Vector Assignment Problems: A General Framework

Leah Epstein[*] Tamir Tassa[†]

Abstract

We present a general framework for vector assignment problems. In such problems one aims at assigning n input vectors to m machines such that the value of a given target function is minimized. While previous approaches concentrated on simple target functions such as max-max, the general approach presented here enables us to design a polynomial time approximation scheme (PTAS) for a wide class of target functions. In particular, thanks to a novel technique of preprocessing the input vectors, we are able to deal with non-monotone target functions. Such target functions arise in vector assignment problems in the context of video transmission and broadcasting.

Keywords: scheduling, bin packing, optimization, makespan, approximation schemes, PTAS.

[*]School of Computer Science, The Interdisciplinary Center, Herzliya, Israel.

[†]Department of Applied Mathematics, Tel-Aviv University, Ramat Aviv, Tel Aviv, Israel.

1 Introduction

In this paper we present a general framework for dealing with assignment problems in general and vector assignment problems in particular. An assignment problem is composed of the following three ingredients:

- Items: $x^1, \ldots, x^n$;
- Containers: $c^1, \ldots, c^m$;
- A target function: $F : \{1, \ldots, m\}^{\{1,\ldots,n\}} \to \mathcal{R}^+$.

Each item is characterized by a parameter or a set of parameters that reflect the "size" of the item. That size may be a scalar, a vector or whatever the application which gave rise to the problem dictates (e.g., a random variable).
The containers may be characterized by their capacity; that capacity would be a scalar or a vector, in accord with the type of the items to be stored.
The set $\{1, \ldots, m\}^{\{1,\ldots,n\}}$ consists of all possible assignments of items to containers. Each assignment is referred to as *a solution* to the problem. In all assignment problems there is a natural addition operation between items. Hence, given a solution $S \in \{1, \ldots, m\}^{\{1,\ldots,n\}}$, we may define the *load* in the kth container as

$$\ell^k = \sum_{S(i)=k} x^i \quad , \quad 1 \le k \le m \ .$$

The target function evaluates for each solution a nonnegative *cost*. That function takes into account the loads ℓ^k and possibly also the container capacities, if such capacities are given.

The assignment problem is the problem of finding an optimal solution in the sense that it has a minimal cost. As such problems are strongly NP-hard, polynomial time approximation schemes (PTAS) are sought. Given a fixed $\varepsilon > 0$, such schemes produce, in polynomial time, a solution whose cost is larger than that of an optimal solution by a factor of no more than $(1 + \text{Const} \cdot \varepsilon)$. We would like to comment that although it is customary to look for a multiplicative factor of exactly $(1 + \varepsilon)$, we prefer, for the sake of simplicity, to allow the more general factor $(1+\text{Const}\cdot\varepsilon)$. Obviously, one may translate the latter to the former by rescaling ε.

The above formulation encompasses all problems that were studied in the art so far. However, the chosen target functions in those studies were limited to a narrow class of "natural" functions, as described below. Motivated by an interesting problem that arises in transmitting multiplexed video streams, see §2, we suggest here a general framework that includes a much wider class of target functions.

Overview. We focus here on Vector Assignment Problems (VAP), where the items $\mathbf{x}^i$, $1 \le i \le n$, and the resulting loads $\boldsymbol{\ell}^k$, $1 \le k \le m$, are vectors in $(\mathcal{R}^+)^d$. We consider target functions of the form:

$$F(S) = f(g(\boldsymbol{\ell}^1), \ldots, g(\boldsymbol{\ell}^m)) \ . \qquad (1)$$

Here:

- S is a given solution and $\boldsymbol{\ell}^k$, $1 \le k \le m$, are the corresponding load vectors.

- $g : (\mathcal{R}^+)^d \to \mathcal{R}^+$ is a function that evaluates a cost per container.
- $f : (\mathcal{R}^+)^m \to \mathcal{R}^+$ is a function that evaluates the final cost over all containers.

In [13] we expand our discussion to include target functions of the form

$$F(S) = f(g^1(\boldsymbol{\ell}^1, \mathbf{c}^1), \ldots, g^m(\boldsymbol{\ell}^m, \mathbf{c}^m)) \ . \tag{2}$$

Namely, the per-container cost evaluator, $g(\cdot)$, may depend also on the container's parameters $\mathbf{c}^k$, and the functional evaluation may be different for each container.

Relation to previously studied problems. This suggested framework includes many problems that are already known in the art. The terminology in those problems may vary. In some scalar problems the containers are referred to as *bins*. In other scalar problems and in most all vector problems the terms *items*, *containers* and *assignment* are replaced with *jobs*, *machines* and *scheduling*, respectively. Since we have in mind applications that do not deal with scheduling, we adopt herein a slightly more general terminology: *vectors, machines* and *assignment.* With these terms, we refer to (1) as the case of identical machines, while (2) will be referred to as the case of non-identical machines. In particular, when the non-identical machine parameters are viewed as part of the input, namely, the running time is required to be independent of those parameters, (2) is referred to as the case of related machines.

Herein, we list some of the previously studied problems. The first five examples are scalar while the last one is a vector problem.

1. The classical problem in this context is the scalar *makespan problem.* In that problem one aims at minimizing the maximal load. It is described by (1) with $d = 1$, $g = id$ and $f = \max$. See [15, 16, 18, 20].

2. The related makespan problem, where the machines are associated with speeds c^k, $1 \le k \le m$, is given by (2) with $d = 1$, $g^k(x, c) = x/c$ for all k and $f = \max$. See [17].

3. The ℓ_p minimization problem is given by (1) with $d = 1$, $g = id$ and $f(y_1, \ldots, y_m) = (\sum_{k=1}^m y_k^p)^{1/p}$. The case $p = 2$ was studied in [4, 6] and was motivated by storage allocation problems. The general case was studied in [1].

4. Problem (2) with $d = 1$, $g^k(x, c) = h(x/c)$ for all k, where $h : \mathcal{R}^+ \to \mathcal{R}^+$ is some fixed function, and f is either the maximum or sum of its arguments, was studied in [2, 12]. Other choices for f are the inverse minimum or the inverse sum. By considering those choices, one aims at **maximizing** the minimal or average completion time. See [8, 21] for the case of identical machines (where all c^k equal 1) and [3] for the case of related machines.

5. The Extensible Bin Packing Problem is given by (2) with $d = 1$, $g^k(x, c) = \max\{x, c\}$ for all k and $f(y_1, \ldots, y_m) = \sum_{k=1}^m y^k$. See [7, 10, 11].

6. The Vector Scheduling Problem coincides with (1) with $f = g = \max$. See [5].

The general framework presented here is a powerful tool. Most of the previous studies concentrated on specific choices of cost functions, depending on the application in mind, or considered narrow classes of cost functions. By defining general classes of cost functions, however, as in

[2, 12], it is possible not only to cover many different problems that were studied separately, but also to identify the properties of the cost functions that play a significant role in the analysis and, consequently, examine which of those properties is truly essential for the existence of a PTAS and which is not.

An example of such a property is monotonicity. In all of the above mentioned problems that were previously studied, except for the scalar problem 4, the target functions were monotone. Namely, when adding an item to a container, the value of the target function increases, or at least does not decrease. Such monotonicity is indeed natural when dealing with bin packing or job scheduling: every item that is stored in a bin decreases the remaining available space in that bin; every job assigned to a machine increases the load on that machine. However, we present in this paper the so called *line-up problem* that arises in video transmission and broadcasting, where the target function has a different nature: it aims at optimizing the quality of the transmitted video. Such functions are not monotone - increasing the size of a vector component may actually decrease the value of the target function. By taking the general framework approach, we are able to identify the place where monotonicity is actually used in the analysis and, consequently, suggest another path that circumvents the need in monotonicity.

As mentioned above, Chekuri and Khanna [5] designed a PTAS for vector assignment problems when $f = g = \max$. To the best of our knowledge, theirs is the only study thus far that offered a PTAS for vector problems. The techniques that we present here, some of which are extensions of the methods that were used in [5], enable us to devise a PTAS for a significantly larger class of cost functions for vector problems.

We note in passing that a related class of problems that we exclude from our discussion is that in which the goal is to minimize the number of containers that are used for packing, subject to some condition (such conditions are usually associated with the capacity of the containers). See [14, 9, 19, 22, 5].

Organization of the paper. In §2 we describe the line-up problem that motivated this study. In §3 we characterize the class of cost functions to which our analysis applies. §4 is devoted to monotone target functions. After describing in §4.1 a simple preprocessing procedure that we apply to the input vectors, we derive in §4.2 lower and upper bounds for the optimal value. We then describe in §4.3 a binary search procedure that converges to the optimal value. §4.4-§4.6 are devoted to the core algorithm which is the main ingredient in the binary search: that algorithm receives a test value and decides whether the problem has a solution whose value is less than or equal the given test value. In case such solutions exist, the core algorithm returns a solution whose value deviates from the given test value by a small multiplicative factor. In §5 we turn to deal with non-monotone target functions. Using a preprocessing technique that is applied to the input vectors, we are able to modify the core algorithm so that it may be applied to such target functions as well.

Notation agreements. Throughout this paper we adopt the following conventions:

- Small case letters denote scalars; bold face small case letters denote vectors.
- A superscript of a vector denotes the index of the vector; a subscript of a vector indicates a component in that vector. E.g., $\boldsymbol{\ell}_j^k$ denotes the jth component of the vector $\boldsymbol{\ell}^k$.
- If $\gamma(k)$ is any expression that depends on k, then $f(\gamma(k))_{1 \le k \le m}$ stands for $f(\gamma(1), \ldots, \gamma(m))$.

- If x is a scalar then $x_+ = \max\{x, 0\}$.
- If $\circ$ is any operation between scalars then $\mathbf{v} \circ c$ is the vector whose jth component is $\mathbf{v}_j \circ c$. Similarly, if $\propto$ is any relation between scalars, then $\mathbf{v} \propto c$ or $\mathbf{v} \propto \mathbf{w}$ mean that the relation holds component-wise.

