after}}]$ for which $(\ell_i^{\text{after}}, \beta) \in \text{constraint}(k)$ as well. Thus, by Observation 2 we get that $(\ell_i^{\text{after}}, \ell_j^{\text{after}}) \in \text{constraint}(k)$.

Other cases are similar to Cases 2 and 3. $\square$

After proving that *OneOnAllImpact* preserves the feasibility property it is possible to use this routine as part of the feasibility algorithm from Figure 3.

Theorem 1: Algorithm *Feasibility* returns a feasible solution if such a solution exists.

Proof: Each recursive call reduces at least one of the ranges (the t'th), thus the execution of the algorithm must terminate. By Lemma 1 if $\text{sat}(\ell, u) = \emptyset$ the algorithm will return "fail". On the other hand, for the case $\text{sat}(\ell, u) \neq \emptyset$ we can prove by induction on $\sum_{i=1}^{n}(u_i - \ell_i)$ that the algorithm finds a feasible solution.

Base: $\sum_{i=1}^{n}(u_i - \ell_i) = 0$ implies $\ell = u$, thus $x = \ell$ is a feasible solution.

Step: By Lemma 1 at least one of the calls to *OneOnAllImpact* terminates without failure. If call *left* was chosen, then by Lemma 2 we know that $\text{sat}(\ell^{left}, u^{left}) \neq \emptyset$. Therefore, by the induction hypothesis we can find a feasible solution for ℓ^{left}, u^{left}. Obviously, a feasible solution $x \in \text{sat}(\ell^{left}, u^{left})$ satisfies $x \in \text{sat}(\ell, u)$. The same applies for call *right*.

This concludes the proof. □

Theorem 2: Algorithm *Feasibility* can be implemented in time $O(mU)$ and space $O(m)$.

Proof: To achieve time complexity of $O(mU)$, we run both calls to *OneOnAllImpact* in parallel (this approach was used for 2SAT by Even et al. [6]), and prefer the faster option of the two, if such a choice exists. After every change in the range of a variable x_i, we need to check the m_i constraints involving this variable, in order to discover the impact of the change. To perform this task efficiently we can store the input in an incidence list, where every variable has its constraints list. As ℓ_i and u_i can change up to $(u_i - \ell_i)$ times, we conclude that the total time complexity of the changes is $O(\sum_{i=1}^{n} m_i(u_i - \ell_i)) = O(mU)$ (the time wasted on uncompleted trials is bounded by the time complexity of the chosen trials). The algorithm uses $O(m)$ space for the input and a constant number of arrays of size n, thus uses linear space. □

3 From Feasibility to Approximation

Before presenting our approximation algorithm, let us first discuss the special case where $|U| = 2$, which is the minimum 2SAT problem, and its special case, the vertex cover problem.

The main idea of Bar-Yehuda and Even's 2-approximation algorithm for the vertex cover problem [2, 3, 4] is as follows: choose an edge $e = (u, v)$ and subtract $\varepsilon = \min\{w(u), w(v)\}$ from both $w(u)$ and $w(v)$. Every vertex cover $C \subseteq V$ must cover e, therefore the subtraction of ε from $w(u), w(v)$, which may cost up to $2 \cdot \varepsilon$, reduces the optimum by at least ε. After repeating this process until such subtractions are no longer possible, the resulting cover is $C = \{v : w(v) = 0\}$.

When considering a 2CNF formula[3], with respect to the minimum 2SAT problem, monotone clauses (e.g., $x_i \vee x_j$) can be treated as in the vertex cover case. But what about clauses with negative literals (of the form $x_i \vee \overline{x_j}$ or $\overline{x_i} \vee \overline{x_j}$)? Even if we knew how to make weight reductions, how would we choose a truth assignment? Thus, the above algorithm should be modified in order to be employed for the minimum 2SAT problem. The idea is that for a given 2CNF formula, if $x_1 \to \cdots \to x_2$ and $\overline{x_1} \to \cdots \to x_3$ we can subtract $\varepsilon = \min\{w_2, w_3\}$ from w_2 and w_3. This leaves us with the task of choosing a feasible truth assignment: when it is possible, assign a zero cost partial assignment, while relying upon the consistency property (Lemma 2).

