University of Pisa

Department of Mathematics

Master's Thesis in Mathematics

A Constrained Maximum Entropy Principle for Data Reduction via Pattern Selection

Candidate:
Renato Budinich

Advisors:
Prof. Giovanni Punzi
Prof. Maria del Viva
Prof. Vladimir Georgiev

Academic Year 2014-2015

Contents

Todo list

1 Introduction

Perception is a complex emergent phenomenon which, in its various different facets and at various levels, modern neuroscience is striving to explain. While we continue to accumulate biological data on many aspects of the human nervous system, this data alone cannot explain more than the low-level interactions happening between the many specialized small parts that eventually add up to form cognitive processes. What is direly needed is some form of coherent organization of all the data, or better still a mathematical framework based on simple abstract principles and powerful enough to actually describe, as a consequence of these principles, as much of the observed neurological processes as possible.

In the particular case of the visual system, the last 50 years have brought many new advancements, and today there are many postulations on the various phases intercurring between the moment the photoreceptors on the retina detect a particular pattern of light and the moment we recognize the object that's in front of us. However we still lack a comprehensive model, detailed enough to make falsifiable predictions regarding how and why information is re-encoded and passed on to the next computational phase.

In this thesis we study a formal mathematical model of a general principle first proposed by M. del Viva, G.Punzi and D. Benedetti in [14], concerning the information reduction that must happen at certain bottlenecks in the nervous complex responsible for vision. The parts of the data that are discarded (or better, those that are preserved) are defined, in our model, as the solution to a combinatorial optimization problem. While the idea of explaining selective perception under a principle of maximization of information is not entirely new (see for example [2]), taking explicitly into account constraints due to limited bandwidth and storage capacity of the system has not, to our knowledge, been done before. The model is corroborated by tests done with human subjects: discarding all the part of the data not in the solution to our optimization problem doesn't seem to affect human's capacity to recognize images.

In the first chapter we will review the original paper by the authors and formalize the principles they propose as a problem of maximizing the Shannon entropy of certain random variables subject to constraints. Before this though, we'll briefly explain the role of Associative Memories in HEP particle detectors, inspiration to the whole model, and expose some basic biological knowledge on the human visual neurological system.

In the second chapter we'll review the Dynamic Programming approach to solving the Knapsack problem, and see how its equivalent formulation as a graph problem can seamlessly be adapted to our optimization problem. We'll see two slightly different reformulations of this type, to which the same general algorithm for optimal path finding can be adapted. We'll try to understand in detail why this approach turns out to be computationally inefficient, and propose a possible approximated algorithm that improves on

these problems.
Finally in the last chapter we'll run some numerical tests, using the developed Python software that implements the proposed algorithm, as well as wrapping some generic mixed integer programming solvers. We'll compare, for different values of the model parameters, various characteristics of the optimal solution with the heuristic solution proposed by the original authors, and we'll draw our conclusions on the whole study.

2 The constrained maximum entropy principle

In [14] the authors propose a new model for pattern selection inspired by the workings of high energy physics (HEP) particle detectors using associative memories (AM) and on the neurological eye-brain system of human vision. Both systems consist of a data acquisition phase (done by particle detectors in HEP and photoreceptor cells in the eye) and a later processing phase; in between a strong reduction of information is known to happen. The authors take as inspiration how this data filtering is done in the electronics implementing AM and postulate that an analogous process happens in the visual cortex. Their main original contribution is making an hypotheses on which patterns are selected for this filtering stage in the brain: they claim that a maximum entropy principle is followed, with the important addition of constraints on storage and bandwidth capacities that both the vision and HEP systems present.

In order to precisely formulate this maximum entropy principle, we must first build a formal framework of the type of systems we want to consider. This model is build upon the AM electronics used in HEP particle detectors and can be reasonably assumed to hold, at least at a functional level, for the human visual neurological system. To justify this assertion, before formalizing the model in section 2.3, we will briefly summarize how AM electronics work and give a basic review of the relevant neurophysiological knowledge on the visual system.

A major benefit of this model is that it lets us, for any system that conforms to it, select a small portion of the input data presenting salient features without the need to define, ad hoc for every such system, what a salient feature is; the selection is completely determined by the statistical properties of the input data. For example, we'll see that many pixel patterns selected by the model are needed for edge detection, which is known to play a relevant role in the reduction of information done by the human visual system. The model is further validated by tests done on human subjects, which show that there's no statistical difference in the success rate of recognizing images drawn using only the selected patterns (sketches) rather than the original images these sketches are derived from.

2.1 Associative Memories in HEP

In [7] the authors describe the architecture of an *associative memory* electronic device that receives data readouts from a particle detector, performs a parallel comparison of the data with a number of hard-wired patterns and finally passes on the results of these

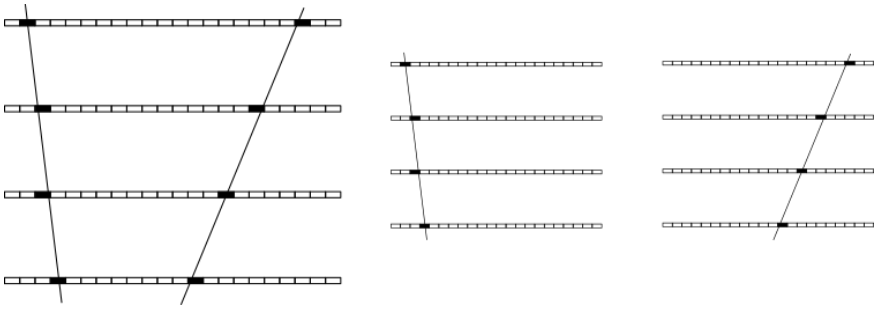


Figure 2.1: An event and the patterns it's composed of

comparisons to the successive processing stages. This architecture, which thanks to its parallel nature allows for very fast and concurrent detection of multiple patterns, has since been implemented with success in many HEP experiments.

referenze...

The detector is assumed to be composed of various parallel layers, each layer having a certain number of bins; particles go across all the layers and hit one bin for each layer (see figure 2.1). This geometrical model is a simplified abstraction supposed to represent a multitude of real detectors, which have more complex layer configurations. The left part of figure 2.1 shows two particle trajectories crossing four layers, and the bins on the trajectories registering a hit: this is called an *event*. The *track finding* problem, which the associative memory is supposed to solve, consists of deducing all the particles trajectories from the combination of bins registering a hit.

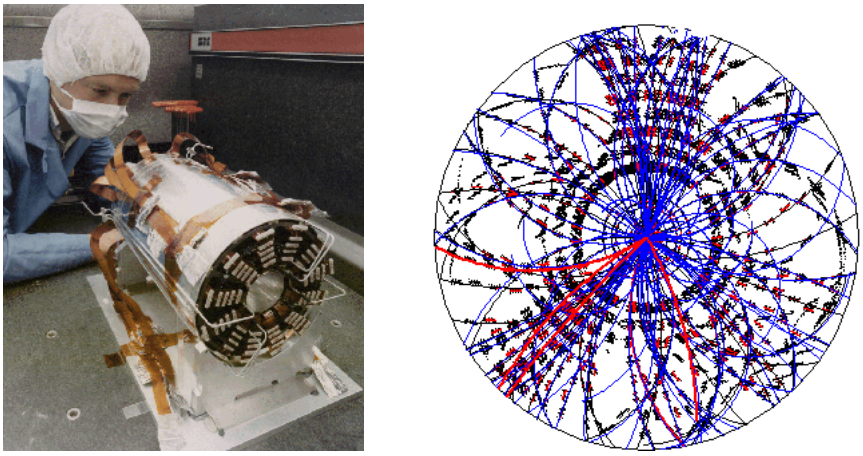


Figure 2.2: A particle detector with layers disposed concentrically and an event registered by it

An event can be thought of as being composed of multiple *patterns*, which are trajectories of single particles (right part of figure 2.1). Among all the possible patterns, a subset is chosen and stored in a pattern bank. The role of the associative memory is to read an incoming event and passing on only the patterns that are present in its pattern

bank.

An important remark must be made here: the patterns that are currently, in the real world, being used in associative memories, are chosen based on mainly[1] physical criteria, i.e. patterns corresponding to impossible trajectories aren't stored in the pattern bank, which has limited storage capacity. The constrained maximum entropy model instead chooses a set of patterns determined only by their statistical frequency, calculated offline in a prior data acquisition phase. The claim is that these patterns turn out to be similar to the ones that are already, for physical considerations, deemed interesting.

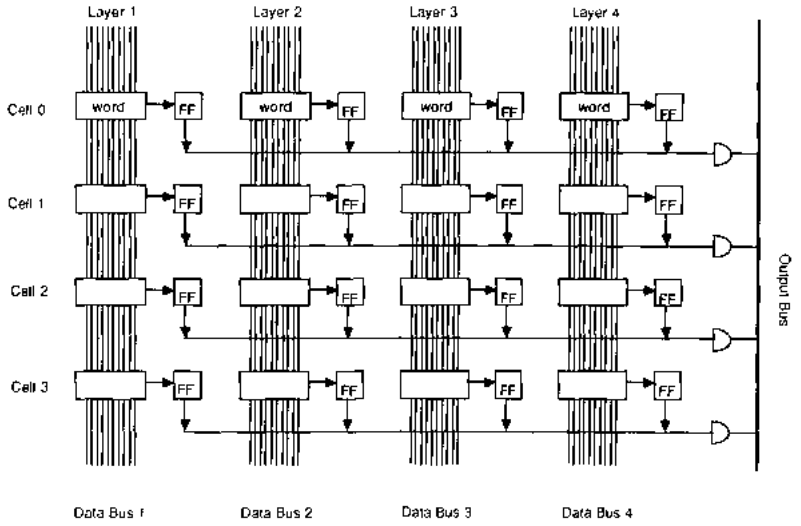


Figure 2.3: Associative memory architecture

In figure 2.3 the authors detail the architecture of the associative memory: every layer of the detector is connected to its own Data Bus, on which all the addresses of the hits it registered for that layer are sent, in parallel for all the layers. Each row represents a pattern in the pattern bank, coded as a sequence of bin addresses (one for each layer), known as *words*. Each word is stored in a cell on the corresponding Data Bus, and this cell reads all the bin addresses being transmitted on its Data Bus; if the word coded in the cell matches one of those transmitted on the Data Bus, it activates a flip-flop. If all the flip-flops for a row are active that means that the corresponding pattern was present in the event being analyzed, and thus an identifier for that pattern is sent on the Output Bus, to signal its presence to the successive processing stages.

Thanks to this highly parallel design the associative memory is able to detect patterns in an event very rapidly (fast enough to keep up with the huge speed events are registered on the detector), orders of magnitude faster than it would take to check the event sequentially against the patterns in the pattern bank.

[1] not entirely, since of course the purpose of the detector in the first place is also to show unanticipated behaviours

2.2 Receptive fields in the visual system

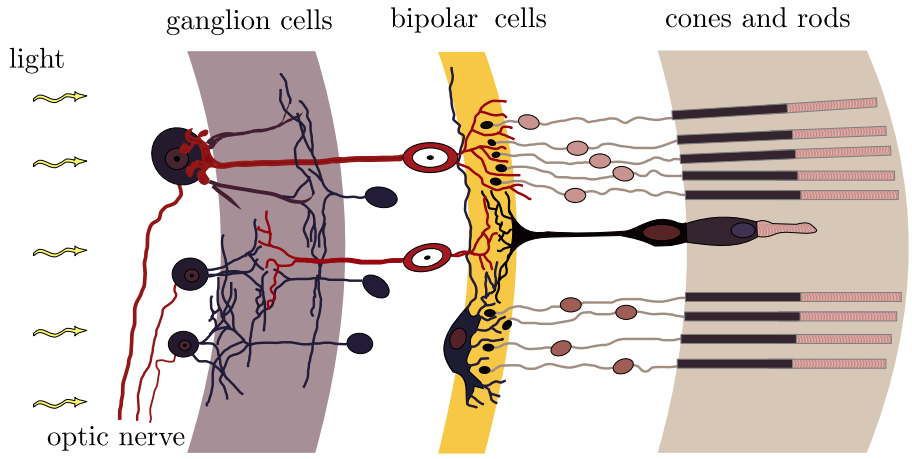


Figure 2.4: Cross-section of the back part of the retina

In this section we want to give a very brief overview of the human visual system, mainly concerned on the characteristics which make it analagous to the Associative Memories described in the previous section; for an in-depth treatise on the functional aspects of the visual system, see for example [13]. Broadly speaking, and greatly simplifying the neurophysiological sophistication of this system, light that enters the eye and hits the retina activates, in this order, these fundamental components of the visual system:

1. **Photoreceptors**, which are of two types, **rods** and **cones**. Rods are on average 92 millions ([6]), are equidistributed in the retina and largely responsible for peripheral and night vision. Cones are on average 4.6 millions ([6]), are clustered in the center of the retina and are less sensible to light than rods, but can percieve colors (which rods cannot).

2. **Bipolar cells**, which transfer information from photoreceptors to ganglion cells.

3. **Ganglion cells**, are responsible for connecting the retina to the visual cortex in the brain, through the **optical nerve fiber**, which is made of the axons of ganglion cells and travels all the way back to the lateral geniculate nucleus.

4. **Lateral geniculate nucleus**, located in the thalamus - this is the first center recieving inputs from both eyes. It sends some feedback signals back to the eye to better focus the viewed image.

5. **Primary visual cortex**, also known as **V1** (visual area one) and **striate cortex**, is in the back of the brain, in the occipital lobe above the cerebellum. Here and in the successive **exastriate** areas (**V2** - **V5**), most of the computation required to recognize edges, lines, surfaces and objects happens.

At each of these passages information gets re-encoded, supposedly in a lossy manner. A simple argument (though by no means the only one) to convince oneself of this are the calculations done by E. Echeverri in [8], which, based on the number and diameter of nerve fibers, estimate the bandwidth capacity of the optic nerve for each eye to be 4 Gb/s, while each retina is estimated to have a capacity closer to 20 Gb/s. This means that processing and filtering of data likely happens already between the photoreceptors and the ganglion cells.

Our model claims to give a theoretical explanation, in particular, of the first in a series of high-level computations done in the visual cortex, concerned with detecting lines, edges, and other simple shapes on a local level: we are concerned with small portions of the whole image percieved by the retina, each processed, in first approximation, independently of one another. The biological evidence that the visual cortex does such computations was originally discovered by Hubel and Wiesel in 1959 and following years (see [9]), expanding on studies by Kufler ([12]) and Barlow ([4]). Kuffler and Barlow inserted a very thin electrode into animals' retina, and in this way were able to read the output (i.e. the firing rate) of a single ganglion cell. They then proceeded to show different visual stimuli to the anesthetized animal on a screen , trying to understand the causation effect between the stimuli and the ganglion's cell response. What they found is that there was a precise location on the screen such that, if a point of light of appropriate diameter was projected onto, would result in the most vigorous response from the ganglion cell. However, if the diameter of the light spot was increased or decreased, the response weakened. This and further studies led to classify receptor fields at the ganglion level as one of two types: they either are on-center off-surround (see figure 2.5), i.e. sensitive to a spot of light surrounded by darkness, or off-center on-surround, i.e. sensitive to a dark spot surrounded by darkness.

Hubel and Wiesel inserted the electrode in the visual cortex instead of in the retina. Through a process of trial and error they eventually discovered that the visual cortex is not sensible to spots of lights, but rather to slits of light: this made them conclude that receptive fields here are not circular, but have an elongated shape. They discerned between three different types of cells: simple, complex and hypercomplex. **Simple** cells are the most numerous and are of two types, eloquently named line and edge detectors (see figure 2.6a); **complex** cells respond only to moving lines or edges (see figure 2.6b) and are thought of as integrating the response from various simple cells. A key difference between simple and complex cells is that the first have approximately a linear response to stimuli (the output of a simple cell detecting a light pattern is roughly the sum of outputs of the response to the singl light spots that pattern is composed of), while the complex cells don't share this property. Finally, Hubel and Wiesel called **hypercomplex** cells those that detected a line shape, but only if its length falled in a determined range.

The discoveries by Hubel and Wiesel, which eventually won the Nobel Prize in 1981, were considered groundbreaking: they were the first to give neurophysiological evidence of a functional level computation done in the visual cortex. Their discoveries led to hierarchical theories of the visual system, which, along its pathway from the retina to the visual cortex, is thought to iteratively detect higher level structures in images: from

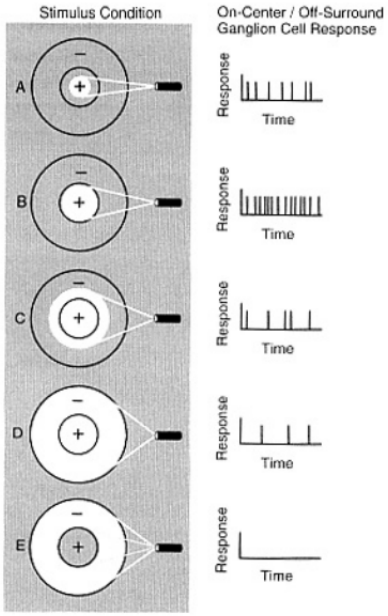
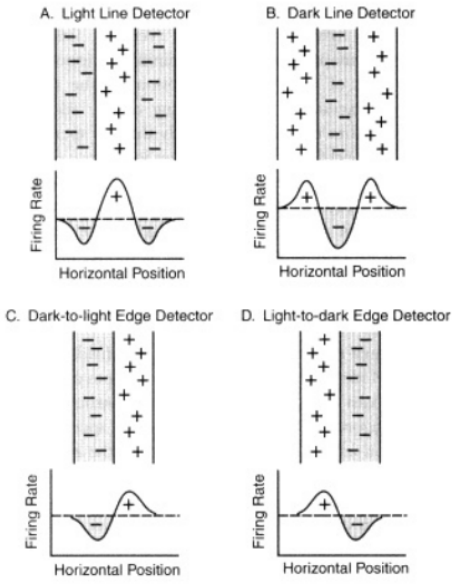


Figure 2.5: Response of on-center ganglion cells

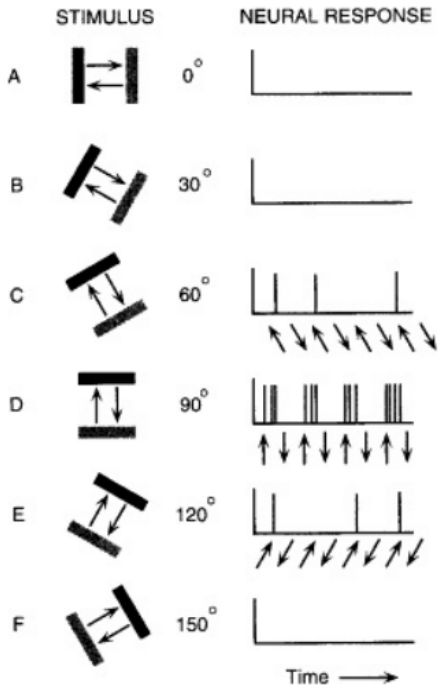
single pixels at the retinal level, to oriented lines and edges in the visual cortex and eventually to surfaces and whole objects. These theories on the functional roles of the various stages of the visual system are still object of debate, and we are not concerned with taking a stance among them.
In fact we don't expect our model to give an explanation of the full visual system, but simply answer the following question: if we define patterns as all the possible inputs going into a cell in the visual cortex, and we suppose that any such pattern with no correspondent cell to detect it gets discarded, then can we describe a set of optimal such patterns, that would somehow be most efficient in discarding redundant or useless portions of information? To be clear: we already know [some of] the patterns that the brain stores, but can we state a theoretical principle from which to deduce them? The starting point of our model, which we will begin to study in detail in the next section, is that these patterns are the ones that maximize the Shannon entropy, when we ask for certain storage and bandwidth constraints to be respected.

2.3 The model and the optimization problem

Taking as inspiration the functioning of Associative Memories described in section 2.1, and taking into account the findings of Hubel and Wiesel on the visual cortex described in section 2.2, we can formulate a simple model that encompasses them both: we will suppose to have a finite number of receptors somehow displaced in space, each recieving a portion of the whole data. While in the case of Associative Memories we're supposing



(a) Response of line and edge-detector simple cells in the visual cortex



(b) Response of complex cells to moving light stimuli

these receptors to correspond to the rows of figure 2.3 (sequence of words), we don't explicate exactly what they correspond to in the visual system: it suffices to say that the whole data can be somehow encoded in small spatial portions, i.e. the whole image can be thought of as composed of many small (low resolution) images. We further suppose that these receptors are all equal and independent among them, the only characteristic differing among them being their spatial displacement.
The flow diagram we wish to model is represented in figure 2.7: for every time unit, each receptor registers one from a common set of possible patterns and passes it on to a filter, which either discards it or passes it on unchanged. The patterns that pass the filtering stage are then collected into the *sketch*, which is thus the lossy compressed version of the data incoming on the receptors. Formally, we'll define:

- $\mathcal{X} = \{1, \dots, \chi\} \subset \mathbb{N}$ the set of indexes identifying the receptors
- $\mathcal{P} = \{1, \dots, n\} \subset \mathbb{N}$ the set of indexes identifying the possible patterns each receptor can register (we suppose this set is equal for all receptors)
- $S \subseteq \mathcal{P}$ the set representing the pattern bank or equivalently a single filter: we wish to discard all patterns that are not in this set.
- $\mathcal{F}_S = S^{\chi}$ the filter bank, which is simply χ copies of the filter S, one for each $i \in \mathcal{X}$. We are thus making the assumption that the output of every receptor gets filtered

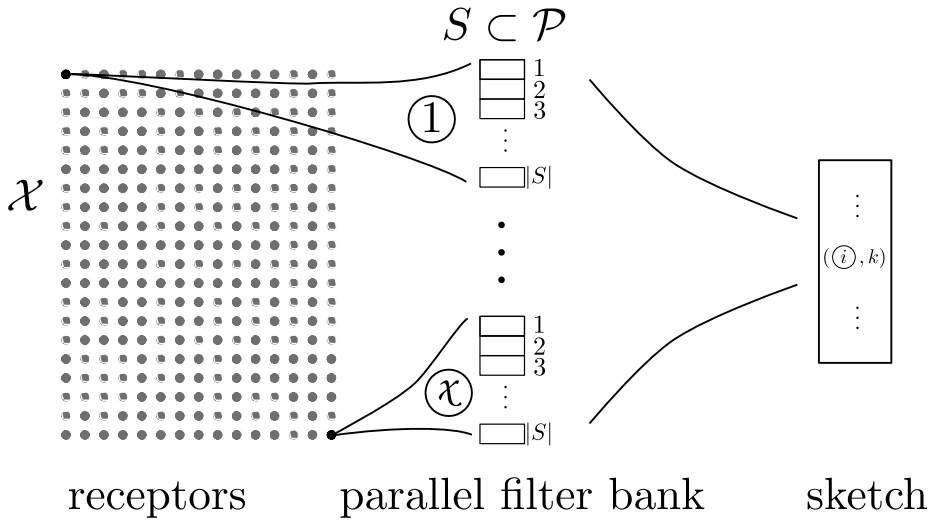


Figure 2.7: Outline of the model

in the same way.