2 The Line Up Problem

One of the greatest advantages of digital encoding of TV programs is that it enables data compression. Consequently, the transition from analog to digital encoding allowed digital TV broadcast centers to transmit a greater number of TV programs using the same infrastructure[1]. In the analog era every TV program was assigned to a physical channel that served solely for the transmission of that specific program. In contrast, when dealing with digital TV programs that were subject to compression and, therefore, require less bandwidth (bits per second),it is possible to transmit several TV programs on the same channel. As the compression ratio that is offered by the ubiquitous MPEG-II standard is roughly 1:10, the number of TV programs that share the same channel is usually between 8 and 12.

Assume a TV broadcast center that broadcasts m analog programs, say $m = 30$. That center may broadcast $n \approx 10m = 300$ digital programs, using the same infrastructure. In order to do so, it is necessary to assign the n digital programs (*inputs*) to the m physical channels (*outputs*) where the number of programs per channel is typically 8-12. Such an assignment, $\{P_1, \dots, P_n\} \to \{C_1, \dots, C_m\}$, is called *a line-up.*

Each digital program is encoded by a digital sequence that is called *a program stream.* Given a line-up, the program streams that correspond to programs that were assigned to the same channel are merged (*multiplexed*) into a single digital stream known as a *transport stream.* Finally, the unified transport stream is converted from digital to analog and is transmitted via cable or satellite to the required destination.

The line-up problem asks for an assignment of programs to channels that would maximize the quality of the multiplexed transport streams. This is explained below.

Each TV program is composed of several *elementary streams*: a video stream, one or several audio streams (several audio streams occur in stereo or multi-lingual programs) and possibly also data streams (e.g., subtitles or close captions). The video stream consumes the larger part of the bandwidth. For efficiency, it is usually encoded by a variable bit-rate stream, according to the complexity of the displayed event. For example, cartoons or talk-shows are typically low bandwidth consumers, while sport events are high bandwidth consumers.

As opposed to the variable bit-rate of the single program streams, the multiplexed transport streams must have a constant bit-rate. A standard transport stream bit-rate is 38.8 Mbits/sec while a typical TV program consumes 3-4 Mbits/sec. The multiplexing (also called *statistical* multiplexing) of several variable bit-rate program streams into one constant bit-rate transport stream assumes that the peaks in one program will be compensated by lows in another program and, hence, the sum of bit-rates of all programs will demonstrate less variation than each of the single programs.

[1]In order to distinguish between logical channels (e.g., CNN, PBS or the Sports Channel) and physical channels, we adopt the terminology used in the Digital Video Broadcasting (DVB) standard that refers to the former as *programs* and to the latter as *channels.*

An underflow is a situation in which the sum of concurrent bit-rates of all programs is less than the channel's capacity. This situation is undesired since then bandwidth, a most valuable resource, is wasted. An overflow is the opposite situation. This is also a bad situation since then compression should be applied in order to decrease the bit-rates of some of the programs and that may reduce the quality of the outgoing video. Unfortunately, underflows and overflows are inevitable. Hence, by studying the behavior of the programs, one may design "clever" assignments that would minimize the effects of underflows and overflows.

To this end, let us assume n programs and m channels. Assume that each program was sampled along a time period of say 1 week (this is a good choice because of the typical 1-week periodicity in TV schedules). Let $\mathbf{x}^k = (\mathbf{x}_1^k, \ldots, \mathbf{x}_d^k)$ describe the bit-rate of the kth program along the sampling period; for example, if we break the week into time frames of 15 minutes, then $d = 4 * 24 * 7 = 672$ and $\mathbf{x}_j^k$ is the average bit-rate of the kth program along the jth time interval. Let us further assume that the channel capacities are given by c_i, $1 \le i \le m$.

Next, we need to formulate a target function. We are looking at target functions of the form (1) or (2). The inner cost function g gives a score for the multiplexed transport stream in a particular channel. Let $\boldsymbol{\ell}$ denote the load vector in that channel and let c be its capacity. Then the following are plausible choices for g:

1. $g(\boldsymbol{\ell}) = \|\boldsymbol{\ell}\|_p$, where $1 \le p < \infty$ ($p = \infty$ seems to be inappropriate in this case because one would like to take into account the behavior of the channel in **all** time frames).

2. $g(\boldsymbol{\ell}) = \|\boldsymbol{\ell}\|_{\ell_p(\mathbf{w})}$ where $\mathbf{w}$ is a weight vector that gives higher weights to time frames in prime time and lower weights to time frames of lower rating.

3. $g(\boldsymbol{\ell}) = \|\boldsymbol{\ell}\|_{1,p} := \|\boldsymbol{\ell}\|_p + \|\boldsymbol{\Delta\ell}\|_p$ where $\boldsymbol{\Delta\ell} \in \mathcal{R}^{d-1}$ and $\boldsymbol{\Delta\ell}_j = \boldsymbol{\ell}_{j+1} - \boldsymbol{\ell}_j$, $1 \le j \le d-1$. This choice of the Sobolev norm, denoted $w^{1,p}$, reflects the goal to avoid, as much as possible, fluctuations in the video quality.

4. $g(\boldsymbol{\ell}, c) = \|\max\{\boldsymbol{\ell}, c\}\|$, where $\|\cdot\|$ is some norm (typically an ℓ_p norm) and c is a minimal bandwidth that is always allocated to the channel. This is the vector analogous to the scalar extensible bin problem.

5. $g(\boldsymbol{\ell}, c) = \|(\boldsymbol{\ell} - c)_+\|$ for some norm. This function penalizes only overflows, and disregards time frames where no quality reduction occurs. This function, with $\|\cdot\|$ being the ℓ_1 norm, aims at minimizing the processing time that is needed to perform the actual video compression, as this time is usually proportional to the amount of data that needs to be removed.

6. $g(\boldsymbol{\ell}, c) = \|\frac{(\boldsymbol{\ell}-c)_+}{\boldsymbol{\ell}}\|$ is a function that aims at optimizing the video quality because it takes into account the compression rate that would need to be imposed, hence, the quality reduction.

More accurate functions may be formulated according to the compression strategy that would be applied in case of an overflow.

The outer cost function f could be any ℓ_p norm. Here, the choice of $p = \infty$ would serve best the goal of having a uniform quality across all channels. The function f could be also a weighted norm to reflect different priorities to different programs or channels.

The above discussion illustrates that VAPs occur in contexts other than job scheduling, and that the relevant target functions may be other than the usual ℓ_p norms. Hence, a more general framework for studying VAPs is needed. In the next sections we present such a framework and we find a PTAS for the VAP in that framework.

3 The Cost Functions

Herein we list the assumptions that we make on the outer cost function $f(\cdot)$ and the inner cost function $g(\cdot)$.

Definition 1

1. *A function $h : (\mathcal{R}^+)^n \to \mathcal{R}^+$ is monotone if*
$$h(\mathbf{x}) \le h(\mathbf{y}) \quad \forall \mathbf{x}, \mathbf{y} \in (\mathcal{R}^+)^n \ \text{ such that } \ \mathbf{x} \le \mathbf{y} \ .$$

2. *A function $h : (\mathcal{R}^+)^n \to \mathcal{R}^+$ dominates a function $\tilde{h} : (\mathcal{R}^+)^n \to \mathcal{R}^+$ if there exists a constant η such that*
$$\tilde{h}(\mathbf{x}) \le \eta h(\mathbf{x}) \quad \forall \mathbf{x} \in (\mathcal{R}^+)^n \ .$$

3. *A function $h : (\mathcal{R}^+)^n \to \mathcal{R}^+$ is Lipschitz continuous if there exists a constant M such that*
$$|h(\mathbf{x}) - h(\mathbf{y})| \le M \|\mathbf{x} - \mathbf{y}\|_\infty \quad \forall \mathbf{x}, \mathbf{y} \in (\mathcal{R}^+)^n \ .$$

4. *A function $h : (\mathcal{R}^+)^n \to \mathcal{R}^+$ is harmonic if*
$$h(\mathbf{x}) \ge h(\bar{\mathbf{x}}) \quad \forall \mathbf{x} \in (\mathcal{R}^+)^n \ ,$$
where $\bar{\mathbf{x}} = (\bar{x}, \ldots, \bar{x})$ and $\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^n \mathbf{x}_i$.

5. *A function $h : (\mathcal{R}^+)^n \to \mathcal{R}^+$ is symmetric if*
$$h(\mathbf{x}) = h(P\mathbf{x}) \quad \forall \mathbf{x} \in (\mathcal{R}^+)^n \ ,$$
where P is an arbitrary permutation matrix, namely, $P_{i,j} = \delta_{i,\pi(j)}$ for some permutation $\pi \in S_n$.

Assumption 1 *The function $f : (\mathcal{R}^+)^m \to \mathcal{R}^+$ is:*

1. *monotone;*

2. *harmonic;*

3. *linear with respect to scalar multiplications, i.e., $f(c\mathbf{x}) = cf(\mathbf{x})$ for all $c \in \mathcal{R}^+$ and $\mathbf{x} \in (\mathcal{R}^+)^m$;*

4. *dominating the max norm with a domination factor η_f that is independent of m;*

5. *Lipschitz continuous with a constant M_f that is independent of m;*

6. *symmetric.*

Assumption 2 *The function $g : (\mathcal{R}^+)^d \to \mathcal{R}^+$ is:*

1. *monotone;*
2. *convex;*
3. *stable under scalar multiplications in the sense that $g(c\mathbf{x}) \le \alpha(c)g(\mathbf{x})$ for all $c \in \mathcal{R}^+$ and $\mathbf{x} \in (\mathcal{R}^+)^d$, where $\alpha(c)$ is such that $\lg\lg\alpha(c)$ is bounded by a polynomial in c for large values of c;*
4. *dominating the ℓ_∞ and the ℓ_1 norms with a domination factor η_g;*
5. *Lipschitz continuous with a constant M_g.*

Next, we see what functions comply with the above assumptions. It turns out that while for f we do not get too far from the popular max function, when it comes to g we do manage to enlarge substantially the class of functions.

