Combinatorial Redundancy Detection*

Komei Fukuda[1], Bernd Gärtner[2], and May Szedlák[2]

1   Department of Mathematics and Department of Computer Science
     Institute of Theoretical Computer Science, ETH Zürich
     CH-8092 Zürich, Switzerland
     komei.fukuda@math.ethz.ch
2   Department of Computer Science
     Institute of Theoretical Computer Science, ETH Zürich
     CH-8092 Zürich, Switzerland
     {gaertner,may.szedlak}@inf.ethz.ch

Abstract

The problem of detecting and removing redundant constraints is fundamental in optimization. We focus on the case of linear programs (LPs) in dictionary form, given by n equality constraints in $n + d$ variables, where the variables are constrained to be nonnegative. A variable x_r is called *redundant*, if after removing $x_r \geq 0$ the LP still has the same feasible region. The time needed to solve such an LP is denoted by $LP(n, d)$.

It is easy to see that solving $n + d$ LPs of the above size is sufficient to detect all redundancies. The currently fastest practical method is the one by Clarkson: it solves $n + d$ linear programs, but each of them has at most s variables, where s is the number of nonredundant constraints.

In the first part we show that knowing all of the finitely many dictionaries of the LP is sufficient for the purpose of redundancy detection. A dictionary is a matrix that can be thought of as an enriched encoding of a vertex in the LP. Moreover – and this is the combinatorial aspect – it is enough to know only the signs of the entries, the actual values do not matter. Concretely we show that for any variable x_r one can find a dictionary, such that its sign pattern is either a redundancy or nonredundancy certificate for x_r.

In the second part we show that considering only the sign patterns of the dictionary, there is an output sensitive algorithm of running time $\mathcal{O}(d \cdot (n + d) \cdot s^{d-1} \cdot LP(s, d) + d \cdot s^d \cdot LP(n, d))$ to detect all redundancies. In the case where all constraints are in general position, the running time is $\mathcal{O}(s \cdot LP(n, d) + (n + d) \cdot LP(s, d))$, which is essentially the running time of the Clarkson method. Our algorithm extends naturally to a more general setting of arrangements of oriented topological hyperplane arrangements.

1998 ACM Subject Classification G.2.1 Combinatorics

Keywords and phrases system of linear inequalities, redundancy removal, linear programming, output sensitive algorithm, Clarkson's method

Digital Object Identifier 10.4230/LIPIcs.SOCG.2015.315

1   Introduction

The problem of detecting and removing redundant constraints is fundamental in optimization. Being able to understand redundancies in a model is an important step towards improvements of the model and faster solutions.

In this paper, we focus on redundancies in systems of linear inequalities. We consider systems of the form

$$
\begin{array}{rcl}
x_B &=& b - Ax_N \\
x_B &\geq& 0 \\
x_N &\geq& 0
\end{array}
\tag{1}
$$

where B and N are disjoint finite sets of variable indices with $|B| = n$, $|N| = d$, $b \in \mathbb{R}^B$ and $A \in \mathbb{R}^{B \times N}$ are given input vector and matrix. We assume that the system (1) has a feasible solution. Any consistent system of linear equalities and inequalities can be reduced to this form.

A variable x_r is called *redundant* in (1) if $x_B = b - Ax_N$ and $x_i \geq 0$ for $i \in B \cup N \setminus \{r\}$ implies $x_r \geq 0$, i.e., if after removing constraint $x_r \geq 0$ from (1) the resulting system still has the same feasible region. Testing redundancy of x_r can be done by solving the linear program (LP)

$$
\begin{array}{rll}
\text{minimize} & x_r & \\
\text{subject to} & x_B = b - Ax_N & \\
& x_i \geq 0, & \forall i \in B \cup N \setminus \{r\}.
\end{array}
\tag{2}
$$

Namely, a variable x_r is redundant if and only if the LP has an optimal solution and the optimal value is nonnegative.

Let $LP(n, d)$ denote the time needed to solve an LP of form (2). Throughout the paper, we are working in the real RAM model of computation, where practical algorithms, but no polynomial bounds on $LP(n, d)$ are known. However, our results translate to the standard Turing machine model, where they would involve bounds of the form $LP(n, d, \ell)$, with ℓ being the bit size of the input. In this case, $LP(n, d, \ell)$ can be polynomially bounded. The notation $LP(n, d)$ abstracts from the concrete representation of the LP, and also from the algorithm being used; as a consequence, we can also apply it in the context of LPs given by the signs of their dictionaries.

By solving $n + d$ linear programs, $\mathcal{O}((n + d) \cdot LP(n, d))$ time is enough to detect all redundant variables in the real RAM model, but it is natural to ask whether there is a faster method. The currently fastest practical method is the one by Clarkson with running time $\mathcal{O}((n + d) \cdot LP(s, d) + s \cdot n \cdot d)$ [4]. This method also solves $n + d$ linear programs, but each of them has at most s variables, where s is the number of nonredundant variables. Hence, if $s \ll n$, this *output-sensitive* algorithm is a major improvement.

A related (dual) problem is the one of finding the extreme points among a set P of n points in $\mathbb{R}^d$. A point $p \in P$ is *extreme* in P, if p is not contained in the convex hull of $P \setminus \{p\}$. It is not hard to see that this problem is a special case of redundancy detection in linear systems.

Specialized (and output-sensitive) algorithms for the extreme points problem exist [14, 6], but they are essentially following the ideas of Clarkson's algorithm [4]. For fixed d, Chan uses elaborate data structures from computational geometry to obtain a slight improvement over Clarkson's method [2].

