Coding and Bounds for Two-Dimensional Constraints

Ido Tal

Coding and Bounds for Two-Dimensional Constraints

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Ido Tal

Submitted to the Senate of
the Technion — Israel Institute of Technology
Tevet 5769 Haifa January 2009

The research thesis was done under the supervision of Prof. Ronny Roth and Prof. Tuvi Etzion in the Computer Science Department.

I would like to sincerely thank my advisers, Prof. Ronny Roth and Prof. Tuvi Etzion, for all of their help and support.

Ronny, thank you for giving me the privilege and pleasure of being your graduate student. Your guidance was truly devoted. The many things you've taught me — and I hope I've learned — are invaluable. Also, thank you for your patience and calm during all the false starts and dead-ends that we'd encountered along the way.

Tuvi, thank you for being my adviser when Ronny was on sabbatical. It was during that time that the three of us achieved the first major result of this thesis, in very large part thanks to the discussions you'd held with me. Apart from the many things I've learned from you, I will always hold dear the fact that you agreed to be my adviser when I was essentially "stuck".

I would like to thank my wife Smadar and son Yoav for their love and encouragement.

The generous financial support of the Technion and the HP corporation is gratefully acknowledged.

Contents

List of Figures

Abstract

Many mass-storage devices have related inputs which, if written on them, have a relatively high probability of being misread. Since this is obviously a problem, one gets around it by allowing only a subset of 'good' inputs to be written on the device. The subset of 'good' inputs is termed a *constrained system*, or a *constraint*.

A one-dimensional (1-D) constraint deals with inputs that are words, and is defined by a graph. The capacity of the constraint is an easily calculated number. Moreover, the state-splitting algorithm gives a very general and practical method for building an encoding/decoding pair for the constraint. If we assume a noiseless case, then the performance of the encoder/decoder pair can be made arbitrarily close to the capacity of the constraint.

A two-dimensional (2-D) constraint deals with allowable two-dimensional arrays, as opposed to the 1-D words. In contrast with 1-D constraints, 2-D constraints are much less understood. There is no known 2-D counterpart of the state-splitting algorithm, and there is an inherent difficulty in calculating the capacity of a general 2-D constraint. Therefore, the known encoding/decoding algorithms are quite specific, and are usually suboptimal. Many algorithms assume noiselessness, and it is not obvious how to make them robust to noise.

In this thesis we present three major results. Our first result is a fixed-rate encoder/decoder pair for a fairly large family of 2-D constraints. Encoding and decoding is done in a row-by-row manner, and is sliding-block decodable.

Essentially, in our first result, the 2-D constraint is turned into a set of independent and relatively simple 1-D constraints; this is done by dividing the array into fixed-width vertical strips. Each row in the strip is seen as a symbol, and a graph presentation of the respective 1-D constraint is con-

structed. The maxentropic stationary Markov chain on this graph is next considered: a perturbed version of the corresponding probability distribution on the edges of the graph is used in order to build an encoder which operates *in parallel* on the strips. This perturbation is found by means of a network flow, with upper and lower bounds on the flow through the edges.

Our second result is a method for bounding the rate of a given bit-stuffing encoder for a 2-D constraint. Instead of considering the original encoder, we consider a related one which is quasi-stationary. We use the quasi-stationary property in order to formulate linear requirements that must hold on the probabilities of the constrained arrays that are generated by the encoder. These requirements are used as part of a linear program. The minimum and maximum of the linear program bound the rate of the encoder from below and from above, respectively.

A lower bound on the rate of an encoder is also a lower bound on the capacity of the corresponding constraint. For some constraints, our second result leads to tighter lower bounds than what was previously known.

Recall that the capacity of 1-D constraints is given by the entropy of a corresponding stationary maxentropic Markov chain. Namely, we maximize the entropy over a finite set of probabilities that must satisfy some requirements. In our third result, we try to extend certain aspects of this characterization to 2-D constraints. The result is a method for calculating an upper bound on the capacity of 2-D constraints.

The key steps are: We look at a maxentropic probability distribution on square arrays, which is stationary. A set of linear equalities and inequalities is derived from this stationarity. The result is a concave program, which can be easily solved numerically.

Abbreviations and Notations

G	—	Edge-labeled directed graph representing a 1-D constraint
$S(G)$	—	1-D constrained system represented by G
$(G_{\mathrm{row}}, G_{\mathrm{col}})$	—	Pair of vertex-labeled directed graph representing a 2-D constraint
$\mathbb{S}(G_{\mathrm{row}}, G_{\mathrm{col}})$	—	2-D constraint system represented by $(G_{\mathrm{row}}, G_{\mathrm{col}})$
cap	—	Capacity
$\log$	—	The base 2 logarithm
U, V	—	Index sets of a configuration
$\mathsf{B}_{M,N}^{(t)}$	—	Parallelogram with M rows, N entries in each row, and slope t
$\mathsf{B}_{M,N}$	—	The index set of an $M \times N$ array
$\mathbb{S}_{M,N}$	—	All $M \times N$ arrays satisfying constraint $\mathbb{S}$
B_M	—	The index set of an $M \times M$ array
$\mathbb{S}_M$	—	All $M \times M$ arrays satisfying constraint $\mathbb{S}$
(i, j)	—	Index of a point in a configuration
$\sigma_{\alpha,\beta}$	—	Shift operator, shifting $(0, 0)$ to (α, β)
$a[\mathsf{U}]$	—	The restriction of a configuration a to the index set U
$\mathbb{S}[\mathsf{U}]$	—	Configurations with index set U that can be extended to configurations in $\mathbb{S}$

Notation specific to Chapter 2

M	—	Number of tracks (columns)
t	—	Stage (row)
k	—	Track (column)
$g_k^{(t)}$	—	Entry written to row t and column k
$\boldsymbol{\gamma}_k$	—	kth track
$\boldsymbol{g}^{(t)}$	—	tth row

m and a	—	Encoder memory and anticipation, respectively
N	—	Number of tracks actually used by the encoder
$G^{\otimes N}$	—	Nth Kronecker power of G
D	—	Multiplicity matrix
Δ	—	The number of typical edges going out of a typical vertex
Q	—	Transition matrix
$\boldsymbol{\pi}$	—	Stationary probability vector
P	—	"Multiplicity matrix," which might have non-integer entries
$\tilde{P}$	—	A good quantization of P

Notation specific to Chapter 3

$\partial_{M,N}$	—	Boundary index set
$\bar{\partial}_{M,N}$	—	Interior index set
Ψ	—	Neighbor set
μ	—	Conditional probability function
$\delta_{M,N}$	—	Border probability function
A	—	Random variable corresponding to the measure defined by the encoder
$A^{(k)}$	—	Quasi-stationary random variable we get from A
Λ	—	Index set of the patch
Γ	—	Index set of the patch's border
$\pi(z)$	—	Probability of border z

Notation specific to Chapter 4

$W^{(M)}$	—	Stationary random variable taking values on $\mathbb{S}_M$
r, s	—	Number of rows and columns in the patch, respectively
Δ	—	Patch index set
$X^{(M)}$	—	Random variable corresponding to the patch
c	—	Number of cases
γ	—	Case value
f	—	A function mapping each (i, j) to one of c cases
(a_γ, b_γ)	—	Index of the point in the patch corresponding to case γ
Ψ_γ	—	All indexes in the patch preceding (a_γ, b_γ)
Υ_γ	—	The index set $\{(a_\gamma, b_\gamma)\} \cup \Psi_\gamma$

Y_γ, Z_γ	—	The random variables defined as $Y_\gamma = X^{(M)}[\Upsilon_\gamma]$, $Z_\gamma = X^{(M)}[\Psi_\gamma]$

Chapter 1

Introduction

In this chapter we introduce and define one-dimensional and two-dimensional constraints, along with basic notions related to them. Moreover, we define the entropy and conditional entropy functions, and state some related theorems. Then, we give a brief overview of the next three chapters.

1.1 1-D constraints

This section contains an overview of known results for one-dimensional (1-D) constraints. Thus, we present and elaborate on certain issues, but do not give a full and rigorous explanation of them. The contents of this section is covered by [31].

1.1.1 Definitions

Let $G = (V, E, L)$ be an edge-labeled (directed) graph, $L : E \to \Sigma$, over an alphabet Σ. We write $e = v \xrightarrow{w} v'$ to mean an edge with initial vertex $\mathsf{s}(e) = v$ and terminal vertex $\mathsf{t}(e) = v'$ labeled $L(e) = w$. Given a path $\gamma = v_1 \xrightarrow{w_1} v_2 \xrightarrow{w_2} \cdots \xrightarrow{w_n} v_{n+1}$ in G, we define the labeling of the path γ as the concatenation of the labels of its edges, namely

$$L(\gamma) = w_1, w_2, \ldots, w_n \ .$$

The constrained system $S = S(G)$ represented by G is the set of labelings of all the paths in G,

$$S(G) = \{L(\gamma) : \gamma \text{ is a path in } G\} \ .$$

In this work, we will use the terms "constrained system" and "constraint" interchangeably.

An edge-labeled graph $G = (V, E, L)$ is termed *deterministic* if there are no two distinct edges in G that emanate from the same vertex and have the same labeling.

We say that a deterministic graph G has *finite memory* if there exists an integer k for which all paths in G of length k that have the same labeling terminate in the same vertex. The smallest such k is termed the memory of G. We say that a constraint S has finite memory if there exists a graph G with finite memory for which $S = S(G)$. The memory of S is the smallest k for which a graph G with memory k satisfying $S = S(G)$ exists.

A graph $G = (V, E, L)$ is *irreducible* if for any pair of vertices $v, v' \in V$, there is a path in G from v to v'. Any graph can be partitioned into irreducible sub-graphs.

The *adjacency matrix* of a graph $G = (V, E, L)$ is a $|V| \times |V|$ matrix indexed by V. For all $v, v' \in V$, entry (v, v') in the adjacency matrix is equal to the number of edges from v to v'. Namely, if $(a_{v,v'})_{v,v' \in V}$ is the adjacency matrix of V, then

$$a_{v,v'} = |\{e \in E \,:\, \mathsf{s}(e) = v \text{ and } \mathsf{t}(e) = v'\}| \ .$$

1.1.2 Examples and motivation

We will now describe two commonly used 1-D constraints.

A commonly used constraint in magnetic and optical recording is the (d, k)-RLL constraint, which requires that there be at most k and at least d, $0 \leq d < k$, consecutive '0's between two successive '1's. RLL is short for run-length-limited — indeed — we are limiting the length of each run of '0's to be between d and k. We can also take $k = \infty$, meaning that there is no upper bound on the length of a run of '0's. For an example of a graph representing the $(2, 5)$-RLL constraint see Figure 1.1. Note that the graph is deterministic, irreducible and has memory equal to 5. We will describe the constraint's merits with respect to magnetic recording, the optical recording case is quite similar.

In magnetic recording, a '1' represents a switch of polarity on the track, while a '0' represents no switch. A (d, k)-RLL constraint is helpful in two aspects.

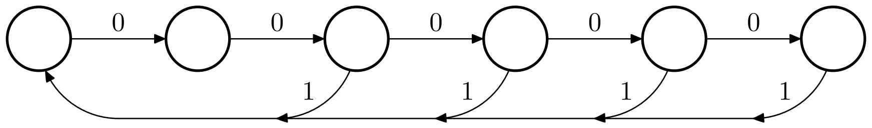


Figure 1.1: Graph representation of the $(2,5)$-RLL constraint.

Timing control: We infer the position of what we are currently reading on the track by means of a clock. The (d,k)-RLL constraint ensures that a '1' (switch of polarity) occurs at least every k time units. This switch of polarity induces a current in the reading head, which is used to synchronize the clock.

Inter-symbol interference: Consider a part of a track magnetized as follows: "North-South-North." When the reading head is above the 'South' section, it will also be affected by the two neighboring 'North' sections. This effect is a negative one, since the two neighbors weaken the magnetic strength of the section in the middle (it is effectively less of a 'South'). Thus, we wish that changes of polarity be somewhat spread apart, and this is achieved by the d parameter of the (d,k)-RLL constraint.

A second constraint commonly used especially in optical recording is the *dc-free* constraint, which is a special case of a spectral null constraint. Our alphabet Σ is now defined as $\Sigma = \{+1, -1\}$. In magnetic recording, $+1$ corresponds to, say, 'North' and -1 to 'South'. A sequence $\mathbf{w} = w_1, w_2, \ldots, w_n$ is in S iff for all $1 \leq i < j \leq n$ we have

$$\left| \sum_{k=i}^{j} w_k \right| \leq B \ ,$$

where the parameter B is a specified constant. For a graph representing the dc-free constraint for $B = 4$ see Figure 1.2. Note that this graph is deterministic, irreducible, and does not have finite memory. The dc-free constraint guarantees a small value at frequency zero in the discrete Fourier transform [32, Chapter 5] of $\mathbf{w} \in S$.

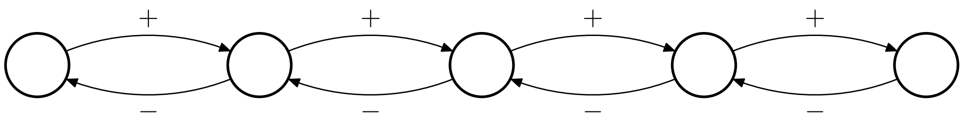


Figure 1.2: Graph representation of the dc-free constraint for $B = 4$.

1.1.3 Capacity of 1-D constraints

The capacity of a 1-D constrained system $S = S(G)$ is defined as

$$\mathsf{cap}(S) = \lim_{n\to\infty} \frac{1}{n} \log |S \cap \Sigma^n| \ ,$$

where from here onward, log is the base 2 logarithm. Note that we are taking the limit of the ratio of information bits ($\log |S \cap \Sigma^n|$) to word length (n). For modest sized graphs, we can easily calculate the capacity of a 1-D constrained system, as we show next.

It turns out [31, Proposition 2.2] that we may assume w.l.o.g. that G is deterministic. So, we henceforth do so. Moreover, it can be shown [31, Theorem 3.12] that at least one of the irreducible sub-graphs of G has the same capacity as G. Thus, we assume from here onward that G is irreducible (and deterministic).

We now show two methods by which the capacity of S can be calculated.

Algebraic characterization of capacity

Let $A = A_G$ be a $|V| \times |V|$ matrix indexed by V, where $A_{v,v'}$ is equal to the number of edges from v to v' in G. Let $\lambda(A)$ be the largest real eigenvalue of A. Except for trivial cases which we henceforth ignore, $\lambda > 1$. We then have [31, Theorem 3.9]

$$\mathsf{cap}(S(G)) = \log \lambda(A_G) \ .$$

Probabilistic characterization of capacity

In this thesis, we will mainly be interested in the probabilistic characterization of capacity, defined next.

A stationary Markov chain $\mathcal{P}$ on G is defined through a row vector $\boldsymbol{\pi} = (\pi_v)_{v \in V}$ and a function $q : E \to [0,1]$. Some conditions must hold. Namely,

1. Each entry in $\boldsymbol{\pi}$ is non-negative, and the entries sum to 1:
$$\sum_{v \in V} \pi_v = 1 \; .$$
Think of π_v as the probability of starting some random path at vertex v.

2. For each $e \in E$, we have that $q(e)$ is non-negative. Moreover, for all $v \in V$, the sum of $q(e)$ over all the edges e emanating from v equals 1:
$$\sum_{e\,:\,\mathsf{s}(e)=v} q(e) = 1 \; .$$
Think of $q(e)$ as the probability of choosing to traverse the edge e, given that we currently at vertex $\mathsf{s}(e)$.

3. Let the *transition matrix* Q associated with $\mathcal{P}$ be defined as follows. The Matrix $Q = (Q_{u,v})_{u,v \in V}$ is a $|V| \times |V|$ matrix indexed by V. For each $u, v \in V$, the value of $Q_{u,v}$ is equal to sum of $q(e)$ on all edges of the form $u \to v$. Namely,
$$Q_{u,v} = \sum_{e\,:\,\mathsf{s}(e)=u, \mathsf{t}(e)=v} q(e) \; .$$
We require that
$$\boldsymbol{\pi} Q = \boldsymbol{\pi} \; .$$
Namely, given that the probability of starting our random path at vertex v is π_v, and that we pick the edges to traverse according to the probability distribution q, the probability of currently being in vertex v is not a function of the number of edges traversed — it is stationary.

The entropy of a stationary Markov chain $\mathcal{P}$ is defined as

$$H(\mathcal{P}) = -\sum_{v \in V} \pi_v \sum_{e\,:\,\mathsf{s}(e)=v} q(e) \log q(e) \; . \tag{1.1}$$

We will have more to say about (1.1) in Section 1.3 below. It turns out that [31, Theorem 3.16]

$$\mathsf{cap}(S) = \max_{\mathcal{P}} H(\mathcal{P}) \ ,$$

where the maximum is over all stationary Markov chains on G. We note that the optimization problem is an instance of convex programming [7], and thus easily solved for graphs of modest size. Moreover, the two characterizations of capacity are in fact dual problems [30, §V].