This approach was used by Gusfield and Pitt [9] for approximating the minimum 2SAT problem. The 2CNF formula can be presented as a digraph where each vertex represents a boolean variable or its negation, and an edge represents a *OneOnOneImpact* propagation (logical "$\to$"). A propagation of an assignment can be viewed as a traversal (e.g., BFS, DFS) of the digraph. In order to be able to propagate forced assignments, Gusfield and Pitt's algorithm starts with a pre-process phase of constructing a transitive closure. This phase uses $\Omega(n^2)$ extra memory, which is expensive. It is much more critical when trying to approximate 2VIP by using the transformation to 2SAT from [11], and then Gusfield and Pitt's [9] algorithm. In this case the pre-process uses $\Omega(n^2U^2)$ extra memory[4].

It seems only natural to try to extend the previous ideas when approximating the 2VIP problem. In the previous section we have seen that it is possible to generalize the 2CNF

[3] Conjunctive Normal Form, see e.g., [5].

[4] This is in addition to the $\Omega(mU^2)$ extra memory needed for the transformation. As far as we know, every algorithm which relies upon direct 2SAT transformation suffers from this drawback.

assignment propagation to a 2VIP bound propagation, but how should we implement weight reductions? We can view a boolean assignment as a bound change, thus the cost of the assignment $x_i = 1$ is actually the cost of raising the lower bound of x_i from 0 to 1. For a 2VIP instance, every unit increase of the variable x_i would cost w_i, or in other words, an increase of the lower bound ℓ_i to ℓ'_i costs $w_i(\ell'_i - \ell_i)$. Bearing that in mind, we define an array called $\hat{\ell}$, which holds the values (of the variables), for which we have already "paid". This means that instead of "reducing" a weight due to a rise of ℓ_i we increase $\hat{\ell}_i$. Note that, unlike ℓ, $\hat{\ell}$ can hold non-integral values. This allows us to avoid using direct weight reductions and the consequent reduction to 2SAT.

We present an $O(nmU)$ time and $O(m)$ space 2-approximation algorithm. This approximation algorithm is a specific implementation of our feasibility algorithm (namely, the algorithm chooses variables and bounds in a specific order to get a 2-approximation). Not only does this algorithm seem natural, but also its complexity, $O(nm)$ time and $O(m)$ space, in the case of 2SAT is lower than that of Gusfield and Pitt's 2SAT algorithm ($O(nm)$ and $O(n^2 + m)$ correspondingly).

In order to use the local-ratio technique [2] we extend the 2VIP definition.

Definition 4: Given $a, b \in I\!R^n$, we write $a \le b$ if $\forall_i\ a_i \le b_i$. Also, we define

$$\begin{aligned} \max\{a,b\} &= (\max\{a_1,b_1\},\ldots,\max\{a_n,b_n\}) \\ \min\{a,b\} &= (\min\{a_1,b_1\},\ldots,\min\{a_n,b_n\}) \ . \end{aligned}$$

Given $\ell, u \in I\!N^n$ and $\hat{\ell}, \hat{u} \in I\!R^n$ for which $\ell \le \hat{\ell} \le \hat{u} \le u$, we define the following Extended 2VIP problem:

$$\begin{array}{lllr} \text{(E2VIP)} & \min \sum_{i=1}^n \Delta(x_i, \hat{\ell}_i, \hat{u}_i) w_i & & \\ & \text{s.t.} & a_k x_{i_k} + b_k x_{j_k} \ge c_k & \forall k \in \{1,\ldots,m\} \\ & & x_i \in [\ell_i, u_i] & \forall i \in \{1,\ldots,n\} \end{array}$$

where

$$\Delta(x, \hat{\ell}_i, \hat{u}_i) = \begin{cases} 0, & x_i < \hat{\ell}_i \\ (x_i - \hat{\ell}_i), & x_i \in [\hat{\ell}_i, \hat{u}_i] \\ (\hat{u}_i - \hat{\ell}_i), & x_i > \hat{u}_i \end{cases}$$

and $1 \le i_k, j_k \le n$, $w_i \ge 0$, $a, b, c \in Z\!\!\!Z^m$, and $\ell, u \in I\!N^n$.