In this paper, we study the *combinatorial* aspects of redundancy detection in linear systems. The basic questions are: What kind of information about the linear system do we need in order to detect all redundant variables? With this restricted set of information, how fast can we detect all of them? Our motivation is to explore and understand the boundary between geometry and combinatorics with respect to redundancy. For example, Clarkson's method [4] uses *ray shooting*, an intrinsically geometric procedure; similarly, the dual extreme

points algorithms [14, 6] use scalar products. In a purely combinatorial setting, neither ray shooting nor scalar products are well-defined notions, so it is natural to ask whether we can do without them.

We will show that our results solely depend on the finite combinatorial information given by the signed dictionaries, i.e., the size is bounded by a function of d and n only. A dictionary can be thought of as an encoding of the associated arrangements of hyperplanes, the corresponding signed dictionary only contains the signs of the encoding (see Section 2). On the other hand Clarkson's algorithm depends on the input data A and b.

Our approach is very similar to the combinatorial viewpoint of linear programming pioneered by Matoušek, Sharir and Welzl [13] in form of the concept of *LP-type problems*. The question they ask is: how quickly can we *optimize*, given only combinatorial information? As we consider redundancy detection and removal as important towards efficient optimization, it is very natural to extend the combinatorial viewpoint to also include the question of redundancy. The results that we obtain are first steps and leave ample space for improvement. An immediate theoretical benefit is that we can handle redundancy detection in structures that are more general than systems of linear inequalities; most notably, our results naturally extend to the realm of *oriented matroids* [1].

Statement of Results

The first point that we will make is that for the purpose of redundancy testing, it is sufficient to know all the finitely many dictionaries associated with the system of inequalities (1). Moreover, we show that it is sufficient to know only the *signed* dictionaries, i.e., the *signs* of the dictionary entries. Their actual numerical values do not matter.

In Theorem 2, we give a characterization of such a redundancy certificate. More precisely, we show that, for every redundant variable x_r there exists at least one signed dictionary such that its sign pattern is a redundancy certificate of x_r. Similarly, as shown in Theorem 4, for every nonredundant variable there exists a nonredundancy certificate. Such a single certificate can be detected in time $LP(n, d)$ (see Section 4.3). The number of dictionaries needed to detect *all* redundancies depends on the LP and can vary between constant and linear in $n + d$ [10, Appendix].

In a second part, we present a Clarkson-type, output-sensitive algorithm that detects all redundancies in running time $\mathcal{O}(d \cdot (n + d) \cdot s^{d-1} LP(s, d) + d \cdot s^d \cdot LP(n, d))$ (Theorem 5). Under some general position assumptions the running time can be improved to $\mathcal{O}((n + d) \cdot LP(s, d) + s \cdot LP(n, d))$, which is basically the running time of Clarkson's algorithm. In these bounds, $LP(n, d)$ denotes the time to solve an LP to which we have access only through signed dictionaries. As in the real RAM model, no polynomial bounds are known, but algorithms that are fast in practice exist.

In general our algorithm's running time is worse than Clarkson's, but it only requires the combinatorial information of the system and not its actual numerical values. If the feasible region is not full dimensional (i.e. not of dimension d), then a redundant constraint may become nonredundant after the removal of some other redundant constraints. To avoid these dependencies of the redundant constraints we assume full dimensionality of the feasible region. Because of our purely combinatorial characterizations of redundancy and nonredundancy, our algorithm works in the combinatorial setting of oriented matroids [1], and can be applied to remove redundancies from oriented topological hyperplane arrangements.

2 Basics

Before discussing redundancy removal and combinatorial aspects in linear programs, we fix the basic notation on linear programming –such as dictionaries and pivots operations – and review finite pivot algorithms. (For further details and proofs see e.g. [3, Part 1], [7, Chapter 4].)

2.1 LP in Dictionary Form

Throughout, if not stated otherwise, we always consider *linear programs* (LPs) of the form

$$
\begin{array}{llll}
\text{minimize} & c^T x_N \\
\text{subject to} & x_B & = & b - A x_N \\
& x_E & \geq & 0,
\end{array}
\tag{3}
$$

where $E := B \cup N$ and as introduced in (1), B and N are disjoint finite sets of variable indices with $|B| = n$, $|N| = d$, $b \in \mathbb{R}^B$ and $A \in \mathbb{R}^{B \times N}$ are given input vector and matrix. An LP of this form is called LP in *dictionary form* and its *size* is $n \times d$. The set B is called a (initial) *basis*, N a (initial) *nonbasis* and $c^T x_N$ the *objective function*.

The *feasible region* of the LP is defined as the set of $x \in \mathbb{R}^E$ that satisfy all constraints, i.e., the set $\{x \in \mathbb{R}^E | x_B = b - A x_N, x_E \geq 0\}$. A feasible solution $\overline{x}$ is called *optimal* if for every feasible solution x, $c^T \overline{x} \leq c^T x$. The LP is called *unbounded* if for every $k \in \mathbb{R}$, there exists a feasible solution x, such that $c^T x \leq k$. If there exists no feasible solution, the LP is called *infeasible*.

The *dictionary* $D(B) \in \mathbb{R}^{B \cup \{f\} \times N \cup \{g\}}$ of an LP (3) w.r.t. a basis B is defined as

$$
D := D(B) = \begin{bmatrix} 0 & c^T \\ b & -A \end{bmatrix},
$$

where f is the index of the first row and g is the index of the first column. For each $i \in B \cup \{f\}$ and $j \in N \cup \{g\}$, we denote by d_{ij} its (i, j) entry, by $D_{i\cdot}$ the row indexed by i, and by $D_{\cdot j}$ the column indexed by j.

Hence by setting $x_f := c^T x_N$, we can rewrite (3) as

$$
\begin{array}{llll}
\text{minimize} & x_f \\
\text{subject to} & x_{B \cup \{f\}} & = & D x_{N \cup \{g\}} \\
& x_E & \geq & 0, \\
& x_g & = & 1.
\end{array}
\tag{4}
$$

Whenever we do not care about the objective function, we may set $c = 0$, and with abuse of notation, set $D = [b, -A]$.