1.1.4 Finite-state encoders

Let $G = (V, E, L)$ be an edge-labeled graph, and define $G^{\mathsf{q}} = (V, E', L')$ as the qth power of G. Namely, each length q path $\gamma = v_1 \xrightarrow{w_1} v_2 \xrightarrow{w_2} \ldots \xrightarrow{w_{\mathsf{q}}} v_{\mathsf{q}+1}$ in G corresponds to an edge $v_1 \xrightarrow{w_1, w_2, \ldots, w_{\mathsf{q}}} v_{\mathsf{q}+1}$ in G^{q}. Note that the edge labels in G^{q} are over the alphabet Σ^{q}. Define the natural correspondence $\phi_{\mathsf{q}} : (\Sigma^{\mathsf{q}})^* \to (\Sigma)^*$ between sequence over Σ^{q} and sequences over Σ by

$$\phi_{\mathsf{q}}((w_1^1, w_1^2, \ldots, w_1^{\mathsf{q}}), (w_2^1, w_2^2, \ldots, w_2^{\mathsf{q}}), \ldots, (w_n^1, w_n^2, \ldots, w_n^{\mathsf{q}})) = \\ w_1^1, w_1^2, \ldots, w_1^{\mathsf{q}}, w_2^1, w_2^2, \ldots, w_2^{\mathsf{q}}, \ldots, w_n^1, w_n^2, \ldots, w_n^{\mathsf{q}} \ .$$

Note that a word $\mathbf{w} \in (\Sigma^{\mathsf{q}})^*$ is in $S(G^{\mathsf{q}})$ iff $\phi_{\mathsf{q}}(\mathbf{w})$ is in $S(G)$.

A rate $\mathsf{p} : \mathsf{q}$, sliding-block decodable (SBD) encoder for a 1-D constrained system S is a (generally non-deterministic) irreducible labeled graph $\mathcal{E}$ with the following properties.

P1 The out-degree of each state in $\mathcal{E}$ is 2^{p}.

P2 We have $S(\mathcal{E}) \subseteq S^{\mathsf{q}}$, where $S^{\mathsf{q}} = S(G^{\mathsf{q}})$.

P3 The edges emanating from each vertex are numbered distinctly, or *tagged*, from 1 to 2^{p}. Thus, each edge in $\mathcal{E}$ has both a label and a tag.

P4 There are fixed parameters $\mathsf{a} \geq 0, \mathsf{m} \geq 0$ such that the following holds. For all $\mathbf{w} = w_{-\mathsf{m}}, w_{-\mathsf{m}+1}, \ldots, w_0, \ldots, w_{\mathsf{a}} \in \Sigma^{\mathsf{q}}$, if γ_1 and γ_2 are two paths in $\mathcal{E}$ whose labeling is $\mathbf{w}$, then the tag of the $(\mathsf{m}+1)$'st edge of γ_1 is equal to the tag of the $(\mathsf{m}+1)$'st edge in γ_2.

We will now justify the name "$\mathsf{p} : \mathsf{q}$ encoder." Suppose we have a long stream $(b_i)_{i=1}^{n\mathsf{p}}$ of bits we wish to encode, that is, to map into a sequence in S. This is done as follows.

1. Divide the stream into blocks of p bits, and let $1 \leq u_i \leq 2^{\mathsf{p}}$ be such that $(b_{i+1}, b_{i+2}, \ldots, b_{i+\mathsf{p}})$ is the binary representation of $u_i - 1$.

2. We set $i = 1$ and let v_1 be some predetermined 'start' vertex in $\mathcal{E}$. We repeat the following n times:

   (a) Find the edge $v_i \xrightarrow{\mathbf{w}} v'$ tagged with u_i.

   (b) Set $v_{i+1} = v'$.

   (c) Output $\phi_{\mathsf{q}}(\mathbf{w})$.

   (d) $i \leftarrow i + 1$

3. It is known that there exists a constant $M = M(G)$ such that for all n there is a path of length exactly $\ell(n)$ from v_{n+1} to v_1 such that $\ell(n) \leq M$. Note that $\ell(n)$ is upper-bounded by a constant, and is a function of n and v_1, but not of v_{n+1}. Find such a path γ from v_{n+1} to v_1, and output $L(\gamma)$.

Properties P1 – P3 ensure the validity of the encoding algorithm. Property P4 leads to a decoding algorithm: Let $\mathbf{w} \in (\Sigma^{\mathsf{q}})^*$ be such that $\phi_{\mathsf{q}}(\mathbf{w})$ was the output of the encoder, and let $\mathbf{w}' \in (\Sigma^{\mathsf{q}})^*$ be such that $\phi_{\mathsf{q}}(\mathbf{w}')$ was read. By Property P4, the value of u_i, $\mathsf{a} < i < \mathsf{m}$, is uniquely determined by $w_{i-\mathsf{m}}, w_{i-\mathsf{m}+1}, \ldots, w_i, \ldots, w_{i+\mathsf{a}}$. Thus, we only need to look at a window $w'_{i-\mathsf{m}}, w'_{i-\mathsf{m}+1}, \ldots, w_i, \ldots, w'_{i+\mathsf{a}}$ of length $\mathsf{m} + \mathsf{a} + 1$ to decode u'_i. Namely, any errors which may have occurred outside this window will not affect the decoding of the current block. The case $i \leq \mathsf{m}$ ($i \geq \mathsf{a}$) can be handled as well, due to the fact that we know the value, v_1, of the start (end) vertex of the path traveled by the encoder.

We may use the state-splitting algorithm [1] to build a rate $\mathsf{p} : \mathsf{q}$ SBD encoder $\mathcal{E}$ for S, where the ratio p/q is arbitrarily close to $\mathsf{cap}(S)$. However, as we get closer to capacity, our window size, $\mathsf{q}(\mathsf{m} + \mathsf{a} + 1)$, will generally grow.

1.2 2-D constraints

2-D constraints deal with allowable 2-D arrays, as opposed to 1-D constraints which deal with words. The usefulness of 2-D constraints in cases where the recording medium is a surface is evident. However, 2-D constraints are useful in a temporal sense as well. For example [10], consider a 1-D track satisfying the 1-D $(1, \infty)$-RLL constraint. We require that each rewrite of the track not alter two adjacent bits. These restrictions can be modeled as a 2-D constraint. The constraint consists of binary matrices. Each row of an array represents a valid recording of a 1-D track. Consecutive rows represent the *same* track, after every rewrite of the track. Namely, the row index is in fact a time index. The 2-D constraint requires that no two 1's are adjacent horizontally, or diagonally. We note that in this scenario, the decoding must be done according to the contents of the current track. Namely, we only know the contents of the current row in the array, the other rows are unavailable.

This section contains an overview of known results for two-dimensional (2-D) constraints. One should note that the results here are quite specific, and in many cases suboptimal, as opposed to 1-D constraints.

1.2.1 Index sets and configurations

Denote the set of integers by $\mathbb{Z}$. A (2-D) index set $\mathsf{U} \subseteq \mathbb{Z}^2$ is a set of integer pairs. A 2-D configuration over Σ with an index set U is a function $w : \mathsf{U} \to \Sigma$. We denote such a configuration as $w = (w_{i,j})_{(i,j)\in\mathsf{U}}$, where for all $(i, j) \in \mathsf{U}$, we have that $w_{i,j} \in \Sigma$. In this thesis, index sets will always be denoted by upper-case Greek letters or upper-case Roman letters in the sans-serif font. We now define a family of index sets which will turn out to be useful. Denote by $\mathsf{B}^{(t)}_{M,N}$ the set

$$\mathsf{B}^{(t)}_{M,N} = \{(i, j) : 0 \leq i < M \ , \quad 0 \leq t \cdot i + j < N\} \ .$$

Thus, $\mathsf{B}^{(t)}_{M,N}$ is the index set of a parallelogram with M rows, N entries in each row, and slope t. For $t = 0$ this is simply the index set of an $M \times N$ rectangular array, which we abbreviate as

$$\mathsf{B}_{M,N} = \mathsf{B}^{(0)}_{M,N} \ .$$

Also, for M equal to N and $t = 0$ we write

$$\mathsf{B}_M = \mathsf{B}_{M,M} \ .$$

For integers α, β we denote the shifting of U by (α, β) as

$$\sigma_{\alpha,\beta}(\mathsf{U}) = \{(i+\alpha, j+\beta) : (i,j) \in \mathsf{U}\} \ .$$

Also, by abuse of notation, let $\sigma_{\alpha,\beta}(w)$ be the shifted configuration (with index set $\sigma(\mathsf{U})$):

$$\sigma_{\alpha,\beta}(w)_{i+\alpha,j+\beta} = w_{i,j} \ .$$

For a configuration w with index set U, and an index set $\mathsf{V} \subseteq \mathsf{U}$, denote the restriction of w to V by $w[\mathsf{V}] = (w[\mathsf{V}]_{i,j})_{(i,j)\in\mathsf{V}}$; namely,

$$w[\mathsf{V}]_{i,j} = w_{i,j} \ , \quad \text{where} \quad (i,j) \in \mathsf{V} \ .$$

1.2.2 Definitions

A two dimensional (2-D) constrained system over Σ is a generalization of a 1-D constrained system; it is a set $\mathbb{S}$ of rectangular configurations over Σ and is defined through a pair of *vertex*-labeled graphs $(G_{\text{row}}, G_{\text{col}})$, where $G_{\text{row}} = (V, E_{\text{row}}, L)$ and $G_{\text{col}} = (V, E_{\text{col}}, L)$. Namely, both graphs share the same vertex set and the same vertex labeling function $L : V \to \Sigma$. The constraint $\mathbb{S} = \mathbb{S}(G_{\text{row}}, G_{\text{col}})$ consists of all finite rectangular configurations $(w_{i,j})$ over Σ with the following property: Let A be the rectangular index set of $(w_{i,j})_{(i,j)\in\mathsf{A}}$. There exists a configuration $(u_{i,j})_{(i,j)\in\mathsf{A}}$ over the vertex set V such that (a) for each $(i,j) \in \mathsf{A}$ we have $w_{i,j} = L(u_{i,j})$; (b) each row in $(u_{i,j})$ is a path in G_{row}; (c) each column in $(u_{i,j})$ is a path in G_{col}.

For a finite index set U, we abuse notation and denote the restriction of $\mathbb{S}$ to U as

$$\mathbb{S}[\mathsf{U}] = \{w : \text{there exists } w' \in \mathbb{S} \text{ such that } w'[\mathsf{U}] = w\} \ . \qquad (1.2)$$

If $\mathsf{U} = \mathsf{B}_{M,N}$, then we abbreviate

$$\mathbb{S}_{M,N} = \mathbb{S}[\mathsf{B}_{M,N}] \ .$$

Also, for M equal to N we write

$$\mathbb{S}_M = \mathbb{S}[\mathsf{B}_M] \ .$$

Examples

The following lists some common 2-D constraints, defined over the binary alphabet.

- The 2-D counterpart of the 1-D (d, k)-RLL constraint is termed the $(d_r, k_r; d_c, k_c)$-RLL constraint; we now require that each row satisfy the (d_r, k_r)-RLL constraint and each column satisfy the (d_c, k_c)-RLL constraint. It is defined for $0 \leq d_r < k_r \leq \infty$ and $0 \leq d_c < k_c \leq \infty$. When the row parameters are equal to the column parameters, we will simply write "2-D (d, k)-RLL" instead of "$(d, k; d, k)$-RLL". Figure 1.3 contains the two graphs which represent the $(1, \infty; 2, \infty)$-RLL constrained system.

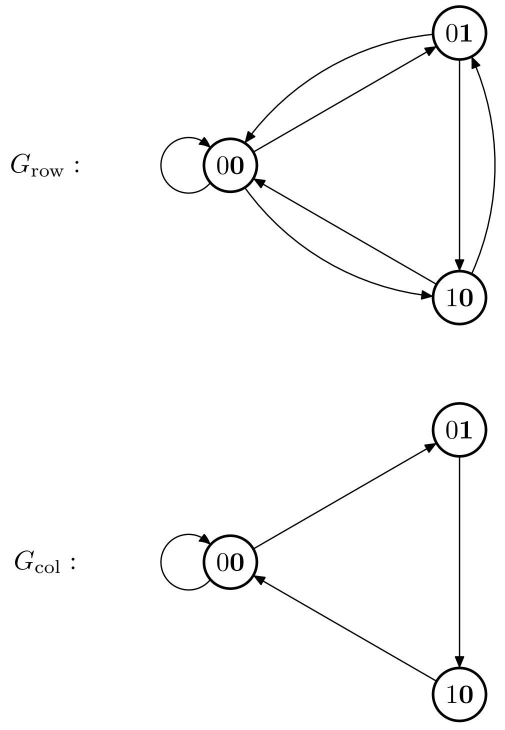


Figure 1.3: Graph representation of the $(1, \infty; 2, \infty)$-RLL constrained system. The label corresponding to vertices 0**0**, 0**1**, and 1**0** is their last (bold) letter; namely 0, 1, and 0, respectively.

- The 2-D $(1, \infty)$-RLL constraint is termed the *hard-square model.* Simply put, we require that a '1' not have a '1' above, below, to the left, or to the right of it.

- The *kings constraint* requires that all neighbors (usually 8, unless we are on the boundary) of each '1' be '0'. If '1' is used to represent a cell with a chess' king on it, then we simply require that there be no attacking kings. This constraint was posed in [46], with the additional requirement that the number of '1's in the array be the largest possible.

- The "*no isolated bits*" constraint, abbreviated as n.i.b., requires that each entry is equal to at least one of the entries immediately to its left, right, top, or bottom (unless we are on the boundary).

- The $(d_{0,r}, k_{0,r}, d_{1,r}, k_{1,r}; d_{0,c}, k_{0,c}, d_{1,c}, k_{1,c})$-SRLL constraint (SRLL is short for *symmetric run-length-limited*) requires that each run of '0's in a row (column) be bounded from below by $d_{0,r}$ ($d_{0,c}$) and from above by $k_{0,r}$ ($k_{0,c}$). Similarly, each run of '1's in a row (column) is bounded from below by $d_{1,r}$ ($d_{1,c}$) and from above by $k_{1,r}$ ($k_{1,c}$). We must have $0 \leq d_{i,s} < k_{i,s} \leq \infty$ for $i = 0, 1$ and $s = r, c$. As before, we abbreviate "$(d, k, d, k; d, k, d, k)$-SRLL" to "2-D (d, k)-SRLL".

1.2.3 Capacity of 2-D constraints

The capacity of a two dimensional constrained system $\mathbb{S}$ is defined as

$$\mathsf{cap}(\mathbb{S}) = \lim_{M,N \to \infty} \frac{1}{M, N} \log |\mathbb{S}_{M,N}| \ .$$

As in the 1-D case, the limit indeed exists (see [28, Appendix], and references therein). Actually, contrary to the 1-D case, it is not obvious when $|\mathbb{S}_{M,N}| = 0$, so we must abuse notation and 'define' $\log 0 = -\infty$. More so, it follows from [6] (but see also [36]), that the following question is undecidable: "Given a pair of vertex-labeled graphs $(G_{\mathrm{row}}, G_{\mathrm{col}})$ with the same vertex set and labeling function, is $\mathsf{cap}(\mathbb{S}(G_{\mathrm{row}}, G_{\mathrm{col}})) \neq -\infty$?". Thus, we have no chance of finding a general method for computing the capacity of a 2-D constrained system.

We next quote some results about capacities of specific constraints.

Theorem 1.1 (Calkin and Wilf [9], Engel [17]) *Let $\mathbb{S}$ be the hard-square model. Then,*

$$0.5878908 \leq \mathsf{cap}(\mathbb{S}) \leq 0.5883392 \ .$$

Theorem 1.1 is rather unique, due to the fact that it is derived through the use of algebraic tools. The lower bound is derived from the ratio of two eigenvalues, while the proof of the upper bound makes use of the trace of a certain matrix.

Both the lower and upper bounds in the method of Calkin and Wilf can be applied to a rather limited family of 2-D constraints. Forchhammer and Justesen extended the method used to derive the upper bound to a larger family of 2-D constraints.

Theorem 1.2 (Forchhammer and Justesen [20]) *The following table (which is taken from [22]) gives upper bounds on the capacities of the 2-D (d, ∞)-RLL constraints, for $d = 2, 3, 4$.*

d	*upper bound on* $\mathsf{cap}(\mathbb{S})$
2	0.4459
3	0.3686
4	0.3188

Kato and Zeger [28] found a necessary and sufficient condition for a $(d, k; d, k)$-RLL constraint to have positive capacity.

Theorem 1.3 (Kato and Zeger [28]) *Let $\mathbb{S}$ be the 2-D (d, k)-RLL constrained system, where $d < k$. Then,*

$$\mathsf{cap}(\mathbb{S}) = 0 \quad \textit{iff} \quad k = d + 1 \ .$$

If $k > d + 1$ then

$$\mathsf{cap}(\mathbb{S}) \geq \max_{2 \leq j \leq 1 + \frac{k-d}{2}} \left\{ \frac{\left\lfloor 1 + \frac{d}{j} \right\rfloor \log(j!) + \log(r!)}{(j + d)^2} \right\} \ . \qquad (1.3)$$

We note that the bound in Equation (1.3) is constructive, that is, it implies an encoding and decoding algorithm. More bounds are derived in [28], which we do not quote here.