- The sketch $\mathcal{Y}$, a set of address-pattern couples of the form $(i, k) \in \mathcal{X} \times \mathcal{P}$; if a couple (i, k) is present in the sketch, this means that pattern k was detected on receptor i and not discarded by the filter.

In order to properly define the sketch, the output of this system, we need to introduce a probability space and random variables representing the data flow.

Let $(\Omega, \Sigma, \mathbb{P})$ be the probability space of experiments' outcome, then for each $i \in \mathcal{X}$ we can define the discrete random variables X_i and Y_i^S, which represent the pattern being registered on receptor i and the output of the associated filter respectively. Formally:

$$\begin{aligned} &\forall i \in \mathcal{X} \quad X_i : \Omega \to \mathcal{P} \\ &\forall i \in \mathcal{X} \quad Y_i^S : \Omega \to \{\texttt{null}\} \cup S \\ &\qquad\qquad Y_i^S(\omega) \stackrel{def}{=} X_i(\omega) \mathbb{1}_{\{X_i \in S\}}(\omega) + \texttt{null}\ \mathbb{1}_{\{X_i \notin S\}}(\omega)\ , \end{aligned} \tag{2.3.1}$$

where `null` is a special symbol used to signal that the pattern processed by the filter does not belong to S. We can then define the sketch as

$$\begin{aligned} &\mathcal{Y} : \Omega \to 2^{\mathcal{X} \times S} \\ &\mathcal{Y}(\omega) \stackrel{def}{=} \{(i, Y_i^S(\omega)) \text{ s.t. } Y_i^S(\omega) \neq \texttt{null}\}\ , \end{aligned}$$

and its size l^S as

$$\begin{aligned} &l^S : \Omega \to \mathbb{N} \\ &l^S(\omega) \stackrel{def}{=} |\mathcal{Y}(\omega)|\ , \end{aligned}$$

where by $|\cdot|$ we denote the cardinality of a set.

We will suppose that all the X_i are independent and identically distributed (and thus also the Y_i), denoting with X and Y^S respectively two representatives of these random variables (i.e. $X \sim X_i$ and $Y^S \sim Y_i^S$); furthermore, for every $k \in S$, we will define $p_k \stackrel{def}{=} \mathbb{P}(X = k)$. While this is certainly a coarse approximation, since in reality receptors that are close together will be strongly correlated, it greatly simplifies successive calculations; with this assumption we will be able to explicitly calculate an exact solution to an otherwise much harder optimization problem.

Now, our main question is, with which criteria do we choose the filter $S \subseteq \mathcal{P}$? Surely we want it to be chosen in some kind of optimal way: since we would like the data coming out of the filter to be most meaningful and descriptive of the data being registered by the sensors, it makes sense to ask for the Shannon entropy of the output of the filter bank to be maximum. The Shannon entropy of a discrete random variable $Z : \Omega \to \mathcal{Z}$ with probability mass function $\rho(z)$ is defined as

$$H(Z) = -\sum_{z \in \mathcal{Z}} \rho(z) \log \rho(z) \ ,$$

where the logarithm is usually in base 2[2]. The random variable is thought of as a *source of information* and $H(Z)$ measures its uncertainty or randomness. It can also be interpreted as the expected value of the random variable $-\log \rho(Z)$, i.e.

$$H(Z) = \mathbb{E}[-\log \rho(z)] \ .$$

This last formula gives a possible intuitive interpretation of entropy: if our source of information outputs a particular element $z \in \mathcal{Z}$, we can think that this gives us more information the less "predictable" z is, i.e. the bigger $-\log \rho(z)$ is. $H(Z)$ can therefore be seen as the average information contained in a received message from source Z.
However the real reason why $H(Z)$ is a meaningful and relevant quantity is that important theorems involving it can be proved, above all Shannon's source coding theorem that, loosely speaking, states that entropy is the lower bound on average word length of any code used to describe Z - it gives a theoretical limit on the achievable lossless compression factor for the information source Z. For a thorough exposure of these concepts see for example [1] or the more modern [5].
Returning to our model, we'll ask for the filter output to be most meaningful by asking that its entropy is maximum; we thus wish to maximize

$$H_{\mathcal{F}_S} \stackrel{def}{=} H((Y_i^S)_{i=1}^{\chi}) = \sum_{y_1,\dots,y_\chi \in \{\texttt{null}\} \cup S} \mathbb{P}(Y_1^S = y_1, \dots, Y_\chi^S = y_\chi) \log \frac{1}{\mathbb{P}(Y_1^S = y_1, \dots, Y_\chi^S = y_\chi)} \ ,$$

which, thanks to the random variables being independent and equally distributed, can

[2] in this thesis we will thus suppose all logarithms to be in base 2

be written as[3]

$$H_{\mathcal{F}_S} = \varkappa \sum_{y \in \{\texttt{null}\} \cup S} \mathbb{P}(Y^S = y) \log \frac{1}{\mathbb{P}(Y^S = y)} \ .$$

However, in order for all this to make sense, we must pose some restrictions on which subsets $S \subseteq \mathcal{P}$ we may choose, otherwise, for the following lemma, $\mathcal{F}_\mathcal{P}$ itself would be the filter bank maximizing entropy (which makes intuitive sense, since choosing $\mathcal{P}$ as a filter means we're not discarding any pattern and thus preserving all information)

Lemma 2.3.2.

$$\forall S \subset \mathcal{P} \quad H_{\mathcal{F}_S} \leq H_{\mathcal{F}_\mathcal{P}}$$

Proof. From definition (2.3.1), we have

$$k \in S \Rightarrow \{Y^S = k\} = \{X = k\}$$
$$\text{and}$$
$$\{Y^S = \texttt{null}\} = \{X \notin S\} \ .$$

Thus we can write:

$$\begin{aligned} H_{\mathcal{F}_S} &= \varkappa \left(\sum_{k \in S} \mathbb{P}(Y^S = k) \log \frac{1}{\mathbb{P}(Y^S = k)} + \mathbb{P}(Y^S = \texttt{null}) \log \frac{1}{\mathbb{P}(Y^S = \texttt{null})} \right) \\ &= \varkappa \left(\sum_{k \in S} p_k \log \frac{1}{p_k} + \mathbb{P}(X \notin S) \log \frac{1}{\mathbb{P}(X \notin S)} \right) \ . \end{aligned}$$

Since

$$\begin{aligned} \mathbb{P}(X \notin S) \log \frac{1}{\mathbb{P}(X \notin S)} &= \sum_{k \in \mathcal{P} \setminus S} p_k \log \frac{1}{\sum_{j \in \mathcal{P} \setminus S} p_j} \\ &\leq \sum_{k \in \mathcal{P} \setminus S} p_k \log \frac{1}{p_k} \ , \end{aligned}$$

we can conclude

$$\begin{aligned} H_{\mathcal{F}_S} &\leq \varkappa \sum_{k \in \mathcal{P}} p_k \log \frac{1}{p_k} \\ &= H_{\mathcal{F}_\mathcal{P}} \ . \end{aligned}$$

■

[3] since, if X and Y are any two independent random variables, the joint entropy $H(X, Y)$ is equal to the sum of the entropies $H(X) + H(Y)$. See for example [5].

Surely the most natural constraint is to impose an upper bound on the cardinality of S, since any real world storage device that could be used to store patterns would have limited capacity. We will thus introduce $N \in \mathbb{N}$ as a model parameter and request

$$|S| \leq N \ . \tag{2.3.3}$$

We'll generally be in the situation of having a great number of possibly registered patterns on the receptors and wishing to select only a few of them, thus $N \ll n$.

Secondly, we must remember that this system we're modeling is operating data reduction at an early stage, i.e. the selected patterns must be transmitted to successive processing units (for example the visual cortex in the human brain). The channel they're transmitted on has limited bandwidth, and this somehow must affect the optimal set of patterns we want to store in the pattern bank: if we choose to store the ones with greatest probability p_k, for example, they will cost us a lot in terms of bandwidth, since they will appear very frequently in the sketch which has to be transmitted on the channel. The idea of introducing a bound on the channel capacity to see how it affects pattern selection is considered by the authors of [14] the main theoretical innovation of their model. Formally we'll introduce a second model parameter $W \in \mathbb{R}$ and ask that, on average, no more than W patterns can be outputted by our system. In other words, the expected length of the sketch must be smaller than W:

W is "rate"?

$$\mathbb{E}[l^S] \leq W \ . \tag{2.3.4}$$

The requirement that this condition has to be met only on average stems from the HEP detector case: there, before transmitting the sketch on the channel, there's always a FIFO[4] buffering phase which allows for the channel capacity to be temporarily exceeded, as long as it later makes up for the accumulated delay.

Thanks to the following lemma we can rewrite this second constraint without reference to l^S or the sketch:

Lemma 2.3.5.

$$\mathbb{E}[l^S] = \varkappa \sum_{k \in S} p_k \ . \tag{2.3.6}$$

Proof. If we define the Bernoulli random variable

$$\begin{aligned} \forall i \in \mathcal{X} \quad & Z_i^S : \Omega \to \{0,1\} \\ & Z_i^S(\omega) = \mathbb{1}_{\{Y_i^S \neq \texttt{null}\}}(\omega) \end{aligned}$$

[4] first in first out

we then can write $l^S(\omega) = \sum_{i=1}^{\varkappa} Z_i^S(\omega)$. Thus

$$\begin{aligned}\mathbb{E}[l^S] &= \sum_{i=1}^{\varkappa} \mathbb{E}[Z_i^S] \\ &= \sum_{i=1}^{\varkappa} \mathbb{P}(Y_i^S \neq \texttt{null}) \\ &= \sum_{i=1}^{\varkappa} \mathbb{P}(X_i \in S) \\ &= \sum_{i=1}^{\varkappa} \sum_{k \in S} p_k \\ &= \varkappa \sum_{k \in S} p_k\end{aligned}$$

■

Since the number of receptors $\varkappa$ is considered a constant of the model, we can rescale W by a factor $\frac{1}{\varkappa}$ and, using the Lemma, rewrite (2.3.4) as

$$\sum_{k \in S} p_k \leq W \ . \tag{2.3.7}$$

We can finally formulate the **constrained maximum entropy principle**: among the sets $S \subset \mathcal{P}$ that satisfy (2.3.3) and (2.3.7), we wish to find the ones that maximize the entropy of the filter bank $\mathcal{F}_S$. We thus have the optimization problem:

$$\max \{H_{\mathcal{F}_S} \text{ where } S \subset \mathcal{P},\ |S| \leq N,\ \sum_{k \in S} p_k \leq W\} \ . \tag{2.3.8}$$

Remark 1. The real world bottleneck on channel bandwidth, represented in our model by (2.3.7), is usually very relevant and must be surely taken into account when choosing which patterns to store in the pattern bank. We thus can suppose that this constraint rules out a big number of subsets $S \subset \mathcal{P}$, the ones composed of the most frequent patterns. We will therefore suppose that the probability of all patterns not in S far outweighs those in S, i.e.:

$$\mathbb{P}(Y^S = \texttt{null}) = \sum_{i \in \mathcal{X} \setminus S} p_i \cong 1 \ .$$

This means that we can write

$$\begin{aligned} H(\mathcal{F}_S) &= \sum_{k\in S} p_k \log\frac{1}{p_k} + \mathbb{P}(Y^S = \texttt{null}) \log\frac{1}{\mathbb{P}(Y^S = \texttt{null})} \\ &\cong \sum_{k\in S} p_k \log\frac{1}{p_k} \ . \end{aligned}$$

For convenience we'll define, for any subset of patterns $S \subset \mathcal{P}$, its **entropy**, **rate** and **cardinality**:

$$\begin{aligned} H(S) &= \sum_{k\in S} p_k \log\frac{1}{p_k} \\ W(S) &= \sum_{k\in S} p_k \\ \kappa(S) &= |S| \ ; \end{aligned} \tag{2.3.9}$$

we'll also use H_{tot} to refer to the unfiltered total entropy:

$$H_{\text{tot}} = \sum_{i=1}^{n} p_i \log\frac{1}{p_i} \ . \tag{2.3.10}$$

Finally, thanks to remark 1, we can consider, in lieu of problem (2.3.8), the optimization problem:

$$\max\left\{H(S) \text{ where } S \subset \mathcal{P},\ \kappa(S) \leq N \text{ and } W(S) \leq W\right\} \ . \tag{2.3.11}$$

Since we're thinking of the pattern set $\mathcal{P}$ as an initial segment of $\mathbb{N}$, we can identify every $S \subset \mathcal{P}$ with a binary string representing the characteristic function $\mathbb{1}_S : \mathcal{P} \to \{0,1\}$. Formally, we're saying that the function $\Phi : \mathcal{P} \to \{0,1\}^n$ defined by

$$\Phi(S) = (\mathbb{1}_S(i))_{i=1}^n$$

is a bijection; further, if we define the **entropy**, **rate** and **cardinality** of a binary string $(x_i)_{i=1}^n \in \{0,1\}^n$ by:

$$\begin{aligned} H((x_i)_{i=1}^n) &= \sum_{k=1}^{n} x_k p_k \log\frac{1}{p_k} \ , \\ W((x_i)_{i=1}^n) &= \sum_{k=1}^{n} x_k p_k \\ \kappa((x_i)_{i=1}^n) &= \sum_{k=1}^{n} x_k \ , \end{aligned} \tag{2.3.12}$$

we'll have that Φ preserves these quantities, i.e.:

$$\begin{aligned} H(\Phi(S)) &= H(S) \\ W(\Phi(S)) &= W(S) \\ \kappa(\Phi(S)) &= \kappa(S) \ . \end{aligned}$$

We can therefore equivalently rewrite (2.3.11) as

$$\max \ \{H((x_i)_{i=1}^n) \text{ st } (x_i)_{i=1}^n \in \{0,1\}^n \text{ and } \kappa((x_i)_{i=1}^n) \leq N \ , W((x_i)_{i=1}^n) \leq W\} \ , \tag{2.3.13}$$

or more explicitly, in what is known as the standard *0-1 integer programming* form:

$$\max \ \left\{ \sum_{k=1}^n x_k p_k \log \frac{1}{p_k} \text{ st } \forall k \ x_k \in \{0,1\} \text{ and } \sum_{k=1}^n x_k \leq N \ , \sum_{k=1}^n x_k p_k \leq W \right\} \ , \tag{2.3.14}$$

The two formulations (2.3.11) and (2.3.14) are of course completely equivalent, but allow us to use two different terminologies: from now on, we'll interchangeably refer to a candidate solution as either a subset $S \subset \mathcal{P}$ or its correspondent binary string $x \in \{0,1\}^n$, depending on what is more convenient.

2.4 A heuristic and initial hypotheses on the exact solution

In [14], the authors propose an heuristic to calculate an approximate solution to this problem: by associating a storage cost $\frac{1}{N}$ and a bandwidth cost $\frac{p_i}{W}$ to each pattern $i \in \mathcal{P}$, we can simply define the maximum between these two as the cost of the pattern. The heuristic then is simply the greedy approach of taking the patterns with best value/cost ratio, i.e. their proposed solution S_{heur} is

$$\begin{aligned} S_{\text{heur}} = \{f(p_i) > c\} &= \{i \in \mathcal{P} \text{ s.t. } f(p_i) > c\} \\ \text{where } f(p) &\stackrel{def}{=} \frac{p \log \frac{1}{p}}{\max\{\frac{1}{N}, \frac{p}{W}\}} \ , \end{aligned} \tag{2.4.1}$$

and where c is determined numerically so as to be the smallest possible without making S_{heur} unfeasible (in an actual implementation there's no need to calculate c, since it suffices to calculate the f_i and iteratively add the index with greatest value of f_i until one of the constraints is violated).
Since f can be written also as

$$f(p) = \begin{cases} Np \log \frac{1}{p} & \text{if} \quad p \leq \frac{W}{N} \\ W \log \frac{1}{p} & \text{if} \quad p \geq \frac{W}{N} \ , \end{cases}$$

we can say that if $\frac{W}{N} \leq \frac{1}{e} = \max\limits_{x \in [0,1]} p \log \frac{1}{p}$, which in practice is always the case, f is an

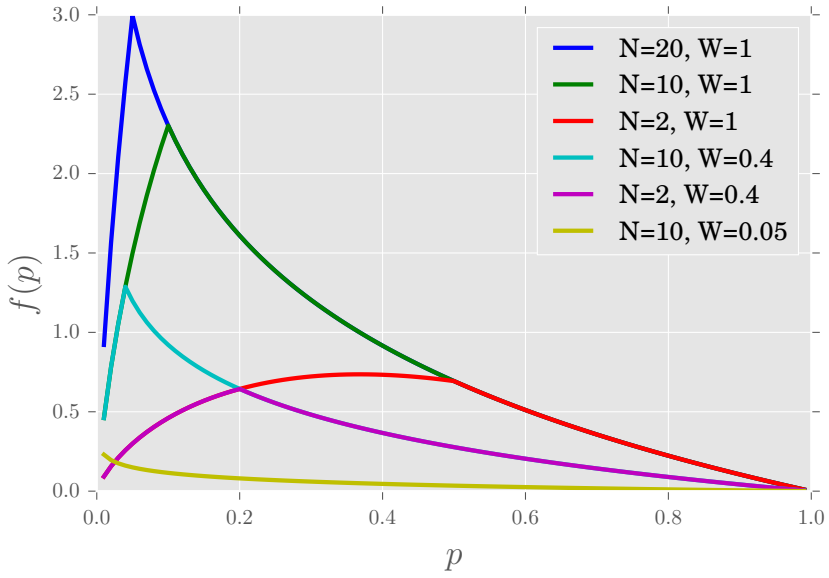


Figure 2.8: Plot of f for various values of N and W.

increasing function for $p \leq \frac{W}{N}$ and decreasing for greater values of p. Therefore f will assume its maximum value in $\frac{W}{N}$, and S_{heur} will always be in the proximity of this value; it might actually happen that $\frac{W}{N}$ is not at all near the center of S_{heur} (see figure 2.9), but we can always say that

$$\sup_{j \in S_{\text{heur}}} |\frac{W}{N} - j| \sim |S_{\text{heur}}| \ .$$

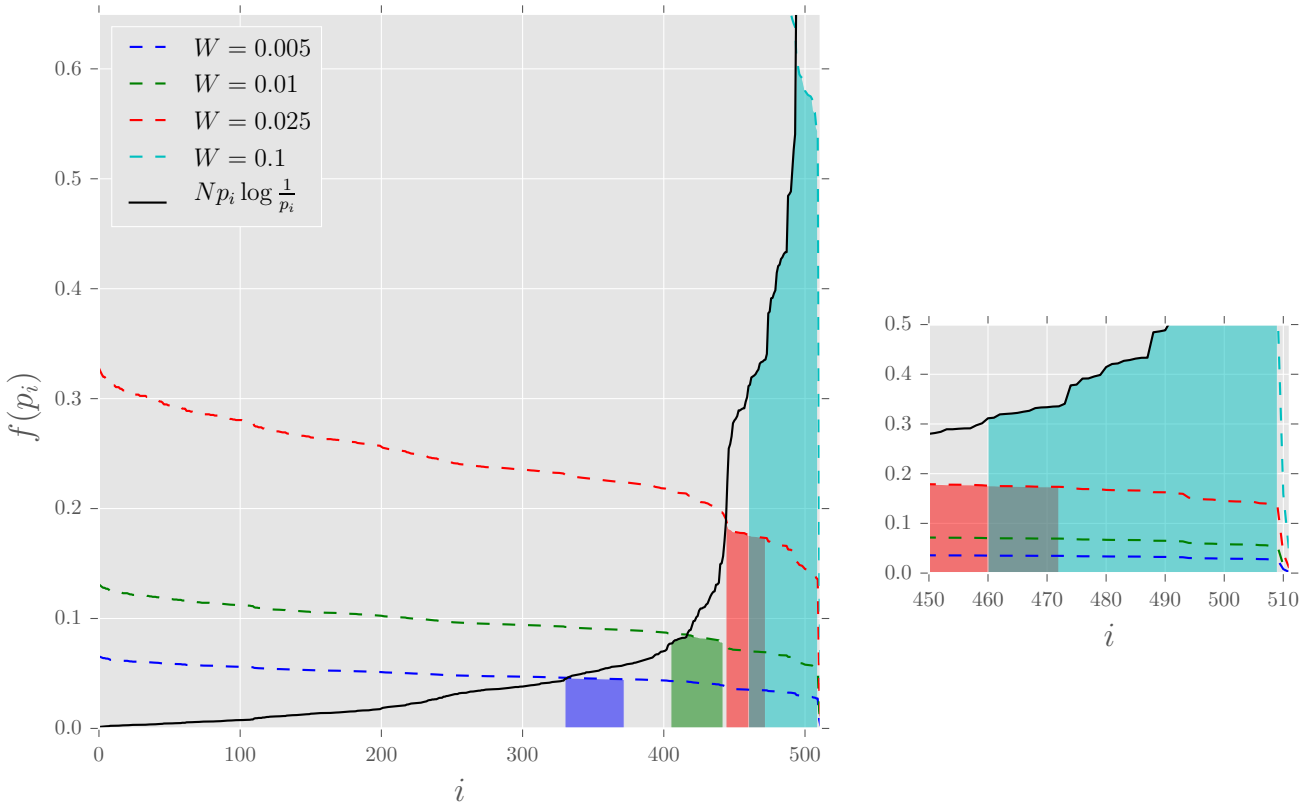


Figure 2.9: S_{heur} regions for various values of W, where the probability distribution $(p_i)_{i=1}^n$ is the pixel pattern distribution, and the pattern indexes $\mathcal{P}$ are reordered such that $(p_i)_{i=1}^n$ is an increasing sequence. The segmented lines are plots of the values $W \log \frac{1}{p_i}$, and thus the plot of f would be the minimum between these and the solid black line, which is the plot of values $Np_i \log \frac{1}{p_i}$.