When the number of machines m is not regarded as part of the input (i.e., the input consists only of the n vectors), the constants η_f and M_f in conditions 4 and 5 in Assumption 1 may depend on m. In that case our results apply to all ℓ_p norms. However, we do wish to consider m as part of the input. The problem then stems from the conjunction of conditions 4 and 5. All ℓ_p norms comply with the first four conditions in Assumption 1, with $\eta_f = 1$ in condition 4; however, they are Lipschitz continuous with a constant $M_f = m^{1/p}$ that depends on m for all $p < \infty$. Unfortunately, there is no normalization of f that would make both constants η_f and M_f independent of m. Also, it seems that both conditions 4 and 5 in Assumption 1 are essential – the first one for limiting the possible values of load vectors in an optimal solution, and the second to enable approximations. Hence, the only ℓ_p norm that satisfies Assumption 1 is the ℓ_∞ norm (as discussed towards the end of §2, this is the only reasonable choice of an ℓ_p norm in the context of the line-up problem). Other functions f for which our results apply are the ℓ_p norms taken on the t largest values in the argument vector, where $t = \min(m_0, m)$ for some constant m_0; e.g., the sum of the largest two components.

As for g, here we have no problem since the dimension of the space on which g acts, d, is considered as a constant. Hence, all ℓ_p norms (even weighted norms) work here. Another natural choice of g for which Assumption 2 holds is

$$g(\mathbf{v}) = \|\max\{\mathbf{v}, \mathbf{c}\}\| \tag{3}$$

where $\mathbf{c}$ is a constant vector and the outer norm is a monotone norm. Later on, in §5, we get rid of the monotonicity condition, Assumption 2-1, and this allows us to include also all Sobolev norms, e.g., $g(\boldsymbol{\ell}) = \|\boldsymbol{\ell}\|_{1,p}$. We note that our analysis does not apply to the choices of $g(\boldsymbol{\ell})$ given in examples 5 and 6 in §2. Those functions do not dominate the ℓ_∞ norm since they may vanish for $\boldsymbol{\ell} \ne 0$. Therefore, the optimal solution may have a zero cost in that case. As PTAS aim at finding solutions whose cost is larger than that of an optimal solution by a *multiplicative* factor close to 1, it is not relevant to talk about a PTAS in this case.

It is interesting to note that the set of functions that comply with either Assumption 1 or 2 is closed under positive linear combinations. For example, if f_1 and f_2 satisfy Assumption 1, so would $c_1 f_1 + c_2 f_2$ for all $c_1, c_2 > 0$.

4 Monotone Target Functions

In this section we present our analysis under the assumptions in §3. In particular, both f and g are monotone. Our algorithm is similar to that of Chekuri and Khanna [5] who studied the case $f = g = \max$.

Let S^o denote henceforth an optimal solution to the problem. We design a PTAS that, given $0 < \varepsilon \leq 1$, finds a solution S for which

$$F(S) \leq (1 + \text{Const} \cdot \varepsilon) F(S^o) \ , \tag{4}$$

where the constant depends solely on d, f and g.

4.1 Preprocessing the vectors by means of truncation

Let I be an instance of the VAP with vectors $\mathbf{x}^i$, $1 \leq i \leq n$. We modify I into another problem $\hat{I}$ where the vectors $\hat{\mathbf{x}}^i$ are defined as follows:

$$\hat{\mathbf{x}}^i_j = \begin{cases} \mathbf{x}^i_j & \text{if } \ \mathbf{x}^i_j \geq \delta \|\mathbf{x}^i\|_\infty \\ 0 & \text{otherwise} \end{cases} \qquad 1 \leq i \leq n \ , \ 1 \leq j \leq d \ ; \tag{5}$$

here, δ is an arbitrary constant, $0 \leq \delta \leq 1$.

Lemma 1 *Let S be a solution to I and let $\hat{S}$ be the corresponding solution to $\hat{I}$. Then*

$$F(S) \leq (1 + C\delta) F(\hat{S}) \tag{6}$$

where C is a constant that depends only on g.

Proof. Let $\boldsymbol{\ell}^k$ and $\hat{\boldsymbol{\ell}}^k$, $1 \leq k \leq m$, denote the load vectors in S and $\hat{S}$ respectively. In view of (5),

$$\hat{\boldsymbol{\ell}}^k \leq \boldsymbol{\ell}^k \leq \hat{\boldsymbol{\ell}}^k + \delta \sum_{S(i)=k} \|\mathbf{x}^i\|_\infty \tag{7}$$

Since $\|\mathbf{x}^i\|_\infty = \|\hat{\mathbf{x}}^i\|_\infty \leq \|\hat{\mathbf{x}}^i\|_1$ we conclude that $\sum_{S(i)=k} \|\mathbf{x}^i\|_\infty \leq \|\hat{\boldsymbol{\ell}}^k\|_1$. As g dominates the max norm, Assumption 2-4, we get that

$$\sum_{S(i)=k} \|\mathbf{x}^i\|_\infty \leq \eta_g g(\hat{\boldsymbol{\ell}}^k) \ . \tag{8}$$

Therefore, by (7) and (8),

$$\hat{\boldsymbol{\ell}}^k \leq \boldsymbol{\ell}^k \leq \hat{\boldsymbol{\ell}}^k + \delta \eta_g g(\hat{\boldsymbol{\ell}}^k) \ . \tag{9}$$

Next, by the Lipschitz continuity of g, Assumption 2-5, we conclude that

$$g(\boldsymbol{\ell}^k) \leq (1 + \eta_g M_g \delta) g(\hat{\boldsymbol{\ell}}^k) \ . \tag{10}$$

Finally, we invoke the monotonicity of f and its linear dependence on scalar multiplications to conclude that

$$F(S) = f(g(\boldsymbol{\ell}^k))_{1 \leq k \leq m} \leq f((1 + \eta_g M_g \delta) g(\hat{\boldsymbol{\ell}}^k))_{1 \leq k \leq m} = (1 + \eta_g M_g \delta) F(\hat{S}) \ . \tag{11}$$

This proves (6). $\square$

As our goal is to find solutions to the VAP whose value differ from the optimal value by a multiplicative factor of no more than $(1 + \text{Const} \cdot \varepsilon)$, we assume henceforth that the input vectors $\mathbf{x}^i$, $1 \le i \le n$, were subject to the above truncation procedure with

$$\delta = \frac{\varepsilon}{\eta_g M_g} \ , \tag{12}$$

so that, in view of (11), the multiplicative contribution of this procedure is bounded by $(1+\varepsilon)$.

4.2 Lower and upper bounds

Let $\mathbf{v}$ denote the average load on each machine, $\mathbf{v} = \frac{1}{m}\sum_{i=1}^{n} \mathbf{x}^i$, and let S be an arbitrary solution with load vectors $\boldsymbol{\ell}^k$, $1 \le k \le m$. Then, as f is harmonic, Assumption 1-2,

$$f(g(\boldsymbol{\ell}^1), \dots, g(\boldsymbol{\ell}^m)) \ge f(t, \dots, t) \quad \text{where} \ \ t = \frac{1}{m}\sum_{k=1}^{m} g(\boldsymbol{\ell}^k) \ . \tag{13}$$

Since g is convex, Assumption 2-2, $t \ge g(\mathbf{v})$. Hence, by (13), the monotonicity of f and its linear dependence on scalar multiplications, we arrive at the following lower bound:

$$f(g(\boldsymbol{\ell}^1), \dots, g(\boldsymbol{\ell}^m)) \ge \lambda := g(\mathbf{v}) \cdot f(1, \dots, 1) \ . \tag{14}$$

To obtain an upper bound, we consider the solution that assigns all n vectors to the first machine. That creates a load of $m\mathbf{v}$ on that machine and a zero load on all other machines. Consequently, we get the upper bound

$$\Lambda := f(g(m\mathbf{v}), 0, \dots, 0) = g(m\mathbf{v}) \cdot f(1, 0, \dots, 0) \ . \tag{15}$$

Finally, we note that the ratio between the upper and lower bounds may be bounded by

$$\frac{\Lambda}{\lambda} = \frac{g(m\mathbf{v}) \cdot f(1, 0, \dots, 0)}{g(\mathbf{v}) \cdot f(1, \dots, 1)} \le \frac{g(m\mathbf{v})}{g(\mathbf{v})} \le 2^{2^{P(m)}} \tag{16}$$

for some polynomial $P(m)$; here, the first inequality is due to the monotonicity of f, Assumption 1-1, while the second one is due to the stability of g under scalar multiplications, Assumption 2-3.

Before turning to describe our scheme, we define the following key term.

Definition 2 *For any value Φ, a solution S to the VAP is called a Φ-solution if $F(S) \le \Phi$.*

4.3 Overview of the scheme

The main ingredient in our PTAS is the *core algorithm.* This algorithm receives a test value Φ and a tolerance value $0 < \varepsilon \le 1$; it returns one of the following two values:

1. **TRUE**. In that case the algorithm returns also a solution S such that

$$F(S) \le (1 + T\varepsilon)\Phi \ , \tag{17}$$

   for some constant $T > 0$ that depends only on d, f and g (the exact value of T is given later on). If the given VAP has a Φ-solution, the algorithm is guaranteed to return a **TRUE** value.

2. **FALSE**. In that case we may conclude that there are no Φ-solutions.

The running time of the core algorithm is polynomial in n and m.

With this core algorithm, one may execute a *geometrical* binary search to approximate the optimal value Φ^o to within a small multiplicative factor. Namely, we may start with $\Phi^- = \lambda$ and $\Phi^+ = \Lambda$, (14)-(15), and execute a geometrical binary search to yield tighter bounds Φ^- and Φ^+ (the former relates to values for which the core algorithm returned **FALSE**, the latter relates to values for which we got **TRUE**) such that

$$\Phi^- \leq \Phi^o \leq (1+T\varepsilon)\Phi^+ \quad \text{and} \quad \frac{\Phi^+}{\Phi^-} \leq 1+\varepsilon \ . \tag{18}$$

If $\hat{S}$ is the solution that the core algorithm returned for the test value Φ^+, then it is a good approximation to the optimal solution because, by (17) and (18),

$$F(\hat{S}) \leq (1+T\varepsilon)\Phi^+ \leq (1+T\varepsilon)(1+\varepsilon)\Phi^o \ . \tag{19}$$

Going back to the original vectors, prior to the truncation procedure described in §4.1, we get a solution S that satisfies

$$F(S) \leq (1+T\varepsilon)(1+\varepsilon)^2\Phi^o \ . \tag{20}$$

Since $0 < \varepsilon \leq 1$, (20) implies (4) with Const $= 4T+3$.