Weeks and Blahut [45] showed that by using a numerical method (Richardson extrapolation), one can estimate the capacity of 2-D constraints. Experimentally, this technique seems to work very well. However, at our current state of knowledge, the numbers they compute are essentially guesses.

1.2.4 Encoders for 2-D constraints

Let $\mathcal{M}$ be some given *message set*, and denote by $\mathcal{M}^*$ the set of all finite length sequences over $\mathcal{M}$. In full generality, an encoder $\mathcal{E}$ for a 2-D constraint $\mathbb{S}$ is a one-to-one function from $\mathcal{M}^*$ to $\mathbb{S}$. The corresponding decoder $\mathcal{D}$ is the inverse of $\mathcal{E}$. Together, these two functions constitute an encoder/decoder pair $\mathcal{E}/\mathcal{D}$.

Given an encoder/decoder pair $\mathcal{E}/\mathcal{D}$ for a 2-D constrained system $\mathbb{S}$, we say that $\mathcal{D}$ is a *block decoder*, and that $\mathcal{E}$ is *block decodable*, if the following two conditions hold. (1) The input of $\mathcal{E}$ and the output of $\mathcal{D}$ is a stream of messages from a message set $\mathcal{M}$. (2) Let $\hat{u}_j$ be the jth output of $\mathcal{D}$. Then, $\hat{u}_j$ is a function of a certain row (usually the jth row). That is, it does not depend on the contents of other rows.

We next survey two rather general coding techniques for 2-D constraints.

Bit stuffing [25, 37, 39]: A bit stuffing encoder is defined through a certain probability distribution on the values of the 2-D array. We randomly choose the value of each element of the array by flipping a biased coin, where the coin tosses are in fact the result of applying distribution transformers on the input stream. The bias of the coin is determined by the value of the already-written neighbors of the current element.

Of course, since we are dealing with constrained arrays, some arrays are not permitted. For example, if we are dealing with the hard-square model, and the element we now wish to write to the array has a neighbor to the left of it, or to the top of it, which is '1' we *must* set the current element to '0'. That is, the 'coin' used has a probability of 1 to be '0'. If, however, we are not in such a forced position, then we may set the bias of the coin to whatever value we choose. In [37], it is shown that for the hard-square constraint there exists a choice of such biases leading to a rate which is only 0.1% below the capacity of the constraint. A precise definition of bit stuffing encoders will be given in Section 3.3. For now, we note that the encoding is variable rate, and not block decodable.

Recently, a generalization of bit-stuffing was presented by Sharov and Roth [38]. Namely, the array index set is partitioned into a finite number of sub-sets, each sub-set corresponding to a different "color" (much like a chess-board is partitioned into white and black cells). The coding is done in phases, the number of phases equaling the number of colors. In the phase corresponding to color c, only the array elements with that color are written to. The utility of the method stems from the fact that the coding rule is a function of the color c.

Parallel constrained coding [26]: Fix a 2-D constraint $\mathbb{S}$. Assume that there exists an integer k such that for any two n row arrays $A, C \in \mathbb{S}$ we can find an $n \times k$ array B such that $ABC \in \mathbb{S}$, where juxtaposition denotes horizontal concatenation. For example, the existence of such an array is shown in [18] for a $(d_0, k_0, d_1, k_1; d_0, k_0, d_1, k_1)$-SRLL constraint. If this is the case, then we can prove the existence of an encoder and a block decoder with performance arbitrarily close to the constraint's capacity. Unfortunately, due to a probabilistic argument, the proof of this fact is not constructive, and generally does not imply efficient algorithms.

Enumerative coding with approximate counts [35]: In (standard) enumerative coding [12], we assume that the set we encode into (such as 2-D arrays or 1-D words) can be efficiently enumerated. Moreover, we assume that this property also holds recursively: if, for example, we encode into 1-D words, then we may efficiently enumerate all the 1-D words we encode into with a given prefix. If these two assumptions are met, then there exists an efficient encoder/decoder pair with constant rate approaching the constraint's capacity.

In enumerative coding with approximate counts, these two assumptions are relaxed: we require that the size of the sets be suitably lower-bounded. These ideas were presented by Immink [27] in the context of speeding up coding for 1-D constraints. Ordentlich and Roth [35] used these ideas in the context of 2-D constraints. Contrary to the 1-D case, the derivation of the lower bounds in [35] is much more intricate, and makes use of probabilistic arguments.

We further note that in [33], [34], [41], and [43], encoders and decoders for specific 2-D constraints are presented. These coding techniques seem quite specialized.

1.3 Entropy

The notions of entropy and conditional entropy of random variables will play an important role in Chapters 3 and 4. We now define these concepts and quote some important results regarding them. The contents of this section is covered in [13, Chapter 2].

Let X and Y be two random variables. Denote

$$p_x = \mathrm{Prob}(X = x) \; .$$

and

$$p_{y|x} = \mathrm{Prob}(X = x, Y = y)/\mathrm{Prob}(X = x) \; .$$

The entropy of X is denoted by $H(X)$ and is equal to

$$H(X) = -\sum_x p_x \log p_x \; ,$$

where $0 \log 0$ is defined to be zero. Similarly, we define the conditional entropy $H(Y|X)$ as

$$H(Y|X) = -\sum_x p_x \sum_y p_{y|x} \log p_{y|x} \; .$$

Recall the definition of a stationary Markov chain $\mathcal{P}$ given in Subsection 1.1.3 and our definition of the entropy of $\mathcal{P}$ given in (1.1). Notice that if we were to define the random variable X as the value of the initial vertex of the random path corresponding to $\mathcal{P}$, and the random variable Y as the value of the first edge traversed along that path, then $H(Y|X)$ would be equal to the RHS of (1.1).

We now state some properties of the entropy function, which will be useful later on.

Theorem 1.4 (The chain rule [13, Page 19])

$$H(X, Y) = H(X) + H(Y|X) \; .$$

Theorem 1.5 ([13, Page 19]) *Let the random variable X take values on the set Σ. Then,*

$$H(X) \leq \log_2 |\Sigma| \ .$$

Theorem 1.6 (Conditioning reduces entropy [13, Page 19])

$$H(X|Y) \leq H(X) \, .$$

1.4 Our results

The rest of this thesis consists of three chapters, each containing one of our major results. The chapters are independent, in the sense that one only needs to read this introductory chapter in order to read any of them (e.g., the reader may now jump directly to the last chapter if he/she wishes to do so). We now survey the contents of the following chapters.

Chapter 2

In Chapter 2, a constant-rate encoder–decoder pair is presented for a fairly large family of two-dimensional (2-D) constraints. Encoding and decoding is done in a row-by-row manner, and is sliding-block decodable.

Essentially, the 2-D constraint is turned into a set of independent and relatively simple one-dimensional (1-D) constraints; this is done by dividing the array into fixed-width vertical strips. Each row in the strip is seen as a symbol, and a graph presentation of the respective 1-D constraint is constructed. The maxentropic stationary Markov chain on this graph is next considered: a perturbed version of the corresponding probability distribution on the edges of the graph is used in order to build an encoder which operates *in parallel* on the strips. This perturbation is found by means of a network flow, with upper and lower bounds on the flow through the edges (Figure 2.5).

A key part of the encoder is an enumerative coder for constant-weight binary words. A fast realization of this coder is shown, using floating-point arithmetic.

Chapter 3

In Chapter 3, we present a method for bounding the rate of bit-stuffing encoders. Instead of considering the original encoder, we consider a related one which is quasi-stationary. We use the quasi-stationary property in order to formulate linear requirements that must hold on the probabilities of the constrained arrays that are generated by the encoder. These requirements are used as part of a linear program (Figure 3.5). The minimum and maximum of the linear program bound the rate of the encoder from below and from above, respectively. Numerical results obtained are summarized in Table 3.1.

A lower bound on the rate of an encoder is also a lower bound on the capacity of the corresponding constraint. For some constraints, our results lead to tighter lower bounds than what was previously known. These lower bounds are summarized in Table 3.2.

Chapter 4

Recall from Section 1.1.3 that the capacity of 1-D constraints is given by the entropy of a corresponding stationary maxentropic Markov chain. Namely, the entropy is maximized over a finite set of probabilities that must satisfy some requirements. In Chapter 4, certain aspects of this characterization are extended to 2-D constraints. The result is a method for calculating an upper bound on the capacity of 2-D constraints.

The key steps are: The stationary maxentropic probability distribution on square configurations is considered. A set of linear equalities and inequalities is derived from this stationarity. The result is a concave program (Figure 4.3), which can be easily solved numerically. The upper bounds are summarized in Table 4.1.

Chapter 2

Row-by-Row Coding for 2-D Constraints

In this chapter, a constant-rate encoder–decoder pair is presented for a fairly large family of two-dimensional (2-D) constraints. Encoding and decoding is done in a row-by-row manner, and is sliding-block decodable.

Essentially, the 2-D constraint is turned into a set of independent and relatively simple one-dimensional (1-D) constraints; this is done by dividing the array into fixed-width vertical strips. Each row in the strip is seen as a symbol, and a graph presentation of the respective 1-D constraint is constructed. The maxentropic stationary Markov chain on this graph is next considered: a perturbed version of the corresponding probability distribution on the edges of the graph is used in order to build an encoder which operates *in parallel* on the strips. This perturbation is found by means of a network flow, with upper and lower bounds on the flow through the edges.

A key part of the encoder is an enumerative coder for constant-weight binary words. A fast realization of this coder is shown, using floating-point arithmetic.

2.1 Introduction

Let $G = (V, E, L)$ be an edge-labeled directed graph (referred to hereafter simply as a graph), where V is the vertex set, E is the edge set, and $L : E \to \Sigma$ is the edge labeling taking values on a finite alphabet Σ (as in

Section 2.5). We require that the labeling L is deterministic: edges that start at the same vertex have distinct labels. We further assume that G has finite memory [31, §2.2.3]. Recall from Section 1.1 that the one-dimensional (1-D) *constraint* $S = S(G)$ that is presented by G is defined as the set of all words that are generated by paths in G (i.e., the words are obtained by reading-off the edge labels of such paths). Also, recall that the capacity of S is given by

$$\mathsf{cap}(S) = \lim_{\ell \to \infty} (1/\ell) \cdot \log_2 \left| S \cap \Sigma^{\ell} \right| .$$

An M-track *parallel encoder* for $S = S(G)$ at rate R is defined as follows (see Figure 2.1).

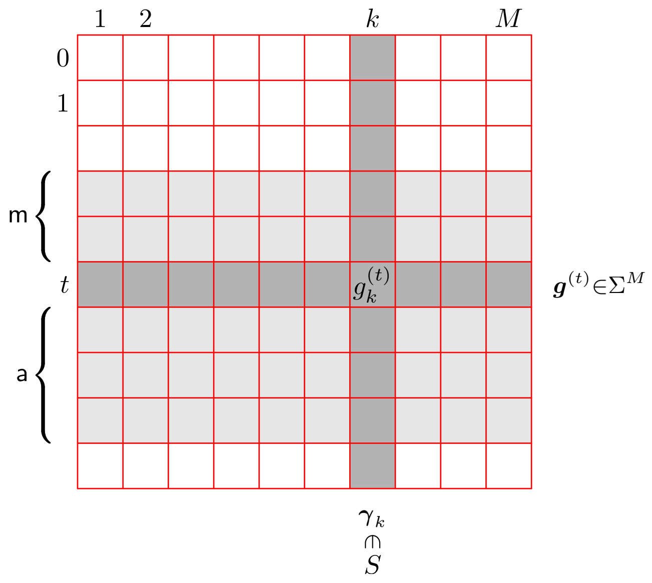


Figure 2.1: Array corresponding to an M-track parallel encoder.

1. At stage $t = 0, 1, 2, \cdots$, the encoder (which may be state-dependent) receives as input $M \cdot R$ (unconstrained) information bits.

2. The output of the encoder at stage t is a word $\boldsymbol{g}^{(t)} = (g_k^{(t)})_{k=1}^{M}$ of length M over Σ.

3. For $1 \le k \le M$, the *kth track* $\boldsymbol{\gamma}_k = (g_k^{(t)})_{t=0}^{\ell-1}$ of any given length ℓ, belongs to S.

4. There are integers $\mathsf{m}, \mathsf{a} \geq 0$ such that the encoder is (m, a)-*sliding-block decodable* (in short, (m, a)-SBD): for $t \geq \mathsf{m}$, the $M \cdot R$ information bits which were input at stage t are uniquely determined by (and can be efficiently calculated from) $\boldsymbol{g}^{(t-\mathsf{m})}, \boldsymbol{g}^{(t-\mathsf{m}+1)}, \ldots, \boldsymbol{g}^{(t+\mathsf{a})}$.

The decoding window size of the encoder is $\mathsf{m} + \mathsf{a} + 1$, and it is desirable to have a small window to avoid error propagation. In this chapter, we will be mainly focusing on the case where $\mathsf{a} = 0$, in which case the decoding requires no look-ahead.

In [26], it was shown that by introducing parallelism, one can reduce the window size, compared to conventional serial encoding. Furthermore, it was shown that as M tends to infinity, there are $(0,0)$-SBD (i.e., block decodable) parallel encoders whose rates approach $\mathsf{cap}(S(G))$. A key step in [26] is using some perturbation of the conditional probability distribution on the edges of G, corresponding to the maxentropic stationary Markov chain on G. However, it is not clear how this perturbation should be done: a naive method will only work for unrealistically large M. Also, the proof in [26] of the $(0,0)$-SBD property is only probabilistic and does not suggest encoders and decoders that have an acceptable running time.

In this chapter, we aim at making the results of [26] more tractable. At the expense of possibly increasing the memory of the encoder (up to the memory of G) we are able to define a suitable perturbed distribution explicitly, and provide an efficient algorithm for computing it. Furthermore, the encoding and decoding can be carried out in time complexity $O(M \log^2 M \log\log M)$, where the multiplying constants in the $O(\cdot)$ term are polynomially large in the parameters of G.

Denote by $\mathrm{diam}(G)$ the diameter of G (i.e., the longest shortest path between any two vertices in G) and let $A_G = (a_{i,j})$ be the adjacency matrix of G, i.e., $a_{i,j}$ is the number of edges in G that start at vertex i and terminate in vertex j. Our main result, specifying the rate of our encoder, is given in the next theorem.

Theorem 2.1 *Let G be a deterministic graph with memory m. For M sufficiently large, one can efficiently construct an M-track $(\mathsf{m}, 0)$-SBD* *parallel encoder for $S = S(G)$ at a rate R such that*

$$R \geq \mathsf{cap}(S(G))\Big(1 - \frac{|V|\,\mathrm{diam}(G)}{2M}\Big) \\ - O\left(\frac{|V|^2 \log\left(M \cdot a_{\max}/a_{\min}\right)}{M - |V|\,\mathrm{diam}(G)/2}\right) , \quad (2.1)$$

where $a_{\min}$ (respectively, $a_{\max}$) is the smallest (respectively, largest) nonzero *entry in A_G.*

The structure of this chapter is as follows. In Section 2.2 we show how parallel encoding can be used to construct an encoder for a 2-D constraint. As we will show, a parallel encoder is essentially defined through what we term a multiplicity matrix. Section 2.3 defines how our parallel encoder works, assuming its multiplicity matrix is given. Then, in Section 2.4, we show how to efficiently calculate a good multiplicity matrix. Although 2-D constraints are our main motivation, Section 2.5 shows how our method can be applied to 1-D constraints. Section 2.6 defines two methods by which the rate of our encoder can be slightly improved. Finally, in Section 2.7 we show a method of efficiently realizing a key part of our encoding procedure.

2.2 Two-dimensional constraints

Our primary motivation for studying parallel encoding is to show an encoding algorithm for a family of two-dimensional (2-D) constraints.

Let $\mathbb{S}$ be a given 2-D constraint over a finite alphabet Σ. Recall the following: we denote by $\mathbb{S}_{\ell,w}$ the set of all $\ell \times w$ arrays in $\mathbb{S}$, and the capacity of $\mathbb{S}$ is given by

$$\mathsf{cap}(\mathbb{S}) = \lim_{\ell,w\to\infty} \frac{1}{\ell \cdot w} \cdot \log_2 |\mathbb{S}_{\ell,w}| \ .$$

Suppose we wish to encode information to an $\ell \times w$ array which must satisfy the constraint $\mathbb{S}$; namely, the array must be an element of $\mathbb{S}_{\ell,w}$. As a concrete example, consider the kings constraint [45]: its elements are all the binary arrays in which no two '1' symbols are adjacent on a row, column, or diagonal.

We first partition our array into two alternating types of vertical strips: *data strips* having width w_{d}, and *merging strips* having width w_{m}. In our example, let $w_{\mathrm{d}} = 4$ and $w_{\mathrm{m}} = 1$ (see Figure 2.2).

$$\begin{array}{cccc|c|cccc|c|cccc}
0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1
\end{array}$$

Figure 2.2: Binary array satisfying the kings constraint, partitioned into data strips of width $w_{\mathrm{d}} = 4$ and merging strips of width $w_{\mathrm{m}} = 1$.