This proposed heuristic brought us to formulate some initial hypotheses on the optimal solution S to the problem:

1. if apply the appropriate permutation to indexes $\mathcal{P}$ so that $(p_i)_{i=1}^n$ is an monotone sequence, then S is an interval, in the sense that

$$i, j \in S,\ i \leq j \Rightarrow \forall k \text{ s.t. } i \leq k \leq j \quad k \in S$$

2. S is unique.

Both these assumptions turned out to be false. In section 4 we'll discuss some numerical trials, and we'll see that the exact solution is always quite close to the interval given by S_{heur} (which in typical conditions commits an error of about 5%), but never an interval itself. In section 3 instead, by studying a possible way to treat problem

(2.3.14), we'll see that the optimal solution corresponds to an optimal path in the *decision graph*, and equivalent paths give rise to equivalent solutions. This will make it easy to construct pathological probability distributions $(p_k)_{k=1}^n$ that give rise to non-unique solutions. However it will prove impossible, with this method, to certify whether a real world example of our model has one or multiple solutions; this is due to a form of numerical instability intrinsic in the decision graph algorithm and the inevitable statistical errors in the estimation of the p_k.

2.5 Pixel patterns

In order to test if this model can be successfully applied to human vision, the authors in [14] conducted two tests with human subjects, of which we'll describe one. The outline of the procedure they employed is this:

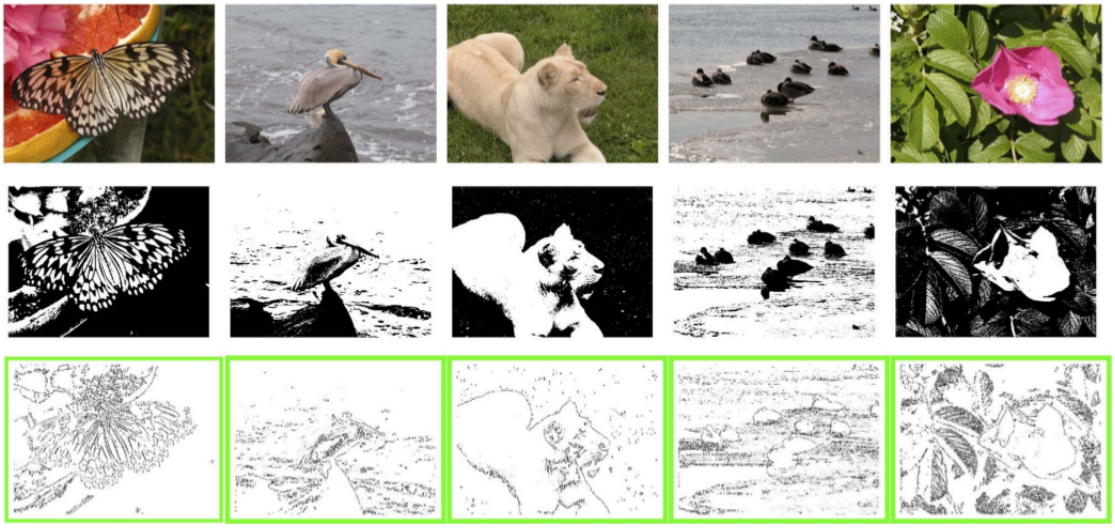


Figure 2.10: Five images from the database in their full color, digitized and sketched versions

1. **Digitalization of images from an image database:** each image from a database of photographs of natural subjects [5] is projected on a 1-bit space, where each pixel can either be black or white. For every image in the database, the luminosity of each pixel is calculated: if its above the average luminosity of the image, the corresponding pixel in the digitalized image is set to 1 (white), otherwise it's 0 (black). Here luminosity of a pixel is defined as the arithmetic mean of the values of its three channels (red,green and blue); see second row of figure 2.10.

2. **Sampling of 3×3 pixel patterns:** patterns are defined as the 2^9 possible 3×3 pixel black and white squares. For each image in the database every square of 3×3

[5] the authors used images from [10], which unfortunately seems to be no longer avaiable

contiguous pixels in the image is considered, including overlapping squares, and the frequency p_k of every pattern k in the database is calculated.

3. **Selection of patterns using heuristic:** S_{heur} is calculated - see figure 2.11.

4. **Creation of sketches:** from every digitized image a sketch is created, obtained by starting with a fully white image and considering again all the 3×3 squares in it, with overlap: if the corresponding pattern in the fully digitized image belongs to S_{heur}, then its black pixels, if not already present due to a matched pattern on an overlapping square, are added to the sketch; see last row of figure 2.10.

5. **Subject testing:** to probe the early stages of human vision, a sketch is presented to a subject on a screen for 20ms, and subsequently, for 700ms, the full digitized version from which the sketch was obtained is shown alongside a distractor, which is simply another image from the database. The subject then has to press one of two computer keys to indicate which image he thinks was the original digitalized image.

In figure 2.12 there is a sample of four types of images that were presented to the four subjects in the experiment and the percentage of correct answers given by each subject for each of the four sets of images; the striped blue bars will be explained below. The color code for the image sets is:

Red: 256 grey levels conversion of the original colored photos, used as a control set

Green: sketches for filter $S_g = S_{\text{heur}}$ with $W = 0.05$ and $N = 50$

Blue: sketches for filter $S_b = S_{\text{heur}}$ with $W = 0.05$ and $N = 15$

Yellow: sketches for filter S_y obtained by recursively adding the least frequent pattern until $H(S_y) = H(S_b)$

The bar graph shows that the reduction of information due to using sketches from S_g and S_b has negligible effects on the capacity of subjects to recognize images. S_y instead, while having the same information content as S_b, causes subjects to respond correctly much less frequently. This would seem to suggest that the filtering of patterns done by the human visual system is quite similar to that done by the heuristic solution to our model.
A natural objection to this conclusion would be that the difference in percentage of correct responses is simply due to the fact that sketches obtained from S_y have very few black pixels, since they are composed of the least frequent patterns. It's clear in fact from figure 2.13 that the average of the number of black pixels in sketches obtained from S_y is lower than the one for sketches obtained from S_b.
To confute this argument, the authors made another test: they artificially assigned weights to the set of sketches obtained from S_b in order for their distribution of number of black pixels to match the one for the sketches obtained from S_y, and presented the sketches obtained from S_b to subjects with frequency given by the inverse of these

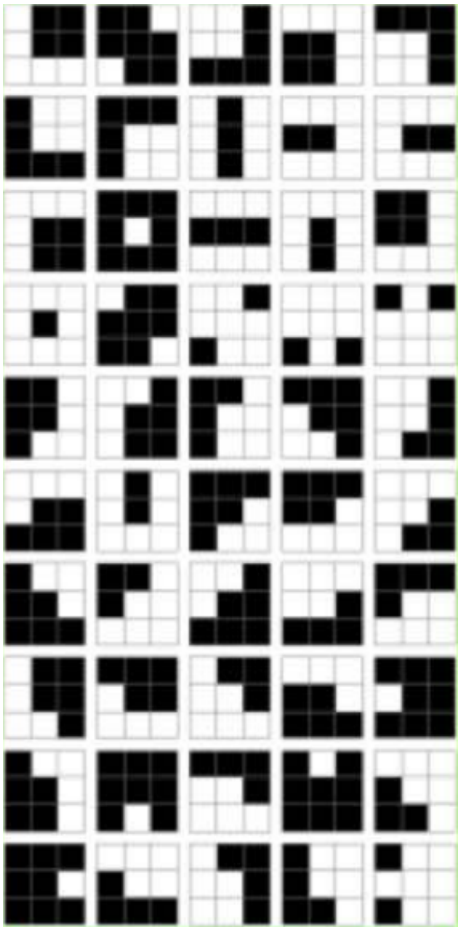


Figure 2.11: Patterns selected by the authors of [14] using the heuristic solution with $N = 50$ and $W = 0.05$

weights, rather than uniform weights as was otherwise done for the other sets of sketches. In other words, they presented subjects with sketches of the S_b family favoring, in the random selection done to choose the next sketch to present to the subject, sketches with fewer black pixels. The percentage of correct answers to this test is represented with the striped blue bars in figure 2.12, and it shows no difference with the blue bars, i.e. the percentage of correct answers given by subjects when presented with sketches drawn uniformly from the S_b family.

Furthermore, by organizing the acquired data in appropriate classes, the authors plotted (figure 2.14) the percentage of correct answers as a function of the number of black pixels rather than the individual subjects. If the increased discrimination capacity was really due to the number of black pixels, these would be plots of increasing functions, or at the very least the percentage of correct answers for images obtained from S_y with many black pixels would be greater than that for images obtained from S_b with few black pixels; this

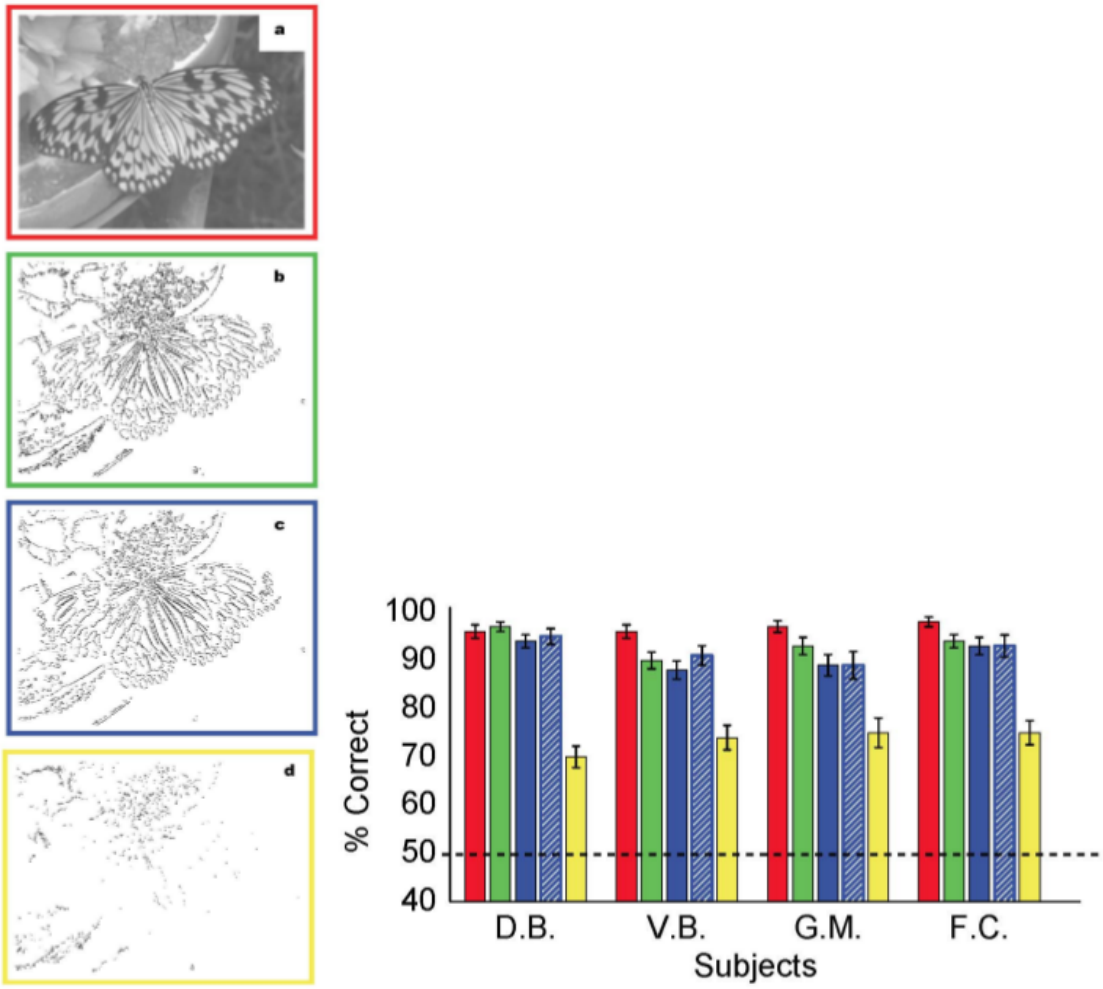


Figure 2.12: See text

is clearly not the case.

pattern selezionati son dello stesso tipo di quelli in corteccia visiva

il modello si presta bene all'apprendimento

Future tests will include presenting to subjects patterns obtained using an optimal filter solution to problem (2.3.11); as we will see in the second part of this thesis, this solution has a high computation cost, and this brings the authors to postulate that the limited neurological resources in our brain couldn't possibly calculate it. On this basis, the heuristic solution might actually select a set of patterns more similar to the ones supposedly selected by the human visual system.

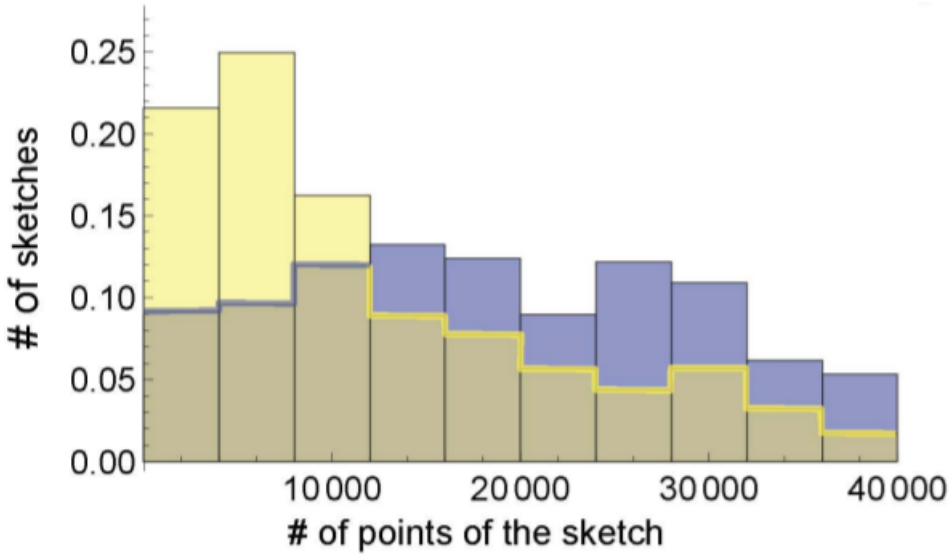


Figure 2.13: Distribution of the number of black pixels in sketches obtained from S_b and S_y

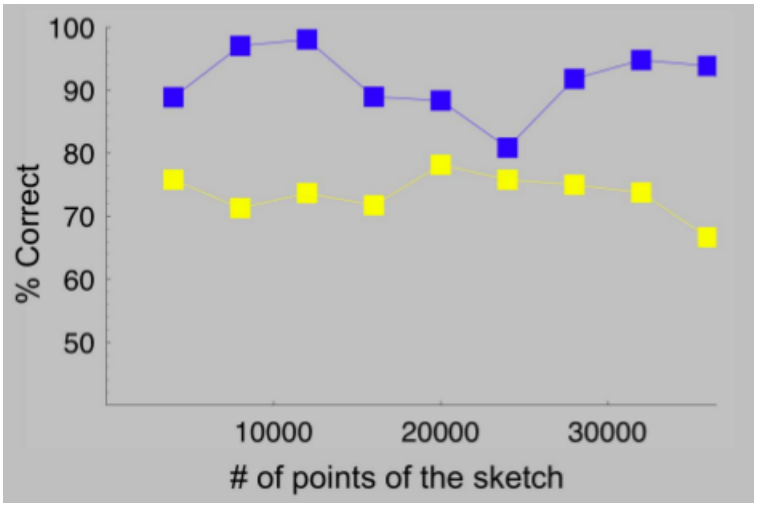


Figure 2.14: Percentage of correct answers for sketches obtained from S_b (in blue) and S_y (in yellow), averaged over all subjects, plotted as a function of the number of black pixels

3 Solving the optimization problem: an explicit algorithm

We have seen that the optimization problem we wish to solve can be written in the standard Integer Programming form as

$$\max \left\{ \sum_{i=1}^{n} x_i p_i \log \frac{1}{p_i} \text{ st } \forall i \; x_i \in \{0,1\} \text{ and } \sum_{i=1}^{n} x_i p_i \leq W, \; \sum_{i=1}^{n} x_i \leq N \right\} . \qquad \text{(BPV)}$$

This is a classic combinatorial optimization problem, and there are many software libraries to solve it numerically, such as the PuLP[1] python module, which is a convenient wrapper for other specialized solvers. However we wanted to try and develop an algorithm specific to BPV, in the hopes of gaining some performance and some insight on the problem. Taking inspiration from the most basic way of solving the Knapsack problem, we developed an algorithm which, while correct, performs very poorly and is of little practical use (the PuLP library solves BPV much, much faster).
We do believe however that the detailed study of this algorithm and its shortcomings has some theoretical relevance: first of all, thanks to the framework we'll develop, it will be clear how it's possible to have multiple optimal solutions, and secondly, the fact that performance is so poor even after some non trivial optimizations suggests us that BPV is not an easy problem, as it often happens with combinatorial optimization problems.
Since the method we developed is closely inspired by the classical dynamic programming approach used for the Knapsack problem, specifically as developed in [11], we will begin this second part of the thesis by reviewing it. Primary purpose of this discussion is to build intuition on the method in this slightly simpler case[2], since much of this intuitive understanding can be brought on to the later sections where we'll be involved in a more complex framework.

For any $W \in \mathbb{R}, W \geq 0$, the *Knapsack problem with integer weights and values* can be stated as

$$\max \left\{ \sum_{i=1}^{n} x_i v_i \text{ st } \forall i \; x_i \in \{0,1\} \text{ and } \sum_{i=1}^{n} x_i w_i \leq W \text{ with } \forall v_i, w_i \in \mathbb{N} \right\} , \qquad \text{(KP)}$$

where v_i, w_i are respectively the values and weights of some given objects. (KP) models the problem of trying to stock a container of fixed capacity W with the most valuable

[1] `https://pypi.python.org/pypi/PuLP/1.6.0`
[2] simpler to understand, not to solve

combination of objects; each object has a weight and the total weight of the chosen objects can't exceed W. Observe that we can always suppose $0 \neq w_i < W$ and $0 \neq v_i$ for every $i \in \{1, \dots, n\}$, otherwise we'd have an object that would always either be or not part of any optimal solution, and as such we could eliminate it's corresponding variable. It's apparent that if we think of patterns as objects, each with entropy yield $p_i \log \frac{1}{p_i}$ and bandwidth weight p_i, (BPV) can be thought of as a Knapsack problem with an additional constraint on the cardinality of the solution.
There are however two more important differences: firstly, while in the Knapsack problem the weights and values of the objects are chosen independently, the entropy yields and the weights of the patterns are both direct functions of the same probability vector. Secondly, the weights and values in (KP) are natural numbers, while in (BPV) rates and probabilities are real numbers. The algorithm we'll describe works without modification for both cases, but having real numbers substantially worsens the time complexity, as we'll see in detail in section 3.6.
The Knapsack problem is NP-hard, which implies that, if P $\neq$ NP, there is no algorithm which is substantially better than just trying all **feasible** combinations (i.e. those that respect the constraint) and choosing one that gives maximum value. Here by "substantially better" I mean an algorithm that has time complexity polynomial in the size of the input data (whereas the naive approach of trying all possible combinations is clearly exponential in n).
Whether (BPV) is or not NP-hard is, to our knowledge, unproven: one would need to study the literature and see if the proofs of (KP) NP-hardness can be adapted to (BPV). The only thing that would seem to set our problem apart from the $D = 2$ case of the *D-dimensional Knapsack problem*, which can be stated as

$$\max \left\{ \sum_{j=1}^{n} x_j v_j \text{ s.t. } \forall j \; x_j \in \{0,1\} \text{ and } \begin{bmatrix} w_{11} & \dots & w_{1n} \\ w_{21} & \dots & w_{2n} \\ \vdots & \dots & \vdots \\ w_{D1} & \dots & w_{Dn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \leq \begin{bmatrix} W_1 \\ W_2 \\ \vdots \\ W_D \end{bmatrix} \right.$$

$$\left. \text{with } \forall i = 1, \dots, D \; \forall j = 1, \dots, n, \; w_{ij} \in \mathbb{N}, W_i \in \mathbb{R}, 0 < w_{ij} \leq W_i \right\},$$

is the already mentioned fact that in BPV weights and values are not chosen independently. However, since we weren't able to effectively leverage the dependency between entropy yields and bandwidth costs at any stage of our study[3], we suspect that BPV is also NP-hard.
In the next sections we will see how our problem can be reformulated (in two different ways) on the **decision graph**: each subset S of $\{1, \dots, k\}$ (for some $k \leq n$) is represented by a path in the graph, the last node of this path having two children; adding one child or the other to the path means either taking or not the $k+1$-th object, i.e. considering

[3] other than, of course, the heuristic solution of section 2.4 which orders objects by value/cost ratio

sets $S_1 = S$ or $S_2 = S \cup \{k+1\}$ respectively. By finding an optimal path on the graph we can find an optimal set that solves our problem.
In section 3.2 we will discuss the decision graph formulation for the Knapsack problem, which is a reinterpretation of the classic dynamic programming approach. This will lead us to state and prove correctness, in section 3.3, of a general algorithm to find an optimal path on the type of graphs we're interested. We'll then be able to reformulate BPV as a decision graph problem in two slightly different ways, in sections 3.4 and 3.5 respectively, to solve which we can adapt the general algorithm for optimal paths. Finally we'll discuss the shortcomings of our approach in section 3.6, and propose a variation of the algorithm that improves on these, albeit giving only an approximation of the optimal solution.

3.1 Solving Knapsack with Dynamic Programming

While, if P $\neq$ NP, there is no polynomial time solution to the Knapsack problem, one can still hope to find an algorithm for solving it which is, if not substantially faster, at least more telling on the nature of the solution than the naive algorithm of trying all possible combinations. Such an algorithm is given by the dynamic programming paradigm, which in its generality "is a method for solving a complex problem by breaking it down into a collection of simpler subproblems"[4].

In practice, for $k \leq n$ and $\mu \leq W$, one can define $V_{k,\mu}$ as the value of a subproblem $KP_{k,\mu}$, like such:

$$V_{k,\mu} \stackrel{def}{=} \max \left\{ \sum_{i=1}^{k} x_i v_i \text{ s.t. } \forall i \; x_i \in \{0,1\} \text{ and } \sum_{i=1}^{k} x_i w_i = \mu \right\}. \tag{3.1.1}$$

I.e. $KP_{k,\mu}$ is the problem of finding, among the set of objects in $\{1,\ldots,k\}$ that have compound weight μ, the set with the best total value. Let us also define $S_{k,\mu}$ as any set of indexes that achieves the optimal value for $KP_{k,\mu}$ - we then have $V_{k,\mu} = \sum_{i \in S_{k,\mu}} v_i$.