We define $W(x, \hat{\ell}, \hat{u}) = \sum_{i=1}^n \Delta(x_i, \hat{\ell}_i, \hat{u}_i) w_i$. A feasible solution x^* is called an *optimal solution* if for every feasible solution x: $W(x^*, \hat{\ell}, \hat{u}) \le W(x, \hat{\ell}, \hat{u})$. We denote $W^*(\hat{\ell}, \hat{u}) = W(x^*, \hat{\ell}, \hat{u})$. A feasible solution x is called an *r-approximation* if $W(x, \hat{\ell}, \hat{u}) \le r \cdot W^*(\hat{\ell}, \hat{u})$.

Observation 5 : Given $\hat{\ell}, \hat{u}, \hat{m} \in I\!R^n$ for which $\hat{\ell} \le \hat{m} \le \hat{u}$, we get

$$W(x, \hat{\ell}, \hat{u}) = W(x, \hat{\ell}, \hat{m}) + W(x, \hat{m}, \hat{u})$$

Similarly to the Decomposition Observation from [2] we have:

Observation 6 (Decomposition Observation):
Given $\hat{\ell}, \hat{u}, \hat{m} \in I\!R^n$ such that $\hat{\ell} \le \hat{m} \le \hat{u}$, then

$$W^*(\hat{\ell}, \hat{m}) + W^*(\hat{m}, \hat{u}) \le W^*(\hat{\ell}, \hat{u}) \ .$$

Proof: Let x^*, y^* and z^* be optimal solutions for the system with respect to $\hat{\ell}, \hat{m}$; $\hat{m}, \hat{u}$, and $\hat{\ell}, \hat{u}$ correspondingly.

$$\begin{array}{rcll} W^*(\hat{\ell},\hat{m}) + W^*(\hat{m},\hat{u}) & = & W(x^*,\hat{\ell},\hat{m}) + W(y^*,\hat{m},\hat{u})) & \text{[By definition]} \\ & \leq & W(z^*,\hat{\ell},\hat{m}) + W(z^*,\hat{m},\hat{u})) & \text{[Optimality of } x^*, y^*\text{]} \\ & \leq & W(z^*,\hat{\ell},\hat{u}) & \text{[Observation 5]} \\ & = & W^*(\hat{\ell},\hat{u}) & \text{[By definition]} \end{array}$$

□

The following is this paper's Local-Ratio Theorem (see [2, 4]):

Theorem 3: If x is an r-approximation with respect to $\hat{\ell}, \hat{m}$ and with respect to $\hat{m}, \hat{u}$ then x is an r-approximation with respect to $\hat{\ell}, \hat{u}$.

Proof:

$$\begin{array}{rcll} W(x,\hat{\ell},\hat{u}) & = & W(x,\hat{\ell},\hat{m}) + W(x,\hat{m},\hat{u}) & \text{[Observation 5]} \\ & \leq & r \cdot W^*(\hat{\ell},\hat{m}) + r \cdot W^*(\hat{\ell},\hat{u}) & \text{[Given]} \\ & \leq & r \cdot W^*(\hat{\ell},\hat{u}) & \text{[Decomposition Observation]} \end{array}$$

□

We are ready to present the 2-approximation algorithm – see Figure 4.

Observation 7 : Algorithm *Approximate* is a specific implementation of Algorithm *Feasibility.*

Theorem 4: Algorithm *Approximate* is a 2-approximation algorithm for E2VIP systems.

Proof: By Observation 7 Algorithm *Approximate* returns a feasible solution. We prove by induction on the depth of the recursion that the algorithm finds a 2-approximation.