The *basic solution* w.r.t. B is the unique solution $\overline{x}$ to $x_{B \cup \{f\}} = D x_{N \cup \{g\}}$ such that $\overline{x}_g = 1$, $\overline{x}_N = 0$ and hence $\overline{x}_{B \cup \{f\}} = D_{\cdot g}$.

The *dual LP* of LP (4) is defined as

$$
\begin{array}{llll}
\text{minimize} & y_g \\
\text{subject to} & y_{N \cup \{g\}} & = & -D^T y_{N \cup \{f\}} \\
& y_E & \geq & 0, \\
& y_f & = & 1.
\end{array}
\tag{5}
$$

It is useful to define the following four different types of dictionaries (and bases) as shown in the figure below, where "+" denotes positivity, "⊕" nonnegativity and similarly "−" negativity and "⊖" nonpositivity.

A dictionary D (or the associated basis B) is called *feasible* if $d_{ig} \geq 0$ for all $i \in B$. A dictionary D (or the associated basis B) is called *optimal* if $d_{ig} \geq 0$, $d_{fj} \geq 0$ for all $i \in B, j \in N$. A dictionary D (or the associated basis B) is called *inconsistent* if there exists $r \in B$ such that $d_{rg} < 0$ and $d_{rj} \leq 0$ for all $j \in N$. A dictionary D (or the associated basis B) is called *dual inconsistent* if there exists $s \in N$ such that $d_{fs} < 0$ and $d_{is} \geq 0$ for all $i \in B$.

feasible

optimal

inconsistent

dual inconsistent
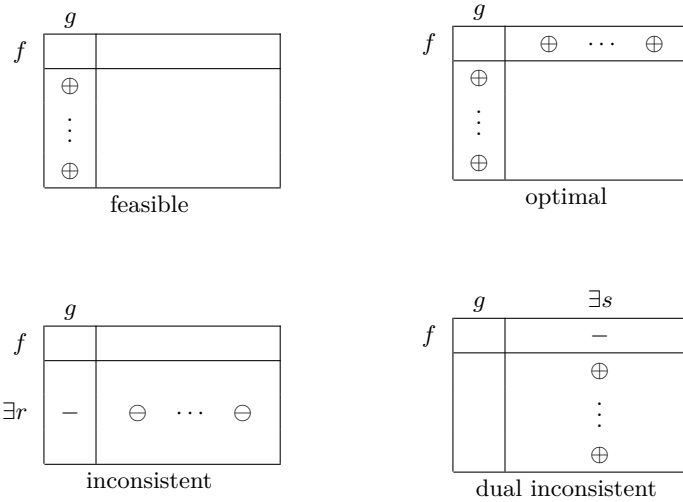
The following proposition follows from standard calculations.

▶ **Proposition 1.** *For any LP in dictionary form the following statements hold.*
1. *If the dictionary is feasible then the associated basic solution is feasible.*
2. *If the dictionary is optimal, then the associated basic solution is optimal.*
3. *If the dictionary is inconsistent, then the LP is infeasible.*
4. *If the dictionary is dual inconsistent, then the dual LP is infeasible. If in addition the LP is feasible, then the LP is unbounded.*

2.2 Pivot Operations

We now show how to transform the dictionary of an LP into a modified dictionary using elementary matrix operation, preserving the equivalence of the associated linear system. This operation is called a *pivot operation*.

Let $r \in B$, $s \in N$ and $d_{rs} \neq 0$. Then it is easy to see that one can transform $x_{B \cup \{f\}} = D x_{N \cup \{g\}}$ to an equivalent system (i.e., with the same solution set) :

$$x_{B' \cup \{f\}} = D' x_{N' \cup \{g\}},$$

where $B' = B \setminus \{r\} \cup \{s\}$ ($N' = N \setminus \{s\} \cup \{r\}$, respectively) is a new (non)basis and

$$d'_{ij} = \begin{cases} \frac{1}{d_{rs}} & \text{if } i = s \text{ and } j = r \\ -\frac{d_{rj}}{d_{rs}} & \text{if } i = s \text{ and } j \neq r \\ \frac{d_{is}}{d_{rs}} & \text{if } i \neq s \text{ and } j = r \\ d_{ij} - \frac{d_{is} \cdot d_{rj}}{d_{rs}} & \text{if } i \neq s \text{ and } j \neq r \end{cases} \quad (i \in B' \cup \{f\} \text{ and } j \in N' \cup \{g\}). \tag{6}$$

We call a dictionary *terminal* if it is optimal, inconsistent or dual inconsistent. There are several finite pivot algorithms such as the simplex and the criss-cross method that transform any dictionary into one of the terminal dictionaries [16, 17, 11],[5, Section 4]. This will be discussed further in Section 4.3.

3   Combinatorial Redundancy

Consider an LP in dictionary form as given in (3). Then $x_r \geq 0$ is *redundant*, if the removal of the constraint does not change the feasible solution set, i.e., if

$$
\begin{array}{lll}
\text{minimize} & c^T x_N & \\
\text{subject to} & x_B & = & b - Ax_N \\
& x_i & \geq & 0, \quad \forall i \in E \setminus \{r\},
\end{array} \tag{7}
$$

has the same feasible solution set as (3). Then the variable x_r and the index r are called *redundant*.

If the constraint $x_r \geq 0$ is not redundant it is called *nonredundant*, in that case the variable x_r and the index r are called *nonredundant*.