We can find a recursive rule which will let us calculate all the $V_{k,\mu}$, by reasoning like this: given an optimal solution $S_{k,\mu}$ for $\mathrm{KP}_{k,\mu}$, the k-th object can either be part of this solution or not. If $k \notin S_{k,\mu}$, then $S_{k,\mu} \subseteq \{1,\ldots,k-1\}$ which implies it is feasible and optimal for $\mathrm{KP}_{k-1,\mu}$ (since a better solution for $\mathrm{KP}_{k-1,\mu}$ would also be feasible for $\mathrm{KP}_{k,\mu}$, violating $S_{k,\mu}$'s optimality). Therefore:

$$V_{k,\mu} = V_{k-1,\mu}$$

If instead $k \in S_{k,\mu}$, then solving $\mathrm{KP}_{k,\mu}$ can be thought of as taking k, setting it apart, and, to find out which of the previous indexes are in $S_{k,\mu}$, solve $\mathrm{KP}_{k-1,\mu-w_k}$. Thus in this case we will have:

$$V_{k,\mu} = v_k + V_{k-1,\mu-w_k}$$

[4] `https://en.wikipedia.org/wiki/Dynamic_programming`, 20/6/2015

Since we don't know a priori if k will belong or not to $S_{k,\mu}$, we'll synthesize the above relations by taking the maximum (if it is at all possible to take the k-th object without violating the constraint):

$$V_{k,\mu} = \begin{cases} V_{k-1,\mu} & \text{if } w_k > \mu \\ & \text{(i.e. we can't take } k\text{)} \\ \max\left\{V_{k-1,\mu}\,, v_k + V_{k-1,\mu-w_k}\right\} & \text{otherwise} \end{cases} \tag{3.1.2}$$

The computation now amounts to filling in a $n \times W$ table, since what we are interested in are the greatest among $V_{n,\mu}$, for $\mu \in \{1, \ldots, W\}$. We need to initialize the first column and first row like this:

$$\begin{aligned} V_{k,0} &= 0 \qquad \forall k \in \{0, \ldots, n\} \\ V_{0,\mu} &= -\infty \quad \forall \mu \in \{1, \ldots, W\} \end{aligned} \tag{3.1.3}$$

Not only can we now iteratively compute the optimal value to (KP), but, if we keep track of which choice is done in the max in (3.1.2) (in each step of the iterative process), we'll also know all sets of indexes that give us this optimal value. In fact, for any k, μ, if $V_{k-1,\mu}$ wins in the maximum, that means that the k-th object was not taken, if instead $v_k + V_{k-1,\mu}$ is bigger, the k-th object is indeed part of the solution. If either choice is equivalent, that means that there will be at least two set of indexes that yield us the value $V_{k,\mu}$ (one with and one without k), and these cases are precisely what gives rise to the non-uniqueness of the solution.
Being that for every cell we simply evaluate (3.1.2), the time complexity is given by the dimension of the table, and is thus $O(nW)$.
To better explain how this can be implemented and gain some intuition on the method, we will detail a simple example.

Example 1. Suppose we have 4 objects with the weights and values in table 3.1, and suppose the capacity is $W = 5$. To calculate all the $V_{k,\mu}$ we can think of filling in a

object	1	2	3	4
weights w_i	2	1	3	2
values v_i	3	1	4	3

Table 3.1: weights and values of the objects

table such that the cell (k, μ) holds the value $V_{k,\mu}$. The first row and the first column of the table are initialized according to (3.1.3) and (3.1.2) is applied iteratively from left to right and from top to bottom (see figure 3.1).
Let's see for example how cell $(2, 3)$ is calculated. The weight of object 2 is 1 which is not greater than $\mu = 3$, so in (3.1.2) we have to evaluate the maximum. We have to compare the cell right above with the one on the same row but farther to the left by the object's weight (in this case 1) - to the latter though we have to add the object's value, in this case 1. Here the left cell wins, thus we write it's value plus the value of the object in cell $(2, 3)$. The algorithm would then move on to calculate cell $(2, 4)$.

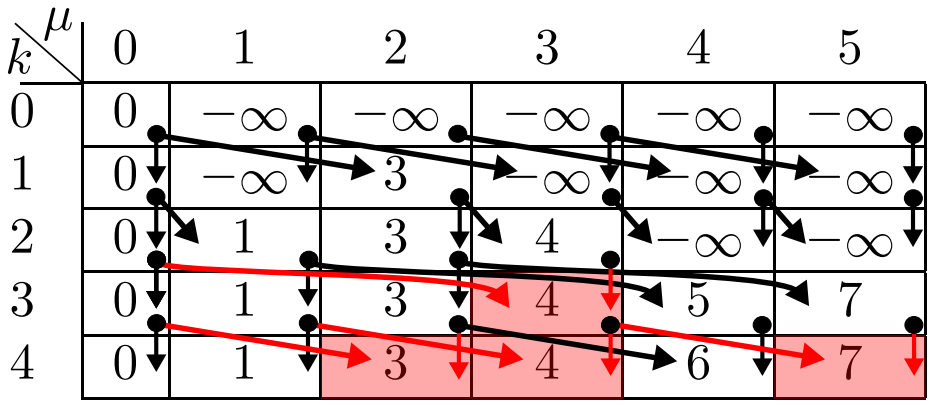


Figure 3.1: The fully computed table of the $V_{k,\mu}$ values. Every cell (k, μ) has an incoming arrow indicating the choice done in (3.1.2) for the computation of $V_{k,\mu}$; cells with two incoming arrows are those for which either choice in the maximum is equivalent, and they are colored in red.

In this case the solution is not unique: object 1 and 4 are interchangeable, therefore if there is a solution containing only one of them, there will be at least another equivalent solution. However, we can explicitly find all set of objects that achieve the optimal value 7: $S_1 = \{1, 3\}, S_2 = \{3, 4\}$ and $S_3 = \{1, 2, 4\}$ (incidentally here they all have maximum weight, but it's not true in general that different solutions that achieve the optimal value will have the same weight). To do this we first have to find the highest valued cell (this will always be in the last row)[5], and from there, having reversed all the arrows, we simply "walk back" until we reach the first column. For every path find like this we have a solution, composed of the first indexes of the cells with an outgoing oblique arrow. In other words we're adding to the solution those k for which the second argument won the maximum in (3.1.2) - see figure 3.2.

Every red cell (those for which the maximum in (3.1.2) was achieved by both the values being compared) now has (if at all) two arrows coming out of it, and thus we can conclude that, if the highest valued cell is red, there are $2 + \xi$ equivalent solutions, where ξ is the number of other red cells encountered when going backwards from the highest valued cell.

[5]Though it won't always be in the last column, since, remembering definition (3.1.1), the highest value of the $V_{k,\mu}$ could be $V_{n,\mu}$ for *any* $\mu \in \{1, \ldots, W\}$

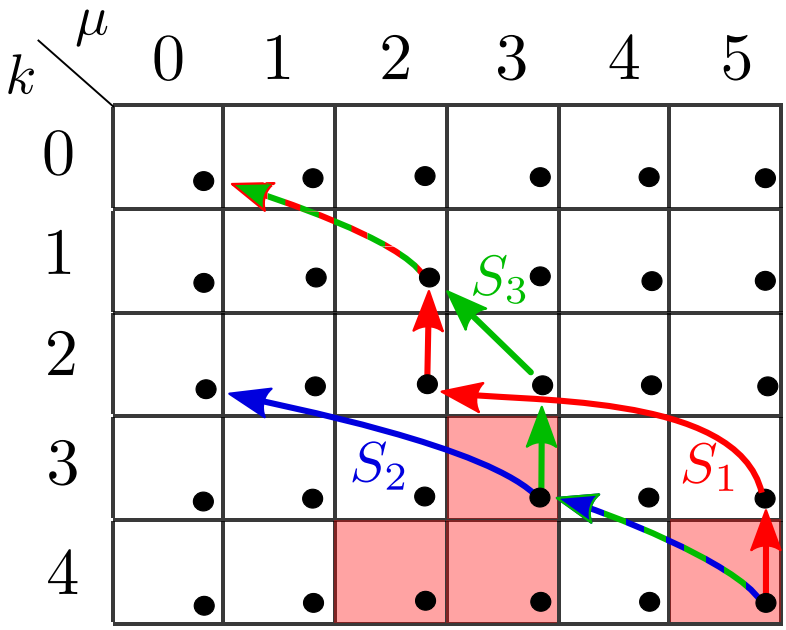


Figure 3.2: Solutions $S_1 = \{1, 3\}, S_2 = \{3, 4\}$ and $S_3 = \{1, 2, 4\}$ as paths

Remark 2. The **table method** we used in example 1 works also if the values v_k are real numbers, but the weights w_k need to be natural numbers - in fact we must be able to add weights to a column index and obtain another column index. In order to use real numbers for weights, we could for example multiply all weights by 10^M, where M is the number of significant figures used to represent them, and change the capacity from W to $W10^M$; with this trick the weights become integers but the solutions don't change, so the approach of example 1 can be used. The problem is that the table now has $W10^M$ columns. In the next section we'll reformulate the problem as a maximum value path on a graph, and this will work without modification if any of v_i, w_i and W are real numbers.

Remark 3. From example 1 it's apparent that all cells with $-\infty$ don't bring anywhere, in the sense that they will never win a comparison against a finite value. Thus in an actual implementation only cell $(0, 0)$ will be calculated from the first row.

Remark 4. The heuristic approximation S_{heur} used for (2.3.11) translates here to ordering objects by decreasing ratio $\frac{v_i}{w_i}$ and adding them to the candidate solution in this order, until the weight constraint is violated. In example 1, this would give us the set of objects $\{1, 4\}$, which has value 6 and is thus not optimal.

Remark 5. There are other ways to define subproblems that, when solved, give the solution to the original problem. For instance, instead of using (3.1.1), one could define $W_{k,\varphi}$ as the minimum weight of a solution consisting of objects in $\{1, \ldots, k\}$ and having

value φ, i.e.:

$$W_{k,\varphi} \stackrel{def}{=} \min \left\{ \sum_{i=1}^{k} y_i w_i \text{ st } \forall i \; y_i \in \{0,1\} \text{ and } \sum_{i=1}^{k} y_i v_i = \varphi \right\}. \tag{3.1.4}$$

Reasoning on whether the k-th object is or not in the solution, exactly like for (3.1.2), we obtain a similar recursion rule:

$$W_{k,\varphi} = \begin{cases} W_{k-1,\varphi} & \text{if } v_k > \varphi \\ & \text{(i.e. we can't take } k\text{)} \\ \min \left\{ W_{k-1,\varphi}, w_k + W_{k-1,\varphi-v_k} \right\} & \text{otherwise} \end{cases} \tag{3.1.5}$$

which can be used to compile a table, exactly as in example 1. The optimal value of (KP) is then

$$\max \left\{ \varphi \mid W_{n,\varphi} \leq W \right\},$$

and can thus be retrieved among the last row of the table.
Remark 2 would translate here in the necessity for the values, not the weights, to be natural numbers. Again one could multiply the values by 10^M for M big enough to make them all integer, at the cost of the number of columns now being $V10^M$ instead of V (where $V \stackrel{def}{=} \sum_{i=1}^{n} v_i$).

3.2 KP's decision graph algorithm

In this section we want to introduce informally the concept of the decision graph and the proposed algorithm to solve an optimal value path problem; in the next section we'll formalize these concepts and prove the correctness of the algorithm.

From example 1 one can already guess how the problem can actually be formulated as a maximum value path graph problem. Instead of a table we draw a plane with k on one axis and μ on the other, and a node at point (k, μ) for every subproblem $KP_{k,\mu}$. Each node $a = (k, \mu)$ is given a label α_a, at all times equal to the best known value of a path leading to it: initially it is not defined (or initialized to $-\infty$) and equal to $V_{k,\mu}$ at the end of the algorithm. If the node is not a leaf (i.e. $k < n$), it has exactly two directed edges coming out of it, going to nodes $(k+1, \mu)$ and $(k+1, \mu + w_{k+1})$ respectively. The first edge is horizontal and corresponds to not taking the $k+1$-th object - it has label 0 and we'll call these **type 0** edges; the second one is not horizontal[6] and corresponds to taking the $k+1$-th object - it has label v_{k+1} and we'll call these **type 1** edges. If there is an edge going from node a to node b, we'll call it's label $\beta_{a,b}$; it can either be 0 if the edge is of type 0 or v_{k+1} if it is of type 1; see figure 3.3

We won't take type 1 edges that would reach a node with second coordinate greater than W, since we don't need to solve subproblems $KP_{k,\mu}$ with $\mu > W$; this is a tree

[6] Remember we can suppose all the weights to be non-zero.

pruning technique which allows us to explore a smaller portion of the decision graph.

Remark 6. The edge labels are constant throughout the algorithm, while the node labels get updated as the graph is explored - as said, they'll finally be equal to $V_{k,\mu}$.

This is called the **decision graph** because a path from the root node to any node a represents a set of objects, precisely all objects k for which the $k-1 \to k$ edge in the path was of type 1 (the correspondence "path $\leftrightarrow$ set of objects" is the same as that in figure 3.2). At every node (k, μ) we can decide whether to take the outgoing type 1 edge or the type 0, i.e. we decide whether to take the $k+1$-th object or not; the edge label represents how much the value of the corresponding solution increases by taking that edge. The label α_a of a node a is the maximum value of all the paths from $(0,0)$ to a, where the value of a path is the sum of the labels of the edges that compose it. Thus, since only type 1 edges have positive label, α_a is the sum of values of the objects in the set of objects corresponding to a.

The idea is then to dynamically construct the decision graph by starting from root node $(0,0)$ and taking all possible paths, computing node coordinates and their labels on the fly, in order to find the node with the highest label and thus the set of objects that compose an optimal solution for (KP). We will do this using a slight adaption of the classic **shortest path tree** algorithm for directed acyclic graphs (**DAG**).

First of all note that the graph is indeed acyclic, since all edges are from a node with k as first coordinate to one with $k+1$. Like any DAG it thus can be **topologically sorted**: one can assign a number $\delta_{k,\mu} \in \mathbb{N}$ to any node (k, μ) in such a way that, if an edge from (k_1, μ_1) to (k_2, μ_2) exists, then $\delta_{k_1,\mu_1} < \delta_{k_2,\mu_2}$ must hold. To establish such an order, it suffices to choose an ordering of nodes that gives precedence to the first coordinate, i.e. for every $k_1 < k_2$, δ_{k_1,μ_1} must be strictly smaller than δ_{k_2,μ_2}, for any choice of μ_1 and μ_2. Visiting the nodes in such an order guarantees us that we'll always have examined all the incoming edges to a node before visiting the node itself.

The procedure of the algorithm is as follows: starting from the root node $(0,0)$, we iterate on the nodes following the topological sorting. For every node τ we consider its two sons, and for each of them we consider the node label of τ plus the edge label connecting them to τ: this number represents the value of the path from the root node to the son that is composed of the optimal path from the root node to τ plus the last edge, connecting τ and the son. If its greater than the current node label of the son (which is initialized to $-\infty$ but could have been already updated with this same procedure, since it could be son to more than one node), then the son's node label is updated, and an array `predecessor` is updated to reflect that the best known predecessor to the son is τ.

More formally, for every node $\tau = (k, \mu)$ and for each $i = 1, 2$, σ_i being its two sons $(k+1, \mu)$ and $(k+1, \mu+w_{k+1})$, this piece of code is executed:

if $\alpha_\tau + \beta_{\tau,\sigma_i} > \alpha_{\sigma_i}$ **then**
  $\alpha_{\sigma_i} = \alpha_\tau + \beta_{\tau,\sigma_i}$
  $\texttt{pred}[\sigma_i] = \tau$
end if

The condition in the first line is known as **Bellman condition**, and it corresponds to

evaluating the max in the second line of equation (3.1.2). If the two quantities being compared are actually equal, this means that τ will give to its son a value equal to the one he already had - in other words there are equivalent valued paths leading from the root node to the son.
Every time a node label α_σ is updated its value is compared to a global variable `best_value`, initialized to $-\infty$; if α_σ is the greater between the two then `best_value` is set to α_σ and `best_node`, another global variable initialized to `None`, is set to σ. In this way at the end of the algorithm, recursively using the `predecessor` array and starting from `best_node`, we can rebuild the highest valued path in the graph.

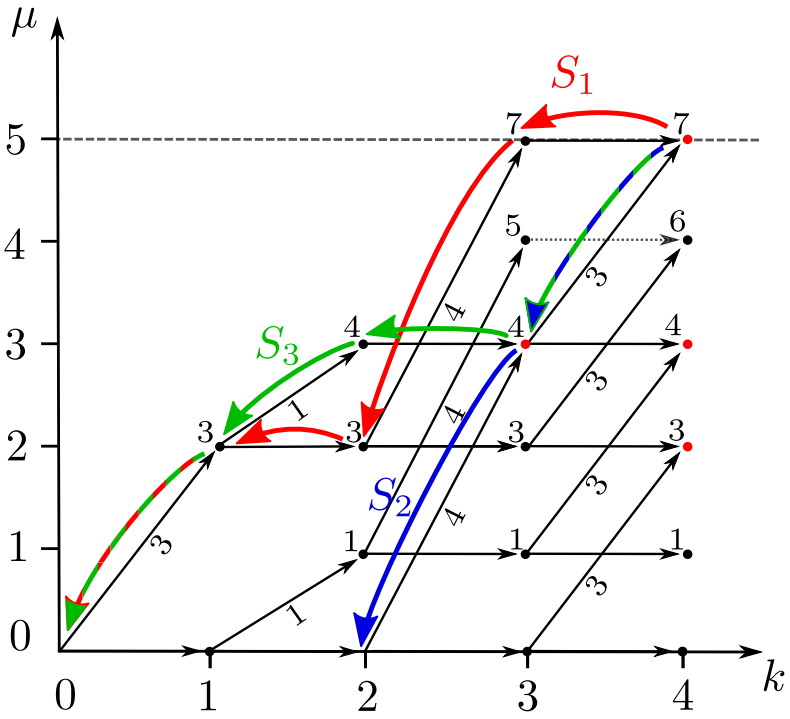


Figure 3.3: Fully computed decision graph of example 1. The small black numbers on the graph are the node and edge labels (only labels of type 1 edges are shown since type 0 edges have always label 0). The red nodes are those for which both incoming edges give the same value. Node $(4, 4)$ has two edges incoming, but the type 0 edge would give it a smaller label than the type 1, so it's discarded. The colored arrows represent the solutions, just like in figure 3.2.

Remark 7. In an actual implementation one could avoid visiting some nodes; for example if we're visiting node (k, μ), we will consider the edge going to $(k, \mu + w_{k+1})$ only if $\mu + w_{k+1} \leq W$. In section 3.6 we'll examine in detail some of these pruning strategies for (2.3.11).

Remark 8. Since the number of operations is constant for every iteration, time complexity is then simply given by the number of nodes. For each k there are at most W nodes

with first coordinate k, thus time complexity is, just like for the table approach, $O(nW)$. This is a **pseudo-polynomial** time complexity, which means that while it's polynomial in the numerical value W, it's exponential in the number of bits needed to represent W in memory: if we call $\theta = \log_2 W$ the approximate length of W's binary representation, time complexity can be written as $O(n2^\theta)$. This observation doesn't apply to n because it's not an input of the algorithm: the inputs are W and the arrays $(v_i)_{i=1}^n$ and $(w_i)_{i=1}^n$, whose size in memory doesn't change based on how we represent n.

Remark 9. The construction of the decision graph (either using $V_{k,\mu}$ or $W_{k,v}$) is exactly the same if any of v_i, w_i or W are real numbers instead of integers.

Remark 10. One could have alternatively stated (3.1.1) as

$$V_{k,\mu} \stackrel{def}{=} \max \left\{ \sum_{i=1}^{k} x_i v_i \text{ st } \forall i \; x_i \in \{0,1\} \text{ and } \sum_{i=1}^{k} x_i w_i \leq \mu \right\}, \tag{3.2.1}$$

i.e. with a $\leq$ instead of $=$ for the constraint (the first row in the table should then be initialized to 0 instead of $-\infty$). While this could seem more intuitive, the equivalent graph problem would not be as clean as the decision graph, since there would be $W - w_1$ root nodes from which to start the graph visit [7] - easily circumvented difficulty, but it can be avoided altogether (and make for a cleaner graph) by using the definition of the subproblems with the $=$ sign in the constraints.

Remark 11. If we wish to find an optimal solution to KP using the subproblems of remark 5, we can still construct a decision graph and find an optimal solution by exploring it. There will be a node (k, v) for every subproblem, the labels on the nodes will now be $W_{k,v}$ instead of $V_{k,\mu}$, and the labels on the edges will be the weights w_k instead of the values v_k. We'll still visit the graph by keeping two lists, one for nodes with first coordinate k and one for their sons, and for every visited node check the Bellman condition, update the `predecessor` array and the `best_node` variable. The number of operations per iteration is again constant, so, given that for fixed k there at most V nodes (k, v), where $V = \sum_{i=1}^{n} v_i$, time complexity is $O(nV)$. The same observation of remark 8 holds here: this time complexity is pseudo-polynomial.

3.3 Correctness of decision graph algorithm

The decision graph algorithm is a special case of an optimal path algorithm for DAGs. In this section we want to formally state this more general problem, illustrate an algorithm and prove its correctness. As a consequence we'll thus have that the decision graph algorithm, both as detailed in the previous section for the Knapsack problem and in the

[7] To see this, think of the second row of the table method: if the first row is set to $-\infty$, only one element in the second row will be non-zero, namely $1, w_1$, while if the first row is entirely 0, all cells $1, \mu$ will be equal to v_1 for $\mu \geq w_1$. This translates to the corresponding graph having respectively one or $W - w_1$ roots.

next one for (BPV), is indeed correct and gives us an optimal solution. Please note the notations used in this section have no relation to the other sections.