Base: $u = \ell$ implies $W(\ell, \hat{\ell}, \hat{u}) = 0$

Step: There are several cases:

Case 1: $\hat{\ell} \not\geq \ell$:
A 2-approximation with respect to $\max\left\{\hat{\ell}, \ell\right\}$ is obviously a 2-approximation solution with respect to $\hat{\ell}$.

case 2: $\hat{u} \not\leq u$:
Trivial.

Case 3: Call *right* failed:
By Lemma 1 there is no feasible solution which satisfies $x_t \geq \alpha + 1$, therefore we do not change the problem by adding the constraint $x_y \leq \alpha$. By Lemma 1 calling *OneOnAllImpact*(ℓ^{left}, u^{left}, t) does not change the problem either.

Case 4: Call *left* failed:
Similar to Case 3.

Algorithm ***Approximate***$(\ell, u \in I\!N^n; \hat{\ell}, \hat{u} \in I\!R^n)$

If $\hat{\ell} \not\geq \ell$ then return *Approximate*$(\ell, u, \max\left\{\hat{\ell}, \ell\right\}, \hat{u})$
If $\hat{u} \not\leq u$ then return *Approximate*$(\ell, u, \hat{\ell}, \min\{\hat{u}, u\})$
If $\ell = u$ then
If $x = \ell$ is a feasible solution
Then return ℓ
Else return "fail"
Choose a variable x_t, for which $\ell_t < u_t$
$\alpha \leftarrow \left\lfloor \frac{1}{2}(\ell_t + u_t) \right\rfloor$
$(\ell^{left}, u^{left}) \leftarrow (\ell, (u_1, \ldots, u_{t-1}, \alpha, u_{t+1}, \ldots, u_n))$
$(\ell^{right}, u^{right}) \leftarrow ((\ell_1, \ldots, \ell_{t-1}, \alpha + 1, \ell_{t+1}, \ldots, \ell_n), u)$
Call *OneOnAllImpact*$(\ell^{left}, u^{left}, t)$ and *OneOnAllImpact*$(\ell^{right}, u^{right}, t)$
If both calls failed then return "fail"
If call *right* failed then return *Approximate*$(\ell^{left}, u^{left}, \hat{\ell}, \hat{u})$
If call *left* failed then return *Approximate*$(\ell^{right}, u^{right}, \hat{\ell}, \hat{u})$
If $W(\ell^{left}, \hat{\ell}, \hat{u}) \leq W(\ell^{right}, \hat{\ell}, \hat{u})$
Then
Find $\hat{m} \in I\!R^n$ such that:
$\hat{\ell} \leq \hat{m} \leq \max\left\{\ell^{right}, \hat{\ell}\right\}$ and $W(\ell^{right}, \hat{\ell}, \hat{m}) = W(\ell^{left}, \hat{\ell}, \hat{u})$
$\hat{m} \leftarrow \max\left\{\hat{m}, \ell^{left}\right\}$
Return *Approximate*$(\ell^{left}, u^{left}, \hat{m}, \hat{u})$
Else
Find $\hat{m} \in I\!R^n$ such that:
$\hat{\ell} \leq \hat{m} \leq \max\left\{\ell^{left}, \hat{\ell}\right\}$ and $W(\ell^{left}, \hat{\ell}, \hat{m}) = W(\ell^{right}, \hat{\ell}, \hat{u})$
$\hat{m} \leftarrow \max\left\{\hat{m}, \ell^{right}\right\}$
Return *Approximate*$(\ell^{right}, u^{right}, \hat{m}, \hat{u})$

Figure 4: 2-Approximation Algorithm

Case 5: Both calls succeeded and $W(\ell^{left}, \hat{\ell}, \hat{u}) \leq W(\ell^{right}, \hat{\ell}, \hat{u})$:

We first show that every feasible solution is a 2-approximation with respect to $\hat{\ell}$ and $\hat{m}$. We examine an optimal solution x^* with respect to $\hat{\ell}$ and $\hat{m}$. By the construction of $\hat{m}$ we know that $\hat{m} \geq \ell^{left}$. Thus, if $\ell^{left} \leq x^* \leq u^{left}$ then the monotonicity of $W(\cdot, \hat{\ell}, \hat{m})$ implies

$$W(x^*, \hat{\ell}, \hat{m}) \geq W(\ell^{left}, \hat{\ell}, \hat{m}) \ .$$

If $\ell^{right} \leq x^* \leq u^{right}$:

$$\begin{array}{rcll} W(x^*, \hat{\ell}, \hat{m}) & \geq & W(\ell^{right}, \hat{\ell}, \hat{m}) & [x^* \geq \ell^{right}] \\ & \geq & W(\ell^{left}, \hat{\ell}, \hat{u}) & [\text{by the construction of } \hat{m}] \\ & = & W(\ell^{left}, \hat{\ell}, \hat{m}) & [\ell^{left} \leq \hat{m}] \end{array}$$

On the other hand, by the construction of $\hat{m}$:

$$W(\hat{m}, \hat{\ell}, \hat{m}) \leq W(\ell^{left}, \hat{\ell}, \hat{m}) + W(\ell^{right}, \hat{\ell}, \hat{m}) \leq 2 \cdot W(\ell^{left}, \hat{\ell}, \hat{m}) \ .$$

Thus, for every feasible solution x:

$$W(x, \hat{\ell}, \hat{m}) \leq 2 \cdot W(\ell^{left}, \hat{\ell}, \hat{m}) \ .$$

Therefore, by Theorem 3 a 2-approximation with respect to $\hat{m}$ and $\hat{u}$ is a 2-approximation with respect to $\hat{\ell}$ and $\hat{u}$.

We need to show that there exists an optimal solution x^* with respect to $\hat{m}$ and $\hat{u}$ for which $x_i^* \leq \alpha$. For every feasible solution y such that $y_i \geq \alpha + 1$ we define y' as:

$$y'_i = \begin{cases} y_i, & y_i \in [\ell_i^{left}, u_i^{left}] \\ \ell_i^{left}, & y_i < \ell_i^{left} \\ u_i^{left}, & y_i > u_i^{left} \end{cases}$$

By Lemma 2 y' is a feasible solution. $\ell^{left} \leq \hat{m}$ implies $W(y', \hat{m}, \hat{u}) \leq W(y, \hat{m}, \hat{u})$, thus there is an optimal solution with respect to $\hat{m}$ and $\hat{u}$ within the bounds ℓ^{left}, u^{left}. Therefore, a 2-approximation within the bounds ℓ^{left}, u^{left} is a 2-approximation with respect to $\hat{m}$ and $\hat{u}$.

Case 6: Both calls succeeded and $W(\ell^{left}, \hat{\ell}, \hat{u}) > W(\ell^{right}, \hat{\ell}, \hat{u})$:

Similar to Case 5.

This concludes the proof. □

Corollary 5: Algorithm *Approximate* is a 2-approximation algorithm for 2VIP systems.

Theorem 6: Algorithm *Approximate* can be implemented in time $O(nmU)$ and space $O(m)$.

Proof: In order to get the required time complexity, we must choose the x_t's carefully. One possibility is to choose the variables in an increasing order, i.e. $x_1, x_2, \ldots, x_n$, and to restart from the beginning after reaching x_n. We call such n iterations on all n variables a *pass*. As stated before, changing the range of x_i might cause changes in the ranges of other variables. The existence of a constraint on x_i and another variable x_j makes x_j a candidate for a range update. This means that we have to check the m_i constraints involving x_i in order to discover the consequences of changing its range each time this range changes. ℓ_i and u_i can change up to $(u_i - \ell_i)$ times, therefore we get that the time complexity of a single iteration is $O(mU + n) = O(mU)$. One pass may involve all n variables, so the time complexity of one pass is $O(nmU)$. By choosing $\alpha = \left\lfloor \frac{1}{2}(\ell_t + u_t) \right\rfloor$, we reduce the possible range for x_t at least by half. Therefore, in a single pass we reduce the possible ranges for all variables at least by half. Thus, we get that the total time complexity is: $\sum_{k=1}^{\log U} O(mn\frac{U}{2^k}) = O(mnU)$. As before, the algorithm uses an incidence list data structure and a constant number of arrays of size n, thus uses linear space. □

4 Generalizations

What if the constraints are not linear? Also, what if the weight functions of the variables are not linear? Can we avoid the expensive transformation to the 2SAT problem? In this section we try to extend our approach to a wider family of problems. In order to do so, we had to identify the properties of linear constraints and linear weight functions which are sufficient for the correctness of our claims.