Secondly, we select a graph $G = (V, E, L)$ with a labeling $L : E \to \mathbb{S}_{1,w_{\mathrm{d}}}$ such that $S(G) \subseteq \mathbb{S}$, i.e., each path of length ℓ in G generates a (column) word which is in $\mathbb{S}_{\ell,w_{\mathrm{d}}}$. We then fill up the data strips of our $\ell \times w$ array with $\ell \times w_{\mathrm{d}}$ arrays corresponding to paths of length ℓ in G. Thirdly, we assume that the choice of w_{m} allows us to fill up the merging strips in a row-by-row (causal) manner, such that our $\ell \times w$ array is in $\mathbb{S}$. Any 2-D constraint $\mathbb{S}$ for which such w_{d}, w_{m}, and G can be found, is in the family of constraints we can code for (for example, the 2-D SRLL constraints belong to this family [18]).

Consider again the kings constraint: a graph which produces *all* $\ell \times w_{\mathrm{d}}$ arrays that satisfy this constraint is given in Figure 2.3. Also, for $w_{\mathrm{m}} = 1$, we can take the merging strips to be all-zero. (There are cases, such as the 2-D SRLL constraints, where determining the merging strips may be less trivial [18].)

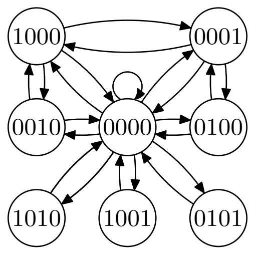


Figure 2.3: Graph G whose paths generate all $\ell \times 4$ arrays satisfying the kings constraint. The label of an edge is given by the label of the vertex it enters.

Suppose we have an $(\mathsf{m}, 0)$-SBD parallel encoder for $S = S(G)$ at rate R

with $M = (w + w_{\mathrm{m}})/(w_{\mathrm{d}} + w_{\mathrm{m}})$ tracks. We may use this parallel encoder to encode information in a row-by-row fashion to our $\ell \times w$ array: at stage t we feed $M \cdot R$ information bits to our parallel encoder. Let $\boldsymbol{g}^{(t)} = (g_k^{(t)})_{k=1}^{M}$ be the output of the parallel encoder at stage t. We write $g_k^{(t)}$ to row t of the kth data strip, and then appropriately fill up row t of the merging strips. Decoding of a row in our array can be carried out based only on the contents of that row and the previous m rows.

Since $M \cdot R$ information bits are mapped to $M \cdot w_{\mathrm{d}} + (M-1) \cdot w_{\mathrm{m}}$ symbols in Σ, the rate at which we encode information to the array is

$$\frac{R}{w_{\mathrm{d}} + w_{\mathrm{m}}(1 - 1/M)} \le \frac{\mathsf{cap}(S(G))}{w_{\mathrm{d}} + w_{\mathrm{m}}(1 - 1/M)} \ .$$

We note the following tradeoff: Typically, taking larger values of w_{d} (while keeping w_{m} constant) will increase the right-hand side of the above inequality. However, the number of vertices and edges in G will usually grow exponentially with w_{d}. Thus, w_{d} is taken to be reasonably small.

Note that in our scheme, a single error generally results in the loss of information stored in the respective vertical sliding-block window. Namely, a single corrupted entry in the array may cause the loss of $\mathsf{m}+1$ rows. Thus, our method is only practical if we assume an error model in which whole rows are corrupted by errors. This is indeed the case if each row is protected by an error-correcting code (for example, by the use of unconstrained positions [15]).

2.3 Description of the encoder

Let N be a positive integer which will shortly be specified. The N words $\boldsymbol{\gamma}_k = (g_k^{(t)})_{t=0}^{\ell-1}$, $1 \le k \le N$, that we will be writing to the first N tracks are all generated by paths of length ℓ in G. In what follows, we find it convenient to regard the $\ell \times N$ arrays $(\boldsymbol{\gamma}_k)_{k=1}^{N} = (g_k^{(t)})_{t=1}^{\ell}{}_{k=1}^{N}$ as (column) words of length ℓ of some new 1-D constraint, which we define next.

The *Nth Kronecker power* of $G = (V, E, L)$, denoted by

$$G^{\otimes N} = (V^N, E^N, L^N) \ ,$$

is defined as follows. The vertex set V^N is simply the Nth Cartesian power of V; that is,

$$V^N = \{\langle v_1, v_2, \ldots, v_N \rangle : v_k \in V\} \ .$$

An edge $\boldsymbol{e} = \langle e_1, e_2, \ldots, e_N \rangle \in E^N$ goes from vertex $\boldsymbol{v} = \langle v_1, v_2, \ldots, v_N \rangle \in V^N$ to vertex $\boldsymbol{v}' = \langle v'_1, v'_2, \ldots, v'_N \rangle \in V^N$ and is labeled $L^N(\boldsymbol{e}) = \langle b_1, b_2, \ldots, b_N \rangle$ whenever for all $1 \leq k \leq N$, e_k is an edge from v_k to v'_k labeled b_k.

Note that a path of length ℓ in $G^{\otimes N}$ is just a handy way to denote N paths of length ℓ in G. Accordingly, the $\ell \times N$ arrays $(\boldsymbol{\gamma}_k)_{k=1}^N$ are the words of length ℓ in $S(G^{\otimes N})$.

Let G be as in Section 2.1 and let $A_G = (a_{i,j})$ be the adjacency matrix of G. Denote by $\mathbf{1}$ the $1 \times |V|$ all-one row vector. The description of our M-track parallel encoder for $S = S(G)$ makes use of the following definition. A $|V| \times |V|$ nonnegative integer matrix $D = (d_{i,j})_{i,j \in V}$ is called a (valid) *multiplicity matrix* with respect to G and M if

$$\mathbf{1} \cdot D \cdot \mathbf{1}^T \leq M \ , \tag{2.2}$$

$$\mathbf{1} \cdot D = \mathbf{1} \cdot D^T \ , \quad \text{and} \tag{2.3}$$

$$d_{i,j} > 0 \text{ only if } a_{i,j} > 0 \ . \tag{2.4}$$

(While any multiplicity matrix will produce a parallel encoder, some will have higher rates than others. In Section 2.4, we show how to compute multiplicity matrices D that yield rates close to $\mathsf{cap}(S(G))$.)

Recall that we have at our disposal M tracks. However, we will effectively be using only the first $N = \mathbf{1} \cdot D \cdot \mathbf{1}^T$ tracks in order to encode information. The last $M - N$ tracks will all be equal to the first track, say.

Write $\boldsymbol{r} = (r_i)_{i \in V} = \mathbf{1} \cdot D^T$. A vertex $\boldsymbol{v} = \langle v_k \rangle_{k=1}^N \in V^N$ is a *typical vertex* (with respect to D) if for all i, the vertex $i \in V$ appears as an entry in $\boldsymbol{v}$ exactly r_i times. Also, an edge $\boldsymbol{e} = \langle e_k \rangle_{k=1}^N \in E^N$ is a *typical edge* with respect to D if for all $i, j \in V$, there are exactly $d_{i,j}$ entries e_k which—as edges in G—start at vertex i and terminate in vertex j.

A simple computation shows that the number of outgoing typical edges from a typical vertex equals

$$\Delta = \frac{\prod_{i \in V} r_i!}{\prod_{i,j \in V} d_{i,j}! \cdot a_{i,j}^{-d_{i,j}}} \tag{2.5}$$

(where $0^0 \triangleq 1$). For example, in the simpler case where G does not contain parallel edges ($a_{i,j} \in \{0, 1\}$), we are in effect counting in (2.5) permutations with repetitions, each time for a different vertex $i \in V$.

The encoding process will be carried out as follows. We start at some fixed typical vertex $\boldsymbol{v}^{(0)} \in V^N$. Out of the set of outgoing edges from $\boldsymbol{v}^{(0)}$, we consider only typical edges. The edge we choose to traverse is determined by the information bits. After traversing the chosen edge, we arrive at vertex $\boldsymbol{v}^{(1)}$. By (2.3), $\boldsymbol{v}^{(1)}$ is also a typical vertex, and the process starts over. This process defines an M-track parallel encoder for $S = S(G)$ at rate

$$R = R(D) = \frac{\lfloor \log_2 \Delta \rfloor}{M} .$$

This encoder is $(\mathsf{m}, 0)$-SBD, where m is the memory of G.

Consider now how we map $M \cdot R$ information bits into an edge choice $\boldsymbol{e} \in E^N$ at any given stage t. Assuming again the simpler case of a graph with no parallel edges, a natural choice would be to use an instance of enumerative coding [12]. Specifically, suppose that for $0 \le \delta \le n$, a procedure for encoding information by an n-bit binary vector with Hamming weight δ were given. Suppose also that $V = \{1, 2, \ldots, |V|\}$. We could use this procedure as follows. First, for $n = r_1$ and $\delta = d_{1,1}$, the binary word given as output by the procedure will define which $d_{1,1}$ of the possible r_1 entries in $\boldsymbol{e}$ will be equal to the edge in E from the vertex $1 \in V$ to itself (if no such edge exists, then $d_{1,1} = 0$). Having chosen these entries, we run the procedure with $n = r_1 - d_{1,1}$ and $\delta = d_{1,2}$ to choose from the remaining $r_1 - d_{1,1}$ entries those that will contain the edge in E from $1 \in V$ to $2 \in V$. We continue this process, until all r_1 entries in $\boldsymbol{e}$ containing an edge outgoing from $1 \in V$ have been picked. Next, we run the procedure with $n = r_2$ and $\delta = d_{2,1}$, and so forth. The more general case of a graph containing parallel edges will include a preliminary step: encoding information in the choice of the $d_{i,j}$ edges used to traverse from i to j ($a_{i,j}$ options for each such edge).

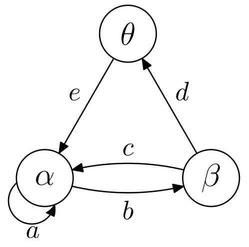
A fast implementation of enumerative coding is presented in Section 2.7. The above-mentioned preliminary step makes use of the Schönhage–Strassen integer-multiplication algorithm [2, §7.5], and the resulting encoding time complexity is proportional[1] to $M \log^2 M \log\log M$. It turns out that this is also the decoding time complexity. Further details are given in Section 2.7.

[1]Actually, the time complexity for the preliminary step can be made linear in M, with a negligible penalty in terms of rate: Fix i and j, and let η be an integer design parameter. Assume for simplicity that $\eta | d_{i,j}$. The number of vectors of length η over an alphabet of size $a_{i,j}$ is obviously $a_{i,j}^{\eta}$. So, we can encode $\lfloor \eta \log_2 a_{i,j} \rfloor$ bits through the choice of such a vector. Repeating this process, we can encode $(d_{i,j}/\eta) \cdot \lfloor \eta \log_2 a_{i,j} \rfloor$ bits through the choice of $d_{i,j}/\eta$ such vectors. The concatenation of these vectors is taken to represent our choice of edges. Note that the encoding process is linear in M for constant η. Also, our losses (due to the floor function) become negligible for modestly large η.

The next section shows how to find a good multiplicity matrix, i.e., a matrix D such that $R(D)$ is close to $\mathsf{cap}(S(G))$.

2.4 Computing a good multiplicity matrix

In order to enhance the exposition of this section, we accompany it by a running example (see Figure 2.4).



$$A_G = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Figure 2.4: **Running Example (1):** Graph G and the corresponding adjacency matrix A_G.

Throughout this section, we assume a probability distribution on the edges of G, which is the maxentropic stationary Markov chain $\mathcal{P}$ on G [31]. Without real loss of generality, we can assume that G is irreducible (i.e., strongly-connected), in which case $\mathcal{P}$ is indeed unique. Let the matrix $Q = (q_{i,j})$ be the transition matrix induced by $\mathcal{P}$, i.e., $q_{i,j}$ is the probability of traversing an edge from $i \in V$ to $j \in V$, conditioned on currently being at vertex $i \in V$.

Let $\boldsymbol{\pi} = (\pi_i)$ be the $1 \times |V|$ row vector corresponding to the stationary distribution on V induced by Q; namely, $\boldsymbol{\pi} Q = \boldsymbol{\pi}$ and $\sum_{i \in V} \pi_i = 1$. Let

$$M' = M - \lfloor |V| \operatorname{diam}(G)/2 \rfloor \ , \tag{2.6}$$

and define

$$\boldsymbol{\rho} = (\rho_i) \ , \ \rho_i = M' \pi_i \ , \quad \text{and} \quad P = (p_{i,j}) \ , \ p_{i,j} = \rho_i q_{i,j}$$

Running Example (2): Taking the number of tracks in our running example (Figure 2.4) to be $M = 12$ gives $M' = 9$. Also, our running example has

$$\boldsymbol{\pi} = \begin{pmatrix} 0.619 & 0.282 & 0.099 \end{pmatrix} \ ,$$

and

$$Q = \begin{pmatrix} 0.544 & 0.456 & 0 \\ 0.647 & 0 & 0.353 \\ 1 & 0 & 0 \end{pmatrix} .$$

Thus,

$$\boldsymbol{\rho} = \begin{pmatrix} 5.57 & 2.54 & 0.89 \end{pmatrix}$$

and

$$P = \begin{pmatrix} 3.03 & 2.54 & 0 \\ 1.65 & 0 & 0.89 \\ 0.89 & 0 & 0 \end{pmatrix} .$$

■

Note that

$$\boldsymbol{\rho} = \mathbf{1} \cdot P^T \quad \text{and} \quad M' = \mathbf{1} \cdot P \cdot \mathbf{1}^T \ .$$

Also, observe that (2.2)–(2.4) hold when we substitute P for D. Thus, if all entries of P were integers, then we could take D equal to P. In a way, that would be the best choice we could have made: by using Stirling's approximation, we could deduce that $R(D)$ approaches $\mathsf{cap}(S(G))$ as $M \to \infty$. However, the entries of P, as well as $\boldsymbol{\rho}$, may be non-integers.

We say that an *integer* matrix $\tilde{P} = (\tilde{p}_{i,j})$ is a *good quantization* of $P = (p_{i,j})$ if

$$\begin{aligned} M' = \textstyle\sum_{i,j \in V} p_{i,j} &= \textstyle\sum_{i,j \in V} \tilde{p}_{i,j} \ , && (2.7) \\ \left\lfloor \textstyle\sum_{j \in V} p_{i,j} \right\rfloor &\le \textstyle\sum_{j \in V} \tilde{p}_{i,j} \le \left\lceil \textstyle\sum_{j \in V} p_{i,j} \right\rceil \ , && (2.8) \\ \lfloor p_{i,j} \rfloor &\le \quad \tilde{p}_{i,j} \quad \le \lceil p_{i,j} \rceil \ , \quad \text{and—} && (2.9) \\ \left\lfloor \textstyle\sum_{i \in V} p_{i,j} \right\rfloor &\le \textstyle\sum_{i \in V} \tilde{p}_{i,j} \le \left\lceil \textstyle\sum_{i \in V} p_{i,j} \right\rceil \ . && (2.10) \end{aligned}$$

Namely, a given entry in $\tilde{P}$ is either the floor or the ceiling of the corresponding entry in P, and this also holds for the sum of entries of a given row or column in $\tilde{P}$; moreover, the sum of entries in both $\tilde{P}$ and P are exactly equal (to M').