Let $G = (N, E)$ be a graph, $N = \{1, \dots, n\}$ its set of nodes and $E \subset N \times N$ its set of edges. For every node $k \in N$ and for every edge $e \in E$, we have a node and an edge *label*, which are simply numbers $\alpha_k, \beta_e \in \mathbb{R}$; the edge labels are considered immutable and an input of our problem, while the node labels are an auxiliary quantity we introduce and which we'll reassign multiple times during the execution of the algorithm. We suppose the graph is directed and acyclic (DAG), which means we can define a **topological sorting**

$$\delta : N \to \mathbb{N}$$

such that

$$(a, b) \in E \Rightarrow \delta(a) < \delta(b) \ .$$

In order to simplify notations, we'll suppose to have reindexed the nodes according to the order provided by δ; this way the previous property can be restated as

$$(a, b) \in E \Rightarrow a < b \ .$$

We'll call node 1 the *root* node, and we'll suppose that all nodes are reachable starting from the root node. A *path* is a subset $S \subset N$

$$S = \{s_1, \dots, s_k\} \text{ such that } \forall j < k \quad (s_j, s_{j+1}) \in E$$

i.e. it's a succession of adjacent edges. We'll be interested in paths that start at the root node, and we'll denote the set of these paths with $\mathcal{S}_1$; $\mathcal{S}_{1,k}$ is the set of paths starting in the root node and terminating in $k \in N$. We'll call the *value* of the path S the quantity

$$V(S) = \sum_{j=1}^{k-1} \beta_{(s_j, s_{j+1})}, \quad \text{where } S = \{s_1, \dots, s_k\}.$$

For every node k we'll define its *forward star* as

$$\gamma(k) = E \cap \bigcup_{j \in N} (k, j) \ ,$$

i.e. $\gamma(k)$ is the set of outgoing edges from k. The fact that the graph is a topologically sorted DAG implies

$$\forall k \in N \quad \gamma(k) \subset \{k + 1, \dots, n\} \ ,$$

and the condition that any node is reachable from the root node may then be written as

$$\forall k \in N \quad \exists h < k \text{ s.t. } k \in \gamma(h) \ . \tag{3.3.1}$$

The problem we want to solve is finding the path starting at the root node with the greatest value; formally:

$$\max_{S \in \mathcal{S}_1} V(S) \ . \tag{3.3.2}$$

Algorithm 1 Optimal path algorithm for DAGs

```
1: ∀k ∈ N  α_k = −∞
2: α_1 = 0
3: predecessor[1] = 1
4: best_node = None
5: best_value = −∞
6: for k = 1, ..., n do
7:    for j ∈ γ(k) do
8:       if α_k + β_{k,j} > α_j then
9:          α_j = α_k + β_{k,j}
10:         predecessor[j] = k
11:         if α_j > best_value then
12:            best_node = j
13:            best_value = α_j
14:         end if
15:      end if
16:   end for
17: end for
```

Theorem 3.3.3. *At the end of algorithm's 1 execution, the path* $\pi($`best_node`$)$ *is a solution to problem* (3.3.2)*, where*

$$\forall k \in N \quad \pi(k) = \cup_{j \in \mathbb{N}} \pi_j(k)$$

and

$$\begin{cases} \pi_1(k) & = k \\ \pi_{j+1}(k) & = \texttt{predecessor}[\pi_j(k)] \end{cases}$$

Proof. Consider, for $\delta = 1, \ldots, n$ the following proposition:

$$\begin{aligned} \mathbf{P}(\delta): \quad & \text{at the end of iteration } \delta, \text{ for every } k \leq n \wedge \delta + 1, \\ & \pi(k) \text{ is solution to } \max_{S \in \mathcal{S}_{1,k}} V(S) \text{ and its value is } \alpha_k. \end{aligned}$$

It's clear that if $\mathbf{P}(n)$ is true then we're finished: `best_node` will be the node with the

best label (since every time we update a label we check to see if its better than the previous best one) and thus searching for the maximum $V(S)$ among S in $\mathcal{S}_{1,\texttt{best\_node}}$ or $\mathcal{S}_1$ is equivalent.

We'll prove $\mathbf{P}(\delta)$ by induction on $\delta = 1, \dots, n$. For the base case $\mathbf{P}(1)$, consider that during the first iteration all nodes in $\gamma(1)$ are visited, and among them must be node 2 (remember we're supposing all nodes are reachable from the root, and node 2 wouldn't be reachable otherwise because of the DAG property). When node 2 is visited, α_2 and `predecessor`$[2]$ are updated to $0 + \beta_{1,2} = \beta_{1,2}$ and 1 respectively. Thus $\pi(2) = \{1, 2\}$ is the only element of $\mathcal{S}_{1,2}$ and has value $V(\pi(2)) = \beta_{1,2} = \alpha_2$, while $\pi(1) = \{1\}$ with value $0 = \alpha_1$ is the only element of $\mathcal{S}_{1,1}$, so $\mathbf{P}(1)$ is true.

Supposing $\mathbf{P}(\delta)$ to be true, we now want to prove $\mathbf{P}(\delta + 1)$. If $k \leq \delta + 1$ then, by inductive hypothesis, $\pi(k)$ is solution to $\max_{S \in \mathcal{S}_{1,k}} V(S)$ and α_k is its value. Then let $k = \delta + 2$; the idea is that $k \in \gamma(h)$ for at least one $h \in \{1, \dots, \delta + 1\}$ and, because of the condition in line 8 in the pseudocode, at the end of iteration $\delta + 1 = k - 1$ the node giving the best possible value to k will be stored in `predecessor`$[k]$. Formally we're saying that, because of 3.3.1,

$$\{h \in \{1, \dots, \delta + 1\} \text{ s.t. } k \in \gamma(h)\} \neq \emptyset$$

holds, and thus we can choose among the best such indexes:

$$\bar{h} \in \underset{h \in \{1,\dots,\delta+1\},\, k \in \gamma(h)}{\operatorname{argmax}} \alpha_h + \beta_{h,k} \ . \tag{3.3.4}$$

Because of line 9 and 10 respectively we'll then have

$$\alpha_k = \alpha_{\bar{h}} + \beta_{\bar{h},k}$$

and

$$\bar{h} = \texttt{predecessor}[k] \ .$$

By definition of π we'll have

$$\begin{aligned} \pi(k) &= k \cup \pi(\bar{h}) \\ &= \{k, \bar{h}, \texttt{predecessor}[\bar{h}], \texttt{predecessor}[\texttt{predecessor}[\bar{h}]], \dots, 1\} \end{aligned}$$

and, by definition of V and inductive hypotheses,

$$\begin{aligned} V(\pi(k)) &= V(\pi(\bar{h})) + \beta_{\bar{h},k} \\ &= \alpha_{\bar{h}} + \beta_{\bar{h},k} \\ &= \alpha_k \ . \end{aligned}$$

We finally need to prove that $\pi(k)$ is a solution to $\max_{S \in \mathcal{S}_{1,k}} V(S)$; suppose $\rho = \{\rho_1, \dots, \rho_l\} \in \mathcal{S}_{1,k}$ to be a better path to k, i.e. $V(\rho) > V(\pi(k))$. If we define $\rho' = \{\rho_1, \dots, \rho_{l-1}\}$, by

inductive hypotheses we have

$$\begin{aligned} V(\rho) &= V(\rho') + \beta_{\rho_{l-1},k} \\ &= \alpha_{\rho_{l-1}} + \beta_{\rho_{l-1},k} \\ &> \alpha_k \\ &= \alpha_{\bar{h}} + \beta_{\bar{h},k} \end{aligned}$$

which is absurd for the definition of $\bar{h}$, since $\rho_{l-1} \in \{1, \dots, \delta{+}1\}$ and $k = \rho_l \in \gamma(\rho_{l-1})$. ■

Remark 12. There is a non-unique solution to (3.3.2) when there are two different paths $S^1, S^2 \in \mathcal{S}_1$ such that $V(S^1) = V(S^2) = \max_{S \in \mathcal{S}_1} V(S)$. This can happen if, during the graph visit done by algorithm 1, one of these two situations happens:

1. there is a choice that must be done in (3.3.4), or equivalently the two sides of the inequality at line 8, known as the **Bellman condition**, are actually equal. This happens when there are two paths with the same value ending in the same node.
2. the two sides of the inequality at line 11 are actually equal. This happens when there are two paths with the same value, which is the best value known up to that point.

If in those two lines we substitutde the strict inequality $>$ with a $\geq$, the algorithm is still correct - it would simply, in case of multiple solutions, output the last optimal path found instead of the first.
In the actual python implementation of algorithm 2, which is an adaption of algorithm 1 to solve (2.3.11) and which will be discussed in detail in the following section, `predecessor` is implemented not as an array, but as a dictionary, and `best_node` is a list of values. The keys of the `predecessor` dictionary are nodes, and the values are lists of nodes that can equivalently be taken as predecessor to them; the items of the `best_node` list are all the nodes achieving, through the optimal path leading to them from the root node, the value `best_value`. At the end of the algorithm, if `best_node` has more than one item or if it has one item but `predecessor[best_node]` is a list with more than 1 element, the algorithm affirms there are multiple optimal solutions.

Remark 13. If we wish to solve

$$\min_{S \in \mathcal{S}_1} V(S) \tag{3.3.5}$$

instead, we can do so with a very simple adaption of algorithm 1: it suffices to initialize `best_value` and the node labels to ∞ instead of $-\infty$ and reverse the inequalities on lines 8 and 11 (maintaining them strict).

Remark 14. Algorithm 1 examines each edge exactly once, and for each edge it does a constant number of operations, thus time complexity is $O(|E|)$. In the Knapsack and in (2.3.11)'s decision graph, every node has at most two outgoing edges, thus time complexity is linear in the number of nodes, i.e., with the notations of this section, it's $O(|N|)$.

3.4 BPV's decision graph algorithm: $\mathcal{H}$-formulation

We can finally extend the ideas from the previous sections to (2.3.11): just as for the Knapsack problem, we can define subproblems of the main problem, and among the solutions of these auxiliary problems will be a solution to the main problem. The optimal values of the subproblems satisfy a recursion rule which we can use to iteratively calculate the solution to one problem by using the previously calculated ones for other problems. All this will have a clear and intuitive interpretation once we reformulated the problem as an optimal path problem on the decision graph.

Drawing a parallel to (3.1.1), we wish to consider the problem of finding the optimal entropy set of pattern indexes, restricting ourselves to solutions contained in $\{1,\dots,k\}$, with cardinality ν and compound rate μ, i.e.:

$$\begin{aligned}
&\forall k \in \{1,\dots,n\}\,,\\
&\forall \nu \in \mathbb{N},\ \nu \leq N\,,\\
&\forall \mu \in \mathbb{R},\ 0 \leq \mu \leq W\,,\\
&\mathcal{H}_{k,\mu,\nu} = \max\left\{ \sum_{i=1}^{k} x_i p_i \log\frac{1}{p_i} \text{ st } \forall i\ x_i \in \{0,1\} \text{ and } \sum_{i=1}^{k} x_i p_i = \mu,\ \sum_{i=1}^{k} x_i = \nu \right\}.
\end{aligned} \tag{3.4.1}$$

With an abuse of language, we'll use $\mathcal{H}_{k,\mu,\nu}$ to refer both to this subproblem and to its optimal value - it should be clear from context which meaning we're using. When we want to refer to the whole set of these subproblems for varying values of k, μ and ν, or simply to this approach at solving the problem, we'll talk about the **$\mathcal{H}$-formulation**.

Remark 15. Since $\sum_{i=1}^{k} x_i p_i$ can be equal to at most 2^k different values in $[0, W]$ (less if there are different sets of patterns with the same rate), the majority of $\mu \in [0, W]$ (in fact all but a set of Lebesgue measure 0) make the subproblem $\mathcal{H}_{k,\mu,\nu}$, for any k and ν, have an empty feasible set, i.e. there are no binary strings satisfying its constraints.

Remark 16. It's easy to see how knowing the solution to all the auxiliary problems (3.4.1) gives us the solution to (2.3.11): the optimal value of the main problem is

$$\max_{\mu \leq W, \nu \leq N} \mathcal{H}_{n,\mu,\nu}$$

and, if $\bar{\mu}, \bar{\nu}$ realize this maximum, then the binary string $\bar{x} \in \{0,1\}^n$ that is a solution to subproblem $\mathcal{H}_{n,\bar{\mu},\bar{\nu}}$ is also a solution to (2.3.11).

We can reason, just like for the Knapsack $K_{k,\mu}$ subproblems, on whether the k-th object will be part or not of the solution to $\mathcal{H}_{k,\mu,\nu}$, and obtain the following recursion rule:

$$\mathcal{H}_{k,\mu,\nu} = \begin{cases} \mathcal{H}_{k-1,\mu,\nu} & \text{if } p_k > \mu \\ & \text{(i.e. we can't take } k\text{)} \\ \max\left\{\mathcal{H}_{k-1,\mu,\nu}\,, p_k \log\frac{1}{p_k} + \mathcal{H}_{k-1,\mu-p_k,\nu-1}\right\} & \text{otherwise.} \end{cases} \tag{3.4.2}$$

Using this relationship between the values $\mathcal{H}_{k,\mu,\nu}$, we can define the decision graph problem, which we will show is equivalent to (2.3.11). This graph problem conforms to the optimal path problem described in section 3.3, and can thus be solved using algorithm 1. We'll give an iterative definition of the nodes and edges that constitute the graph, which will let us visit the graph starting from the root node $(0,0,0)$ without prior knowledge of the set of nodes and edges; if we call $M^{\mathcal{H}}$ the set of nodes and $E \subset M^{\mathcal{H}} \times M^{\mathcal{H}}$ the set of edges, we'll have:

$$\begin{cases} (0,0,0) & \in M^{\mathcal{H}} \\ (k,\mu,\nu) & \in M^{\mathcal{H}} \Rightarrow \begin{cases} \sigma_0(k,\mu,\nu) \in M^{\mathcal{H}} & \text{and } ((k,\mu,\nu),\sigma_0(k,\mu,\nu)) \in E \\ \text{if } k+1 \leq n & \\ \sigma_1(k,\mu,\nu) \in M^{\mathcal{H}} & \text{and } ((k,\mu,\nu),\sigma_1(k,\mu,\nu)) \in E \\ \text{if } k+1 \leq n, & \mu + p_{k+1} \leq W \text{ and } \nu + 1 \leq N \end{cases} \end{cases} , \tag{3.4.3}$$

where σ_0 and σ_1 denote the two sons a node can have:

$$\begin{aligned} \sigma_0, \sigma_1 &: M^{\mathcal{H}} \to M^{\mathcal{H}} \\ \sigma_0(k,\mu,\nu) &= (k+1,\mu,\nu) \\ \sigma_1(k,\mu,\nu) &= (k+1,\mu+p_{k+1},\nu+1) \ . \end{aligned}$$

An explicit but less useful definition of $M^{\mathcal{H}}$ and E would be:

$$\begin{aligned} M^{\mathcal{H}} &= \{(k,\mu,\nu) \text{ s.t. } \mathcal{H}_{k,\mu,\nu} \text{ has a non-empty feasible set}\} \\ E &= \{(m_1,m_2) \text{ s.t. } m_1, m_2 \in M^{\mathcal{H}} \text{ and } m_2 = \sigma_0(m_1) \vee m_2 = \sigma_1(m_1)\} \ . \end{aligned}$$

For convenience we'll call the two type of edges **type 0** and **type 1** edges, formally defining an application τ assigning to every edge its type:

$$\begin{aligned} \tau &: E \to \{0,1\} \\ \tau(m_1,m_2) &= \begin{cases} 0 & \text{if } \ m_2 = \sigma_0(m_1) \\ 1 & \text{if } \ m_2 = \sigma_1(m_1) \end{cases} \ . \end{aligned}$$

Finally we need to define edge and node labels:

- for every $(k,\mu,\nu) \in M^{\mathcal{H}}$, the node label $\alpha_{k,\mu,\nu} \in \mathbb{R}$ is at all times (during the graph visit) equal to the best known entropy among the feasible solutions in $\mathcal{H}_{k,\mu,\nu}$,
- for every type 0 edge $(m_1,\sigma_0(m_1)$ the edge label will be $\beta_{m_1,\sigma_0(m_1)} = 0$,
- for every type 1 edge $(m_1,\sigma_1(m_1))$, if $m_1 = (k,\mu,\nu)$, the edge label will be $\beta_{m_1,\sigma_1(m_1)} = p_{k+1} \log \frac{1}{p_{k+1}}$.

Remark 17. Just as for the Knapsack, this graph is a DAG, though now it's three dimensional and not planar, since the first coordinate can increase independently of the other two. To have a topological sorting it suffices to order nodes by their first coordinate,

is this a correct explanation?

and it doesn't matter how we order nodes that have the same first coordinate: in fact nodes with the same coordinate have no edges between them.

The optimal path problem we want to solve on this graph is

$$\max_{S \in \mathcal{S}_r} H(S) \ , \tag{3.4.4}$$

where $\mathcal{S}_r$ is the set of paths in the graph starting from the root node $r = (0, 0, 0)$, and, if $S = \{s_0, \dots, s_j\}$, $j \leq n$ is such a path, then

$$H(S) = \sum_{k=0}^{j-1} \tau(s_k, s_{k+1}) p_{k+1} \log \frac{1}{p_{k+1}} \ .$$

For an intuitive understanding of the decision graph, we must understand how every path in it corresponds to a choice of pattern indexes. If we think of starting a graph visit at the root node $(0, 0, 0)$, at each step taking one of the two outgoing edges means either taking a pattern index or not: if we are in node (k, μ, ν), choosing the type 1 edge means adding pattern $k + 1$ to our solution, and thus gaining its contribution $p_{k+1} \log \frac{1}{p_{k+1}}$ to the solution entropy, which is exactly the label of the edge we've chosen. Choosing the type 0 edge instead means we ignore pattern $k + 1$, and that is why the label of type 0 edges is 0.
In order to prove equivalence of (3.4.4) with (2.3.11), we need to formalize this correspondence between paths in the graph and binary strings $(x_i)_{i=1}^n$. For every path $S \in \mathcal{S}_r$, $S = \{s_0, \dots, s_n\}$[8], we'll define the values

$$W(S) = \sum_{k=0}^{n-1} \tau(s_k, s_{k+1}) p_{k+1} \ ,$$
$$\kappa(S) = \sum_{k=0}^{n-1} \tau(s_k, s_{k+1}) \ .$$

We can then define the feasible sets of the two optimization problems (3.4.4) and (2.3.11) as, respectively [9]:

$$\mathcal{F} = \{S \in \mathcal{S}_r \text{ s.t. } W(S) \leq W, \ \kappa(S) \leq N\}$$
$$\mathcal{G} = \{(x_i)_{i=1}^n \in \{0,1\}^n \text{ s.t. } W((x_i)_{i=1}^n) \leq W, \ \kappa((x_i)_{i=1}^n) \leq N\} \ .$$

We then have:

Theorem 3.4.5. *The function $F : \mathcal{F} \to \mathcal{G}$ defined on $S = \{s_0, \dots, s_n\} \in \mathcal{F}$ as*

$$F(S) = (\tau(s_{i-1}, s_i))_{i=1}^n \ ,$$

[8] from now on we'll always suppose a path S to have exactly n nodes; if it has less, we can iteratively append to it type 0 sons without affecting the values of $H(S), W(S)$ and $\kappa(S)$

[9] see equation (2.3.12) for the definitions of $H(\cdot), W(\cdot)$ and $\kappa(\cdot)$ on binary strings

is a bijection and it preserves the entropy, rate and cardinality of solutions, i.e.:

$$\begin{aligned} H(F(S)) &= H(S) \\ W(F(S)) &= W(S) \\ \kappa(F(S)) &= \kappa(S) \ . \end{aligned} \tag{3.4.6}$$

Proof. Relationships (3.4.6) follow immediately from the definitions of $H(\cdot), W(\cdot), \kappa(\cdot)$ on $\mathcal{F}$ and $\mathcal{G}$ and the definition of F.

To prove injectiveness, let $S^1 = \{s_0^1, \ldots, s_n^1\}$ and $S^2 = \{s_0^2, \ldots, s_n^2\}$ be two different feasible paths in $\mathcal{F}$, and let

$$i = \min_{j=1,\ldots,n} \{s_j^1 \neq s_j^2\} \ .$$

Since $s_0^1 = s_0^2 = r$, $i \geq 1$ must hold and we can say $s_{i-1}^1 = s_{i-1}^2$. Now, both s_i^1 and s_i^2 must be sons of s_{i-1}^1, because a path is a succession of nodes each one son of the preceding one; we'll then have that $\tau(s_{i-1}^1, s_i^1)$ and $\tau(s_{i-1}^2, s_i^2)$ will be different, and these are exactly the i-th component of $F(S^1)$ and $F(S^2)$, which therefore must be different binary strings.

To prove surjectiveness, let $(x_i)_{i=1}^n \in \mathcal{G}$; we'll define

$$\begin{cases} s_0 & = (0,0,0) \\ s_{k+1} & = \begin{cases} \sigma_0(s_k) & \text{if } x_{k+1} = 0 \\ \sigma_1(s_k) & \text{if } x_{k+1} = 1 \ . \end{cases} \end{cases}$$

This implies that

$$\tau(s_k, s_{k+1}) = x_{k+1} \ , \tag{3.4.7}$$

and thus $S = \{s_0, \ldots, s_n\}$ is indeed an element of $\mathcal{F}$, because

$$\begin{aligned} W(S) &= \sum_{k=0}^{n-1} \tau(s_k, s_{k+1}) p_{k+1} \\ &= \sum_{k=0}^{n-1} x_{k+1} p_{k+1} \\ &= \sum_{k=1}^{n} x_k p_k \\ &= W((x_i)_{i=1}^n) \\ &\leq W \ , \end{aligned}$$

and similarly $\kappa(S) = \kappa((x_i)_{i=1}^n)) \leq N$. Finally, it follows trivially from (3.4.7) that $F(S) = (x_i)_{i=1}^n$. ∎

Algorithm 2 BPV decision graph algorithm in the $\mathcal{H}_{k,\mu,\nu}$ formulation, with pruning. The `add_node` function takes care of updating the `alpha` and `predecessor` dictionaries, the `best_entropy` and `best_node` to keep track of the best known node and finally adding, if not already present, the children to `next_visitlist`.

visitlist = list($(-1, 0, 0)$)
next_visitlist = list()
while visitlist.notEmpty() **do**
  cur = (k, μ, ν) = visitlist.pop()
  $\sigma_0 = (k + 1, \mu, \nu)$
  $\sigma_1 = (k + 1, \mu + p_{k+1}, \nu + 1)$
  if $k + 1 < n$ **and** $\mu + p_{k+1} \leq W$ **and** $\nu + 1 \leq N$ **and** α[cur] + $\eta[k + 1] \geq$ best_entropy **then**
    add_node(σ_0)
    add_node(σ_1)
  end if
  if visitlist.empty() **then**
    visitlist = next_visitlist
    next_visitlist = list()
  end if
end while

To solve (3.4.4) we'll use algorithm 2, which is essentially algorithm 1 with a few tweaks for this specific case; the structure is slightly different and is quite close to the actual python implementation of the `decgraph_solver` in the `BPV` class. A few explanations are in order:

- We're supposing the patterns to be indexed in such a way that $(p_k)_{k=0}^{n-1}$ is an increasing sequence
- Since arrays in python start with index 0, the first pattern in the code will be 0, and thus the root node is $(-1, 0, 0)$
- In order to apply the main `for` loop structure of algorithm 1, we would have to first explicitly build a topological sorting to number the nodes accordingly. We would then need to explore all the graph twice: once for finding out the nodes and edges and build the topological sorting and once to actually solve the optimal path problem. However, we can comply to the structure of algorithm 1, and thus be sure of finding an optimal solution thanks to theorem 3.3.3, by visiting the graph by *any* topological sorting, we don't care which. Thanks to remark 17, we can do this simply by maintaining a `visitlist` consisting only of nodes with the same first coordinate, and `next_visitlist` where we add their sons. When we visited all nodes in `visitlist`, we copy `next_visitlist` onto it and initialize `next_visitlist` to a new list.