As for the number of steps in the geometrical binary search: the search begins with the interval $[\lambda, \Lambda]$ where the ratio between the two end points is bounded by $2^{2^{P(m)}}$, (16), $P(m)$ being a polynomial. This implies that the termination condition (18) will occur after no more than

$$[P(m) - \lg\lg(1+\varepsilon)] \tag{21}$$

steps.

4.4 The core algorithm

Let Φ be the given test value. Assume that there exists a Φ-solution S and let $\boldsymbol{\ell}^k$, $1 \leq k \leq m$, denote the load vectors in that solution. Hence,

$$f(g(\boldsymbol{\ell}^k))_{1\leq k\leq m} \leq \Phi \ .$$

Since f dominates the max norm, Assumption 1-4, we conclude that

$$\max_{1\leq k\leq m} g(\boldsymbol{\ell}^k) \leq \eta_f \cdot f(g(\boldsymbol{\ell}^k))_{1\leq k\leq m} \ .$$

Furthermore, as g dominates the max-norm too, Assumption 2-4,

$$\max_{1\leq j\leq d} \boldsymbol{\ell}_j^k \leq \eta_g \cdot g(\boldsymbol{\ell}^k) \quad 1 \leq k \leq m \ .$$

Putting the last three inequalities together we arrive at the conclusion that all components of all load vectors in a Φ-solution are upper bounded as follows,

$$\boldsymbol{\ell}^k \leq \eta_f\eta_g\Phi \quad 1 \leq k \leq m \ . \tag{22}$$

This upper bound enables us to determine the range in which we would search for Φ-solutions. Next, we decompose the set of input vectors $\{\mathbf{x}^i\}_{1\le i\le n}$ into two disjoint subsets: the subset of large vectors,

$$\mathcal{L} = \{\mathbf{x}^i \ : \|\mathbf{x}^i\|_\infty \ge \Phi\varepsilon,\ 1 \le i \le n\} \ , \tag{23}$$

and the complement set of small vectors,

$$\mathcal{S} = \{\mathbf{x}^i \ : \|\mathbf{x}^i\|_\infty < \Phi\varepsilon,\ 1 \le i \le n\} \ . \tag{24}$$

Let $S \in \{1,\dots,m\}^{\{1,\dots,n\}}$ be a solution of the VAP and let $\boldsymbol{\ell}^k$, $1 \le k \le m$, be its corresponding load vectors. Those load vectors may be decomposed into $\boldsymbol{\ell}^k = \mathbf{a}^k + \mathbf{b}^k$, where $\mathbf{a}^k$ equals the sum of large vectors $\mathbf{x}^i \in \mathcal{L}$ that were assigned to the kth machine, while $\mathbf{b}^k$ denotes the sum of the small vectors $\mathbf{x}^i \in \mathcal{S}$ in that machine. We consider the following discretizations of those vectors:

$$\hat{\mathbf{a}}^k = \mathcal{P}\mathbf{a}^k \ , \ \hat{\mathbf{b}}^k = \mathcal{P}\mathbf{b}^k \qquad \text{where} \quad (\mathcal{P}\mathbf{v})_j := \left\lceil \frac{\mathbf{v}_j}{\Phi\varepsilon} \right\rceil \Phi\varepsilon \ , \ 1 \le j \le d \ . \tag{25}$$

Hence, we may associate with any solution S and any k, $1 \le k \le m$, a pair of vectors $(\hat{\mathbf{a}}^k, \hat{\mathbf{b}}^k)$ that describes, in a discretized manner, the load configuration in the kth machine in that solution. We shall refer to such a pair as a *Discretized Load Configuration*, or *DLC*, and will denote it by $\mathbf{DLC}^k$. Furthermore, the two vectors of which it consists will be denoted $\mathbf{DLC_a}^k$ and $\mathbf{DLC_b}^k$, while $\mathbf{DLC}_{\boldsymbol{\ell}}{}^k := \mathbf{DLC_a}^k + \mathbf{DLC_b}^k$.

Lemma 2 *Let S be a Φ-solution for some test value $\Phi > 0$. Let $\boldsymbol{\ell}^k$, $1 \le k \le m$, denote its load vectors and let $\mathbf{DLC}^k$ denote its corresponding DLCs. Then:*
(a) $\mathbf{DLC}^k$, $1 \le k \le m$, take values in a finite set $\Omega = \Omega(\Phi, \varepsilon)$ of size $(\mu+1)^{2d}$, where

$$\mu := \left\lceil \frac{\eta_f \eta_g}{\varepsilon} \right\rceil \ . \tag{26}$$

(b) The following inequality holds:

$$f(g(\mathbf{DLC}_{\boldsymbol{\ell}}{}^k))_{1\le k\le m} \le (1 + 2M_f M_g \varepsilon)\Phi \ . \tag{27}$$

Proof. In view of (22) and the discretization process, all components of both $\mathbf{DLC_a}^k$ and $\mathbf{DLC_b}^k$ take values in the set $\{k\Phi\varepsilon \ : \ 0 \le k \le \mu\}$ where μ is given in (26); this proves part (a) of the lemma. As $0 \le \hat{\boldsymbol{\ell}}^k - \boldsymbol{\ell}^k \le 2\Phi\varepsilon$, the Lipschitz continuity of f and g imply that $f(g(\mathbf{DLC}_{\boldsymbol{\ell}}{}^k))_{1\le k\le m} \le f(g(\boldsymbol{\ell}^k))_{1\le k\le m} + 2M_f M_g \Phi\varepsilon$. This inequality implies (27) due to the assumption that the solution S with load vectors $\boldsymbol{\ell}^k$ is a Φ-solution. $\Box$

The basic idea now is to go over all possible assignments of DLCs from the set Ω to the m machines and to look for an assignment that satisfies (27) – a necessary condition for the existence of Φ-solutions. To this end, we order the set Ω,

$$\Omega = \{DLC_1, \dots, DLC_t \ : \ t = (\mu+1)^{2d}\} \ . \tag{28}$$

Each assignment of DLCs from Ω to the m machines may be uniquely described by an *Assignment Configuration Vector*, or *ACV*, of the following form:

$$\mathbf{ACV} = (m_1, \dots, m_t) \quad \text{where } 0 \le m_\tau \le m, \quad 1 \le \tau \le t \ \text{ and } \sum_{\tau=1}^{t} m_\tau = m \ . \tag{29}$$

The number of ACVs is bounded by m^t. As t is a constant that depends solely on d, f, g and ε, the number of ACVs to consider is polynomial in m.

The Core Algorithm

Perform a loop over all ACVs. For each ACV do as follows:

1. Assign a DLC to each machine in an arbitrary manner according to the present ACV (an arbitrary assignment is allowed since the target function is symmetric in its m arguments, Assumption 1-6). Let $\hat{\mathbf{a}}^k$, $\hat{\mathbf{b}}^k$ and $\hat{\boldsymbol{\ell}}^k = \hat{\mathbf{a}}^k + \hat{\mathbf{b}}^k$ denote the vectors $\mathbf{DLC_a}^k$, $\mathbf{DLC_b}^k$ and $\mathbf{DLC}_{\boldsymbol{\ell}}{}^k$, $1 \le k \le m$.

2. If $f(g(\hat{\boldsymbol{\ell}}^k))_{1\le k\le m} > (1 + 2M_f M_g \varepsilon)\Phi$ skip to the next ACV.

3. Exercise algorithm A_L, described in §4.5. If it succeeds it returns an assignment of the large $\mathcal{L}$-vectors to the m machines, $A_L : \mathcal{L} \to \{1, \ldots, m\}$, such that
$$\mathbf{a}^k := \sum_{i\in\mathcal{L}, A_L(i)=k} \mathbf{x}^i \le (1+\varepsilon)\hat{\mathbf{a}}^k \quad \forall k,\ 1 \le k \le m\ . \tag{30}$$
The algorithm fails only if there is no assignment $A_L : \mathcal{L} \to \{1, \ldots, m\}$ for which
$$\mathbf{a}^k := \sum_{i\in\mathcal{L}, A_L(i)=k} \mathbf{x}^i \le \hat{\mathbf{a}}^k \quad \forall k,\ 1 \le k \le m\ . \tag{31}$$
In that case we skip to the next ACV.

4. Exercise algorithm A_S, described in §4.6. If it succeeds, it returns an assignment of the small $\mathcal{S}$-vectors to the m machines, $A_S : \mathcal{S} \to \{1, \ldots, m\}$, such that
$$\mathbf{b}^k := \sum_{i\in\mathcal{S}, A_S(i)=k} \mathbf{x}^i \le \hat{\mathbf{b}}^k + d\Phi\varepsilon \quad \forall k,\ 1 \le k \le m\ . \tag{32}$$
The algorithm fails only if there is no assignment $A_S : \mathcal{S} \to \{1, \ldots, m\}$ for which
$$\mathbf{b}^k := \sum_{i\in\mathcal{S}, A_S(i)=k} \mathbf{x}^i \le \hat{\mathbf{b}}^k \quad \forall k,\ 1 \le k \le m\ . \tag{33}$$
In that case we skip to the next ACV.

5. If this point in the algorithm is reached then we found an ACV for which both algorithm A_L and algorithm A_S were successful. We combine the assignments A_L and A_S in (30)-(32) into a solution $S : \{1, \ldots, n\} \to \{1, \ldots, m\}$,
$$S|_{\mathcal{L}} = A_L \quad , \quad S|_{\mathcal{S}} = A_S\ . \tag{34}$$
It is shown in Theorem 1 below that this solution satisfies (17) with
$$T = \eta_g M_g + (d+2)M_f M_g + 2\eta_g M_f M_g^2\ , \tag{35}$$
for all $0 < \varepsilon \le 1$. Return this solution together with a **TRUE** value.