Lemma 2.2 *There exists a matrix $\tilde{P}$ which is a good quantization of P. Furthermore, such a matrix can be found by an efficient algorithm.*

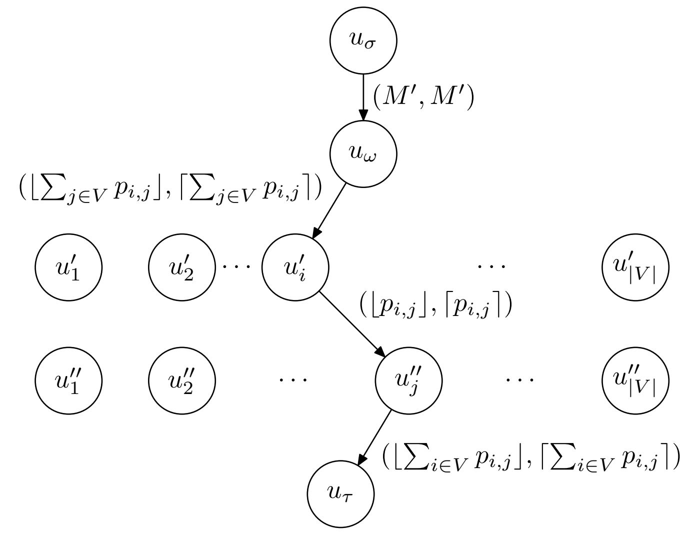

u_σ
(M', M')
u_ω
$(\lfloor\sum_{j\in V} p_{i,j}\rfloor, \lceil\sum_{j\in V} p_{i,j}\rceil)$
u'_1
u'_2
u'_i
$u'_{|V|}$
$(\lfloor p_{i,j}\rfloor, \lceil p_{i,j}\rceil)$
u''_1
u''_2
u''_j
$u''_{|V|}$
$(\lfloor\sum_{i\in V} p_{i,j}\rfloor, \lceil\sum_{i\in V} p_{i,j}\rceil)$
u_τ

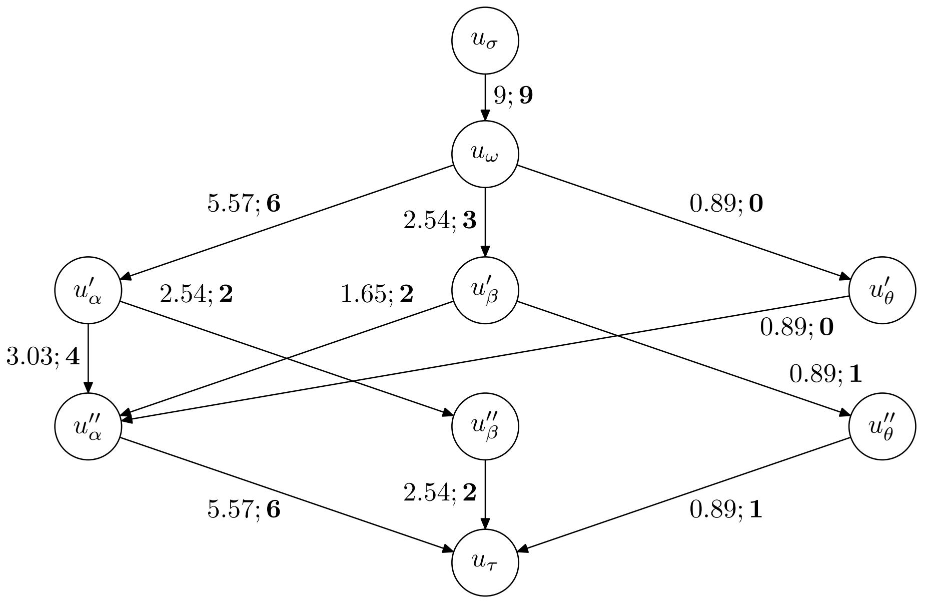
Figure 2.5: Flow network for the proof of Lemma 2.2. An edge labeled (a, b) has lower and upper bounds a and b, respectively.

Proof. We recast (2.7)–(2.10) as an integer flow problem (see Figures 2.5 and 2.6). Consider the following flow network, with upper and lower bounds on the flow through the edges [3, §6.7]. The network has the vertex set

$$\{u_\sigma\} \cup \{u_\omega\} \cup \{u_\tau\} \cup \{u'_i\}_{i\in V} \cup \{u''_j\}_{j\in V} ,$$

with source u_σ and target u_τ. Henceforth, when we refer to the upper (lower) bound of an edge, we mean the upper (lower) bound on the flow through it. There are four kinds of edges:

1. An edge $u_\sigma \to u_\omega$ with upper and lower bounds both equaling to M'.
2. $u_\omega \to u'_i$ for every $i \in V$, with the upper and lower bounds $\lfloor\sum_{j\in V} p_{i,j}\rfloor$ and $\lceil\sum_{j\in V} p_{i,j}\rceil$, respectively.
3. $u'_i \to u''_j$ for every $i, j \in V$, with the upper and lower bounds $\lfloor p_{i,j}\rfloor$ and $\lceil p_{i,j}\rceil$, respectively.
4. $u''_j \to u_\tau$ for every $j \in V$, with the upper and lower bounds $\lfloor\sum_{i\in V} p_{i,j}\rfloor$ and $\lceil\sum_{i\in V} p_{i,j}\rceil$, respectively.



$$P = \begin{pmatrix} 3.03 & 2.54 & 0 \\ 1.65 & 0 & 0.89 \\ 0.89 & 0 & 0 \end{pmatrix}, \qquad \tilde{P} = \begin{pmatrix} 4 & 2 & 0 \\ 2 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}.$$

Figure 2.6: **Running Example (3):** Flow network derived from P in Running Example 2. An edge labeled $a;\mathbf{b}$ has lower and upper bounds $\lfloor a \rfloor$ and $\lceil a \rceil$, respectively. A legal real flow is given by a. A legal integer flow is given by $\mathbf{b}$. The matrix $\tilde{P}$ resulting from the legal integer flow is given, as well as the matrix P (again).

We claim that (2.7)–(2.10) can be satisfied if a legal integer flow exists: simply take $\tilde{p}_{i,j}$ as the flow on the edge from u'_i to u''_j.

It is well known that if a legal *real* flow exists for a flow network with integer upper and lower bounds on the edges, then a legal *integer* flow exists as well [3, Theorem 6.5]. Moreover, such a flow can be efficiently found [3, §6.7]. To finish the proof, we now exhibit such a legal real flow:

1. The flow on the edge $u_\sigma \to u_\omega$ is $\sum_{i,j \in V} p_{i,j} = M'$.

2. The flow on an edge $u_\omega \to u'_i$ is $\sum_{j \in V} p_{i,j}$.

3. The flow on an edge $u'_i \to u''_j$ is $p_{i,j}$.
4. The flow on an edge $u''_j \to u_\tau$ is $\sum_{i\in V} p_{i,j}$.

■

For the remaining part of this section, we assume that $\tilde{P}$ is a good quantization of P (say, $\tilde{P}$ is computed by solving the integer flow problem in the last proof). The next lemma states that $\tilde{P}$ "almost" satisfies (2.3).

Lemma 2.3 *Let $\tilde{\boldsymbol{\rho}} = (\tilde{\rho}_i) = \mathbf{1}\cdot\tilde{P}^T$ and $\tilde{\boldsymbol{r}} = (\tilde{r}_i) = \mathbf{1}\cdot\tilde{P}$. Then, for all $i \in V$,*

$$\tilde{\rho}_i - \tilde{r}_i \in \{-1, 0, 1\} \ .$$

Proof. From (2.8), we get that for all $i \in V$,

$$\lfloor \textstyle\sum_{j\in V} p_{i,j} \rfloor \le \tilde{\rho}_i \le \lceil \textstyle\sum_{j\in V} p_{i,j} \rceil \ . \tag{2.11}$$

Recall that (2.3) is satisfied if we replace D by P. Thus, by (2.10), we have that (2.11) also holds if we replace $\tilde{\rho}_i$ by $\tilde{r}_i$. We conclude that $|\tilde{\rho}_i - \tilde{r}_i| \le 1$. The proof follows from the fact that entries of $\tilde{P}$ are integers, and thus so are those of $\tilde{\boldsymbol{\rho}}$ and $\tilde{\boldsymbol{r}}$. ■

The following lemma will be the basis for augmenting $\tilde{P}$ so that (2.3) is satisfied.

Lemma 2.4 *Fix two distinct vertices $s, t \in V$. We can efficiently find a $|V| \times |V|$ matrix $F^{(s,t)} = F = (f_{i,j})_{i,j\in V}$ with non-negative integer entries, such that the following three conditions hold.*

(i)

$$\mathbf{1}\cdot F\cdot \mathbf{1}^T \le \operatorname{diam}(G) \ .$$

(ii) For all $i, j \in V$,

$$f_{i,j} > 0 \ \textit{ only if } \ a_{i,j} > 0 \ .$$

(iii) Denote $\boldsymbol{\xi} = \mathbf{1}\cdot F^T$ and $\boldsymbol{x} = \mathbf{1}\cdot F$. Then, for all $i \in V$,

$$x_i - \xi_i = \begin{cases} -1 & \textit{if } i = s, \\ 1 & \textit{if } i = t, \\ 0 & \textit{otherwise.} \end{cases}$$

Proof. Let $k_1 = s, k_2, k_3 \ldots, k_{\ell+1} = t$ be the vertices along a shortest path from s to t in G. For all $i, j \in V$, define

$$f_{i,j} = |\{1 \le h \le \ell : k_h = i \text{ and } k_{h+1} = j\}| \ . \tag{2.12}$$

Namely, $f_{i,j}$ is the number of edges from i to j along the path.

Conditions (i) and (ii) easily follow from (2.12). Condition (iii) follows from the fact that ξ_i (x_i) is equal to the number of edges along the path for which i is the start (end) vertex of the edge. ■

The matrix $\tilde{P}$ will be the basis for computing a good multiplicity matrix D, as we demonstrate in the proof of the next theorem.

Theorem 2.5 *Let $\tilde{P} = (\tilde{p}_{i,j})$ be a good quantization of P. There exists a multiplicity matrix $D = (d_{i,j})$ with respect to G and M, such that*

1. *$d_{i,j} \ge \tilde{p}_{i,j}$ for all $i, j \in V$, and—*
2. *$M' \le \mathbf{1} \cdot D \cdot \mathbf{1}^T \le M$*

(where M' is as defined in (2.6)). Moreover, the matrix D can be found by an efficient algorithm.

Proof. Consider a vertex $i \in V$. If $\tilde{r}_i > \tilde{\rho}_i$, then we say that vertex i has a *surplus* of $\tilde{r}_i - \tilde{\rho}_i$. In this case, by Lemma 2.3, we have that the surplus is equal to 1. On the other hand, if $\tilde{r}_i < \tilde{\rho}_i$ then vertex i has a *deficiency* of $\tilde{\rho}_i - \tilde{r}_i$, which again is equal to 1.

Of course, since $\sum_{i \in V} \tilde{\rho}_i = \sum_{i \in V} \tilde{r}_i = M'$, the total surplus is equal to the total deficiency, and both are denoted by Surp:

$$\mathrm{Surp} = \sum_{i \in V} \max\{0, \tilde{r}_i - \tilde{\rho}_i\} = -\sum_{i \in V} \min\{0, \tilde{r}_i - \tilde{\rho}_i\} \ . \tag{2.13}$$

Denote the vertices with surplus as $(s_k)_{k=1}^{\mathrm{Surp}}$ and the vertices with deficiency as $(t_k)_{k=1}^{\mathrm{Surp}}$. Recalling the matrix F from Lemma 2.4, we define

$$D = \tilde{P} + \sum_{k=1}^{\mathrm{Surp}} F^{(s_k, t_k)} \ .$$

We first show that D is a valid multiplicity matrix. Note that $\mathrm{Surp} \le |V|/2$. Thus, (2.2) follows from (2.6), (2.7), and (i). The definitions of surplus and deficiency vertices along with (iii) give (2.3). Lastly, recall that (2.4) is satisfied if we replace $d_{i,j}$ by $p_{i,j}$. Thus, by (2.9), the same can be said for $\tilde{p}_{i,j}$. Combining this with (ii) yields (2.4).

Since the entries of $F^{(s_k,t_k)}$ are non-negative for every k, we must have that $d_{i,j} \geq \tilde{p}_{i,j}$ for all $i, j \in V$. This, together with (2.2) and (2.7), implies in turn that $M' \leq \mathbf{1} \cdot D \cdot \mathbf{1}^T \leq M$.

■

Running Example (4): For the matrix $\tilde{P}$ in Figure 2.6, we have

$$\tilde{\boldsymbol{r}} = \begin{pmatrix} 6 & 2 & 1 \end{pmatrix}, \quad \tilde{\boldsymbol{\rho}} = \begin{pmatrix} 6 & 3 & 0 \end{pmatrix}.$$

Thus, Surp = 1. Namely, the vertex θ has a surplus while the vertex β has a deficiency. Taking $s = \theta$ and $t = \beta$ we get

$$F^{(s,t)} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \quad \text{and} \quad D = \begin{pmatrix} 4 & 3 & 0 \\ 2 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

■

Now that Theorem 2.5 is proved, we are in a position to prove our main result, Theorem 2.1. Essentially, the proof involves using the Stirling approximation and taking into account the various quantization errors introduced into D. The proof itself is given in the Appendix.

2.5 Enumerative coding into sequences with a given Markov type

The main motivation for our methods is 2-D constrained coding. However, in this section, we show that they might be interesting in certain aspects of 1-D coding as well. Given a labeled graph G, a classic method for building an encoder for the 1-D constraint $S(G)$ is the state-splitting algorithm [1]. The rate of an encoder built by [1] approaches the capacity of $S(G)$. Also, the word the encoder outputs has a corresponding path in G, with the following favorable property: the probability of traversing a certain edge approaches the maxentropic probability of that edge (assuming an unbiased source distribution). However, what if we'd like to build an encoder with a different probability distribution on the edges? This scenario may occur, for example, when there is a requirement that all the output words of a given length N that are generated by the encoder have a prescribed Hamming weight[2].

More formally, suppose that we are given a labeled graph $G = (V, E, L)$; to make the exposition simpler, suppose that G does not contain parallel edges. Let Q and $\boldsymbol{\pi}$ be a transition matrix and a stationary probability distribution corresponding to a stationary (but not necessarily maxentropic) Markov chain $\mathcal{P}$ on G. We assume w.l.o.g. that each edge in G has a positive conditional probability. We are also given an integer M, which we will shortly elaborate on.

We first describe our encoder in broad terms, so as that its merits will be obvious. Let D and N be as previously defined, and let $R_T(D)$ be specified shortly. We start at some fixed vertex $v_0 \in V$. Given $M \cdot R_T(D)$ information bits, we traverse a soon to be defined cyclic path of length N in G. The concatenation of the edge labels along the path is the word we output. Of course, since the path is cyclic, the concatenation of such words is indeed in $S(G)$. Moreover, the path will have the following key property: the number of times an edge from i to j is traversed equals $d_{i,j}$. Namely, if we uniformly pick one of the N edges of the path, the probability of picking a certain edge e is constant (not a function of the input bits), and is equal to the probability of traversing e on the Markov chain $\mathcal{P}$, up to a small quantization error. The rate R_T of our encoder will satisfy (2.1), where we replace R by R_T and $\mathsf{cap}(S)$ by the entropy of $\mathcal{P}$. We would like to be able to exactly specify the path length N as a design parameter. However, we specify M and get an N between M and $M - \lfloor |V| \operatorname{diam}(G)/2 \rfloor$.

Our encoding process will make use of an *oriented tree*, a term which we will now define. A set of edges $T \subseteq E$ is an oriented tree of G with root v_0 if $|T| = |V| - 1$ and for each $u \in V$ there exists a path from u to v_0 consisting entirely of edges in T (see Figure 2.7). Note that if we reverse the edge directions of an oriented tree, we get a directed tree as defined in [19, Theorem 2.5]. Since reversing the directions of all edges in an irreducible

[2]We remark in passing that one may use convex programming techniques (see [30, §V]) in order to efficiently solve the following optimization problem: find a probability distribution on the edges of G yielding a stationary Markov chain with largest possible entropy, subject to a set of edges (such as the set of edges with label '1') having a prescribed cumulative probability.

graph results in an irreducible graph, we have by [19, Lemma 3.3] that an oriented tree T indeed exists in G, and can be efficiently found. So, let us fix some oriented tree T with root v_0. By [19, Theorem 2.5], we have that every vertex $u \in V$ which is not the root v_0 has an out-degree equal to 1. Thus, for each such vertex u we may define $\text{parent}(u)$ as the destination of the single edge in T going out of u.

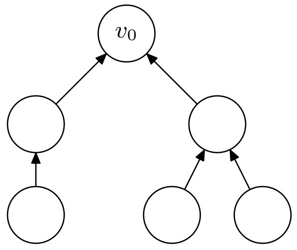


Figure 2.7: Oriented tree with root v_0.

We now elaborate on the encoding process. The encoding consists of two steps. In the first step, we map the information bits to a collection of lists. In the second step, we use the lists in order to define a cyclic path.

First step: Given $M \cdot R_T(D)$ information bits, we build for each vertex $i \in V$ a list $\boldsymbol{\lambda}^{(i)}$ of length r_i,

$$\boldsymbol{\lambda}^{(i)} = (\lambda_1^{(i)}, \lambda_2^{(i)}, \ldots, \lambda_{r_i}^{(i)}) .$$

The entries of each $\boldsymbol{\lambda}^{(i)}$ are vertices in V. Moreover, the following properties are satisfied for all i:

- The number of times j is an entry in $\boldsymbol{\lambda}^{(i)}$ is exactly $d_{i,j}$.
- If $i \neq v_0$, then the last entry of the list equals the parent of i. Namely,

$$\lambda_{r_i}^{(i)} = \text{parent}(i) .$$

Recalling (2.5), a simple calculation shows that the number of possible list collections is

$$\Delta_T = \Delta \cdot \prod_{i \in V \setminus \{v_0\}} \frac{d_{i,\text{parent}(i)}}{r_i} . \tag{2.14}$$

Thus, we define the rate of encoding as

$$R_T = \frac{\lfloor \log_2 \Delta_T \rfloor}{M} \ .$$

Also, note that as in the 2-D case, we may use enumerative coding in order to efficiently map information bits to lists.

Second step: We now use the lists $\boldsymbol{\lambda}^{(i)}$, $i \in V$, in order to construct a cyclic path starting at vertex v_0. We start the path at v_0 and build a length-N path according to the following rule: when exiting vertex i for the kth time, traverse the edge going into vertex $\lambda_k^{(i)}$.