- Line 7 consists of various checks that allow us to not visit sons to the current node under certain conditions, which effectively limits the portion of graph we explore; for now we'll make some brief remarks, and in section 3.6 we'll analyze these conditions in more detail.
  - $\eta : \{1, \dots, n\} \to [0, H_{\text{tot}}]$ is defined as

$$\eta(j) = \sum_{i=j}^{n} p_i \log \frac{1}{p_i} \ , \tag{3.4.8}$$

    i.e. the entropy given by taking in the solution all the patterns with index greater than j. In the code, $\alpha[\text{cur}]+\eta[k+1]$ is the entropy we would obtain by taking the optimal path to the current node plus all successive type 1 edges; if this is not greater than the previous best known entropy, there's no point in exploring this part of the graph.
  - Because $(p_k)_{k=0}^{n-1}$ is an increasing sequence,

$$\mu + p_{k+1} > W \Rightarrow \forall j > k+1,\ \mu + p_j > W \ ,$$

    and thus any type 1 descendant of (k, μ, ν) would be non-feasible.
  - If only the first of these conditions ($k + 1 < n$) is met and any of the others are false, we could still add son σ_0, i.e. it would still be a regular element of $M^{\mathcal{H}}$. We don't add it because it would be pointless: we wouldn't be able to take any successive type 1 edge (see also previous point), which are the only ones increasing entropy.

3.5 BPV's decision graph algorithm: $\mathcal{W}$-formulation

Alternatively to the $\mathcal{H}$-formulation, we can use the **$\mathcal{W}$-formulation**, defined by another set of subproblems, aiming at minimizing rate instead of maximizing entropy:

$$\begin{aligned}
&\forall k \in \{1, \dots, n\}\,, \\
&\forall \varphi \in \mathbb{R},\ 0 \le \varphi \le H_{\text{tot}}\,, \\
&\forall \nu \in \mathbb{N},\ \nu \le N\,, \\
&\mathcal{W}_{k,\varphi,\nu} = \min \left\{ \sum_{i=1}^{k} y_i p_i \text{ st } \forall i\ y_i \in \{0,1\} \text{ and } \sum_{i=1}^{k} y_i p_i \log \frac{1}{p_i} = \varphi,\ \sum_{i=1}^{k} y_i = \nu \right\} .
\end{aligned} \tag{3.5.1}$$

Again, we'll use the notation $\mathcal{W}_{k,\varphi,\nu}$ to denote both the subproblem as a whole and its optimal value. A similar remark to 15 holds here: for all $\varphi \in [0, H_{\text{tot}}]$ except a set of Lebesgue measure 0, $\mathcal{W}_{k,\varphi,\nu}$ will have an empty feasible set.
If we know the solutions to all subproblems (3.5.1) we know once more the solution to the main problem - its optimal value is the largest φ for which there exists a non-empty

subproblem with value $\mathcal{W}_{n,\varphi,\nu} \leq W$ and $\nu \leq N$: a solution to this subproblem is also a solution to (BPV).
Once more we'll have a recursive rule between the values $\mathcal{W}_{k,\varphi,\nu}$:

$$\mathcal{W}_{k,\varphi,\nu} = \begin{cases} \mathcal{W}_{k-1,\varphi,\nu} & \text{if } p_k > \varphi \\ & \text{(i.e. we can't take } k\text{)} \\ \min\left\{\mathcal{W}_{k-1,\varphi,\nu}\,, p_k + \mathcal{W}_{k-1,\varphi-p_k \log \frac{1}{p_k},\nu-1}\right\} & \text{otherwise,} \end{cases} . \tag{3.5.2}$$

We could retrace all the formal definitions leading up to, and including, theorem 3.4.5; the changes we would have to apply however are minor and we will limit ourselves to informally listing them:

- Nodes are (k, φ, ν) with labels $\alpha_{k,\varphi,\nu}$. Each of these nodes represents a non-empty subproblem and its label is, at all times, equal to the best known rate among the feasible solutions to $\mathcal{W}_{k,\varphi,\nu}$.

- Edges of type 0 go from node (k, φ, ν) to $(k+1, \varphi, \nu)$ and have label 0.

- Edges of type 1 go from node (k, φ, ν) to $(k+1, p_{k+1} \log \frac{1}{p_{k+1}} + \varphi, \nu+1)$ and have label p_{k+1}.

- The graph is still a decision graph, in the sense that the correspondence F of theorem 3.4.5 between paths in the graph and binary strings still stands: adding a type 1 edge to a path means adding a pattern to a candidate solution. However, because edge and node labels are exchanged, the value of a path in the graph is now equal to the rate of the corresponding solution rather than its entropy, and thus the analogous to (3.4.4)

  $$\min_{S \in \mathcal{S}_r} W(S) \tag{3.5.3}$$

  is now meaningless for our goals. However, from a practical standpoint, little changes: we can still use algorithm 2 if we make some simple changes:

  – change the definition of the current node and its sons to

  $$\begin{aligned} &(k, \varphi, \nu)\ , \\ \sigma_0 &= (k+1, \varphi, \nu)\ , \\ \text{and } \sigma_1 &= (k+1, p_{k+1} \log \frac{1}{p_{k+1}} + \varphi, \nu+1) \end{aligned}$$

  respectively

  – in line 7 substitute p_{k+1} with $\alpha[k+1]$ and $\alpha[\text{cur}]$ with φ

– as noted in remark 13, change the direction of the inequalities that check if `best_value` needs to be updated; in the pseudocode of algorithm 2 these lines are not shown since they're hidden in the `add_node` function

3.6 Pitfalls of the decision graph algorithm

The decision graph algorithm performs poorly under two aspects:

1. time complexity: the number of nodes in the graph, and thus time complexity, exhibits quasi-exponential dependence on N and W, the $\mathcal{H}$-formulation being slightly better than the $\mathcal{W}$-formulation

2. memory usage: the `predecessor` array is effectively storing in memory one edge for every node of the graph, thus memory usage is proportional to time complexity

A natural idea to try and solve this last problem would be to, every time a new path has been found which raises the value of `best_entropy`, remove from `predecessor` all the paths which have already been explored to the end[10] and have worse than such a value. However potentially some of these paths share edges in common with the new optimal path, or with future optimal paths yet to be found, and thus it's not trivial to what extent such a clean up operation on `predecessor` can be done, let alone in a time effective manner.
Regarding time complexity, we know that it's asymptotically equal to the number of nodes in the graph, since the number of operations for every iteration of algorithm 2 is constant (see also remark 14). In section 4.4 we'll see the results of some numerical trials of the decision graph algorithm, using the pixel pattern distribution as input, which will show us the actual number of nodes in the graph and its dependence on N and W. Here instead we want to try to give an analytical estimate of the number of nodes in the graph: it will however prove difficult to actually compute the bound because of the complex combinatorics required to calculate the cardinality of certain sets.

We start by defining $\mathcal{P}_k = \{1, \dots, k\} \subset \mathcal{P}$ and the **k-level** sets of the graphs as the set of all nodes with first coordinate k, i.e.:

$$\begin{aligned} M_k^{\mathcal{H}} &= \{(k, \mu, \nu) \in M^{\mathcal{H}} \text{ s.t. } \mu \leq W,\ \nu \leq N\} \\ &= \bigcup_{S \subset \mathcal{P}_k,\ W(S) \leq W,\ \kappa(S) \leq N} (k, W(S), \kappa(S)) \end{aligned} \tag{3.6.1}$$

and

$$\begin{aligned} M_k^{\mathcal{W}} &= \{(k, \varphi, \nu) \in M^{\mathcal{W}} \text{ s.t. } \varphi \leq H_{\text{tot}},\ \nu \leq N\} \\ &= \bigcup_{S \subset \mathcal{P}_k,\ \kappa(S) \leq N} (k, H(S), \kappa(S)) \end{aligned} \tag{3.6.2}$$

[10] since we don't add type 0 sons indefinitely most paths get terminated way before the end of the algorithm.

where $M^{\mathcal{H}}$ and $M^{\mathcal{W}}$ are the set of nodes of the decision graph in the $\mathcal{H}$-formulation and in the $\mathcal{W}$-formulation respectively, then the number of nodes in the decision graph of the the two formulation is

$$|M^{\mathcal{H}}| \stackrel{def}{=} \sum_{k=1}^{n} |M_k^{\mathcal{H}}|$$

and

$$|M^{\mathcal{W}}| \stackrel{def}{=} \sum_{k=1}^{n} |M_k^{\mathcal{W}}|$$

respectively. From now on we'll be omitting the superscript when we're referring to both formulations independently.

Remark 18. In the Knapsack problem, since weights and values are natural numbers, the cardinality of the k-level set can be at most W and V, depending on the chosen formulation, where $V = \sum_{i=1}^{n} v_i$. The time complexities then, since the number of total nodes is the sum of the cardinality of the k-sets, are $O(nW)$ and $O(nV)$.

By removing all constraints on candidate solutions $S \subset \mathcal{P}$, we obtain a new decision graph $\mathcal{G} = (\mathcal{M}, \mathcal{E})$,[11] which we'll call the **complete decision graph**, of which our decision graph is a subgraph; we can then try and estimate $|M|$ by estimating the number of nodes we don't visit (because of the conditions in line 7 of algorithm 2) and subtracting it from $|\mathcal{G}|$. In order to formalize this idea, we define functions

$$\begin{aligned} \gamma_k^{\mathcal{H}} : \mathcal{P}_k \to & \{k\} \times [0,1] \times \{1,\dots,n\} \\ \gamma_k^{\mathcal{H}}(S) = & (k, W(S), \kappa(S)) \end{aligned} \tag{3.6.3}$$

and

$$\begin{aligned} \gamma_k^{\mathcal{W}} : \mathcal{P}_k \to & \{k\} \times [0, H_{\text{tot}}] \times \{1,\dots,n\} \\ \gamma_k^{\mathcal{W}}(S) = & (k, H(S), \kappa(S)) \ . \end{aligned} \tag{3.6.4}$$

Then, if we define k-level sets for the complete graphs analogously to (3.6.1) and (3.6.2),

$$\begin{aligned} \mathcal{M}_k^{\mathcal{H}} &= \{(k,\mu,\nu) \in \mathcal{M}^{\mathcal{H}} \text{ s.t. } \mu \le 1,\ \nu \le n\} \\ \mathcal{M}_k^{\mathcal{W}} &= \{(k,\varphi,\nu) \in \mathcal{M}^{\mathcal{W}} \text{ s.t. } \varphi \le H_{\text{tot}},\ \nu \le n\} \ , \end{aligned} \tag{3.6.5}$$

[11] we're really defining two new graphs here, $\mathcal{G}^{\mathcal{H}} = (\mathcal{M}^{\mathcal{H}}, \mathcal{E}^{\mathcal{H}})$ and $\mathcal{G}^{\mathcal{W}} = (\mathcal{M}^{\mathcal{W}}, \mathcal{E}^{\mathcal{W}})$, depending on the chosen formulation

we can view these two sets as the image of their respective γ_k:

$$\mathcal{M}_k^{\mathcal{H}} = Im(\gamma_k^{\mathcal{H}}) = \bigcup_{S \subset \mathcal{P}_k} (k, W(S), \kappa(S))$$
$$\mathcal{M}_k^{\mathcal{W}} = Im(\gamma_k^{\mathcal{W}}) \bigcup_{S \subset \mathcal{P}_k} (k, H(S), \kappa(S)) \ .$$

If the γ_k functions were injective for every k, this would tell us that

$$\forall k = 1, \ldots, n \quad |\mathcal{M}_k| = |2^{\mathcal{P}_k}| = 2^k$$

and therefore

$$|\mathcal{M}| = 2^{n+1} \ .$$

In general we'll have $M_k \subset \mathcal{M}_k \subset 2^{\mathcal{P}_k}$, and thus 2^{n+1} can serve as a first approximation for the number of nodes $|M|$.

We now want to subtract from $2^{\mathcal{P}_k}$ those sets whose image under γ_k is a node in the complete graph, but not in the actual decision graph computed by algorithm 2; for the time being we'll refer only to the $\mathcal{H}$-formulation. To begin, let's look at line 7 of the pseudocode, and analyze the 4 cases when the condition in the `if` fails (see figure 3.4 for a visual representation of how these conditions affect the graph visit):

1. $k \geq n$[12]. This simply means we've reached the end of the graph and don't have any more sons to add to the graph.

2. $\mu + p_{k+1} > W$. We surely can't add the type 1 son because it would violate the bandwidth constraint; the type 0 son instead would still be a valid node, since it's second coordinate would be equal to the one of the current node, i.e. μ. However, as already noted in the discussion of the algorithm, we're supposing the patterns to be ordered such that $(p_k)_{k=1}^n$ is an increasing sequence, exactly because this lets us not add the type 0 son, and thus have fewer nodes in the $k+1$-level. In fact, if we were to add it to the graph, we would only be able to add more type 0 sons to it, without affecting the value of the corresponding solution: any type 1 son to any of its descendants would violate the bandwidth constraint. From another point of view, we are simply pruning, from all paths, the tails consisting only of type 0 edges.

3. $\nu + 1 > N$. Analogously to the previous point, we don't need to add neither son to the path if adding a type 1 edge, now or after a tail of type 0 edges, would violate the cardinality constraint.

[12]Remember that in the pseudocode k goes from 0 to $n-1$, while here we're supposing it goes from 1 to n; "$k+1 < n$" then translates to "$k < n$". We account for this difference only in this border case, while for the other conditions we're supposing k to be a non extreme value.

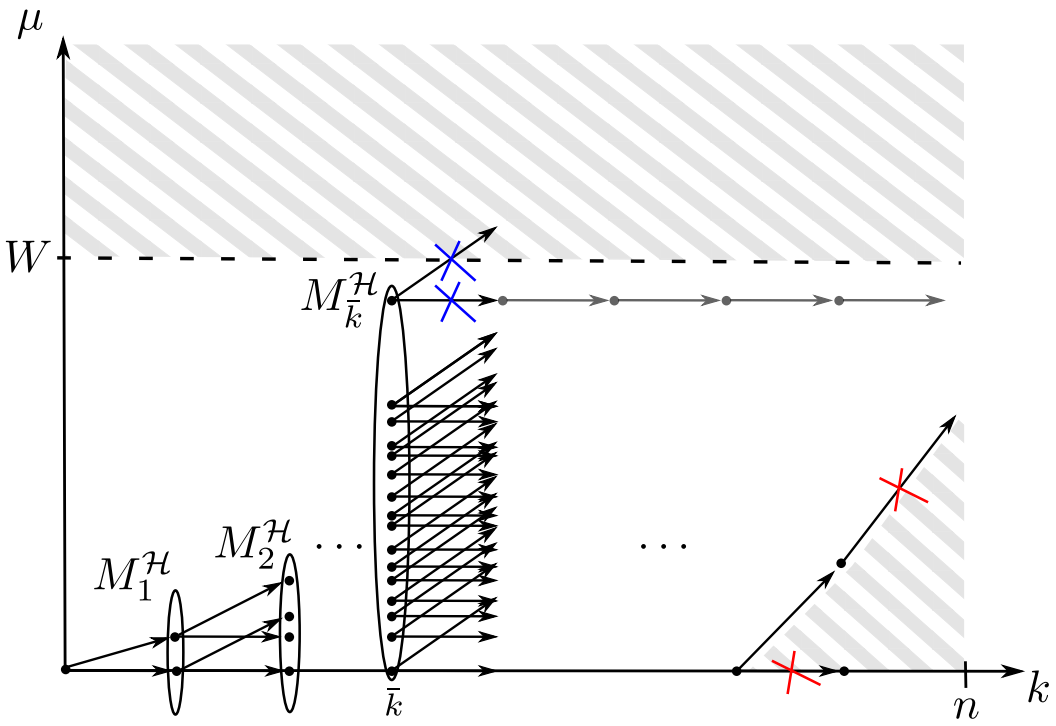


Figure 3.4: Projection on the $\nu = 0$ plane of the decision graph in the $\mathcal{H}$-formulation; for definition of $\bar{k}$ see (3.6.9). The blue-crossed edges are pruned due to violating the bandwidth constraint; pruning also type 0 edges allows to not visit type 0 tails, shown in grey; the cardinality constraint (not shown) operates exactly in the same way. The red-crossed edges are pruned due to the `best_entropy` condition; we don't really know which parts of the graph they exclude, though certainly this type of pruning will be most effective in the final part of the graph visit.

4. $\alpha_{k,\mu,\nu} + \sum_{i=k+1}^{n} p_i \log \frac{1}{p_i} <$ `best_entropy` (remember how we defined $\eta(\cdot)$ in (3.4.8)). This means we've already found a path S with $H(S) =$ `best_entropy`, and even if we took all edges of type 1 after (k, μ, ν) we couldn't do better than that.

While there's no trivial way of to explicitly write how many nodes the algorithm won't visit thanks to the last condition, we can easily use the second and third condition to restrict the domain of $\gamma^{\mathcal{H}}_k$, or, more precisely, $\gamma^{\mathcal{H}}_{k+1}$. In fact, suppose we've just extracted node (k, μ, ν) from `visitlist` (which is at the k-level), then the second and third condition tell us that we won't add certain nodes to `next_visitlist`, which is at the $k+1$-level ; we can then restrict γ_{k+1} so that these nodes are no longer part of its image. Formally, we want to define A_k and B_k as the sets of subsets of $\mathcal{P}_k$ whose corresponding nodes are in the k-level in the graph but not in the successive levels,

because they violate the cardinality constraint (A_k) or the bandwidth one (B_k), i.e.:

$$\begin{aligned} \gamma_k^{\mathcal{H}}(A_k) \subseteq M_k^{\mathcal{H}}, \gamma_k^{\mathcal{H}}(B_k) \subseteq M_k^{\mathcal{H}} \\ \forall j > k \ \gamma_j^{\mathcal{H}}(A_k) \nsubseteq M_j^{\mathcal{H}}, \gamma_j^{\mathcal{H}}(B_k) \nsubseteq M_j^{\mathcal{H}} \ . \end{aligned} \tag{3.6.6}$$

These sets are precisely:

$$\begin{aligned} A_k &= \{S \in 2^{\mathcal{P}_k} \text{ s.t. } \kappa(S) \leq N \ , \kappa(S \cup \{k+1\}) > N\} \\ &= \{S \in 2^{\mathcal{P}_k} \text{ s.t. } \kappa(S) = N\} \end{aligned} \tag{3.6.7}$$

and

$$B_k = \{S \in 2^{\mathcal{P}_k} \text{ s.t. } \sum_{i \in S} p_i \leq W \ , \sum_{i \in S \cup \{k+1\}} p_i > W\} \ . \tag{3.6.8}$$

Remark 19. We have $k < N \Rightarrow A_k = \emptyset$, and similarly if we define

$$\bar{k} = \min \left\{ k \text{ s.t. } \sum_{i=1}^{k} p_i > W \right\} \ , \tag{3.6.9}$$

we have $k < \bar{k} \Rightarrow B_k = \emptyset$.

Now, we wish to define $\delta_k^{\mathcal{H}}$ as a restriction of $\gamma_k^{\mathcal{H}}$, ideally shaving of as much as possible from the domain, but not too much, so as to still have $M_k^{\mathcal{H}} \subseteq Im(\delta_k^{\mathcal{H}})$ and thus being able to bound $|M_k^{\mathcal{H}}|$. Equations (3.6.6) tell us that we can safely take away

$$\bigcup_{j=1}^{k-1} A_j \cup B_j$$

from the domain of $\gamma_k^{\mathcal{H}}$, but since both the A_j and the B_j form a chain for the inclusion relation, i.e.

$$\forall j \ A_j \subseteq A_{j+1} \text{ and } B_j \subseteq B_{j+1} \ ,$$

we can simply define

$$\begin{aligned} \delta_k^{\mathcal{H}} : 2^{\mathcal{P}_k} \setminus (A_{k-1} \cup B_{k-1}) &\longrightarrow \{k\} \times [0, W] \times \{1, \ldots, N\} \\ \delta_k^{\mathcal{H}}(S) = \gamma_k^{\mathcal{H}}(S) &= (k, W(S), \kappa(S)) \ . \end{aligned}$$

Finally, because of $M_k^{\mathcal{H}} \subseteq Im(\delta_k^{\mathcal{H}})$, we can say

$$|M_k| \leq 2^k - |A_{k-1} \cup B_{k-1}| \ . \tag{3.6.10}$$

Remember that this bound is still non-optimal, since we're not accounting for non-injectiveness of $\delta_k^{\mathcal{H}}$, which may cause a reduction of the number of nodes on the k-level

independently of the pruning effects discussed here, and the parts of the graph not visited because of the fourth condition in the above list, i.e. those subgraphs rooted in a node (k, μ, ν) such that $\alpha_{k,\mu,\nu} + \sum_{i=k+1}^{n} p_i \log \frac{1}{p_i} < \texttt{best\_entropy}$. While $|A_k| = \binom{k}{N}$, there's no trivial way of calculating $|B_k|$ or more importantly $|A_k \cup B_k|$.

In section 4.4 we'll see the actual values of $|M_k^{\mathcal{H}}|$ and $|M_k^{\mathcal{W}}|$ in the case of the pixel pattern probability distribution; furthermore we'll see the total number of nodes $|M^{\mathcal{H}}|$ and $|M^{\mathcal{W}}|$ in function of N, W and n.