If we found no ACV for which both algorithm A_L and algorithm A_S were successful, then the only conclusion is that there are no Φ-solutions, as shown in Theorem 2. In that case, the algorithm returns a **FALSE** value.

Theorem 1 *Assume that for some ACV, algorithm A_L succeeded with an assignment $A_L : \mathcal{L} \to \{1, \ldots, m\}$ that satisfies (30) and algorithm A_S succeeded with an assignment $A_S : \mathcal{S} \to \{1, \ldots, m\}$ that satisfies (32). Then the solution $S : \{1, \ldots, n\} \to \{1, \ldots, m\}$ defined by (34) satisfies (17) with T given in (35) for all $0 < \varepsilon \le 1$.*

Proof. Let $\hat{\mathbf{a}}^k$, $\hat{\mathbf{b}}^k$ and $\hat{\boldsymbol{\ell}}^k = \hat{\mathbf{a}}^k + \hat{\mathbf{b}}^k$ denote the vectors $\mathbf{DLC_a}^k$, $\mathbf{DLC_b}^k$ and $\mathbf{DLC}_{\boldsymbol{\ell}}{}^k$ that are associated with the kth machine, $1 \le k \le m$, by the ACV. In view of Step 2 in the core algorithm,

$$f(g(\hat{\boldsymbol{\ell}}^k))_{1\le k\le m} \le (1 + 2M_f M_g \varepsilon)\Phi \ . \tag{36}$$

Let $\mathbf{a}^k$ and $\mathbf{b}^k$ be as defined in (30) and (32) and let $\boldsymbol{\ell}^k := \mathbf{a}^k + \mathbf{b}^k$ be the load vectors in the solution S, (34). Then, by (30) and (32),

$$\boldsymbol{\ell}^k \le (1+\varepsilon)\hat{\boldsymbol{\ell}}^k + d\Phi\varepsilon \quad \forall k \ .$$

Hence, as g is monotone,

$$g(\boldsymbol{\ell}^k) \le g\left(\hat{\boldsymbol{\ell}}^k + \left(\hat{\boldsymbol{\ell}}^k + d\Phi\right)\varepsilon\right) \quad \forall k \ .$$

We now invoke the Lipschitz continuity of g, Assumption 2-5, to conclude that

$$g(\boldsymbol{\ell}^k) \le g(\hat{\boldsymbol{\ell}}^k) + \left(\|\hat{\boldsymbol{\ell}}^k\|_\infty + d\Phi\right) M_g\varepsilon \quad \forall k \ .$$

As g dominates the max norm, Assumption 2-4, we conclude from the last inequality that

$$g(\boldsymbol{\ell}^k) \le (1 + \eta_g M_g \varepsilon) g(\hat{\boldsymbol{\ell}}^k) + dM_g\Phi\varepsilon \quad \forall k \ . \tag{37}$$

Next, we apply the function f on both sides of (37). Due to the monotonicity and Lipschitz continuity of f we get that

$$f(g(\boldsymbol{\ell}^k))_{1\le k\le m} \le f\left((1 + \eta_g M_g\varepsilon)g(\hat{\boldsymbol{\ell}}^k)\right)_{1\le k\le m} + dM_f M_g\Phi\varepsilon \ .$$

As f depends linearly on scalar multiplications, Assumption 1-3, we get that

$$f(g(\boldsymbol{\ell}^k))_{1\le k\le m} \le (1 + \eta_g M_g \varepsilon) f(g(\hat{\boldsymbol{\ell}}^k))_{1\le k\le m} + dM_f M_g \Phi\varepsilon \ . \tag{38}$$

Finally, (38) together with (36) yield the desired bound,

$$f(g(\boldsymbol{\ell}^k))_{1\le k\le m} \le (1 + \eta_g M_g\varepsilon)(1 + 2M_f M_g\varepsilon)\Phi + dM_f M_g\Phi\varepsilon \le (1 + T\varepsilon)\Phi \quad \forall \varepsilon \le 1 \ ,$$

where T is given in (35). That completes the proof. $\square$

Theorem 2 *Assume that there was no ACV for which both algorithm A_L and algorithm A_S were successful. Then the VAP has no Φ-solutions.*

Proof. Assume that a Φ-solution does exist, S'. Let $\boldsymbol{\ell}^k = \mathbf{a}^k + \mathbf{b}^k$, $1 \le k \le m$, be the load vectors in that solution and their decomposition to the large and small parts, respectively. Let $\{(\hat{\mathbf{a}}^k, \hat{\mathbf{b}}^k) \ : \ 1 \le k \le m\}$ be the set of DLCs of that solution, (25). This set is described by some ACV. Then when the core algorithm would have reached that ACV in the loop both algorithm A_L and algorithm A_S should have succeeded with $A_L = S'|_{\mathcal{L}}$ and $A_S = S'|_{\mathcal{S}}$; in fact, the resulting loads $\mathbf{a}^k$ and $\mathbf{b}^k$ would have even satisfied the tighter bounds (31)+ (33) rather than (30)+ (32). As that did not occur, we conclude that there are no Φ-solutions. □

4.5 Algorithm A_L : Assigning the large vectors

The input to this algorithm is:

- the set $\mathcal{L}$ of large $\mathbf{x}^i$ vectors;
- vectors $\hat{\mathbf{a}}^k$, $1 \le k \le m$.

The algorithm aims at finding an assignment $A_L : \mathcal{L} \to \{1, \ldots, m\}$ such that (30) holds. Assume that Φ-solutions exist. Then (22) provides an upper bound for all vectors $\mathbf{x}^i$, $1 \le i \le n$. On the other hand, the definition of $\mathcal{L}$, (23), and the truncation procedure, (5)+(12), provide a lower bound for all nonzero components of $\mathbf{x}^i \in \mathcal{L}$. Those bounds are given by:

$$\frac{\Phi\varepsilon^2}{\eta_g M_g} \le \mathbf{x}^i_j \le \eta_f \eta_g \Phi \qquad \forall \mathbf{x}^i \in \mathcal{L} \ \text{ and } \ \mathbf{x}^i_j > 0 \ . \tag{39}$$

Next, as in [5], we define a geometric mesh on the interval given in (39):

$$\xi_0 = \frac{\Phi\varepsilon^2}{\eta_g M_g} \quad ; \quad \xi_i = (1+\varepsilon)\xi_{i-1} \quad , \quad 1 \le i \le q \quad ; \quad q := \left\lfloor \frac{\lg(\eta_f \eta_g^2 M_g \varepsilon^{-2})}{\lg(1+\varepsilon)} \right\rfloor + 1 \ . \tag{40}$$

In view of the above, every nonzero component of $\mathbf{x}^i \in \mathcal{L}$ lies in an interval $[\xi_{i-1}, \xi_i)$ for some $1 \le i \le q$. We use this in order to define a new set of vectors, $\hat{\mathcal{L}} = \{\hat{\mathbf{x}}^i = \mathcal{H}\mathbf{x}^i \ : \ \mathbf{x}^i \in \mathcal{L}\}$, where the operator $\mathcal{H}$ replaces each nonzero component in the vector on which it operates by the left end point of the interval $[\xi_{i-1}, \xi_i)$ where it lies. The vectors in $\hat{\mathcal{L}}$ may take

$$s = (q+1)^d - 1 \tag{41}$$

values. Hence, any assignment of vectors from $\hat{\mathcal{L}}$ to a machine may be described by an assignment vector $(k_1, \ldots, k_s)$ where k_j equals the number of vectors of the jth type that were assigned to that machine. Next we observe that

$$\sum_{j=1}^{s} k_j \le \eta_f \eta_g d \cdot \frac{1+\varepsilon}{\varepsilon} \ . \tag{42}$$

This is because the ℓ_∞ norm of each vector in $\hat{\mathcal{L}}$ is at least $\Phi\varepsilon/(1+\varepsilon)$, (23), and because the vectors $\hat{\mathbf{a}}^k$ are bounded from above by $\eta_f\eta_g\Phi$, (22). Hence, the number of assignment vectors is bounded from above by

$$R := \left(\eta_f\eta_g d \cdot \frac{1+\varepsilon}{\varepsilon}\right)^s \ , \tag{43}$$

a constant that depends only on f, g, d and ε. Applying standard dynamic programming techniques, we may run a simple algorithm that either finds an assignment of the $\hat{\mathcal{L}}$ vectors to the m machines such that the resulting load on the kth machine is not larger than $\hat{\mathbf{a}}^k$ (success), or decides that such an assignment does not exist (failure). The running time of that algorithm is $\mathcal{O}(Rmn^s)$, where R is given above. The reader is referred to [5, Lemma 2.3] for more details. Clearly, if such an assignment is found, then applying it to the original $\mathcal{L}$ vectors would result in an assignment that satisfies (30).

4.6 Algorithm A_S : Assigning the small vectors

The input to this algorithm is:

- the set $\mathcal{S}$ of small $\mathbf{x}^i$ vectors;
- vectors $\hat{\mathbf{b}}^k$, $1 \le k \le m$.

The algorithm aims at finding an assignment $A_S : \mathcal{S} \to \{1,\ldots,m\}$ such that (32) holds.