It is not hard to see that solving $n + d$ LPs of the same size as (7) suffices to find all redundancies. Hence running time $\mathcal{O}((n + d) \cdot LP(n, d))$ suffices to find all redundancies, where $LP(n, d)$ is the time needed to solve an LP of size $n \times d$. Clarkson showed that it is possible to find *all* redundancies in time $\mathcal{O}((n + d) \cdot LP(s, d) + s \cdot n \cdot d)$, where s is the number of nonredundant variables [4]. In case where $s \ll n$ this is a major improvement. To be able to execute Clarkson's algorithm, one needs to assume full dimensionality and an interior point of the feasible solution set. In the LP setting this can be done by some preprocessing, including solving a few $(O(d))$ LPs [9, Section 8].

In the following we focus on the combinatorial aspect of redundancy removal. We give a combinatorial way, the *dictionary oracle*, to encode LPs in dictionary form, where we are basically only given the signs of the entries of the dictionaries. In Section 4 we will show how the signs suffice to find all redundant and nonredundant constraints of an LP in dictionary form.

Consider an LP of form (3). For any given basis B, the *dictionary oracle* returns a matrix

$$
D^{\sigma} = D^{\sigma}(B) \in \{+, -, 0\}^{B \times N \cup \{g\}}, \text{ with } d_{ij}^{\sigma} = sign(d_{ij}), \forall i \in B, j \in N \cup \{g\}.
$$

Namely, for basis B, the oracle simply returns the matrix containing the signs of $D(B)$, without the entries of the objective row f.

4   Certificates

We show that the dictionary oracle is enough to detect all redundancies and nonredundancies of the variables in E. More precisely for every $r \in E$, there exists a basis B such that $D^{\sigma}(B)$ is either a redundancy or nonredundancy certificate for x_r. We give a full characterization of the certificates in Theorems 2 and 4. The number of dictionaries needed to have *all* certificates depend on the LP. See [10, Appendix] for examples where constantly many suffice and where linearly many are needed.

For convenience throughout we make the following assumptions, which can be satisfied with simple preprocessing.

1. The feasible region of (3) is full dimensional (and hence nonempty).
2. There is no $j \in N$ such that $d_{ij} = 0$ for all $i \in B$.

In Section 4.3 we will see that both the criss-cross and the simplex method can be used on the dictionary oracle for certain objective functions. Testing whether the feasible solution set is empty can hence be done by solving one linear program in the oracle setting. As mentioned in the introduction the full-dimensionality assumption is made to avoid dependencies between

the redundant constraints. This can be achieved by some preprocessing on the LP, including solving a few ($O(d)$) LPs [9].

It is easy to see that if there exists a column j such that $d_{ij} = 0$ for all $i \in B$, then x_j is nonredundant and we can simply remove the column.

4.1 A Certificate for Redundancy in the Dictionary Oracle

We say a that basis B is *r-redundant* if $r \in B$ and $D_{r.}^\sigma \geq 0$ i.e. if $D^\sigma(B)$ is as given in the figure below.
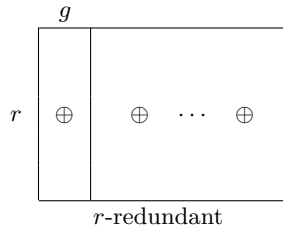
r-redundant

Since the r-th row of the dictionary represents $x_r = d_{rg} + \sum_{j \in N} d_{rj} x_j$, $x_r \geq 0$ is satisfied as long as $x_j \geq 0$ for all $j \in N$. Hence $x_r \geq 0$ is redundant for (3).

▶ **Theorem 2** (**Redundancy Certificate**). *An inequality $x_r \geq 0$ is redundant for the system (3) if and only if there exists an r-redundant basis.*

Proof. We only have to show the "only if" part.

Suppose $x_r \geq 0$ is redundant for the system (3). We will show that there exists an r-redundant basis.

Consider the LP minimizing the variable x_r subject to the system (3) without the constraint $x_r \geq 0$. Since $x_r \geq 0$ is redundant for the system (3), the LP is bounded. By assumption 1 and the fact that every finite pivot algorithm terminates in a terminal dictionary the LP has an optimal dictionary.

If the initial basis contains r, then we can consider the row associated with r as the objective row. Apply any finite pivot algorithm to the LP. Otherwise, r is nonbasic. By assumption 2, one can pivot on the r-th column to make r a basic index. This reduces the case to the first case.

Let's consider an optimal basis and optimal dictionary for the LP where x_r is the objective function. Since it is optimal, all entries d_{rj} for $j \in N$ are nonnegative. Furthermore, d_{rg} is nonnegative as otherwise we would have found a solution that satisfies all constraints except $x_r \geq 0$, implying nonredundancy of x_r.   ◀

From the proof of Theorem 2 the following strengthening of Theorem 2 immediately follows.

▶ **Corollary 3.** *An inequality $x_r \geq 0$ is redundant for the system (3) if and only if there exists a* feasible *r-redundant basis.*

feasible r-redundant
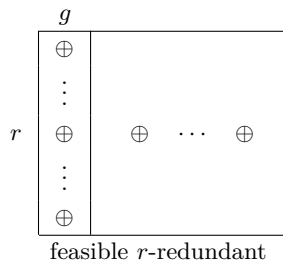
4.2 A Certificate for Nonredundancy in the Dictionary Oracle

Similarly as in the redundancy case, we introduce a certificate for nonredundancy using the dictionary oracle. A basis B is called *r-nonredundant* if B is feasible, $r \in N$ and $d_{tg} = 0$ implies $d_{tr} \leq 0$ for all $t \in B$ i.e. $D^{\sigma}(B)$ is of the following form.

r-nonredundant

▶ **Theorem 4** (**Nonredundancy Certificate**). *An inequality $x_r \geq 0$ is nonredundant for the system (3) if and only if there exists an r-nonredundant basis.*

Before proving the theorem, we observe the following.

1. Unlike in the redundancy certificate an r-nonredundant basis needs to be feasible. To verify the correctness of a nonredundancy certificate we need to check between n and $2n$ entries, which is typically much larger than the $d + 1$ entries we need for the redundant case.