Of course, our encoding method is valid (and invertible) iff we may always abide by the above-mentioned rule. Namely, we don't get "stuck", and manage to complete a cyclic path of length N. This is indeed the case: define an auxiliary graph $G(D)$ with the same vertex set, V, as G and $d_{i,j}$ parallel edges from i to j (for all $i, j \in V$). First, recall that for sufficiently large M, the presence of an edge from i to j in G implies that $d_{i,j} > 0$. Thus, since G was assumed to be irreducible, $G(D)$ is irreducible as well. Also, an edge in T from i to j implies the existence of an edge in $G(D)$ from i to j. Secondly, note that by (2.3), the number of times we are supposed to exit a vertex is equal to the number of times we are supposed to enter it. The rest of the proof follows from [40, p. 56, Claim 2], applied to the auxiliary graph $G(D)$. Namely, our encoder follows directly from van Aardenne-Ehrenfest and de Bruijn's [42] theorem on counting Eulerian cycles in a graph.

We now return to the rate, R_T, of our encoder. From (2.6), (2.9), (2.10) and Theorem 2.5, we see that for M sufficiently large, Δ_T is greater than some positive constant times Δ. Thus, (2.1) still holds if we replace R by R_T and $\mathsf{cap}(S)$ by the entropy of $\mathcal{P}$.

2.6 An example, and two improvement techniques

Recall from Section 2.2 the square constraint: its elements are all the binary arrays in which no two '1' symbols are adjacent on a row, column, or diagonal. By employing the methods presented in [9], we may calculate an upper bound on the rate of the constraint. This turns out to be 0.425078. We will show an encoding/decoding method with rate slightly larger than 0.396 (about 93% of the upper bound). In order to do this, we assume that the array has 100,000 columns. Our encoding method has a fixed rate and has a vertical window of size 2 and vertical anticipation 0.

We should point out now that a straightforward implementation of the methods we have previously defined gives a rate which is strictly *less* than 0.396. Namely, this section also outlines two improvement techniques which help boost the rate.

We start out as in the example given in Section 2.2, except that the width of the data strips is now $w_{\mathrm{d}} = 9$ (the width of the merging strips remains $w_{\mathrm{m}} = 1$). The graph G we choose produces all width-w_{d} arrays satisfying the kings constraint, and we take the merging strips to be all-zero. Our array has 100,000 columns, so we have $M = 10{,}000$ tracks (the last, say, column of the array will essentially be unused; we can set all of its values to 0).

Define the normalized capacity as

$$\frac{\mathsf{cap}(S(G))}{w_{\mathrm{d}} + w_{\mathrm{m}}} .$$

The graph G has $|V| = 89$ vertices and normalized capacity

$$\frac{\mathsf{cap}(S(G))}{w_{\mathrm{d}} + w_{\mathrm{m}}} \approx \frac{\mathsf{cap}(S(G))}{w_{\mathrm{d}} + w_{\mathrm{m}}(1 - 1/M)} \approx 0.402 .$$

This number is about 94.5% from the upper bound on the capacity of our 2-D constraint. Thus, as expected, there is an inherent loss in choosing to model the 2-D constraint as an essentially 1-D constraint. Of course, this loss can be made smaller by increasing w_{d} (but the graph G will grow as well).

From Theorem 2.1, the rate of our encoder will approach the normalized capacity of 0.402 as the number of tracks M grows. So, once the graph G is chosen, the parameter we should be comparing ourselves to is the normalized capacity. We now apply the methods defined in Section 2.4 and find a multiplicity matrix D. Recall that the matrix D defines an encoder. In our case, this encoder has a rate of about 0.381. This is 94% of the normalized capacity, and is quite disappointing (but the improvements shown in Sections 2.6.1 and 2.6.2 below are going to improve this rate). On the other hand, note that if we had limited ourselves to encode to each track independently of the others, then the best rate we could have hoped for with 0 vertical anticipation turns out to be 0.3 (see [29, Theorem 5]).

2.6.1 Moore-style reduction

We now define a graph G which we call the reduction of G. Essentially, we will encode by constructing paths in G, and then translate these to paths in G. In both G and G, the maxentropic distributions have the same entropy. The main virtue of G is that it often has less vertices and edges compared to G. Thus, the penalty in (2.1) resulting from using a finite number of tracks will often be smaller.

For $s \geq 0$, we now recursively define the concept of s-equivalence (very much like in the Moore algorithm [31, page 1660]).

- For $s = 0$, any two vertices $v_1, v_2 \in V$ are 0-equivalent.
- For $s > 0$, two vertices $v_1, v_2 \in V$ are s-equivalent iff 1) the two vertices v_1, v_2 are $(s-1)$-equivalent, and 2) for each $(s-1)$-equivalence class c, the number of edges from v_1 to vertices in c is equal to the number of edges from v_2 to vertices in c.

Denote by Π_s the partition induced by s-equivalence. For the graph G given in Figure 2.3,

$$\Pi_0 = \{0000,0001,0010,0100,0101,1000,1001,1010\} \ ,$$
$$\Pi_{s\geq 1} = \{0000\},\{0010,0100\},\{1000,0001\},\{1010,1001,0101\} \ .$$

Note that, by definition, Π_{s+1} is a refinement of Π_s. Thus, let s' be the smallest s for which $\Pi_s = \Pi_{s+1}$. The set $\Pi_{s'}$ can be efficiently found (essentially, by the Moore algorithm [31, page 1660]).

Define a (non-labeled) graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ as follows. The vertex set of G is

$$\mathsf{V} = \Pi_{s'} \ .$$

For each $\mathsf{c} \in \mathsf{V}$, let $v(\mathsf{c})$ be a fixed element of c (if c contains more than one vertex, then pick one arbitrarily). Also, for each $v \in V$, let $\mathsf{c}(v)$ be the class $\mathsf{c} \in \mathsf{V}$ such that $v \in \mathsf{c}$. Let $\sigma_G(e)$ ($\sigma_{\mathsf{G}}(e)$) and $\tau_G(e)$ ($\tau_{\mathsf{G}}(e)$) denote the start and end vertex of an edge e in G (G), respectively. The edge set E is defined as

$$\mathsf{E} = \bigcup_{\mathsf{c}\in\mathsf{V}} \{e \in E : \sigma_G(e) = v(\mathsf{c})\} \ , \tag{2.15}$$

where

$$\sigma_{\mathsf{G}}(e) = \mathsf{c}(\sigma_G(e)) \quad \text{and} \quad \tau_{\mathsf{G}}(e) = \mathsf{c}(\tau_G(e)) \ .$$

Namely, the number of edges from c_1 to c_2 in G is equal to the number of edges in G from some fixed $v_1 \in \mathsf{c}_1$ to elements of c_2, and, by the definition of s', this number does not depend on the choice of v_1. The graph G is termed the *reduction* of G. The reduction of G from Figure 2.3 is given in Figure 2.8. Note that since G was assumed to be irreducible, we must have that G is irreducible as well.

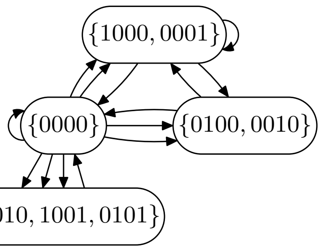


Figure 2.8: Reduction of the graph G from Figure 2.3.

Lemma 2.6 *The entropies of the maxentropic Markov chains on G and G are equal.*

Proof. Let $\mathsf{A} = \mathsf{A}_{\mathsf{G}}$ be the adjacency matrix of G, and recall that $A = A_G$ is the adjacency matrix of G. Let λ' and $\boldsymbol{x}' = (x'_{\mathsf{c}})_{\mathsf{c} \in \mathsf{V}}$ be the Perron eigenvalue and right Perron eigenvector of A, respectively [31, §3.1]. Next, define the vector $\boldsymbol{x} = (x_v)_{v \in V}$ as

$$x_v = x'_{\mathsf{c}(v)} \ .$$

It is easily verifiable that $\boldsymbol{x}$ is a right eigenvector of A, with eigenvalue λ'. Now, since $\boldsymbol{x}'$ is a Perron eigenvector of an irreducible matrix, each entry of it is positive. Thus, each entry of $\boldsymbol{x}$ is positive as well. Since A is irreducible, we must have that $\boldsymbol{x}$ is a Perron eigenvector of A. So, the Perron eigenvalue of A is also λ'. ∎

The next lemma essentially states that we can think of paths in G as if they were paths in G.

Lemma 2.7 *Let $\ell \geq 1$. Fix some $\mathsf{c}_0, \mathsf{c}_{\ell+1} \in \mathsf{V}$, and $v_0 \in \mathsf{c}_0$. There exists a one-to-one correspondence between the following sets. First set: paths of* *length ℓ in G with start vertex c_0 and end vertex $\mathsf{c}_{\ell+1}$. Second set: paths of length ℓ in G with start vertex v_0 and end vertex in $\mathsf{c}_{\ell+1}$.*

Moreover, for $1 \leq t \leq \ell - 1$, the first t edges in a path belonging to the second set are a function of only the first t edges in the respective path in the first set.

Proof. We prove this by induction on ℓ. For $\ell = 1$, we have

$$\begin{aligned}&|\{e \in \mathsf{E} : \sigma_{\mathsf{G}}(e) = \mathsf{c}_0\ , \quad \tau_{\mathsf{G}}(e) = \mathsf{c}_1\}| = \\ &|\{e \in E : \sigma_G(e) = v_0\ , \quad \tau_G(e) \in \mathsf{c}_1\}|\ .\end{aligned}$$

To see this, note that we can assume w.l.o.g. that $v_0 = v(\mathsf{c}_0)$, and then recall (2.15). For $\ell > 1$, combine the claim for $\ell - 1$ with that for $\ell = 1$. ■

Notice that $\mathrm{diam}(\mathsf{G}) \leq \mathrm{diam}(G)$. We now show why G is useful.

Theorem 2.8 *Let D be the multiplicity matrix found by the methods previously outlined, where we replace G by G. Let $\mathsf{N} = \mathbf{1} \cdot \mathsf{D} \cdot \mathbf{1}^T$. We may efficiently encode (and decode) information to $G^{\otimes \mathsf{N}}$ in a row-by-row manner at rate $R(\mathsf{D})$.*

Proof. We conceptually break our encoding scheme into two steps. In the first step, we "encode" (map) the information into N paths in G, each path having length ℓ. We do this as previously outlined (through typical vertices and edges in G). Note that this step is done at a rate of $R(\mathsf{D})$. In the second step, we map each such path in G to a corresponding path in G. By Lemma 2.7, we can indeed do this (take c_0 as the first vertex in the path, $\mathsf{c}_{\ell+1}$ as the last vertex, and $v_0 = v(\mathsf{c}_0)$).

By Lemma 2.7 we see that this two-step encoding scheme can easily be modified into one that is row-by-row. ■

Applying the reduction to our running example (kings constraint with $w_{\mathrm{d}} = 9$ and $w_{\mathrm{m}} = 1$), reduces the number of vertices from 89 in G to 34 in G. The computed D increases the rate to about 0.392, which is 97.5% of the normalized capacity.

2.6.2 Break-merge

Let $\mathsf{G}^{\otimes \mathsf{N}}$ be the Nth Kronecker power of the Moore-style reduction G. Recall that the rate of our encoder is

$$R(\mathsf{D}) = \frac{\lfloor \log_2 \Delta \rfloor}{M}\ ,$$

where Δ is the number of typical edges in $\mathsf{G}^{\otimes \mathsf{N}}$ going out of a typical vertex. The second improvement involves expanding the definition of a typical edge, thus increasing Δ. This is best explained through an example. Suppose that G has Figure 2.9 as a subgraph; namely, we show all edges going out of vertices α and β. Also, let the numbers next to the edges be equal to the corresponding entries in D. The main thing to notice at this point is that if the edges to ϵ and ζ are deleted ("break"), then α and β have exactly the same number of edges from them to vertex j, for all $j \in V$ (after the deletion of edges, vertices α and β can be "merged").

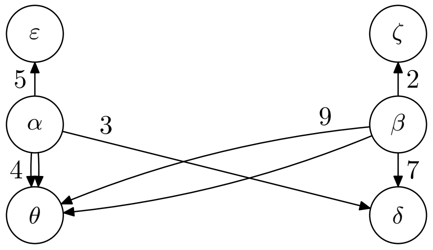


Figure 2.9: Break-merge example graph.

Let $\boldsymbol{v}$ be a typical vertex. A short calculation shows that the number of entries in $\boldsymbol{v}$ that are equal to α (β) is $5+4+3=12$ ($9+7+2=18$). Recall that the standard encoding process consists of choosing a typical edge $\boldsymbol{e}$ going out of the typical vertex $\boldsymbol{v}$ and into another typical vertex $\boldsymbol{v}'$. We now briefly review this process. Consider the 12 entries in $\boldsymbol{v}$ that are equal to α. The encoding process with respect to them will be as follows (see Figure 2.10):

- Out of these 12 entries, choose 5 for which the corresponding entry in $\boldsymbol{v}'$ will be ε. Since there is exactly one edge from α the ε in G, the corresponding entries in $\boldsymbol{e}$ must be equal to that edge.

- Next, from the remaining 7 entries, choose 4 for which the corresponding entries in $\boldsymbol{v}'$ will be θ. There are two parallel edges from α to θ, so choose which one to use in the corresponding entries in $\boldsymbol{e}$.

- We are left with 3 entries, the corresponding entries in $\boldsymbol{v}'$ will be δ. Also, we have one option as to the corresponding entries in $\boldsymbol{e}$.

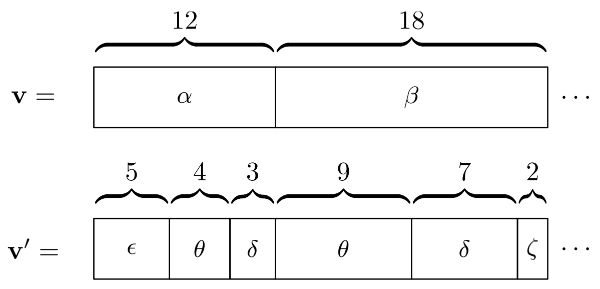


Figure 2.10: Illustration of the entries in two typical vertices $\boldsymbol{v}$, $\boldsymbol{v}'$, where we got from $\boldsymbol{v}$ to $\boldsymbol{v}'$ by the standard encoding process.

A similar process is applied to the entries in $\boldsymbol{v}$ that are equal to β. Thus, the total number of options with respect to these entries is

$$\frac{12! \cdot 2^4}{5! \cdot 4! \cdot 3!} \cdot \frac{18! \cdot 2^9}{2! \cdot 9! \cdot 7!} \approx 3.97 \cdot 10^{14} .$$

Next, consider a different encoding process (see Figure 2.11).

- Out of the 12 entries in $\boldsymbol{v}$ that are equal to α, choose 5 for which the corresponding entry in $\boldsymbol{v}'$ will be ε. As before, the corresponding entries in $\boldsymbol{e}$ have only one option.
- Out of the 18 entries in $\boldsymbol{v}$ that are equal to β, choose 2 for the corresponding entry in $\boldsymbol{v}'$ will be ζ. Again, one option for entries in $\boldsymbol{e}$.
- Now, of the remaining 23 entries in $\boldsymbol{v}$ that are equal to α or β, choose $4 + 9 = 13$ for which the corresponding entry in $\boldsymbol{v}'$ will be θ. We have two options for the entries in $\boldsymbol{e}$.
- We are left with $3+7 = 10$ entries in $\boldsymbol{v}$ that are equal to α or β. These will have δ as the corresponding entry in $\boldsymbol{v}'$, and one option in $\boldsymbol{e}$.

Thus, the total number of options is now

$$\binom{12}{5} \cdot \binom{18}{2} \cdot \frac{23! \cdot 2^{13}}{13! \cdot 10!} \approx 1.14 \cdot 10^{15} .$$

The important thing to notice is that in both cases, we arrive at a typical vertex $\boldsymbol{v}'$.

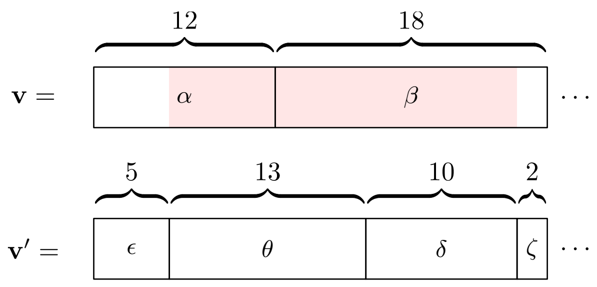


Figure 2.11: Illustration of the entries in two typical vertices $\boldsymbol{v}$, $\boldsymbol{v}'$, where we got from $\boldsymbol{v}$ to $\boldsymbol{v}'$ by the improved encoding process. The shaded part corresponds to vertices that were merged.