As in §4.5, we employ similar techniques to those used in [5]. For convenience reasons, we assume that $\mathbf{x}^i$, $1 \le i \le n$, are ordered in a non-decreasing order according to their ℓ_∞ norm. Therefore, $\mathcal{S} = \{\mathbf{x}^i : \ 1 \le i \le |\mathcal{S}|\}$. We define indicator variables ξ_i^k, $1 \le i \le |\mathcal{S}|$, $1 \le k \le m$, such that $\xi_i^k = 1$ if $\mathbf{x}^i$ is assigned to the kth machine and $\xi_i^k = 0$ otherwise. Hence, the algorithm aims at finding $\xi_i^k \in \{0,1\}$ such that:

$$\sum_{i=1}^{|\mathcal{S}|} \xi_i^k \mathbf{x}_j^i \le \hat{\mathbf{b}}_j^k \quad 1 \le k \le m \ \ , \ \ 1 \le j \le d \tag{44}$$

and

$$\sum_{k=1}^{m} \xi_i^k = 1 \quad 1 \le i \le |\mathcal{S}| \ . \tag{45}$$

In order to turn this into a linear programming problem, we replace the constraint $\xi_i^k \in \{0,1\}$ by

$$\xi_i^k \ge 0 \quad 1 \le i \le |\mathcal{S}| \ , \ 1 \le k \le m \ . \tag{46}$$

Let us denote (44)-(46) by *(LP)*. *(LP)* is a linear programming problem consisting of $(md + |\mathcal{S}| + m|\mathcal{S}|)$ constraints in the $m|\mathcal{S}|$ unknowns ξ_i^k. A *basic solution* to this problem is an extreme solution in the sense that it is a vertex in the polyhedron determined by *(LP)*. Such basic solutions are identified by the property that the number of equalities that they satisfy is at least as the number of unknowns. Hence, any integral solution, $\xi_i^k \in \{0,1\}$, is a basic solution, as can be seen from (45)+(46).

We may now apply a standard polynomial time algorithm to find a basic solution for this problem. If such a solution was not found, the algorithm fails. Otherwise, as this basic solution satisfies (46) rather than the integrality conditions $\xi_i^k \in \{0,1\}$, the algorithm proceeds to the next stage of translating that fractional solution to an integral one. Any basic solution of *(LP)* satisfies at least $m|\mathcal{S}|$ equalities, or, equivalently, at most $(md + |\mathcal{S}|)$ inequalities. Therefore, there could be at most $(md+|\mathcal{S}|)$ indicator variables that satisfy (46) with an inequality. Since there is at least one positive ξ_i^k for each $1 \le i \le |\mathcal{S}|$ we arrive at the conclusion that there could be no more than md vectors that are assigned fractionally to more than one machine. In view of the above, we spread those vectors in an arbitrary manner among the m machines such that each machine gets no more than d vectors of that sort. This is the solution that the algorithm returns. It is easy to see why this solution satisfies (32): as the ℓ_∞ norm of all small vectors is bounded by $\Phi\varepsilon$, this spreading might increase the upper bound in (44) by no more than $d\Phi\varepsilon$.

4.7 Overall complexity of the scheme

We summarize herein the steps of the entire scheme and the (worst case) cost of each step:

1. The outer loop executes the geometrical binary search. The number of steps in that search is $[P(m) - \lg\lg(1+\varepsilon)]$, (21), where $P(m)$ is the polynomial given in (16).

2. In each step in the binary search we perform the core algorithm. The core algorithm executes a loop of length m^t at the most where $t = \left(\left\lceil\frac{\eta_f\eta_g}{\varepsilon}\right\rceil + 1\right)^{2d}$, (28)+(26). In some steps, both algorithms A_L and A_S will be executed; in others, only algorithm A_L will be executed or none (depending on the conditions that are verified in steps 2 and 3 in the core algorithm).

3. The running time of algorithm A_L is bounded by $O(Rmn^s)$ where
$$R = \left(\eta_f\eta_g d \cdot \frac{1+\varepsilon}{\varepsilon}\right)^s \quad \text{and} \quad s = \left(\left\lfloor \frac{\lg(\eta_f\eta_g^2 M_g \varepsilon^{-2})}{\lg(1+\varepsilon)} \right\rfloor + 2\right)^d - 1 \ ,$$
(43), (41) and (40).

4. The running time of the linear programming algorithm A_S is polynomial in the length of the input (i.e. in the number of bits in the input).

5 Non-monotone Target Functions

In this section we extend our results to include also cost functions g that are not monotone; namely, we remove Assumption 2-1. We elect to separate the discussion of this case from §4 in order not to overload the presentation of our scheme with the technical details that follow.

The monotonicity of g was needed only in the proof of Theorem 1. The reason why that monotonicity was necessary is manifested in the one sided estimates (30) and (32): the core algorithm scanned all possible ACVs for which the resulting target function value was in accord with the current test value Φ, (36). It then used algorithms A_L and A_S to find a solution that yields loads lower than the loads dictated by the current ACV. When both f and g are monotone, such load distributions give even lower costs than the cost of the ACV. However, when g is not monotone, this is not enough. We need two-sided estimates in order to use the Lipschitz continuity of g. Such a two-sided estimate may be obtained for the output of algorithm A_L, but not for the output of algorithm A_S. Hence, this is how we plan to overcome this difficulty:

1. We redefine the subsets $\mathcal{S}$ and $\mathcal{L}$ of small and large vectors.
2. We replace $\mathcal{S}$ with another set $\tilde{\mathcal{S}}$ of vectors that are large according to the new definition, and we show that the two instances of the VAP – $\mathcal{S} \cup \mathcal{L}$ and $\tilde{\mathcal{S}} \cup \mathcal{L}$ – are close in terms of their cost.
3. In view of the above, we may assume that all vectors are large, according to the new definition. We describe how to modify the core algorithm and algorithm A_L in wake of those changes.
4. We finally return to Theorem 1 and prove it without using the monotonicity of g.

Step 1.

Every test value Φ induced a decomposition of the set of input vectors $\{\mathbf{x}^i\}_{1\le i\le n}$ into the subset $\mathcal{L}$ of large vectors (23) and the complement set of small vectors $\mathcal{S}$ (24). We modify those definitions as follows:

$$\mathcal{L} = \{\mathbf{x}^i \ : \|\mathbf{x}^i\|_\infty \ge \Phi\varepsilon^{2d+1},\ 1 \le i \le n\} \ , \tag{47}$$

$$\mathcal{S} = \{\mathbf{x}^i \ : \|\mathbf{x}^i\|_\infty < \Phi\varepsilon^{2d+1},\ 1 \le i \le n\} \ . \tag{48}$$

What we present below is a technique to replace $\mathcal{S}$ with another set of vectors $\tilde{\mathcal{S}} = \{\mathbf{z}^1, \dots, \mathbf{z}^{\tilde{\nu}}\}$ where

$$\tilde{\nu} = |\tilde{\mathcal{S}}| \le \nu = |\mathcal{S}| \quad \text{and} \quad \|\mathbf{z}^i\|_\infty = \Phi\varepsilon^{2d+1} \quad 1 \le i \le \tilde{\nu} \ . \tag{49}$$

In other words, all vectors in $\tilde{\mathcal{S}}$ are large (47). That would allow us to treat all $n - \nu + \tilde{\nu}$ vectors in the same manner. We also show that the original and new problems are close in the sense that to every solution of one there exists a solution of the other such that the ratio between their costs is within $\mathrm{Const} \cdot \varepsilon$ from 1.

Step 2.

Let $\mathbf{x} \in \mathcal{S}$. Then, in view of the truncation procedure (5),

$$\delta \le \frac{\mathbf{x}_j}{\|\mathbf{x}\|_\infty} \le 1 \qquad \forall \mathbf{x}_j > 0 \ , \ 1 \le j \le d \ , \tag{50}$$

where δ is given in (12). Next, as in §4.5, we define a geometric mesh on the interval $[\delta, 1]$:

$$\xi_0 = \delta \quad ; \quad \xi_i = (1+\varepsilon)\xi_{i-1} \quad , \quad 1 \le i \le q \quad ; \quad q := \left\lfloor \frac{-\lg\delta}{\lg(1+\varepsilon)} \right\rfloor + 1 \ . \tag{51}$$

In view of the above, every nonzero component of $\mathbf{x}/\|\mathbf{x}\|_\infty$ lies in an interval $[\xi_{i-1}, \xi_i)$ for some $1 \le i \le q$. Next, we define

$$\hat{\mathbf{x}} = \|\mathbf{x}\|_\infty \mathcal{H}\left(\frac{\mathbf{x}}{\|\mathbf{x}\|_\infty}\right) \ , \tag{52}$$

where the operator $\mathcal{H}$ retains components that are 0 or 1 and replaces every other component by the left end point of the interval $[\xi_{i-1}, \xi_i)$ where it lies. Hence, the vector $\hat{\mathbf{x}}$ may be in one of

$$s = (q+2)^d - 1 \tag{53}$$

linear subspaces of dimension 1 in $\mathcal{R}^d$; we denote those subspaces by W^σ, $1 \le \sigma \le s$. In view of the above, we define the set

$$\hat{\mathcal{S}} = \{\hat{\mathbf{x}} \ : \ \mathbf{x} \in \mathcal{S}\} \ . \tag{54}$$

Next, we define for each type $1 \le \sigma \le s$

$$\mathbf{w}^\sigma = \sum \{\hat{\mathbf{x}} \ : \ \hat{\mathbf{x}} \in \hat{\mathcal{S}} \cap W^\sigma\} \qquad 1 \le \sigma \le s \ ; \tag{55}$$

namely, $\mathbf{w}^\sigma$ aggregates all vectors $\hat{\mathbf{x}}$ of type σ. We now slice this vector into large identical "slices", where each of those slices and their number are given by:

$$\tilde{\mathbf{w}}^\sigma = \frac{\mathbf{w}^\sigma}{\|\mathbf{w}^\sigma\|_\infty} \cdot \Phi\varepsilon^{2d+1} \quad \text{and} \quad \kappa_\sigma = \left\lceil \frac{\|\mathbf{w}^\sigma\|_\infty}{\Phi\varepsilon^{2d+1}} \right\rceil \ . \tag{56}$$

Finally, we define the set $\tilde{\mathcal{S}}$ as follows:

$$\tilde{\mathcal{S}} = \cup_{\sigma=1}^{s} \{\mathbf{z}^{\sigma,k} = \tilde{\mathbf{w}}^\sigma \ : \ 1 \le k \le \kappa_\sigma\} \ . \tag{57}$$

Namely, the new set $\tilde{\mathcal{S}}$ includes for each type σ the "slice"-vector $\tilde{\mathbf{w}}^\sigma$, (56), repeated κ_σ times. As implied by (56), all vectors in $\tilde{\mathcal{S}}$ have a max norm of $\Phi\varepsilon^{2d+1}$, in accord with (49). Also, the number of vectors in $\tilde{\mathcal{S}}$, $\tilde{\nu} = \sum_{\sigma=1}^{s} \kappa_\sigma$, is obviously no more than ν as the construction of the new vectors implies that $\kappa_\sigma \le |\hat{\mathcal{S}} \cap W^\sigma|$ (recall that $\|\hat{\mathbf{x}}\|_\infty < \Phi\varepsilon^{2d+1}$ for all $\hat{\mathbf{x}} \in \hat{\mathcal{S}}$).