3.7 Instableness with regards to uniqueness

From remark 12 we know that the solution to the decision graph problem, and thus to (2.3.11), may not be unique: there can be paths with the same optimal value, corresponding to different sets of objects that both satisfy the constraints and have the same entropy. If we define the subset of feasible patterns as

$$\mathcal{F} = \{S \in 2^{\mathcal{P}} \mid W(S) \leq W,\ \kappa(S) \leq N\} \ ,$$

we can define $\Psi : \mathcal{F} \to [0, H_{\text{tot}}]$ by

$$\Psi(S) = H(S) \ , \tag{3.7.1}$$

and view non-uniqueness of the solution to (2.3.11) as a consequence of non-injectivity of Ψ rather than equivalent paths on the decision graph. In fact, if we call H^* the optimal value to (2.3.11), we can affirm that there are multiple optimal solutions iff

$$\exists S^1, S^2 \in \mathcal{F} \text{ s.t. } \Psi(S^1) = \Psi(S^2) = H^* \ , \tag{3.7.2}$$

i.e. iff

$$\exists\ S^1, S^2 \in \mathcal{F} \text{ s.t. } \sum_{i \in S^1} p_i \log \frac{1}{p_i} = \sum_{i \in S^2} p_i \log \frac{1}{p_i} = H^* \ . \tag{3.7.3}$$

This last condition is clearly highly sensitive to any type of perturbation introduced on the probabilities p_i: a change in the entropies of the pattern sets, even very small, might change the entropies of previously optimal sets thus negating the injectivity condition above, or augment the entropy of other non-optimal sets, potentially making them optimal if they weren't by only a small amount. The number of solutions to (2.3.11) would thus seem to depend chaotically on the probabilities p_i, and this type of dependence is inherent in the problem, not in the particular algorithm used to solve it. Any procedure used to enumerate the optimal solutions to (2.3.11) will be highly sensitive to statistical errors arising from the sampling of the $(p_k)_{k=1}^n$ distribution and to numerical errors introduced in the computations involving these probabilities.

3.8 An ϵ-solution

To partially improve time complexity, a natural idea would be to round the floating point coordinates of the nodes in the graph to a discrete grid: this would reduce the number of nodes, since by using a grid with c points the nodes with fixed first coordinate would be at most cN. Simultaneously this quantization process, though obviously resulting in a loss of precision, would solve the instableness problem: sets that have close but different entropies (rates) would get mapped to the same nodes in the $\mathcal{H}$-formulation ($\mathcal{W}$-formulation).

In this section we adapt theorem (11.38) of [11] for the approximation of the Knapsack's problem solution: we prove that by rounding the entropies to a fine enough grid we can obtain a solution committing a relative error on the objective function (entropy of the selected subset $S \subset \mathcal{P}$) as small as we like. If we apply the $\mathcal{W}$-formulation algorithm to this rounded problem we improve time complexity (see also remark 20).
Supposing the probability vector $(p_k)_{k=1}^n$ is ordered increasingly and having defined

$$e_i = p_i \log \frac{1}{p_i}$$
$$e_* = \max_{i=1,\dots,n} e_i \ ,$$

we can subdivide $[0, e_*]$ in a grid of step $\frac{1}{c}$ (the last step will be smaller, unless e_* is a multiple of $\frac{1}{c}$). Having defined $\tilde{e_i} \stackrel{def}{=} \lceil ce_i \rceil$, we'll have

$$\forall i = 1, \dots, n \quad e_i \leq \frac{1}{c}\tilde{e_i} \leq e_i + \frac{1}{c} \ . \tag{3.8.1}$$

We can now prove the following proposition:

Proposition 3.8.2. *For any $\epsilon \in (0, 1]$, let $c = \frac{2N}{\epsilon e_*}$ and let S be an optimal set of indexes for (2.3.11) with values $\frac{1}{c}\tilde{e_i}$ instead of e_i, i.e. S is solution to the rounded problem*

$$max \left\{ \frac{1}{c} \sum_{i=1}^{n} x_i \tilde{e_i} \ st \ \forall i \ x_i \in \{0, 1\} \ and \ \sum_{i=1}^{n} x_i p_i \leq W, \ \sum_{i=1}^{n} x_i \leq N \right\} \ ; \tag{3.8.3}$$

then, if $\bar{S}$ is any other feasible solution to (2.3.11),

$$H(\bar{S}) \leq H(S)(1 + \epsilon) \tag{3.8.4}$$

or equivalently

$$\frac{H(\bar{S}) - H(S)}{H(S)} \leq \epsilon \tag{3.8.5}$$

holds.

Proof. We can suppose $\epsilon^{-1} \in \mathbb{N}$; this guarantees us that $\frac{1}{c}\tilde{e_*} = \frac{1}{c}\lceil ce_* \rceil = e_* \frac{\epsilon}{2N} \lceil \frac{2N}{\epsilon} \rceil = e_*$.

Now, from the first inequality in (3.8.1) and S's optimality for the rounded problem, we have

$$\sum_{i\in\bar{S}} e_i \leq \frac{1}{c}\sum_{i\in\bar{S}} \tilde{e}_i \leq \frac{1}{c}\sum_{i\in S} \tilde{e}_i \ .$$

Observe both problems have the same constraints, so both sets are feasible for both problems. If we now apply the second inequality in (3.8.1), we obtain

$$\sum_{i\in\bar{S}} e_i \leq \frac{1}{c}\sum_{i\in\bar{S}} \tilde{e}_i \leq \frac{1}{c}\sum_{i\in S} \tilde{e}_i \leq \sum_{i\in S}(e_i + \frac{1}{c}) \leq \frac{N}{c} + \sum_{i\in S} e_i \ ,$$

and thus the numerator in inequality (3.8.5) is smaller than $\frac{N}{c}$. From this inequality chain we also obtain a lower bound for the denominator:

$$\sum_{i\in S} e_i \geq \frac{1}{c}\sum_{i\in S} \tilde{e}_i - \frac{N}{c} \ .$$

Since S is optimal for the rounded problem, it will surely be better than the solution containing only one index, thus

$$\frac{1}{c}\sum_{i\in S} \tilde{e}_i \geq \frac{1}{c}\tilde{e}_* = e_* = \epsilon^{-1}\frac{2N}{c}$$

and we can write

$$\sum_{i\in S} e_i \geq \frac{N}{c}(2\epsilon^{-1} - 1) \ .$$

Being that, if $\epsilon \in (0, 1]$, $2\epsilon^{-1} - 1 \geq \epsilon^{-1}$, we finally conclude

$$\frac{\sum_{i\in\bar{S}} e_i - \sum_{i\in S} e_i}{\sum_{i\in S} e_i} \leq \frac{\frac{N}{c}}{\frac{N}{c}\epsilon^{-1}} = \epsilon \ .$$

■

Corollary 3.8.6. *For any $\epsilon \in (0, 1]$, let $c = \frac{2N}{\epsilon e_*}$, let S be a solution to* (3.8.3) *and S^* an optimal solution to* (2.3.11)*; then the relative error on the entropies is smaller than ϵ, i.e.:*

$$\frac{H(S^*) - H(S)}{H(S^*)} \leq \frac{\epsilon}{1+\epsilon} \leq \epsilon \ . \tag{3.8.7}$$

Proof. Since S^* is optimal we have $H(S^*) \geq H(S)$, and thus from (3.8.5) follows immediately that the relative error is smaller than ϵ. To obtain the slightly tighter bound $\frac{\epsilon}{1+\epsilon}$

we can write the following inequalities implied by (3.8.4):

$$H(S^*) \leq H(S)(1+\epsilon) \leq H(S^*)(1+\epsilon)$$

$$-\epsilon H(S^*) \leq (1+\epsilon)(H(S) - H(S^*) \leq 0$$

$$1 \geq \frac{H(S^*) - H(S)}{H(S^*)} \frac{1+\epsilon}{\epsilon} \geq 0 \ .$$

■

If we now apply the $\mathcal{W}$-formulation decision graph algorithm on the rounded problem 3.8.3, we can very easily calculate the time complexity: $|M_k^{\mathcal{W}}|$ can be bounded with cN, and thus time complexity is $O(\sum_{i=1}^{n} |M_k^{\mathcal{W}}|) \leq O(nNc) = O(nN^2\epsilon^{-1})$.

Remark 20. We could round the rates instead of the entropies and, using the $\mathcal{H}$-formulation algorithm, obtain the same asymptotic time complexity, but we wouldn't be able to prove inequality 3.8.7. We could prove an analogous inequality for the relative error on bandwidth usage, but since ultimately we're interested in maximizing the entropy, not minimizing bandwidth, this wouldn't help us.

4 Numerical tests

In this section we want to illustrate and comment the various numerical tests that were done in the course of this study. In the first section we'll describe the python software developed to conduct these tests and create the plots. In section 4.2 we'll briefly comment on the pixel pattern distribution (which we already defined in section 2.5), and in section 4.3 we'll use this probability distribution to see how various properties of the exact and heuristic solution vary with the model parameters N and W. Finally, in section 4.4, using a truncated version of the pixel pattern distribution we'll see how the number of nodes in the decision graph, and thus time complexity of the algorithm we proposed, varies in function of N and W.

4.1 Software used

At `https://github.com/nareto/BPV` there's a git repository with the software developed in python to run the numerical tests of this section. There are two modules, `BPV` and `pattern_manipulation`; the latter leverages the python imaging library (PIL) to digitalize images in a directory tree and calculate the pixel pattern distribution, as long as convenience functions to convert patterns between their two representations as numpy arrays or binary strings. The `BPV` module is more complex and contains two classes, `Data` and `BPV`, used to store problem data and problem instances respectively.
The main features of a `Data()` instance are its `.read_csv` method and `.df` variable; the method reads a CSV file as outputted by `pattern_manipulation` (simply defined by having an `index,probability` line for every pattern) and stores it in the `.df` variable, which is simply a nicely formatted pandas DataFrame, with columns `pattern-string`, `pattern-matrix`, `p` and `plog1onp`.
To make an instance of a `BPV` object, the following arguments must be passed, in this order:

1. `solver`: can be one of those listed in table 4.1
2. `data`: a `Data()` instance, whose DataFrame should contain columns `p` and `plog1onp`
3. N: the maximum allowed cardinality of the solution
4. W: the maximum allowed bandwidth usage of the solution
5. `time_solver` (optional): boolean value indicating whether the solver should be timed; defaults to `False`

Table 4.1: Solvers provided by `BPV` class

solver	description
`pulp`	Utilizes the PuLP framework (using the default CBC solver[1]) to calculate an exact solut
`glpk`	Makes a call to `glpsol` to calculate an exact solution
`heuristic`	Calculates S_{heur}
`decgraphH`	Calculates all exact solutions using the $\mathcal{H}$-formulation decision graph algorithm
`decgraphW`	Calculates all exact solutions using the $\mathcal{W}$-formulation decision graph algorithm

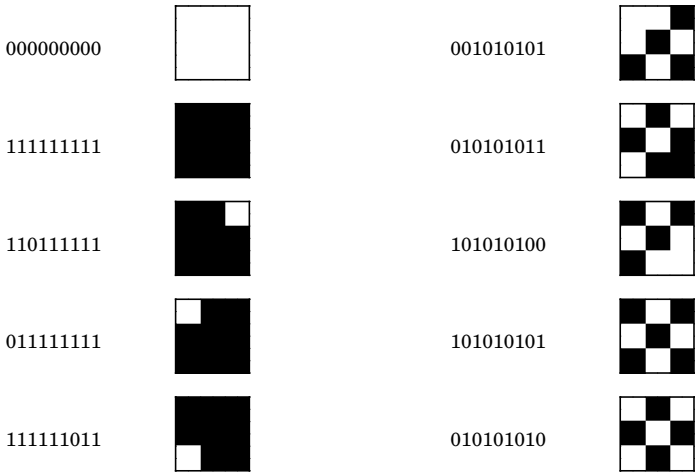
An example of typical usage would be:

```
N = 50
W = 0.025

data = BPV.Data()
data.read_csv("pixel.dist.csv",False)

#optionally sort the DataFrame by increasing (or decreasing)
#pattern probability and reindex accordingly:
#data.df.sort_index(by="p",inplace=True,ascending=True)
#data.df.set_index(pd.Index([j for j in range(len(data.df))]), inplace=True)

#solve with pulp
prbl_pulp = BPV.BPV("pulp",df,N,W,time_solver=False)
prbl_pulp.solve()
prbl_pulp.pprint_solution()

#solve with heuristic
prbl_heur = BPV.BPV("heuristic",df,N,W,time_solver=False)
prbl_heur.solve()
prbl_heur.pprint_solution()

#take the first (depending on how the DataFrame is indexed)
#100 patterns and use the decision graph algorithm to solve the problem
data_head = data.data_head(100)
N = 20
W = 0.03
prbl_decgH = BPV.BPV("decgraphH",df,N,W,time_solver=False)
prbl_decgH.solve()
prbl_decgH.pprint_solution()
```

[15] Coin-or branch and cut - `https://projects.coin-or.org/Cbc`

4.2 Pixel patterns distribution

The pixel pattern distribution was obtained by sampling the 3×3 pixel squares from digitized versions of images in [10], as explained in section 2.5. Here $\mathcal{P} = 2^9$: each pattern is identified by a binary string of length 9, the first bit representing the value of the top-left pixel in the square and the others following the natural top-to-bottom left-to-right order. In figure 4.1 the 5 most and least probable patterns are shown, alongside their corresponding binary string.



(a) The 5 most frequent patterns (b) The 5 least frequent patterns

Figure 4.1: Patterns from the digitilazed images from [10], alongside their corresponding binary string.

The pixel pattern distribution has an interesting symmetry property that can be seen in figure 4.2a: if we order patterns by their binary string (thinking of it as a natural number written in base 2), the values seem to be approximately symmetrical with respect to the middle point 255.5 (we're plotting from 0, the all-white pattern, to 511). Symmetrical patterns with respect to the middle point are exactly **negative patterns**, i.e. the ones with all the bits opposite to one another. In fact, if $b = (b_i)_{i=0}^8 \in \{0,1\}^9$ is a pattern, its negative being defined as $\bar{b} = (1 - b_i)_{i=0}^8$, their corresponding natural numbers β and $\bar{\beta}$

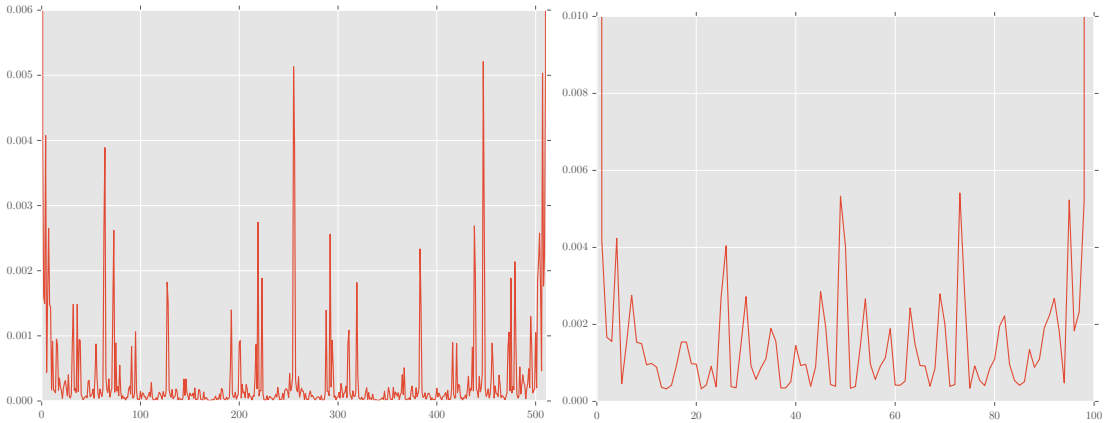
are complementary with respect to 511, i.e.:

$$\begin{aligned}\bar{\beta} &= \sum_{i=0}^{8}(1-b_i)2^i \\ &= \sum_{i=0}^{8}2^i - \sum_{i=0}^{8}b_i 2^i \\ &= 511 - \beta \ ,\end{aligned}$$

and thus $|255.5 - \bar{\beta}| = |255.5 - \beta|$.

This symmetrical property tells us that the pixel pattern probability distribution will potentially give non unique solutions to (2.3.11), since as we've seen having multiple patterns with the same probability leads to non-uniqueness. However, since the probabilities we're using are taken from a statistical sampling, this symmetrical property will always be true only to a certain tolerance, and we've seen that uniqueness of the solution is highly unstable with respect to errors on the probability distribution.

Due to the inefficiency of the decision graph algorithm, to run the tests in section 4.4 we couldn't use the whole set of 2^9 pixel patterns, and thus took the 100 patterns appearing with greatest frequency in the image database and renormalized them; from figure 4.2b we can see that the symmetry property is still present.



(a) Full set of 2^9 patterns (b) 100 most probable patterns renormalized

Figure 4.2: Pixel pattern's probabilities when ordered by their corresponding binary number

4.3 Phenomenological behavior of the solution in function of N and W

Using the full pixel pattern distribution, the `pulp` exact solver[2] and the `heuristic` approximate solver, we calculated both optimal and approximated solutions for 900 couples of the model parameters N and W: they spanned the ranges from 10 to 150 and from 0.001 to 0.25 respectively in 30 equal steps. In this section we want to explore and comment the generated data in the hopes of gaining some insight on our model.
In figure 4.3 we plotted entropies, rates and cardinalities of the exact and heuristic solutions. Looking at the exact rates and entropies we can identify three major regions on these surfaces:

1. For low values of W a quasi-planar region, approximately independent of N
2. For midrange values of W a more curved region, approximately independent of W
3. For large values of W and small values of N there's an abrupt increase in values

The dependency only on one of the two parameters in the first two regions can be explained like this: when one of the two parameters is very small, the corresponding constraint in (2.3.11) becomes predominant in selecting the feasible subsets $S \in \mathcal{P}$.
The discontinuity in the third region is also easily explained, since, as can be seen from table 4.2, the two most frequent patterns appear two magnitude orders more frequently than the third[3]; when W is greater than 0.20469, the second pattern can finally be accepted into a solution and brings its substantially larger contribute to the solution's entropy and especially rate.

[2] integrating with the `glpk` solver, since for some isolated values of N and W the `pulp` solver silently failed to give a solution.

[3] looking back at figure 4.2a its now clear that the y axis had to be limited; if it had been large enough to include the two highest probabilities the plot would have been practically invisible

index	probability	entropy
1	0.661722	0.273231
2	0.204690	0.324691
3	0.004366	0.023726
4	0.004213	0.023044
5	0.003695	0.020693
6	0.003619	0.020343
7	0.003260	0.018668
8	0.003185	0.018310
9	0.003161	0.018196
10	0.003140	0.018099

Table 4.2: 10 largest probabilities and entropies of single patterns for the pixel pattern distribution

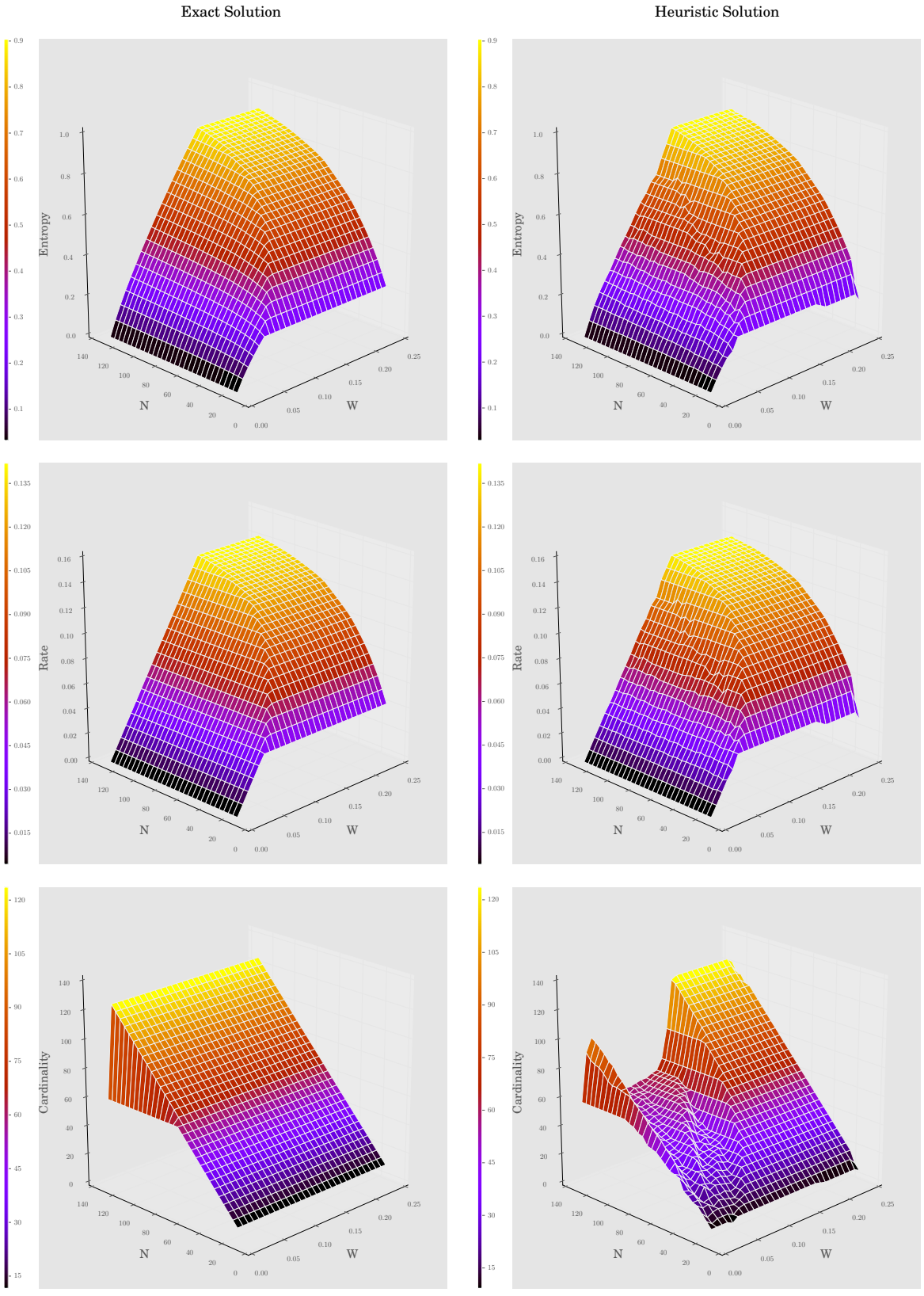


Figure 4.3: Entropies, rates and cardinalities of the exact and heuristic solution in function of the model parameters N and W

It's interesting to see how the entropy and rate of the heuristic solution mimic quite closely those of the exact solution, but the cardinality behaves very differently: the exact solution's cardinality seems to not depend on W and have a linear dependence on N, while the heuristic solution's cardinality has a more complex dependence on the model parameters.
In figure 4.4a we've plotted the relative error of the heuristic solution's entropy with respect to the optimal entropy: for most values of N and W it's below 10%, but again it increases abruptly for large Ws and small Ns (bare in mind that figure 4.4a has the axes oriented opposite of figure 4.3 because of better visual readability of the plot). The different qualitative regions of this plot correspond to the different regions of the optimal entropy surface of figure 4.3: in fact the black lines, which seem to demarcate quite well these different regions, are the contour lines of the norm of the gradient of the optimal entropy surface.