2. If the g-column of a feasible basis does not contain any zeros, then all nonbasic variables are nonredundant. In general when $x_r \geq 0$ is nonredundant, not necessarily every feasible basis B with $r \in N$ is r-nonredundant. Consider the system:

$$x_3 = x_1 + x_2$$
$$x_1, x_2, x_3 \geq 0.$$

Then the basis $\{3\}$ is not a certificate of nonredundancy of x_1, as $d_{31}^{\sigma} = +$ in the associated dictionary. On the other hand, the basis $\{2\}$ is 1-nonredundant:

	g	1	2			g	1	3
3	0	+	+		2	0	−	+

Proof of Theorem 4. Let (LP) be of form (3) and suppose that $x_r \geq 0$ is nonredundant. Then it follows that for ϵ small enough $x_r \geq -\epsilon$ is nonredundant in

$$
\begin{array}{lrcl}
\text{minimize} & x_r \\
\text{subject to} & x_B & = & b - Ax_N \\
& x_i & \geq & 0, \quad \forall i \in B \cup N \setminus \{r\} \\
& x_r & \geq & -\epsilon.
\end{array}
\tag{8}
$$

Note that this LP can easily be transformed to an LP of form (3) by the straight forward variable substitution $x'_r = x_r + \epsilon$.

LP (8) attains its minimum at $-\epsilon$ and hence there exists an optimal dictionary where r is nonbasic. Let B be such a feasible optimal basis of (LP^{ϵ}) with $r \in N$. We show that if we choose ϵ small enough, B is r-nonredundant in (LP).

Let $B_1, B_2, \ldots, B_m$ be the set of all bases (feasible and infeasible) of (LP), that have r as a nonbasic variable. Choose $\epsilon > 0$ such that

$$\epsilon < \min \left\{ \frac{d_{tg}}{d_{tr}} \,\middle|\, t \in B_i : d_{tg}, d_{tr} < 0; i = 1, 2, \ldots, m \right\}.$$

If the right hand side (RHS) is undefined, we choose any $\epsilon < \infty$.

Geometrically this means that if for $t \in B_i$ $x_t \geq 0$ is violated in the basic solution w.r.t. B_i in (LP), then it is still violated in the corresponding basic solution (LP^ϵ). Let D and D^ϵ be the dictionaries w.r.t. B in (LP) and (LP^ϵ) respectively.

D and D^ϵ only differ in their entries of column g, where

$$d_{tg}^\epsilon = d_{tg} - \epsilon \cdot d_{tr}, \forall t \in B. \tag{9}$$

We need to show that B is r-nonredundant in (LP). To show that B is a feasible basis we need that $d_{tg} \geq 0$ for all $t \in B$. If $d_{tr} \geq 0$, then this is clear. In the case where $d_{tr} < 0$ it follows that $\epsilon \geq \frac{d_{tg}}{d_{tr}}$ and hence $d_{tg} \geq 0$ by choice of ϵ. Hence B is feasible and if $d_{tg} = 0$, then by equation (9) it follows that $d_{tr} \leq 0$. Therefore B is r-nonredundant.

For the other direction let B be r-nonredundant and D and D^ϵ the corresponding dictionaries in (LP) and (LP^ϵ), respectively. Choose $\epsilon > 0$ such that

$$\epsilon \leq \min\left\{\frac{d_{tg}}{d_{tr}} \,\bigg|\, t \in B : d_{tg}, d_{tr} > 0\right\}.$$

If the RHS is undefined, we choose any $\epsilon < \infty$.

We claim that for such an ϵ, B is still feasible for (LP^ϵ) and hence $x_r \geq 0$ is nonredundant. Again the two dictionaries only differ in row g, where

$$d_{tg}^\epsilon = d_{tg} - \epsilon \cdot d_{tr}, \forall t \in B.$$

In the case where $d_{tg} = 0$, it follows that $d_{tg}^\epsilon \geq 0$ by r-nonredundancy. If $d_{tg} > 0$, then

$$d_{tg}^\epsilon = d_{tg} - \epsilon \cdot d_{tr} \geq d_{tg} - \min\left\{\frac{d_{t'g}}{d_{t'r}} \,\bigg|\, t' \in B : d_{t'g}, d_{t'r} > 0\right\} \frac{d_{t'g}}{d_{t'r}} \cdot d_{tr} \geq 0.$$

◀

4.3 Finite Pivot Algorithms for Certificates

In this section we discuss how to design finite pivot algorithms for the dictionary oracle model. Both the criss-cross method and the simplex method can be used for the dictionary oracle to find redundancy and nonredundancy certificates. A finite pivot algorithm chooses in every step a pivot according to some given rule and terminates in an optimal, inconsistent or dual inconsistent basis in a finite number of steps. Note that both the criss-cross method and the simplex method may not be polynomial in the worst case, but are known to be fast in practice [12, 15]. Furthermore there exits no known polynomial algorithm to solve an LP given by the dictionary oracle. Fukuda conjectured that the randomized criss-cross method is an expected polynomial time algorithm [8].

By the proof of Theorem 2, in order to find a redundancy certificate in (3) it is enough to solve (3) with objective function x_r. Similarly by the proof of Theorem 4, for a nonredundancy certificate it is enough to solve the ϵ-perturbed version (8).

For the criss-cross method, the pivot rule is solely dependent on the signs of the dictionary entries and not its actual values [7, Chapter 4], [11]. Standard calculations show that the signs in the ϵ-perturbed dictionary (for $\epsilon > 0$ small enough) are completely determined by the signs of the original dictionary. We recall that the dictionary oracle does not output the objective row, but since we minimize in direction of x_r the signs of the objective row are completely determined. (If r is basic then the objective row has the same entries as the r-th row and if r nonbasic then $d_{fr} = +$ and all other entries of the objective row are

zero.) Therefore the dictionary oracle is enough to decide on the pivot steps of the criss-cross method.