4.1 Generalized Constraints

In this subsection we are interested in problems of the form:

$$\begin{array}{lll} \text{(G2VIP)} & \min \sum_{i=1}^n w_i x_i & \\ & \text{s.t.} \quad (x_{i_k}, x_{j_k}) \in \text{constraint}(k) & \forall k \in \{1, \ldots, m\} \\ & \qquad\quad x_i \in [\ell_i, u_i] & \forall i \in \{1, \ldots, n\} \end{array}$$

where $1 \leq i_k, j_k \leq n$, $w_i \geq 0$ and $\ell, u \in I\!N^n$. Also each constraint(k) satisfies the following: If $(\alpha_1, \alpha_2), (\beta_1, \beta_2) \in$ constraint(k) then there exists a shortest path in $I\!N^2$ (lattice) connecting (α_1, α_2) and (β_1, β_2) such that all its points satisfy constraint k.

A constraint which satisfies this property is called an *axis-convex* constraint. We assume that for an axis-convex constraint we have an $O(1)$ time oracle *OneOnOneImpact* which returns a tight range on x_j when given a range for x_i.

The following is implied by the definition of an axis-convex constraint:

Observation 8 : Given an axis-convex constraint $\mathcal{C}$, if $(\alpha_1, \alpha_2), (\beta_1, \beta_2) \in \mathcal{C}$ then for all $\beta \in [\beta_1, \beta_2]$ there exists $\alpha \in [\alpha_1, \alpha_2]$ such that $(\alpha, \beta) \in \mathcal{C}$.

Observations 1 and 4 hold for axis-convex constraints. A weaker version of Observation 2 also holds for axis-convex constraints. The proof uses Observation 8.

Observation 9 : Given an axis-convex constraint $\mathcal{C}$ and $(\alpha_1, \alpha_2), (\beta_1, \beta_2), (\gamma_1, \gamma_2) \in \mathcal{C}$, if (α_1, β_2) is inside the triangle induced by $(\alpha_1, \alpha_2), (\beta_1, \beta_2)$ and (γ_1, γ_2) then $(\alpha_1, \beta_2) \in \mathcal{C}$.

It is easy to see that Lemma 1 and Lemma 2 remain valid with axis-convex constraints. Thus, Algorithm *Feasibility* and Algorithm *Approximate* can be applied to G2VIP systems.

Corollary 7: A feasible solution, if such a solution exists, can be found for G2VIP systems in time $O(mU)$ and space $O(m)$.

Corollary 8: A 2-approximation, if a feasible solution exists, can be found for G2VIP systems in time $O(nmU)$ and space $O(m)$.

4.2 Generalized Weight Functions

In the original definition of a 2VIP system we used linear weight functions ($w_i x_i$ for every variable x_i). The following definition generalizes the weight function of a variable x_i:

Definition 5: A non-negative weight function ω is called a *monotone weight function* with respect to an interval if $\omega(\alpha) \leq \omega(\alpha+1)$ for all $\alpha \in \mathbb{Z}$ in the interval.

We assume, without loss of generality, that a monotone weight function is monotone over the real numbers and is invertible. If it is not we can always define $\omega'(z) = \omega(\lfloor z \rfloor) + (\omega(\lceil z \rceil) - \omega(\lfloor z \rfloor)) \cdot (z - \lfloor z \rfloor)$.