Since one of the initial hypotheses of this study was that, when we reindex the patterns so as to make $(p_i)_{i=1}^n$) a monotone sequence, the optimal set of indexes S would be composed of successive numbers (i.e. it would be an interval like the heuristic solution S_{heur}), we wanted to measure by how much it's not. We therefore define, for any $S \subset \mathbb{N}$, the quantity $\mathcal{I}(S)$ as the ratio of holes in S to the possible number of holes for sets with same cardinality as S, where by hole we mean a point in between the extrema of S but not in S. Some simple trials show that the correct formula is

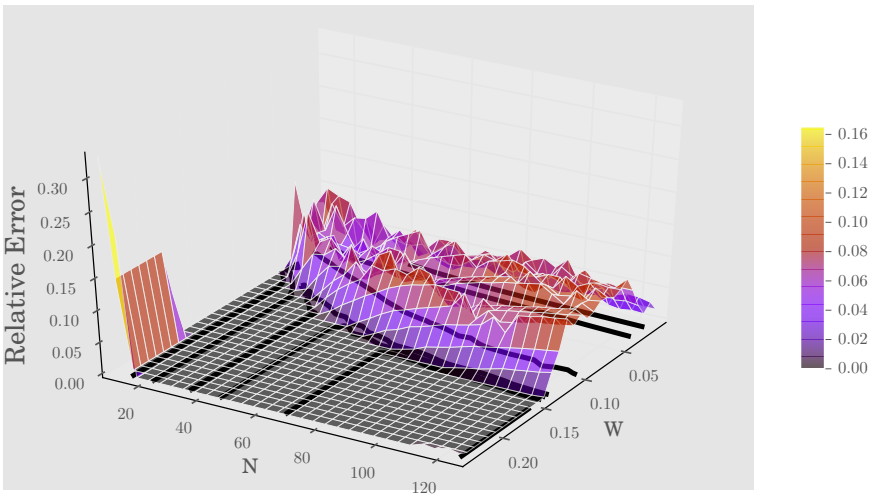
$$\mathcal{I}(S) = \frac{\max S - \min S + 1 - |S|}{\max S - \min S - 1} \ .$$

We can call this the **not-interval measure**: $\mathcal{I}(S)$ is a rational number in $[0, 1]$, equal to 1 for sets consisting of only two points and to 0 for intervals - in fact it's easy to see that

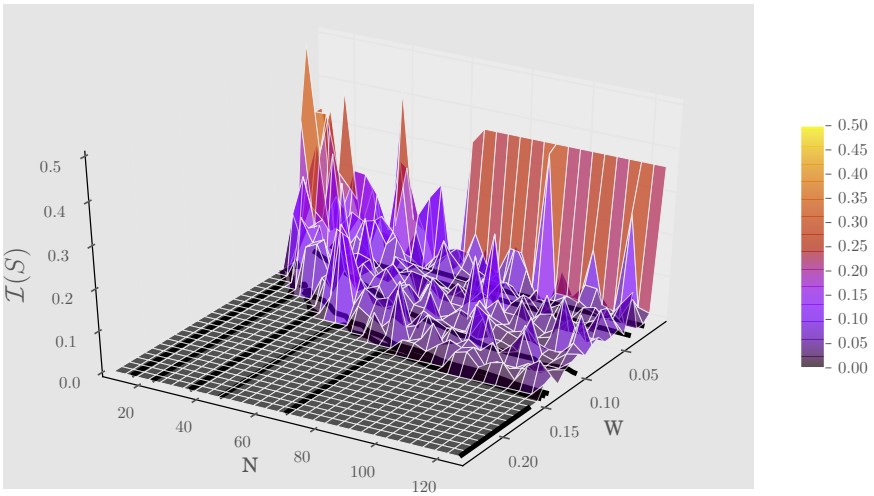
$$\begin{aligned} &\mathcal{I}(S) = 1 \Rightarrow S = \{\min S, \max S\} \\ &\mathcal{I}(S) = 0 \Rightarrow S = \{\min S, \min S + 1, \min S + 2, \ldots, \max S\} \end{aligned}$$

holds. Comparing figure 4.4b to figure 4.4a we can see that, in first approximation, the values of N and W that make the optimal solution be closest to an interval are the same values that guarantee a small relative error of the heuristic solution.

Remark 21. Figure 4.4b gives us a possible insight to write a new algorithm for solving 2.3.11 with greatly improved time complexity: if we are in the flat part of the graph, we can artificially add a new constraint, asking for all solutions to be intervals. This would let us simply consider the first point of $\mathcal{P}$ and add the successive ones until the bandwidth or cardinality constraint is violated, then iteratively take away the first point on the left and add as many patterns possible to the right. This simple procedure would let us find the best interval and has the minimum possible time complexity $O(n)$. Time complexity would thus be given by the sorting algorithm used to initially sort the $(p_k)_{k=1}^n$ vector - for example if we use quicksort that would be $O(n \log n)$ in the average case and



(a) Relative errors



(b) Not-interval measure

Figure 4.4: Black lines are contour lines of the norm of the gradient of the optimal entropy surface of figure 4.3

$O(n^2)$ in the worst case.

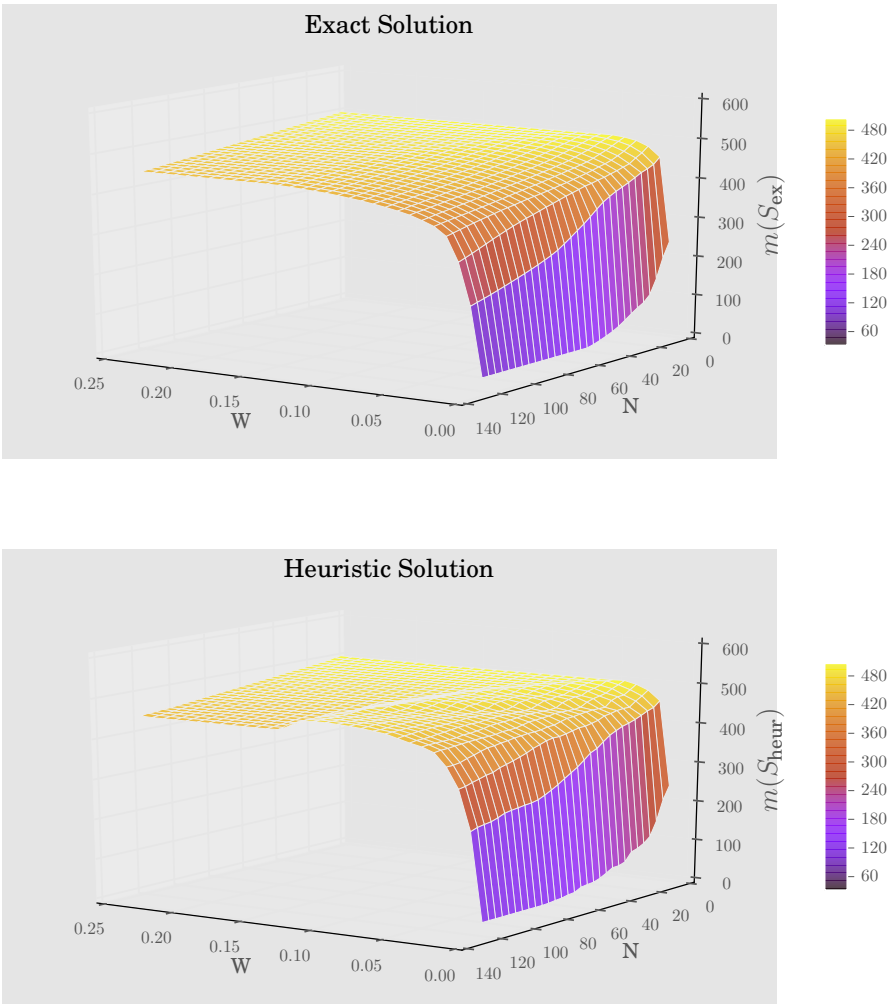


Figure 4.5: Medium index of exact and heuristic solutions

Finally, supposing the patterns to be reindexed so that the probabilities are an increasing sequence, we wanted to understand where the solutions are localized. We thus calculated, for every solution $S = \{s_1, s_2, \dots, s_h\}$, its medium index as

$$m(S) = \frac{1}{h} \sum_{j=1}^{h} s_j \ ,$$

and we've plotted this quantity for both the optimal S_{ex} and the heuristic solution S_{heur} in figure 4.5. We can see that, apart from very low values of W that exclude the most probable patterns (which have higher index because of how we've reordered

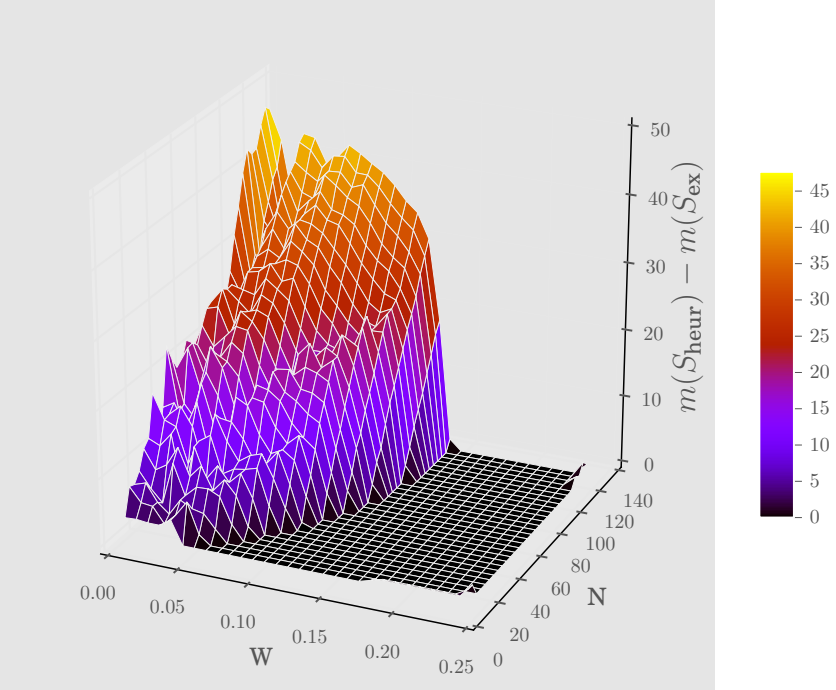


Figure 4.6: Difference in medium indexes

the pattern indexes), the solutions are always approximately concentrated around index 500. Furthermore, in figure 4.6, we've plotted $m(S_{\text{heur}}) - m(S_{\text{ex}})$; this quantity is always positive, which means the heuristic solution is always concentrated to the right of the exact one, i.e. it contains more of the most probable patterns.

4.4 Decision graph algorithm number of nodes

Remember from section 3.6 the definitions of $M_k^{\mathcal{H}}$ and $M_k^{\mathcal{W}}$, i.e. (3.6.1) and (3.6.2) - they are the k-level sets, the sets of nodes in the decision graph (in the respective formulation) with first coordinate k. We wanted to plot actual values of their cardinalities using the pixel pattern distribution, however this was not possible due to the inefficiencies of the algorithm; we therefore truncated the distribution to the 100 most probable patterns, renormalizing the probabilities to 1. Using this somewhat artificial probability distribution, we plotted in figure 4.8 the actual values of $|M_k^{\mathcal{H}}|$ and $|M_k^{\mathcal{W}}|$, for various values of W, alongside the function 2^k, shown in black. Initially there's an almost perfect fit with the exponential curve, due to remark 19: for small values of k the graph is approximately

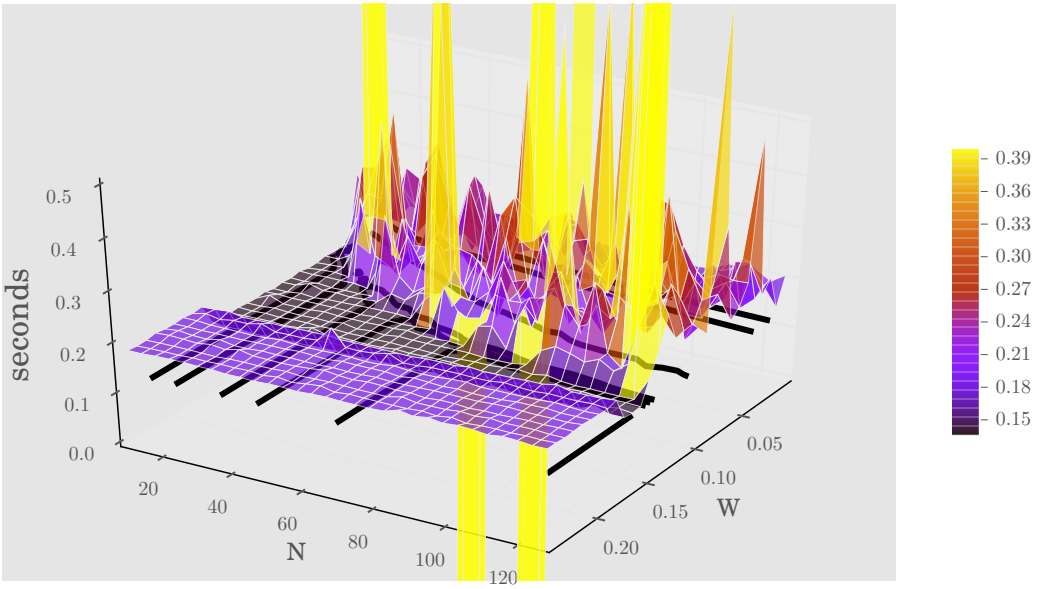


Figure 4.7: Execution times of the default PuLP solver

the complete binary graph - approximately because as always we need to consider non injectivity of function Ψ defined at (3.7.1) or equivalently of functions $\gamma_k^{\mathcal{H}}$ and $\gamma_k^{\mathcal{W}}$ defined at (3.6.3) and (3.6.4), in other words the possibility of having different paths ending in the same node.

In figure 4.9 instead we can see the total number of nodes (in log scale) in the graph in dependency of N and W. It's apparent that complexity has a strong dependency on W, which was to be expected from the analogy with the Knapsack problem, where the formulation correspondent to the $\mathcal{H}$-formulation has exactly $O(nW)$ time complexity. From these plots we see that for the $\mathcal{W}$-solver time complexity is exponential in both W and N, while the $\mathcal{H}$-formulation is slightly better.

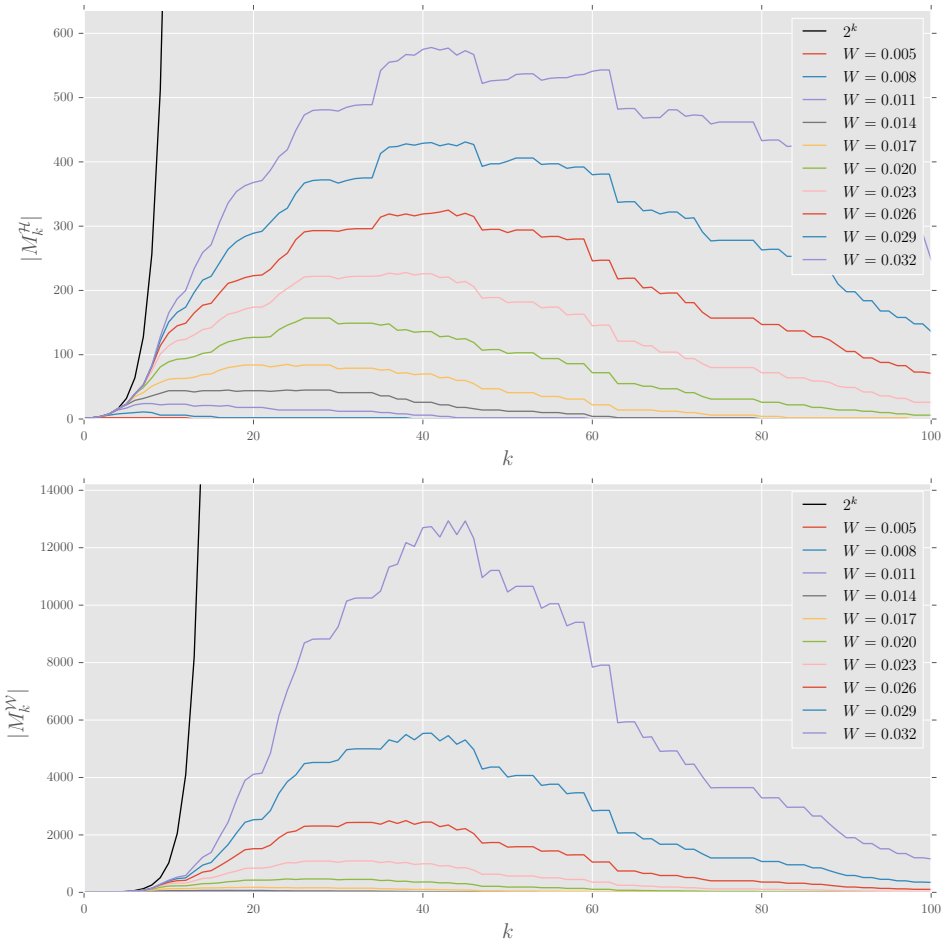


Figure 4.8: Plot of $|M_k^{\mathcal{H}}|$ and $|M_k^{\mathcal{W}}|$ for various values of W and $n = 100$, $N = 15$. The probability vector used is given by the 100 greatest values of the natural image pixel patterns distribution used in [14], renormalized.

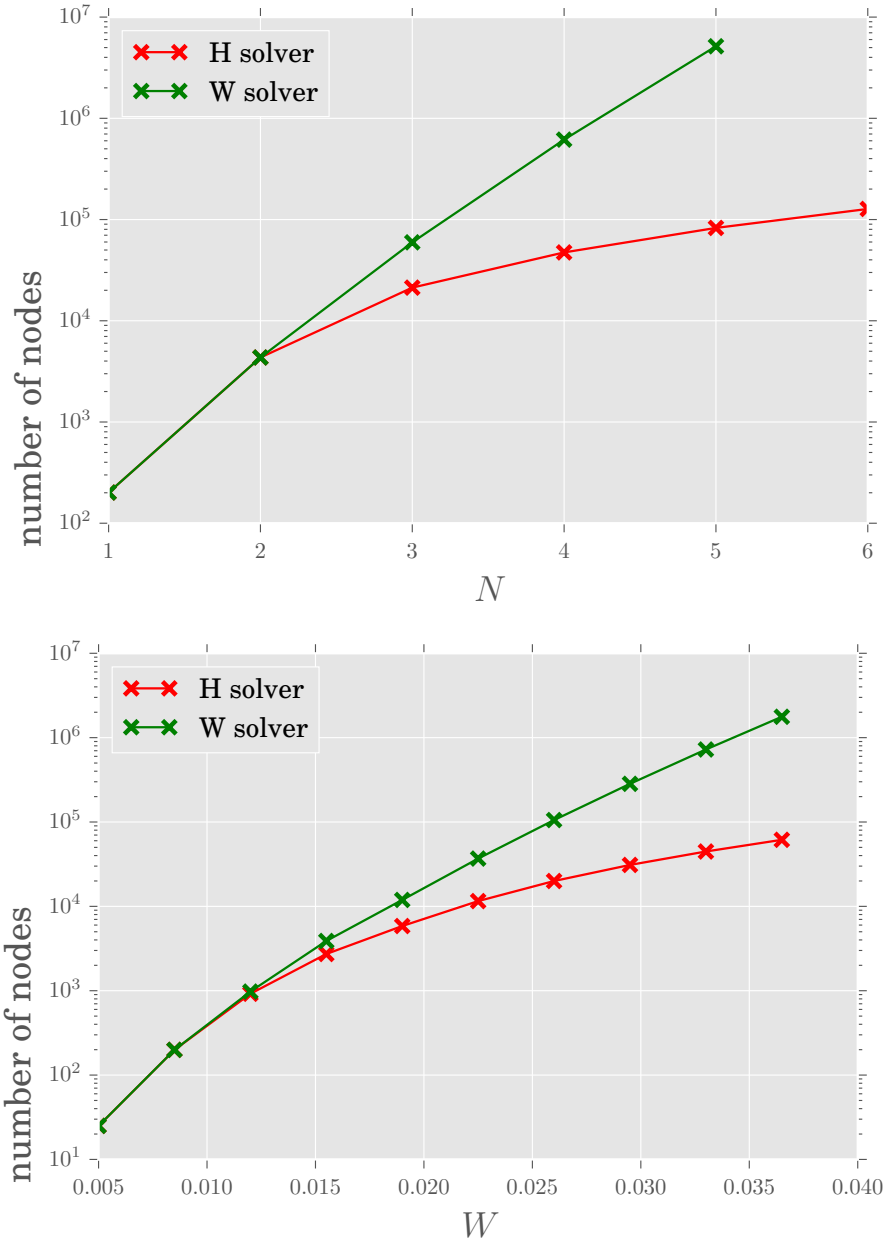


Figure 4.9: Number of nodes, in log scale, in function of N for $W = 0.2$ and in function of W for $N = 15$. The probability vector used is given by the 100 greatest values of the natural image pixel patterns distribution used in [14], renormalized.

Bibliography

[1] Robert B. Ash. *Information Theory.* Courier Dover Publications, 1990.

[2] Joseph Atick. Could information theory provide an ecological theory of sensory processing? *Network: Computation in Neural Systems*, pages 213–251, May 1992.

[3] H. Barlow. Possible principles underlying the transformation of sensory messages. *Sensory Communication*, pages 217–234, 1961.

[4] H. B. Barlow. Summation and inhibition in the frog's retina. *Journal of Physiology*, 119(1):69–88, January 1953.

[5] Thomas M. Cover and Joy A. Thomas. *Elements of information theory.* Wiley, New York, 1991.

[6] C. A. Curcio, K. R. Sloan, R. E. Kalina, and A. E. Hendrickson. Human photoreceptor topography. *The Journal of comparative neurology*, 292(4):497–523, February 1990.

[7] Mauro dell'Orso and Luciano Ristori. Vlsi structures for track finding. *Nuclear Instruments and Methods in Physics Research*, 1989.

[8] Elkin Echeverri. Limits of capacity for the exchange of information in the human nervous system. *IEEE Transactions on Information Technology in Biomedicine*, 10(4), 2006.

[9] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148:574–591, 1959.

[10] Fred Kingdom. McGill Calibrated Colour Image Database. `http://tabby.vision.mcgill.ca/html/welcome.html`.

[11] Jon M. Kleinberg and Éva Tardos. *Algorithm design.* Addison-Wesley, 2006.

[12] S. W. Kuffler. Discharge patterns and functional organization of mammalian retina. *J Neurophysiol*, 16(1):37–68, January 1953.

[13] Stephen E. Palmer. *Vision science: photons to phenomenology.* The MIT Press, 2002.

[14] Maria M. Del Viva, Giovanni Punzi, and Daniele Benedetti. Information and perception of meaningful patterns. *PLoS ONE*, 2013.