For the simplex method with the smallest index rule, we are given a feasible basis and the nonbasic variable of the pivot element is chosen by its sign only [3, Part 1 Section 3]. The basic variable of the pivot is chosen as the smallest index such that feasibility is preserved after a pivot step. Using the dictionary oracle one can test the at most n possibilities and choose the appropriate pivot.

5   An Output Sensitive Redundancy Detection Algorithm

Throughout this section, we denote by S' the set of nonredundant indices and by R' the set of redundant indices. Denote by $LP(n, d)$ the time needed to solve an LP. By the discussion in Section 4.3, for any x_r, $r \in E$, we can find a certificate in time $LP(n, d)$. Theorem 5 presents a Clarkson type, output sensitive algorithm with running time $\mathcal{O}(d \cdot (n + d) \cdot s^{d-1} \cdot LP(s, d) + d \cdot s^d \cdot LP(n, d))$, that for a given LP outputs the set S', where $s = |S'|$. Typically s and d are much smaller than n.

5.1   General Redundancy Detection

> Redundancy Detection Algorithm(D,g,f);
> **begin**
> $R := \emptyset, S := \emptyset$;
>   **while** $R \cup S \neq E$ **do**
>   Pick any $r \notin R \cup S$ and test if r is redundant w.r.t. S;
>   **if** r redundant w.r.t. S **then**
> $R = R \cup \{r\}$;
>   **else** /* r nonredundant w.r.t. S */
>   test if r is redundant w.r.t. $E \setminus R$;
>   **if** r is nonredundant w.r.t. $E \setminus R$ **then**
> $S = S \cup \{r\}$;
>   **else** /* r redundant w.r.t. $E \setminus R$ */
>   Find some sets $S^F \subseteq S'$ and $R^F \subseteq R'$ such that $S^F \nsubseteq S$;
> $R = R \cup R^F$, $S = S \cup S^F$;
>   **endif**;
>   **endif**;
>   **endwhile**;
> $S^* := S$;
>   output S^*;
> **end**.

Since in every round at least one variable is added to S or R, the algorithm terminates. The correctness of the output can easily be verified: If in the outer loop r is added to R, r is redundant w.r.t. S and hence redundant w.r.t. $S^* \supseteq S$. If in the inner loop r is added to S, r is nonredundant w.r.t. $E \setminus R$ and hence nonredundant w.r.t. $S^* \subseteq E \setminus R$.

The main issue is how to find the sets S^F and R^F efficiently in the last step. This will be discussed in (the proof of) Lemma 6.

A technical problem is that we cannot test for redundancy in the dictionary oracle when S does not contain a nonbasis. Therefore as long as this is the case, we fix an arbitrary

nonbasis N and execute the redundancy detection algorithm on $S \cup N$ instead of S. Since this does not change correctness or the order of the running time, we will omit this detail in the further discussion.

▶ **Theorem 5.** *The redundancy detection algorithm outputs S', the set of nonredundant constraints in time*

$$R(n,d,s) = \mathcal{O}\left(\sum_{i=0}^{d-1}((n+d) \cdot s^i \cdot LP(s,d-i) + s^{i+1} \cdot LP(n,d-i))\right)$$

and consequently in time

$$R(n,d,s) = \mathcal{O}\left(d \cdot (n+d) \cdot s^{d-1} \cdot LP(s,d) + d \cdot s^d \cdot LP(n,d)\right).$$

The following Lemma implies Theorem 5.

▶ **Lemma 6.** *Let $R(n,d,s)$ be the running time of the redundancy detection algorithm in n basic variables, d nonbasic variables and s the number of nonredundant variables. Then in the last step of the inner loop some sets $S^F \subseteq S'$ and $R^F \subseteq R'$, with $S^F \nsubseteq S$, can be found in time $\mathcal{O}(R(n,d-1,s) + LP(n,d))$.*

Proof of Theorem 5. Termination and correctness of the algorithm are discussed above. The iteration of the outer loop of the algorithm takes time $\mathcal{O}(LP(s,d))$ and is executed at most $n+d$ times. By Lemma 6, the running time of the inner loop is $\mathcal{O}(R(n,d-1,s) + LP(n,d))$ and since in each round at least one variable is added to S, it is executed at most s times. Therefore the total running time is given recursively by

$$R(n,d,s) = \mathcal{O}\left((n+d) \cdot LP(s,d) + s \cdot (R(n,d-1,s) + LP(n,d))\right).$$

The claim follows by solving the recursion and noting that $R(n,0,s)$ can be set to $\mathcal{O}(n)$. ◄

It remains to prove Lemma 6, for which we first prove some basic results below, using the dictionary oracle setting.

▶ **Lemma 7.** *Let $D = D(B)$ be a feasible dictionary of an LP of form (3) and assume $F := \{i \in B | b_i = 0\} \neq \emptyset$. We consider the subproblem of the LP denoted LP^F (with dictionary D^F,) that only contains the rows of D indexed by F. Then $r \in F \cup N$ is nonredundant in LP if and only if it is nonredundant in LP^F.*

Proof. We only need to show the "if" part. Let $r \in F \cup N$ be nonredundant in LP^F with certificate $\overline{D}^F$. Then there exists a sequence of pivot steps from D^F to $\overline{D}^F$. Using the same ones on D and obtaining dictionary $\overline{D}$, this is a nonredundancy certificate for r, since $\overline{d}_{ig} = d_{ig} > 0$ for all $i \in B \setminus F$ by the definition of F. ◄

▶ **Lemma 8.** *Let $D = [b, -A]$ be the dictionary of an LP of form (3). Then a variable $r \in E$ is nonredundant in the LP given by D if and only if it is nonredundant in the LP given by $D^0 = [0, b, -A]$.*

Proof. If $D(B)$ is a redundancy certificate for r for some basis B, then $D^0(B)$ is a redundancy certificate for r as well.