To recap, we first “broke” the entries in $\boldsymbol{v}$ that are equal to α into two groups: Those which will have ε as the corresponding entry in $\boldsymbol{v}'$ and those which will have θ or δ as the corresponding entry. Similarly, we broke entries in $\boldsymbol{v}$ that are equal to β into two groups. Next, we noticed that of these four groups, two could be “merged”, since they were essentially the same. Namely, removing some edges from the corresponding vertices in G resulted in vertices which were mergeable.

Of course, these operations can be repeated. The hidden assumption is that the sequence of breaking and merging is fixed, and known to both the encoder and decoder. The optimal sequence of breaking and merging is not known to us. We used a heuristic. Namely, choose two vertices such that the sets of edges emanating from both have a large overlap. Then, break and merge accordingly. This was done until no breaking or merging was possible. We got a rate of about 0.396, which is 98.5% of the normalized capacity.

2.7 Fast enumerative coding

Recall from Section 2.3 that in the course of our encoding algorithm, we make use of a procedure which encodes information into fixed-length binary words of constant weight. A way to do this would be to use enumerative coding [12]. Immink [27] showed a method to significantly improve the running time of an instance of enumerative coding, with a typically negligible penalty in terms of rate. We now briefly show how to tailor Immink's method to our needs.

Denote by n and δ the length and Hamming weight, respectively, of the binary word we encode into. Some of our variables will be *floating-point* numbers with a mantissa of μ bits and an exponent of ϵ bits: each floating-point number is of the form $\overline{x} = a \cdot 2^b$ where a and b are integers such that

$$2^\mu \le a < 2^{\mu+1} \quad \text{and} \quad -2^{\epsilon-1} \le b < 2^{\epsilon-1} \ .$$

Note that $\mu + \epsilon$ bits are needed to store such a number. Also, note that every positive real x such that

$$2^\mu \cdot 2^{-2^{\epsilon-1}} \le x \le (2^{\mu+1} - 1) \cdot 2^{2^{\epsilon-1}-1}$$

has a floating point approximation $\overline{x}$ with relative precision

$$\left(1 - \frac{1}{2^\mu}\right) \le \frac{\overline{x}}{x} \le \left(1 + \frac{1}{2^\mu}\right) \ . \tag{2.16}$$

We assume the presence of two look-up tables. The first will contain the floating-point approximations of $1!, 2!, \ldots, n!$. The second will contain the floating-point approximations of $f(0), f(1), \ldots, f(\delta)$, where

$$f(\chi) = f_\mu(\chi) = 1 - \frac{32\chi + 16}{2^\mu} \ .$$

In order to exclude uninteresting cases, assume that $\mu \ge 10$ and is such that $f(\delta) \ge 1/2$. Also, take ϵ large enough so that $n!$ is less than the maximum number we can represent by floating point. Thus, we can assume that $\mu = O(\log \delta)$ and $\epsilon = O(\log n)$.

Notice that in our case, we can bound both n and δ from above by the number of tracks M. Thus, we will actually build beforehand two look-up tables of size $2M(\mu + \epsilon)$ bits.

Let $\overline{x}$ denote the floating-point approximation of x, and let $*$ and $\div$ denote floating-point multiplication and division, respectively. For $0 \le \chi \le \kappa \le n$ we define

$$\left\lceil \begin{matrix} \kappa \\ \chi \end{matrix} \right\rceil = \left\lceil (\overline{\kappa!} * \overline{f(\chi)}) \div (\overline{\chi!} * \overline{(\kappa - \chi)!}) \right\rceil \ .$$

Note that since we have stored the relevant numbers in our look-up table, the time needed to calculate the above function is only $O(\mu^2 + \epsilon)$. The encoding procedure is given in Figure 2.12. We note the following points:

- The variables n, ψ, δ and ι are integers (as opposed to floating-point numbers).
- In the subtraction of $\left\lceil{n-\iota \atop \delta-1}\right\rceil$ from ψ in line 5, the floating-point number $\left\lceil{n-\iota \atop \delta-1}\right\rceil$ is "promoted" to an integer (the result is an integer).

Name: EnumEncode(n, δ, ψ)

Input: Integers n, δ, ψ such that $0 \leq \delta \leq n$ and $0 \leq \psi < \left\lceil{n \atop \delta}\right\rceil$.
Output: A binary word of length n and weight δ.

```
if ($\delta == 0$) // stopping condition:                          /* 1 */
    return $\underbrace{00\ldots0}_{n}$;                                   /* 2 */
for ($\iota \leftarrow 1$; $\iota \leq n - \delta + 1$; $\iota$++) {                     /* 3 */
    if ($\psi \geq \left\lceil{n-\iota \atop \delta-1}\right\rceil$)                                        /* 4 */
        $\psi \leftarrow \psi - \left\lceil{n-\iota \atop \delta-1}\right\rceil$;                                /* 5 */
    else                                                          /* 6 */
        return $\underbrace{00..0}_{\iota-1}1\|$EnumEncode$(n-\iota, \delta-1, \psi)$;         /* 7 */
}                                                                 /* 8 */
```

Figure 2.12: Enumerative encoding procedure for constant-weight binary words.

We must now show that the procedure is valid, namely, that given a valid input, we produce a valid output. For our procedure, this reduce to showing two things: 1) If the stopping condition is not met, a recursive call will be made. 2) The recursive call is given valid parameters as well. Namely, in the recursive call, ψ is non-negative. Also, for the encoding to be invertible, we must further require that 3) $\left\lceil{n \atop 0}\right\rceil = 1$ for $n \geq 0$.

Condition 2 is clearly met, because of the check in line 4. Denote

$$\left\langle{\kappa \atop \chi}\right\rangle = (\overline{\kappa!} * \overline{f(\chi)}) \div (\overline{\chi!} * \overline{(\kappa-\chi)!})$$

(and so, $\left\lceil{\kappa \atop \chi}\right\rceil = \left\lceil\left\langle{\kappa \atop \chi}\right\rangle\right\rceil$). Condition 3 follows from the next lemma.

Lemma 2.9 *Fix $0 \le \delta \le n$. Then,*

$$\binom{n}{\delta} \cdot \left(1 - \frac{32(\delta+1)}{2^\mu}\right) \le \left\langle {n \atop \delta} \right\rangle \le \binom{n}{\delta} \cdot \left(1 - \frac{32\delta}{2^\mu}\right) .$$

Proof. The proof is essentially repeated invocations of (2.16) on the various stages of computation. We leave the details to the reader. ■

Finally, Condition 1 follows easily from the next lemma.

Lemma 2.10 *Fix $0 \le \delta \le n$. Then,*

$$\left\lceil {n \atop \delta} \right\rceil \le \sum_{\iota=1}^{n-\delta+1} \left\lceil {n-\iota \atop \delta-1} \right\rceil .$$

Proof. The claim will follow if we show that

$$\left\langle {n \atop \delta} \right\rangle \le \sum_{\iota=1}^{n-\delta+1} \left\langle {n-\iota \atop \delta-1} \right\rangle .$$

This is immediate from Lemma 2.9 and the binomial identity

$$\binom{n}{\delta} = \sum_{\iota=1}^{n-\delta+1} \binom{n-\iota}{\delta-1} .$$

■

Note that the penalty in terms of rate one suffers because of using our procedure (instead of plain enumerative coding) is negligible. Namely, $\log_2 \left\lceil {n \atop \delta} \right\rceil$ can be made arbitrarily close to $\log_2 \binom{n}{\delta}$. Since we take $\epsilon = O(\log n)$ and $\mu = O(\log \delta)$, we can show by amortized analysis that the running time of the procedure is $O(n \log^2 n)$. Specifically, see [11, Section 17.3], and take the potential of the binary vector corresponding to ψ as the number of entries in it that are equal to '0'. The decoding procedure is a straightforward "reversal" of the encoding procedure, and its running time is also $O(n \log^2 n)$.

Appendix

Proof of Theorem 2.1: Let $\tilde{\Delta}$ be as in (2.5), where we replace $d_{i,j}$ by $\tilde{p}_{i,j}$ and r_i by $\tilde{\rho}_i$. By the combinatorial interpretation of (2.5), and the fact that $d_{i,j} \geq \tilde{p}_{i,j}$ for all $i, j \in V$, it easily follows that $\Delta \geq \tilde{\Delta}$. Thus,

$$R(D) \geq \frac{\lfloor \log_2 \tilde{\Delta} \rfloor}{M} = \frac{M'}{M} \cdot \frac{\lfloor \log_2 \tilde{\Delta} \rfloor}{M'} \ .$$

Denote by e the base of natural logarithms. By Stirling's formula we have

$$\log_2(t!) = t \log_2(t/\mathsf{e}) + O(\log t) \ ,$$

and from (2.5) we get that

$$\log_2 \tilde{\Delta} = \sum_{i \in V} \tilde{\rho}_i \log_2(\tilde{\rho}_i/\mathsf{e}) - \sum_{i,j \in V} \tilde{p}_{i,j} \log_2(\tilde{p}_{i,j}/\mathsf{e}) \\ + \sum_{i,j \in V} \tilde{p}_{i,j} \log_2(a_{i,j}) - O(|V|^2 \log M) \ .$$

By (2.7) and (2.9),

$$\sum_{i,j \in V} \tilde{p}_{i,j} \log_2(a_{i,j}) \ = \ \sum_{i,j \in V} p_{i,j} \log_2(a_{i,j}) \ - \ O\left(|V|^2 \log_2(a_{\max}/a_{\min})\right) \ .$$

Since $\sum_j \tilde{p}_{i,j} = \tilde{\rho}_i$, we have

$$\sum_{i \in V} \tilde{\rho}_i \log_2(\tilde{\rho}_i/\mathsf{e}) - \sum_{i,j \in V} \tilde{p}_{i,j} \log_2(\tilde{p}_{i,j}/\mathsf{e}) \\ = \sum_{i \in V} \tilde{\rho}_i \log_2(\tilde{\rho}_i) - \sum_{i,j \in V} \tilde{p}_{i,j} \log_2(\tilde{p}_{i,j}) \ .$$

Moreover, by (2.8) and (2.9), the RHS of the last equation equals

$$\sum_{i \in V} \rho_i \log_2(\rho_i) - \sum_{i,j \in V} p_{i,j} \log_2(p_{i,j}) - O(|V|^2) \ .$$

We conclude that

$$\log_2 \tilde{\Delta} = \sum_{i \in V} \rho_i \log_2(\rho_i) - \sum_{i,j \in V} p_{i,j} \log_2(p_{i,j}) \\ + \sum_{i,j \in V} p_{i,j} \log_2(a_{i,j}) - O\left(|V|^2 (\log M \cdot a_{\max}/a_{\min})\right) \ .$$

Lastly, recall that $\rho_i = M'\pi_i$ and $p_{i,j} = \rho_i q_{i,j}$. Thus,

$$\log_2 \tilde{\Delta} = M' H(\mathcal{P}) - O\left(|V|^2 (\log M \cdot a_{\max}/a_{\min})\right) \ ,$$

where $H(\mathcal{P})$ is the entropy of the stationary Markov chain $\mathcal{P}$ with transition matrix Q. Recall that $\mathcal{P}$ was selected to be maxentropic: $H(\mathcal{P}) = \mathsf{cap}(S(G))$. This fact, along with (2.6) and a short calculation, finishes the proof. ■

Chapter 3

Bounds on the Rate of 2-D Bit-Stuffing Encoders

In this chapter, we present a method for bounding the rate of bit-stuffing encoders. Instead of considering the original encoder, we consider a related one which is quasi-stationary. We use the quasi-stationary property in order to formulate linear requirements that must hold on the probabilities of the constrained arrays that are generated by the encoder. These requirements are used as part of a linear program. The minimum and maximum of the linear program bound the rate of the encoder from below and from above, respectively.

A lower bound on the rate of an encoder is also a lower bound on the capacity of the corresponding constraint. For some constraints, our results lead to tighter lower bounds than what was previously known.

3.1 Introduction

Consider a 2-D constraint $\mathbb{S}$ defined over some finite alphabet Σ. Informally, a bit-stuffing encoder for $\mathbb{S}$ operates as follows. We encode information to an $M \times N$ rectangular array; namely, we produce an array $a \in \mathbb{S} \cap \Sigma^{M \times N}$. We first initialize the "boundaries" of the array (formally defined later) according to some fixed probability distribution. Then, we write to the "interior" of the array in raster fashion: row-by-row. The symbol currently written is the result of a coin toss. The probability distribution of the

$$\begin{array}{cccccccc}
1 & \mathbf{0} & 0 & 0 & \mathbf{0} & 1 & \mathbf{0} & 0 \\
\mathbf{0} & \mathbf{0} & 0 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
0 & 0 & 0 & 0 & 0 & \mathbf{0} & 1 & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & 1 & \mathbf{0} & 0 & 0 & 0 & 0 & 0
\end{array}$$

Figure 3.1: Binary array satisfying the kings constraint. If we flip any one (or more) of the highlighted "0" bits to "1", then the resulting array will not satisfy the kings constraint.

coin is a function of neighboring symbols, which have already been written. However, the "coins" used are in fact (invertible) probability transformers, the input of which is the information we wish to encode. Thus, information can be encoded, and decoded.

A bit-stuffing encoder is "variable-rate." The bit-stuffing technique was initially devised for encoding one-dimensional (1-D) constraints [5]. In [25] and [37], bit-stuffing encoders for specific 2-D constraints were presented and analyzed. In [22], a slightly different definition of bit-stuffing was used to give lower bounds on the capacity of specific 2-D constraints.

In this chapter, we derive upper and lower bounds on the rate of a general bit-stuffing encoder. A lower bound on the rate of an encoder is also a lower bound on the capacity of the corresponding constraint:

$$\mathsf{cap}(\mathbb{S}) = \lim_{M,N\to\infty} \frac{1}{M \cdot N} \cdot \log_2 \left|\mathbb{S} \cap \Sigma^{M\times N}\right| \ .$$

For some constraints, our results lead to tighter lower bounds on capacity than what was previously known.

Fix some 2-D constraint $\mathbb{S}$ over an alphabet Σ. As a running example, consider the kings constraint $\mathbb{S}_{\text{kgs}}$, defined over the binary alphabet $\Sigma_{\text{kgs}} = \{0, 1\}$ (see Figure 3.1). A binary array satisfies the kings constraint if each entry set to "1" has all of its eight neighbors set to "0". Namely, two entries equal to "1" may not appear consecutively along a row, column, or diagonal.

The rest of this chapter is organized as follows. In Sections 3.2 and 3.3, we define our notation and our model of a bit-stuffing encoder, respectively. In Section 3.4, we define the concept of quasi-stationarity. We also prove that, w.l.o.g., we may assume that our encoder is quasi-stationary. In Section 3.5, we take advantage of the quasi-stationary property and define a linear program. The minimum (maximum) of the linear program bounds the rate of our encoder from below (above). Finally, Section 3.6 states a generic lower bound on capacity, and contains examples where this bound improves on previous results.

We note at this point that although this chapter deals with 2-D constraints, our method can be easily generalized to higher dimensions as well.

3.2 Notation

We first recall the relevant notation from Section 1.2, and add some new notation.

Parallelogram and rectangle: For $M, N > 0$ and $t \geq 0$, denote

$$\mathsf{B}^{(t)}_{m,n} = \{(i,j) : 0 \leq i < M \ , \quad 0 \leq t \cdot i + j < N\} \ .$$

Also, for $t = 0$, denote

$$\mathsf{B}_{M,N} = \mathsf{B}^{(0)}_{m,n} \ .$$

Configuration: Let $a = (a_{i,j})_{(i,j) \in \mathsf{U}}$ be a 2-D configuration over Σ. Namely, the index set satisfies $\mathsf{U} \subseteq \mathbb{Z}^2$, and for all $(i,j) \in \mathsf{U}$ we have that $a_{i,j} \in \Sigma$.

Shifts: For integers α, β we denote the shifted index set as

$$\sigma_{\alpha,\beta}(\mathsf{U}) = \{(i+\alpha, j+\beta) : (i,j) \in \mathsf{U}\} \ .$$

Also, by abuse of notation, let $\sigma_{\alpha,\beta}(a)$ be the shifted configuration (with index set $\sigma(\mathsf{U})$):

$$\sigma_{\alpha,\beta}(a)_{i+\alpha,j+\beta} = a_{i,j} \ .$$

Restriction of configuration: For an index set $\Psi \subseteq \mathsf{U}$, denote the restriction of a to Ψ by $a[\Psi] = (a[\Psi]_{i,j})_{(i,j) \in \Psi}$. Namely,

$$a[\Psi]_{i,j} = a_{i,j} \ , \quad \text{where} \quad (i,j) \in \Psi \ .$$

Shift and restrict: Let $\tau_{\alpha,\beta}(a, \Psi)$ be shorthand for

$$\tau_{\alpha,\beta}(a, \Psi) = (\sigma_{-\alpha,-\beta}(a))[\Psi] \ .$$

Namely, shift the configuration a such that index (α, β) is now index $(0,0)$, and then restrict to Ψ.