For a linear weight function ω_i we get that $\forall_{\alpha \geq 0}\ \omega_i(\alpha+1) - \omega_i(\alpha) = w_i$. For a general monotone weight function this is not necessarily the case. Therefore, in order to make Algorithm *Approximate* applicable to such weight functions we replace $\Delta(x_i, \hat{\ell}_i, \hat{u}_i) w_i$ in the definition of E2VIP systems by:

$$\Delta'(x_i, \hat{\ell}_i, \hat{u}_i) = \begin{cases} \omega_i(x_i) - \omega_i(\hat{\ell}_i), & x_i \in [\hat{\ell}_i, \hat{u}_i] \\ \omega_i(\hat{u}_i) - \omega_i(\hat{\ell}_i), & x_i > \hat{u}_i \\ 0, & x_i < \hat{\ell}_i \end{cases}$$

Now Algorithm *Approximate* is applicable as is.

Corollary 9: A 2-approximation, if a feasible solution exists, can be found for systems with monotone weight functions in time $O(nmU)$ and space $O(m)$.

Remark 1: If $x_i \in \{s_{i,1}, \ldots, s_{i,n_i}\}$, we can define a new variable $x'_i \in \{0, \ldots, n_i - 1\}$ and a new monotone weight function $w'_i(\alpha) = w_i(s_{i,\alpha+1})$. Thus, we get that the same results hold for $\ell_i = 0$ and $u_i = n_i - 1$.

5 Optimization Algorithm for Monotone Systems

In this section we show how to improve the complexity of Hochbaum and Naor's [12] optimization algorithm for a specific case of 2VIP systems called monotone systems:

Definition 6: A *monotone inequality* is an inequality on two variables with coefficients of opposite signs. A system with two variables per inequality is called *monotone* if all the constraints in the system are monotone.

We formulate the following monotone system:

$$\begin{array}{lll} \text{(M2VIP)} & \min \sum_{i=1}^{n} w_i x_i & \\ & \text{s.t.} \quad a_k x_{i_k} - b_k x_{j_k} \geq c_k & \forall k \in \{1, \ldots, m\} \\ & \qquad x_i \in [\ell_i, u_i] & \forall i \in \{1, \ldots, n\} \end{array}$$

where $1 \leq i_k, j_k \leq n$, $w_i \geq 0$, $a, b \in \mathbb{N}^n$, $c \in \mathbb{Z}^n$ and $\ell, u \in \mathbb{N}$.

Lagarias [13] proved that deciding whether a monotone integer program has a feasible solution is NP-complete. Hochbaum and Naor [12] showed that the set of all feasible

Algorithm ***Monotone***$(\ell, u \in \mathbb{Z}^n)$

For $t = 1$ to n do
  Call *OneOnAllImpact*(ℓ, u, t)
  If *OneOnAllImpact* fails then return "fail"
$x^* \leftarrow \ell$
Return x^*

Figure 5: Optimization Algorithm for Monotone Systems

solutions of a monotone system form a *distribution lattice* (this has been observed before by Veinott [16]). Using this property they presented an $O(mn^2 \log m + nmU)$ time feasibility algorithm for monotone systems, which finds the top (or bottom) of this lattice. They also presented an $O(nmU^2 \log(\frac{n^2U}{m}))$ time optimization algorithm for such systems with general (possibly negative) weights.

By limiting the discussion to non-negative weights, one can construct an $O(mU)$ time optimization algorithm: use the transformation to a 2SAT instance [11], then assign 0 to all boolean variables which did not get an assignment by the transformation. As before, this transformation uses $O(mU)$ space.

In Figure 5 we present an $O(mU)$ time and $O(m)$ space optimality algorithm for monotone systems with non-negative weights. This algorithm is actually a feasibility algorithm which finds the bottom of the feasible solutions lattice[5]. Due to the non-negative weights, the bottom of the lattice is also an optimal solution.

Theorem 10: An optimal solution can be found for non-negative monotone systems in time $O(mU)$ and space $O(m)$.

Proof: We prove that algorithm *Monotone* finds an optimal solution for M2VIP systems with non-negative weights in time $O(mU)$ and space $O(m)$.