So we have defined three problem instances:

- The original problem I with n input vectors $\mathcal{L} \cup \mathcal{S}$.
- An intermediate problem $\hat{I}$ with n input vectors $\mathcal{L} \cup \hat{\mathcal{S}}$, (52)+(54).
- The modified problem $\tilde{I}$ with $\tilde{n} = n - \nu + \tilde{\nu}$ input vectors $\mathcal{L} \cup \tilde{\mathcal{S}}$, (55)-(57).

We are now ready to prove that all of the above problems are close.

Theorem 3 *For each solution $S \in \{1, \ldots, m\}^{\{1,\ldots,n\}}$ of I there exists a solution $\tilde{S} \in \{1, \ldots, m\}^{\{1,\ldots,\tilde{n}\}}$ of $\tilde{I}$ such that*

$$(1 - C_1\varepsilon) \cdot \left(F(\tilde{S}) - C_2\Phi\varepsilon\right) \le F(S) \le (1 + C_1\varepsilon) \cdot \left(F(\tilde{S}) + C_2\Phi\varepsilon\right) \ , \tag{58}$$

where the constants C_1 and C_2 depend only on d, f and g. Conversely, for each solution $\tilde{S} \in \{1, \ldots, m\}^{\{1,\ldots,\tilde{n}\}}$ of $\tilde{I}$ there exists a solution $S \in \{1, \ldots, m\}^{\{1,\ldots,n\}}$ of I that satisfies (58).

Proof. Let S be a solution of I and $\hat{S}$ be its counterpart solution of $\hat{I}$. Let $\boldsymbol{\ell}^k$ and $\hat{\boldsymbol{\ell}}^k$, $1 \le k \le m$, denote the load vectors in S and $\hat{S}$, respectively. By (52), $1 \le \boldsymbol{\ell}^k / \hat{\boldsymbol{\ell}}^k \le 1 + \varepsilon$. Hence, as g dominates the max-norm, Assumption 2-4,

$$\|\boldsymbol{\ell}^k - \hat{\boldsymbol{\ell}}^k\|_\infty \le \varepsilon \eta_g g(\hat{\boldsymbol{\ell}}^k) \ . \tag{59}$$

Therefore, by the Lipschitz continuity of g, Assumption 2-5,

$$(1 - C_1\varepsilon)g(\hat{\boldsymbol{\ell}}^k) \le g(\boldsymbol{\ell}^k) \le (1 + C_1\varepsilon)g(\hat{\boldsymbol{\ell}}^k) \quad 1 \le k \le m \qquad \text{where } \ C_1 = \eta_g M_g \ . \tag{60}$$

Applying the monotonicity of f, Assumption 1-1, and its linear dependence with respect to scalars, Assumption 1-3, on (60) we get that

$$(1 - C_1\varepsilon)F(\hat{S}) \le F(S) \le (1 + C_1\varepsilon)F(\hat{S}) \ . \tag{61}$$

We now proceed to find a solution $\tilde{S}$ of $\tilde{I}$ for which (58) holds. To this end, we fix $1 \le \sigma \le s$ and define for every machine k the following vector:

$$\mathbf{y}^{\sigma,k} = \sum \{\hat{\mathbf{x}}^i \ : \ \hat{\mathbf{x}}^i \in \hat{S} \cap W^\sigma \ , \ \hat{S}(i) = k\} \ ; \tag{62}$$

i.e., $\mathbf{y}^{\sigma,k}$ is the sum of small vectors of type σ that are assigned to the kth machine. Recalling (56), $\tilde{\mathcal{S}}$ includes the vector $\tilde{\mathbf{w}}^\sigma$ repeated κ_σ times, where

$$\kappa_\sigma = \left\lceil \sum_{k=1}^{m} \frac{\|\mathbf{y}^{\sigma,k}\|_\infty}{\Phi\varepsilon^{2d+1}} \right\rceil \ . \tag{63}$$

We may now select for each k an integer $t_{\sigma,k}$ such that

$$\left| t_{\sigma,k} - \frac{\|\mathbf{y}^{\sigma,k}\|_\infty}{\Phi\varepsilon^{2d+1}} \right| \le 1 \tag{64}$$

and

$$\sum_{k=1}^{m} t_{\sigma,k} = \kappa_\sigma \ . \tag{65}$$

With this, the solution $\tilde{S}$ is the one that assigns to the kth machine, $1 \le k \le m$, $t_{\sigma,k}$ vectors $\tilde{\mathbf{w}}^\sigma$ for all $1 \le \sigma \le s$. In view of (64),

$$\|t_{\sigma,k} \cdot \tilde{\mathbf{w}}^\sigma - \mathbf{y}^{\sigma,k}\|_\infty \le \Phi\varepsilon^{2d+1} \ . \tag{66}$$

Therefore, summing (66) over $1 \le \sigma \le s$, we conclude that $\tilde{\boldsymbol{\ell}}^k$ and $\hat{\boldsymbol{\ell}}^k$ – the loads on the kth machine in $\tilde{S}$ and $\hat{S}$ respectively – are close,

$$\|\tilde{\boldsymbol{\ell}}^k - \hat{\boldsymbol{\ell}}^k\|_\infty \le s\Phi\varepsilon^{2d+1} \ . \tag{67}$$

However, as (12), (51) and (53) imply that

$$s \le C_s\varepsilon^{-2d} \qquad \text{for all } \ 0 < \varepsilon \le 1 \tag{68}$$

where C_s depends on d and g, we conclude by (67) and (68) that

$$\|\tilde{\boldsymbol{\ell}}^k - \hat{\boldsymbol{\ell}}^k\|_\infty \le C_s\Phi\varepsilon \ . \tag{69}$$

The Lipschitz continuity of both g and f imply that

$$F(\tilde{S}) - C_2\Phi\varepsilon \le F(\hat{S}) \le F(\tilde{S}) + C_2\Phi\varepsilon \qquad \text{where } \ C_2 = C_s M_f M_g \ . \tag{70}$$

Finally, (58) follows from (61) and (70).

The proof of the second assertion of the theorem goes along the same lines. Given a solution $\tilde{S}$ of $\tilde{I}$, we may construct a solution $\hat{S}$ of $\hat{I}$ for which (70) holds. This latter solution corresponds to at least one solution S of I for which (61) holds. Hence, for any solution $\tilde{S}$ of $\tilde{I}$ we may find a solution S of I that satisfies (58). $\square$

Step 3.

We made two modifications: changing the definition of $\mathcal{L}$ from (23) to (47) and transforming the original instance I to $\tilde{I}$ where all input vectors are large.

We begin by considering the first modification and observing that it has no significant effect on algorithm A_L. That algorithm still acts in the same way as described in §4.5, with the following changes:

1. The lower bound in (39) changes to $\Phi\varepsilon^{2(d+1)}/(\eta_g M_g) \leq \mathbf{x}^i_j$ for all $\mathbf{x}^i \in \mathcal{L}$ and $\mathbf{x}^i_j > 0$. This affects only the value of q in (40) that changes to
$$q = \left\lfloor \frac{\lg(\eta_f \eta_g^2 M_g \varepsilon^{-2(d+1)})}{\lg(1+\varepsilon)} \right\rfloor + 1 \ . \tag{71}$$

2. The value of the upper bound in (42) changes to $\eta_f \eta_g d \cdot (1+\varepsilon)\varepsilon^{2d+1}$. This changes the value of R in (43) to
$$R := \left(\eta_f \eta_g d \cdot \frac{1+\varepsilon}{\varepsilon^{2d+1}}\right)^s \ , \tag{72}$$
where s is still given by (41) and q is as in (71).

These two changes, (71) and (72), will have an effect only on the running time of the dynamical programming algorithm. As ε is considered a constant when dealing with running time, the redefinition of $\mathcal{L}$ does not have a significant effect on A_L.

We proceed to describe the necessary modifications in the core algorithm and in algorithm A_L due to the fact that the input vectors in $\tilde{I}$ are all large. In light of this simplification, the DLCs consist now of only one vector $\hat{\mathbf{a}}^k$, as $\hat{\mathbf{b}}^k$ became redundant. Hence, given a solution of $\tilde{I}$ with load vectors $\boldsymbol{\ell}^k$, $1 \leq k \leq m$, and corresponding DLCs $\hat{\mathbf{a}}^k$, the definition of $\hat{\mathbf{a}}^k$, (25), implies that
$$\hat{\mathbf{a}}^k - \Phi\varepsilon \leq \boldsymbol{\ell}^k \leq \hat{\mathbf{a}}^k \quad , \quad 1 \leq k \leq m \ . \tag{73}$$

In view of (73) we refer herein to a set of vectors $\{\hat{\mathbf{a}}^k \ : \ 1 \leq k \leq m\}$ as Φ-*good* if there exists a Φ-solution with load vectors $\boldsymbol{\ell}^k$ for which (73) holds.