For the converse, let $D = D(B)$ be a nonredundancy certificate for r for some basis B. For simplicity assume that $B = \{1, 2, \ldots, n\}$. For now assume that $b_i > 0$ for all $i \in B$ and let D^i the dictionary obtained from D^0 by pivoting on b_i, $i = 1, 2, \ldots, n$. We will show that at least one of the D^i, $i \in \{0, 1, \ldots, n\}$ is a nonredundancy certificate for r. Since after any pivot the first column of D^i stays zero, D^i is a nonredundancy certificate if and only if $D^i_{\cdot r} \le 0$. Let $R^i = (r^i_1, r^i_2, \ldots r^i_n)^T := D^i_{\cdot r}$ for $i \ge 1$ and $R = (r_1, r_2, \ldots, r_n)^T := D^0_{\cdot r}$.

Claim: Assume that $r_i^i < 0$ for any fixed i and there are at least $i-1$ additional nonpositive entries (w.l.o.g. we assume them to be $r_1^i, r_2^i, \ldots, r_{i-1}^i$). If R^i has a positive entry (which w.l.o.g. we assume to be r_{i+1}^i), then $r_{i+1}^{i+1} < 0$ and $r_1^{i+1}, r_2^{i+1}, \ldots, r_i^{i+1}$ are nonpositive.

If D^0 is not a certificate for r, then w.l.o.g. $r_1 > 0$ and hence $r_1^1 = -\frac{r_1}{b_1} < 0$. Therefore by induction the lemma follows from the claim.

Assume that $r_1^i, r_2^i, \ldots, r_{i-1}^i \le 0$, $r_i^i < 0$ and $r_{i+1}^i > 0$. Then we have $r_i > 0$ and

$$r_{i+1}^i = r_{i+1} - \frac{r_i b_{i+1}}{b_i} > 0 \Leftrightarrow r_i b_{i+1} < r_{i+1} b_i \Rightarrow r_{i+1} > 0, \tag{10}$$

$$\forall j < i : r_j^i = r_j - \frac{r_i b_j}{b_i} \le 0 \Leftrightarrow r_j b_i \le r_i b_j. \tag{11}$$

The following calculations show the claim.

$$r_{i+1}^{i+1} = -\frac{r_{i+1}}{b_{i+1}} < 0 \Leftrightarrow r_{i+1} > 0 \text{ which holds by (10)}.$$

$$r_i^{i+1} = r_i - \frac{r_{i+1} b_i}{b_{i+1}} \le 0 \Leftrightarrow r_i b_{i+1} \le r_{i+1} b_i \text{ which holds by (10)}.$$

$$\forall j < i : r_j^{i+1} = r_j - \frac{r_{i+1} b_j}{b_{j+1}} \le 0 \Leftrightarrow r_j b_{i+1} \le r_{i+1} b_j,$$

and by (10) and (11), $r_j b_{i+1} = (r_j b_i)(r_i b_{i+1}) \cdot \frac{1}{r_i b_i} \le r_{i+1} b_j.$

Now suppose that $b_i = 0$ for some i. Then by the nonredundancy certificate $r_i \le 0$, and it is easy to see that $r_i^j = r_i \le 0$ for all admissible pivots on b_j. Hence we can use the above construction on the nonzero entries of b. ◀

Proof of Lemma 6. Suppose that during the execution of the algorithm, r is nonredundant w.r.t. the current set S, and redundant w.r.t. $E \setminus R$, with *feasible* redundancy certificate $D = [b, -A]$, which exists by Corollary 3. If $b > 0$, then all nonbasic indices in N are nonredundant by Theorem 4. Choose $S^F = N$, $R^F = \emptyset$. It holds that $S^F \nsubseteq S$, since otherwise r would be redundant w.r.t. S. The running time of the inner loop in this case is $LP(n, d)$.

Now if there exists $i \in B$ such that $b_i = 0$, define $F = \{i \in B | b_i = 0\}$, LP^F and D^F as in Lemma 7. We now recursively find all redundant and nonredundant constraints in the LP^F using Lemma 8 as follows. From LP^F we construct another LP, denoted LP^- with one less nonbasic variable, by deleting $D_{\cdot g}^F$ (the column of all zeros), choosing any element $t \in N$ and setting $t = g$. Finding all redundancies and nonredundancies in LP^- takes time $R(|F|, d-1, s)$. By Lemma 8 redundancies and nonredundancies are preserved for LP^F.

Therefore finding them in LP^F takes time $R(|F|, d-1, s) + LP(n, d) \le R(n, d-1, s) + LP(n, d)$, where the $LP(n, d)$ term is needed to check separately whether t is redundant. Choose S^F as the set of nonredundant indices of LP^F and R^F as the set of redundant ones. By Lemma 7 $S^F \subseteq S'$ and $R^F \subseteq R'$. Since by Lemma 7 r is redundant in LP^F, $S^F \nsubseteq S$, since otherwise r would be redundant w.r.t. S. ◀

5.2   Strong Redundancy Detection

In this section we show how under certain assumptions the running time of the redundancy algorithm can be improved. If we allow the output to also contain some *weakly redundant* constraints (see definition below), it is basically the same as the running time of Clarkson's method.

A redundant variable r is called *strongly redundant* if for any basic feasible solution $\overline{x}$, $\overline{x}_r > 0$. In particular for any basic feasible solution, $r \in B$. If r is redundant but not strongly redundant r is called *weakly redundant*.

As before let s be the number of nonredundant constraints and let R_s, (with $|R_s| = r_s$,) and R_w, (with $|R_w| = r_w$,) be the set of strongly and weakly redundant constraints respectively.