It is easy to prove by induction using Lemma 1 that $\mathrm{sat}(\ell^{\text{after}}, u^{\text{after}}) = \mathrm{sat}(\ell, u)$, where ℓ^{after} and u^{after} are the values of ℓ and u after the loop. Thus, if one of the calls to *OneOnAllImpact* fails, then no feasible solution exists. The lemma also implies that if the instance of M2VIP is satisfiable then the above algorithm terminates without failure.

We now prove that $x^* = \ell$ is a feasible solution by showing that every constraint k: $a_k x_{i_k} - b_k x_{j_k} \geq c_k$ is satisfied by $x^* = \ell$. We examine the value of the lower bound of x_{i_k} after the last call of *OneOnOneImpact*(ℓ, u, j_k, i_k, k) during the algorithm (the call is made at least once). After this call we get $\ell_{i_k} \geq \left\lceil \frac{c_k + b_k \ell_{j_k}}{a_k} \right\rceil$. ℓ_{j_k} will not change from this point until the algorithm terminates (as a change of ℓ_{j_k} implies another call), and other lower bounds only increase during the execution algorithm, thus ℓ_{i_k}, ℓ_{j_k} must satisfy constraint k.

Due to Lemma 1, we get that $x^* = \ell$ is an optimal solution (and it is the only optimal solution if $\forall_i w_i > 0$).

Based on the same arguments as in Theorem 6 we get that the time complexity of finding an optimal solution for a monotone system is $O(\sum_{i=1}^{n} m_i(u_i - \ell_i)) = O(mU)$. The algorithm uses an incidence list data structure and therefore uses linear space. $\square$

[5]Note that when given ℓ as the initial value, the feasibility algorithm from [12] can be implemented in time $O(mU)$ and space $O(m)$.

As in the previous section we try to extend our approach to some non-linear systems. We use an important property of monotone constraints:

Definition 7: Let constraint k be an axis-convex constraint on x_i and x_j. Constraint k is called a *monotone axis-convex* constraint if for every $(\alpha_1, \alpha_2), (\beta_1, \beta_2) \in \text{constraint}(k)$ also $(\min\{\alpha_1, \beta_1\}, \min\{\alpha_2, \beta_2\}) \in \text{constraint}(k)$. A system of monotone axis-convex constraints is called a *generalized monotone system.*

By using similar arguments to those used in the previous section we get:

Corollary 11: An optimal solution can be found for generalized monotone systems with monotone weight functions in time $O(mU)$ and space $O(m)$.

Acknowledgments

We would like to thank Arie Freund, Niv Gilboa, Avigail Orni and especially Hadas Heier for their careful reading and suggestions.

References

[1] B. Aspvall and Y. Shiloach. A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. *SIAM Journal on Computing*, 9:827–845, 1980.

[2] R. Bar-Yehuda. One for the price of two: A unified approach for approximating covering problems. In *1st International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 49–62, July 1998. To appear in Algorithmica.

[3] R. Bar-Yehuda and S. Even. A linear time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2:198–203, 1981.

[4] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–46, 1985.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[6] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multi-commodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[8] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988.

[9] D. Gusfield and L. Pitt. A bounded approximation for the minimum cost 2-SAT problem. *Algorithmica*, 8:103–117, 1992.

[10] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982.

[11] D. S. Hochbaum, N. Megiddo, J. Naor, and A. Tamir. Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. *Mathematical Programming*, 62:69–83, 1993.

[12] D. S. Hochbaum and J. Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM Journal on Computing*, 23:1179–1192, 1994.

[13] J. C. Lagarias. The computational complexity of simultaneous diophantine approximation problems. *SIAM Journal on Computing*, 14:196–209, 1985.

[14] G. L. Nemhauser and L. E. Trotter. Vertex packings: structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.

[15] R. Shostak. Deciding linear inequalities by computing loop residues. *Journal of the ACM*, 28:769–779, 1981.

[16] A. F. Veinott. Representation of general and polyhedral subsemilattices and sublattices of product spaces. *Linear Algebra and its Applications*, 114/115:681–704, 1989.