The original algorithm A_L rescaled all vectors down by a factor of no more than $(1+\varepsilon)$ and tried to find an assignment such that the resulting load on the kth machine, $\hat{\boldsymbol{\ell}}^k$, would satisfy $\hat{\boldsymbol{\ell}}^k \leq \hat{\mathbf{a}}^k$. However, when we call this algorithm with a *good* set of vectors $\hat{\mathbf{a}}^k$ then
$$\frac{\hat{\mathbf{a}}^k - \Phi\varepsilon}{1+\varepsilon} \leq \hat{\boldsymbol{\ell}}^k \ , \tag{74}$$

because of (73) and the rescaling process. Hence, the modified algorithm A_L, given a set of vectors $\hat{\mathbf{a}}^k$, will look for assignments of the rounded down vectors to the m machines such that the resulting load on each machine satisfies

$$\frac{\hat{\mathbf{a}}^k - \Phi\varepsilon}{1+\varepsilon} \leq \hat{\boldsymbol{\ell}}^k \leq \hat{\mathbf{a}}^k \ . \tag{75}$$

Clearly, the modified A_L will fail more times than the original A_L, but it will not miss the Φ-solutions. Then, when we perform the same assignment with the original vectors (i.e., prior to rescaling), we would get in the kth machine a load $\boldsymbol{\ell}^k$ that satisfies

$$\frac{\hat{\mathbf{a}}^k - \Phi\varepsilon}{1+\varepsilon} \leq \boldsymbol{\ell}^k \leq (1+\varepsilon)\hat{\mathbf{a}}^k \ . \tag{76}$$

Finally, there are two straightforward modifications to the core algorithm: the outer loop on the ACVs may be shortened to include DLCs that consist of one vector only, $\hat{\mathbf{a}}^k$, and, as there are no longer small vectors, steps 4 and 5 in the algorithm become redundant. Hence, the simplified core algorithm goes like this: scan all ACVs; for each ACV that passed the check in step 2, exercise algorithm A_L, step 3; if it succeeds - return the corresponding assignment; otherwise skip to the next ACV; if there was no successful ACV there are no Φ-solutions.

Step 4.

We now revisit the proof Theorem 1 and see how the fact that we have only large vectors helps prove the theorem for non-monotone g functions.

Theorem 4 *Let f be a function that satisfies Assumption 1 and let g be a function that satisfies Assumption 2 except possibly the monotonicity condition 1. Let I be an instance of the VAP with input vectors $\{\mathbf{x}^i\}_{1\leq i\leq n}$, all of which are large, (47). Let Φ be a test value. Assume that for some ACV algorithm A_L succeeded with an assignment $A_L : \{1,\dots,n\} \to \{1,\dots,m\}$. Then this assignment satisfies (17) for all $0 < \varepsilon \leq 1$, where T is a constant that depends only on f and g.*

Proof. Let $\hat{\mathbf{a}}^k$ denote the load vectors that are associated with the kth machine, $1 \leq k \leq m$, by the ACV. In view of Step 2 in the core algorithm,

$$f(g(\hat{\mathbf{a}}^k))_{1\leq k\leq m} \leq (1 + 2M_f M_g \varepsilon)\Phi \ . \tag{77}$$

Let $\boldsymbol{\ell}^k$ be the load vector in the kth machine in the solution that algorithm A_L returned. Then, as implied by (76) and by Assumption 2-4, it satisfies

$$\|\boldsymbol{\ell}^k - \hat{\mathbf{a}}^k\|_\infty \leq \varepsilon\eta_g g(\hat{\mathbf{a}}^k) + \varepsilon\Phi \ . \tag{78}$$

Applying the Lipschitz continuity of g we get that

$$|g(\boldsymbol{\ell}^k) - g(\hat{\mathbf{a}}^k)| \leq \varepsilon\eta_g M_g g(\hat{\mathbf{a}}^k) + \varepsilon M_g \Phi \ ,$$

and, consequently,

$$g(\boldsymbol{\ell}^k) \leq g(\hat{\mathbf{a}}^k) \cdot (1 + \varepsilon\eta_g M_g) + \varepsilon M_g \Phi \ . \tag{79}$$

Applying f on (79), using its monotonicity, Lipschitz continuity and then its linear dependence on scalar multipliers, Assumption 1-1, 5 and 3, we arrive at the following estimate:

$$f(g(\boldsymbol{\ell}^k))_{1\le k\le m} \le (1+\varepsilon\eta_g M_g)\cdot f(g(\hat{\mathbf{a}}^k))_{1\le k\le m} + \varepsilon M_f M_g \Phi \ . \qquad (80)$$

Combining (80) with (77), we conclude that this assignment satisfies (17) with a value of T that depends solely on f and g. $\square$

Summary

When the function g is not monotone, each step in the binary search goes as follows:

1. Translate the problem instance I into a problem instance $\tilde{I}$ having only large vectors .
2. Apply the modified core algorithm to find a solution to $\tilde{I}$ that satisfies (17).
3. If the core algorithm succeeded with a solution $\tilde{S}$ of $\tilde{I}$, we translate it into a solution S of I along the lines of the proof of Theorem 3. As $\tilde{S}$ satisfies (17), we conclude in view of (58) that S also satisfies (17) with a different value of the constant T that depends on d, f and g.
4. If the core algorithm failed then, by Theorem 2, $\tilde{I}$ has no Φ-solutions. Hence, in view of (58), all solutions of I have a cost greater than $(1-C\varepsilon)\Phi$ where $C = C_1 + C_2 - C_1C_2$ and C_1 and C_2 are as in (58). This is a weaker result than before (where a failure of the algorithm implied that there are no Φ-solutions), but it is sufficient for the binary search to converge to a solution that satisfies (4).

5.1 Overall complexity of the scheme

We summarize herein the steps of the entire scheme when the inner cost function is non-monotone and the (worst case) cost of each step:

1. As in §4, the outer loop executes the geometrical binary search. The number of steps in that search remains $\lceil P(m) - \lg\lg(1+\varepsilon)\rceil$, (21), where $P(m)$ is the polynomial given in (16).
2. In each step in the binary search we perform the core algorithm. The core algorithm is composed of the following steps:
   (a) Translating the original set of small vectors $\mathcal{S}$ to $\hat{\mathcal{S}}$. The running time of this stage is $\Theta(|\mathcal{S}|d)$, where $|\mathcal{S}| \le n$.
   (b) Translating the set $\hat{\mathcal{S}}$ to $\tilde{\mathcal{S}}$, consisting of large vectors only. The running time of this stage is bounded by $s\cdot n$ steps of aggregating vectors of equal type, where

$$s = (q+2)^d - 1 \quad \text{and} \quad q = \left\lfloor \frac{\lg(\eta_g M_g) - \lg\varepsilon}{\lg(1+\varepsilon)} \right\rfloor + 1 \ ,$$

   (53), (51) and (12), followed by no more than n steps of slicing the aggregated vectors into the new input vectors (57).

(c) Executing a loop over all possible ACVs. The number of ACVs is bounded by m^t, where $t = \left(\lceil \frac{\eta_f \eta_g}{\varepsilon} \rceil + 1\right)^d$. This value of t is the square root of the value of t in the monotone case since here we first got rid of the small vectors and, therefore, the DLCs are single discretized load vectors and not pairs of vectors as they were in §4.

(d) In each step in the core algorithm we may perform algorithm A_L. The running time of that algorithm is bounded by $O(Rmn^s)$ where

$$R = \left(\eta_f \eta_g d \cdot \frac{1+\varepsilon}{\varepsilon^{2d+1}}\right)^s \quad , \quad s = \left(\left\lfloor \frac{\lg(\eta_f \eta_g^2 M_g \varepsilon^{-2(d+1)})}{\lg(1+\varepsilon)} \right\rfloor + 2\right)^d - 1 \ ,$$

(72), (41) and (71).

References

[1] N. Alon, Y. Azar, G. Woeginger, and T. Yadid. Approximation schemes for scheduling. In *Proc. 8th ACM-SIAM Symp. on Discrete Algorithms*, pages 493–500, 1997.

[2] N. Alon, Y. Azar, G. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1:1:55–66, 1998.

[3] Y. Azar and L. Epstein. Approximation schemes for covering and scheduling on related machines. In *1st Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX98)*, pages 39–47, 1998.

[4] A.K. Chandra and C.K. Wong. Worst-case analysis of a placement algorithm related to storage allocation. *SIAM Journal on Computing*, 4(3):249–263, 1975.

[5] C. Chekuri and S. Khanna. On multi-dimensional packing problems. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 185–194, 1999.

[6] R.A. Cody and E.G. Coffman, Jr. Record allocation for minimizing expected retrieval costs on drum-like storage devices. *J. Assoc. Comput. Mach.*, 23(1):103–115, 1976.

[7] E. G. Coffman, Jr. and George S. Lueker. Approximation algorithms for extensible bin packing. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-01)*, pages 586–588, 2001.

[8] J. Csirik, H. Kellerer, and G. Woeginger. The exact lpt-bound for maximizing the minimum completion time. *Operations Research Letters*, 11:281–287, 1992.

[9] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

[10] P. Dell'Olmo, H. Kellerer, M. G. Speranza, and Zs. Tuza. A 13/12 approximation algorithm for bin packing with extendable bins. *Information Processing Letters*, 65(5):229–233, 1998.

[11] P. Dell'Olmo and M. G. Speranza. Approximation algorithms for partitioning small items in unequal bins to minimize the total size. *Discrete Applied Mathematics*, 94:181–191, 1999.

[12] L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. In *7th Annual European Symposium on Algorithms (ESA'99)*, pages 151–162, 1999.

[13] L. Epstein and T. Tassa. Graph based vector assignment schemes, manuscript, 2002.

[14] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. C. Yao. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory (Series A)*, 21:257–298, 1976.

[15] R.L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[16] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*, 17:416–429, 1969.

[17] D. Hochbaum and D. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.

[18] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987.

[19] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one–dimensional bin–packing problem. In *Proc. 23rd Ann. Symp. on Foundations of Computer Science*, 1982.

[20] S. Sahni. Algorithms for scheduling independent tasks. *Journal of the Association for Computing Machinery*, 23:116–127, 1976.

[21] G. Woeginger. A polynomial time approximation scheme for maximizing the minimum machine completion time. *Operations Research Letters*, 20:149–154, 1997.

[22] Gerhard J. Woeginger. There is no asymptotic PTAS for two-dimensional vector packing. *Information Processing Letters*, 64(6):293–297, 1997.