▶ **Theorem 9.** *It is possible to find a set $S^* \supseteq S'$, $S^* \cap R_s = \emptyset$ in time $\mathcal{O}((n + d) \cdot LP(s + r_w, d) + (s + r_w) \cdot LP(n, d))$.*

The following corollary follows immediately.

▶ **Corollary 10.** *If there are no weakly redundant constraints, the set S' of nonredundant constraints can be found in time $\mathcal{O}((n + d) \cdot LP(s, d) + s \cdot LP(n, d))$.*

The theorem is proven using the following two lemmas, which can be verified with straight forward variable substitutions.

▶ **Lemma 11.** *[3, Part 1 Section 3] Let (LP) of form (3), where (LP) is not necessarily full dimensional. W.l.o.g. $B = \{1, 2, \ldots, n\}$. For each $i \in \{1, 2, \ldots, n\}$ replace the nonnegativity constraint $x_i \geq 0$ by $x_i \geq -\epsilon^i$, for $\epsilon > 0$ sufficiently small. Denote the resulting LP by (LP^ϵ). Let D^σ be the output of the dictionary oracle for an arbitrary dictionary D of (LP). Then (LP^ϵ) is full dimensional. Furthermore in $D^{\sigma,\epsilon}$, the corresponding output for the ϵ-perturbed version, all signs can be determined by D^σ, and $D^{\sigma,\epsilon}_{.g}$ has no zero entries.*

▶ **Lemma 12.** *[3, Part 1 Section 3] Let (LP) and (LP^ϵ) be as in Lemma 11. Then any nonredundant constraint in (LP) is nonredundant in (LP^ϵ) and any strongly redundant constraint in (LP) is strongly redundant in (LP^ϵ).*

Proof of Theorem 9. Replace the given LP by it's ϵ-perturbed version as in Lemma 11 and run the redundancy removal algorithm, which is possible by the same lemma. By Lemma 12, $S^* \supseteq S'$ and $S^* \cap R_s = \emptyset$. Since by Lemma 11, the entries of the g-column of any dictionary $D^{\sigma,\epsilon}$ are strictly positive the algorithm never runs the recursive step and the running time follows.   ◀

▶ Remark. The ϵ-perturbation makes every feasible LP full dimensional, therefore the full dimensionality assumption can be dropped for Theorem 9.

5.3   Discussion

In this paper, we presented new combinatorial characterizations of redundancy and nonredundancy in linear inequality systems. We also presented a combinatorial algorithm for redundancy removal.

In contrast to the Clarkson algorithm our redundancy detection algorithm does not need the whole LP but only the combinatorial information of the dictionaries. Although in general the running time is worse, assuming that we have no weak redundancies, our redundancy removal algorithm basically has the same running time as the Clarkson algorithm. Still, a natural goal is to improve the runtime of our algorithm in the general case and get it closer to that of Clarkson's method. We do have a first output-sensitive algorithm for combinatorial redundancy detection, but the exponential dependence on the dimension d is prohibitive already for moderate d.

Our algorithm works in a more general setting of oriented matroids. This means one can remove redundancies from oriented pseudo hyperplane arrangements efficiently. Furthermore,

the algorithm can be run in parallel. Yet, analyzing the performance may not be easy because checking redundancy of two distinct variables simultaneously may lead to the discovery of the same (non)redundant constraint. This is an interesting subject of future research.

References

1 A. Björner, M. Las Vergnas, B. Sturmfels, N. White, and G. Ziegler. *Oriented Matroids.* Cambridge University Press, 1993.

2 T. M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete & Computational Geometry*, 16(4):369–387, 1996.

3 V. Chvatal. *Linear Programming.* W. H. Freeman, 1983.

4 K. L. Clarkson. More output-sensitive geometric algorithms. In *Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 695–702, 1994.

5 G. B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, NJ, 1963.

6 J. H. Dulá, R. V. Helgason, and N. Venugopal. An algorithm for identifying the frame of a pointed finite conical hull. *INFORMS J. Comput.*, 10(3):323–330, 1998.

7 K. Fukuda. Introduction to optimization. `http://www.ifor.math.ethz.ch/teaching/ Courses/Fall_2011/intro_fall_11`, 2011.

8 K. Fukuda. Walking on the arrangement, not on the feasible region. Efficiency of the Simplex Method: Quo vadis Hirsch conjecture?, IPAM, UCLA, 2011. presentation slides available as http://helper.ipam.ucla.edu/publications/sm2011/sm2011_9630.pdf.

9 K. Fukuda. Lecture: Polyhedral computation. `http://www-oldurls.inf.ethz.ch/ personal/fukudak/lect/pclect/notes2015/`, 2015.

10 K. Fukuda, B. Gärtner, and M. Szedlák. Combinatorial redundancy removal. *Preprint: arXiv:1412.1241*, 2014.

11 K. Fukuda and T. Terlaky. Criss-cross methods: A fresh view on pivot algorithms. *Mathematical Programming*, 79:369–395, 1997.

12 V. Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities III*, pages 159–175. Academic Press, 1972.

13 J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498–516, 1996.

14 Th. Ottmann, S. Schuierer, and S. Soundaralakshmi. Enumerating extreme points in higher dimensions. In E.W. Mayer and C. Puech, editors, *STACS 95: 12th Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 900, pages 562–570. Springer-Verlag, 1995.

15 C. Roos. An exponential example for Terlaky's pivoting rule for the criss-cross simplex method. *Mathematical Programming*, 46:79–84, 1990.

16 T. Terlaky. A finite criss-cross method for the oriented matroids. *Journal of Combinatorial Theory Series B*, 42:319–327, 1987.

17 Z. Wang. A finite conformal-elimination free algorithm over oriented matroid programming. *Chinese Annals of Math.*, 8B:120–125, 1987.