Boundary: Denote by $\partial(\mathsf{U}, \Psi)$ the set of all the indexes $(\alpha, \beta) \in \mathsf{U}$ for which the "shift and restrict" operation is invalid.

$$\partial(\mathsf{U}, \Psi) = \{(\alpha, \beta) \in \mathsf{U} : \sigma_{\alpha,\beta}(\Psi) \not\subseteq \mathsf{U}\} \ .$$

The index set $\partial(\mathsf{U}, \Psi)$ is termed the "boundary," and the "interior" is

$$\bar{\partial}(\mathsf{U}, \Psi) = \mathsf{U} \setminus \partial(\mathsf{U}, \Psi) \ .$$

When $\mathsf{U} = \mathsf{B}_{M,N}$ and Ψ is understood from the context, we abbreviate

$$\partial_{M,N} = \partial(\mathsf{B}_{M,N}, \Psi) \ , \quad \bar{\partial}_{M,N} = \bar{\partial}(\mathsf{B}_{M,N}, \Psi) \ .$$

Figure 3.2 shows an example of such sets, where

$$\Psi = \{(0, -2), (0, -1), (-1, -1), (-1, 0), (-1, 1)\} \ . \tag{3.1}$$

Restriction of constraint: Denote the restriction of $\mathbb{S}$ to U by

$$\mathbb{S}[\mathsf{U}] = \{a : \text{there exists } a' \in \mathbb{S} \text{ such that } a'[\mathsf{U}] = a\} \ .$$

If $\mathsf{U} = \mathsf{B}_{M,N}$, then we abbreviate

$$\mathbb{S}_{M,N} = \mathbb{S}[\mathsf{B}_{M,N}] \ .$$

Lexicographic ordering: We define a lexicographic ordering $\prec_{\text{lex}}$ on $\mathbb{Z}^2$ as

$$(i', j') \prec_{\text{lex}} (i, j) \quad \Longleftrightarrow \quad (i' < i) \ \text{ or } \ (i' = i \quad \text{and} \quad j' < j) \ .$$

Also, we define the index set

$$\mathsf{T}_{i,j} = \big\{(i', j') : (i', j') \prec_{\text{lex}} (i, j)\big\} \ . \tag{3.2}$$

3.3 Bit stuffer definitions

In this section, we present the formal definition of bit-stuffing encoders. A bit-stuffing encoder for $\mathbb{S}$ is defined through a triple

$$\mathcal{E} = (\Psi, \mu, \boldsymbol{\delta} = (\delta_{M,N})_{M,N>0}) \ .$$

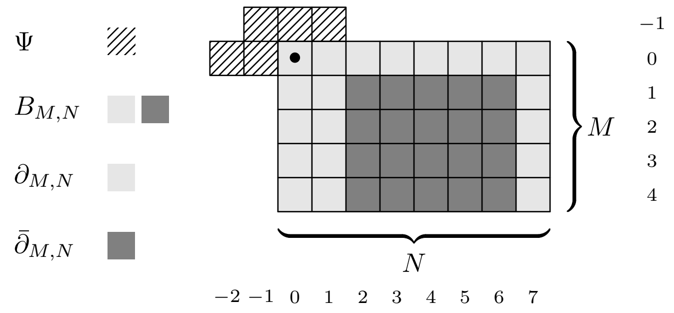


Figure 3.2: The index $(0,0)$ is represented by $\bullet$. We take Ψ as in (3.1), and it is represented by the diagonally striped cells. We set $M=5$ and $N=8$. The index set $\mathsf{B}_{M,N}$ is represented by the shaded part (both light and dark). The boundary $\partial_{M,N}$ is represented by the lighter shaded part, while the interior $\bar{\partial}_{M,N}$ is represented by the darker shaded part.

The set

$$\Psi \subseteq \mathsf{T}_{0,0} \tag{3.3}$$

is termed the *neighbor set.* The *conditional probability function* μ,

$$\mu(\cdot|\cdot)\ , \quad \mu : \Sigma \times \mathbb{S}[\Psi] \to [0,1]\ ,$$

is a conditional probability distribution on Σ, given an element of $\mathbb{S}[\Psi]$. For $M, N > 0$, the *boundary probability function*

$$\delta_{M,N} : \mathbb{S}[\partial_{M,N}] \to [0,1]$$

is a probability distribution on $\mathbb{S}[\partial_{M,N}]$. From here onward, we fix $\mathcal{E}$.

For our running example, let the neighbor set $\Psi_{\mathrm{kgs}} = \Psi$ be as in (3.1), and define $\varphi^{(0)}, \varphi^{(1)} \in \mathbb{S}_{\mathrm{kgs}}[\Psi]$ as

$$\begin{array}{lllll} \varphi^{(0)}_{0,-2}{=}0 & \varphi^{(0)}_{0,-1}{=}0 & \varphi^{(0)}_{-1,-1}{=}0 & \varphi^{(0)}_{-1,0}{=}0 & \varphi^{(0)}_{-1,1}{=}0 \\ \varphi^{(1)}_{0,-2}{=}1 & \varphi^{(1)}_{0,-1}{=}0 & \varphi^{(1)}_{-1,-1}{=}0 & \varphi^{(1)}_{-1,0}{=}0 & \varphi^{(1)}_{-1,1}{=}0 \end{array}$$

(see Figure 3.3). Also, take the conditional probability function as

$$\mu_{\mathrm{kgs}}(1|\varphi) = 1 - \mu_{\mathrm{kgs}}(0|\varphi) = \begin{cases} 0.258132 & \varphi = \varphi^{(0)} \\ 0.312231 & \varphi = \varphi^{(1)} \\ 0 & \text{otherwise}\ . \end{cases} \tag{3.4}$$

$$\varphi^{(0)} = \begin{array}{cccc} & 0 & 0 & 0 \\ 0 & 0 & \bullet & \end{array} \qquad \varphi^{(1)} = \begin{array}{cccc} & 0 & 0 & 0 \\ 1 & 0 & \bullet & \end{array}$$

Figure 3.3: The two non-trivial configurations for μ in our running example, where $\bullet$ designates coordinate $(0, 0)$.

Thus, $\mu_{\text{kgs}}(\cdot|\cdot)$ can be implemented using two coins (one for the context $\varphi^{(0)}$ and one for $\varphi^{(1)}$). For our running example, we take $\delta_{M,N}$ as the function equal to 1 for the all zero boundary $(0)_{(i,j)\in\partial_{M,N}}$, and 0 for all other members of $\mathbb{S}_{\text{kgs}}[\partial_{M,N}]$.

Given integers $M, N > 0$, the bit-stuffing encoder $\mathcal{E}$ defines a probability measure on the elements $a = (a_{i,j})_{(i,j)\in\mathsf{B}_{M,N}}$ of $\mathsf{B}_{M,N}$, in the following manner. As a first step, we set the boundary $a[\partial_{M,N}]$, according to the probability distribution $\delta_{M,N}$. Next, we write the contents of the interior of a in raster fashion: row-by-row, from left to right. The probability of writing $w \in \Sigma$ in entry $(i, j) \in \bar{\partial}_{M,N}$ is given by

$$\text{Prob}(a_{i,j} = w) = \mu(w|(\tau_{i,j}(a, \Psi)) \ .$$

Specifically, note that when writing entry (i, j), we have by (3.3) that $\tau_{i,j}(a)$ is a function of entries of a which have already been written. A fundamental requirement for Ψ and μ is that for every M, N, and $\delta_{M,N}$, the support of the probability measure thus defined is contained in $\mathbb{S}_{M,N}$.

Let

$$A(\mathcal{E}, M, N) = A = (A_{i,j})_{(i,j)\in\mathsf{B}_{M,N}}$$

be a random variable taking values on $\mathbb{S}_{M,N}$ according to the measure we have just defined. Namely,

$$\text{Prob}(A = a) = \delta_{M,N}(a[\partial_{M,N}]) \cdot \prod_{(i,j)\in\bar{\partial}_{M,N}} \mu(a_{i,j}|\tau_{i,j}(a, \Psi)) \ . \qquad (3.5)$$

We now explain how $\mathcal{E}$ is used to actually encode information. The "coin tosses" corresponding to the invocations of μ are, in effect, a function of the information we wish to encode. Specifically, the values of the tosses are the output of distribution transformers on the input stream (the mapping from the input stream to the sequence of coin tosses is one-to-one) [37]. Thus, we may encode information, and also decode it. So, we define the *rate* of our encoder as

$$R(\mathcal{E}) \triangleq \liminf_{M,N\to\infty} \frac{H(A[\bar{\partial}_{M,N}]|A[\partial_{M,N}])}{M \cdot N} \ ,$$

where

$$A = A(\mathcal{E}, M, N) \ .$$

Note that since

$$\liminf_{M,N\to\infty} \frac{|\bar{\partial}_{M,N}|}{M \cdot N} = 1 \ ,$$

we also have that

$$R(\mathcal{E}) = \liminf_{M,N\to\infty} \frac{H(A(\mathcal{E}, M, N))}{M \cdot N} \ .$$

3.4 Quasi-stationarity

Fix $k > 0$. Define the random variable

$$A^{(k)}(\mathcal{E}, M, N) = A^{(k)} = (A^{(k)}_{i,j})_{(i,j)\in \mathsf{B}_{M,N}}$$

taking values on $\mathbb{S}_{M,N}$ as follows. For $w \in \mathbb{S}_{M,N}$, we have

$$\text{Prob}(A^{(k)}{=}w) = \frac{1}{k^2} \sum_{0\le i,j<k} \text{Prob}(\sigma_{-i,-j}(A'[\mathsf{B}_{M,N}]){=}w) \ ,$$

where

$$A' = A(\mathcal{E}, M + k - 1, N + k - 1) \ .$$

Namely, given A', we randomly and uniformly pick an $M\times N$ sub-configuration of it, and shift accordingly. The usefulness of $A^{(k)}$ is that it is "quasi-stationary" [25, §6].

Lemma 3.1 ([25, Proposition 6.1]) *Let $\mathcal{E}$, M, N, and k be given. Let $\mathsf{U} \subseteq \mathsf{B}_{M,N}$ be an index set, and let $w \in \mathbb{S}[\mathsf{U}]$ be given. Suppose that for given integers α, β we have that $\sigma_{\alpha,\beta}(\mathsf{U}) \subseteq \mathsf{B}_{M,N}$. Denote $A^{(k)} = A^{(k)}(\mathcal{E}, M, N)$. Then,*

$$\Big|\text{Prob}(A^{(k)}[\mathsf{U}] = w) - \text{Prob}(A^{(k)}[\sigma_{\alpha,\beta}(\mathsf{U})] = \sigma_{\alpha,\beta}(w))\Big| \le \frac{|\alpha| + |\beta|}{k} \ .$$

Next, we show that $A^{(k)}$ is a random variable corresponding to an encoder very similar to $\mathcal{E}$. First, define $\boldsymbol{\delta}^{(k)} = (\delta^{(k)}_{M,N})_{M,N>0}$, where

$$\delta^{(k)}_{M,N} : \mathbb{S}[\partial_{M,N}] \to [0,1]$$

(that is, $\delta^{(k)}_{M,N}$ is a probability distribution on $\mathbb{S}[\partial_{M,N}]$), and for every $d \in \mathbb{S}[\partial_{M,N}]$,

$$\delta^{(k)}_{M,N}(d) = \text{Prob}(A^{(k)}(\mathcal{E}, M, N)[\partial_{M,N}] = d) \ .$$

Next, define the encoder $\mathcal{E}^{(k)}$ as

$$\mathcal{E}^{(k)} = (\Psi, \mu, \boldsymbol{\delta}^{(k)}) \ . \tag{3.6}$$

Lemma 3.2 ([25, Proposition 6.2]) *The probability distributions of $A^{(k)}(\mathcal{E}, M, N)$ and $A(\mathcal{E}^{(k)}, M, N)$ are equal.*

The next lemma essentially states that the normalized entropies of A and $A^{(k)}$ are asymptotically equal (for $M, N \to \infty$ and k fixed). The proof is straightforward.

Lemma 3.3 *Fix an integer $k > 0$. Then,*

$$R(\mathcal{E}) = R(\mathcal{E}^{(k)}) \ .$$

It follows from Lemma 3.3 that we can obtain bounds on $R(\mathcal{E})$ by bounding instead the rate of the quasi-stationary encoder $\mathcal{E}^{(k)}$. And, indeed, quasi-stationarity will turn out to be useful for this purpose.

3.5 Linear program

In this section, we present lower and upper bounds on $R(\mathcal{E})$. The bounds will be expressed as values of corresponding linear programs.

For $r, s > 0$ and $t \geq 0$, we say that the parallelogram $\mathsf{B}^{(t)}_{r,s}$ is valid with respect to the neighbor set Ψ if the set

$$\left\{ (\alpha, \beta) : (\Psi \cup (0,0)) \subseteq \sigma_{\alpha,\beta}(\mathsf{B}^{(t)}_{r,s}) \right\} \tag{3.7}$$

is non-empty. Namely, some shift of the parallelogram includes the neighbor set Ψ and $(0,0)$. From here onward, we fix r, s, and t so that $\mathsf{B}^{(t)}_{r,s}$ is valid.

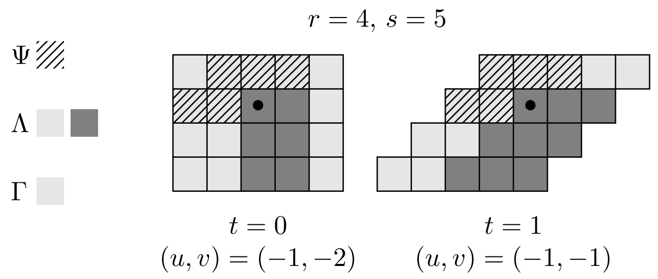


Figure 3.4: The index sets Ψ, Λ, and Γ. The index sets are shown for $r = 4$, $s = 5$, and for both $t = 0$ and $t = 1$. The index $(0, 0)$ is represented by $\bullet$. We take Ψ as in (3.1), and it is represented by the diagonally striped cells. The index set Λ is represented by the shaded part (both light and dark). The boundary Γ is represented by the lighter shaded part. Note that $\Psi \subseteq \Gamma \subseteq \Lambda$.

Also, we fix u and v, where (u, v) is the largest element of (3.7), with respect to the ordering $\prec_{\text{lex}}$.

Denote (see Figure 3.4)

$$\Lambda = \sigma_{u,v}(\mathsf{B}^{(t)}_{r,s}) \ , \quad \Gamma = \partial(\Lambda, \Psi) \ .$$

For an as yet unspecified probability distribution over $\mathbb{S}[\Gamma]$

$$\pi(z) \ , \quad z \in \mathbb{S}[\Gamma] \ ,$$

define the random variable Y taking values on $\mathbb{S}[\Lambda]$ as follows. For $y \in \mathbb{S}[\Lambda]$,

$$\text{Prob}(Y = y) = \pi(y[\Gamma]) \prod_{(i,j) \in \Lambda \setminus \Gamma} \mu(y_{i,j} | \tau_{i,j}(y, \Psi)) \tag{3.8}$$

(compare to (3.5)). Note that $\text{Prob}(Y = y)$ is a linear function of the various $\pi(z)$'s. Next, define

$$\Lambda' = \sigma_{u,v}(\mathsf{B}^{(t)}_{r-1,s}) \ , \quad \Lambda'' = \sigma_{u,v}(\mathsf{B}^{(t)}_{r,s-1}) \ ,$$

and

$$\Gamma' = \partial(\Lambda', \Psi) \ , \quad \Gamma'' = \partial(\Lambda'', \Psi) \ .$$

Consider the linear program in Figure 3.5. First, note that it is indeed a linear program. Namely, recall that by (3.8), the probability distribution of Y is a linear function of the $\pi(z)$'s. Thus, both sides of (3.9) and (3.10) are also linear functions of the $\pi(z)$'s. For example, the LHS of (3.9) equals

$$\sum_{y \in \mathbb{S}[\Lambda] \,:\, y[\Gamma'] = z'} \pi(y[\Gamma]) \prod_{(i,j) \in \Lambda \setminus \Gamma} \mu(y_{i,j} | \tau_{i,j}(y, \Psi)) \ .$$

Denote the value of the linear program when minimizing by $\mathrm{lp}^*_{\min} = \mathrm{lp}^*_{\min}(\mathcal{E})$, and when maximizing by $\mathrm{lp}^*_{\max} = \mathrm{lp}^*_{\max}(\mathcal{E})$. Since (3.5) and (3.8) are very similar, we may intuitively say that $\mathcal{E}$ outputs Y. The optimization is over the probability distribution of the boundary $Y[\Gamma]$. The linear requirements (3.9) and (3.10) are added to force the distribution of Y to be stationary. The objective function is the rate at point $(0, 0)$.

The following theorem is our main result.

Theorem 3.4 *For the linear program in Figure 3.5, we have that*

$$\mathrm{lp}^*_{\min} \le R(\mathcal{E}) \le \mathrm{lp}^*_{\max} \ .$$

In order to prove the theorem, we first state and prove a lemma, on a slightly modified linear program.

Lemma 3.5 *Fix $k > 0$, and replace (3.9) and (3.10) in Figure 3.5 by*

$$\Big| \mathrm{Prob}(Y[\Gamma'] = z') - \mathrm{Prob}(Y[\sigma_{0,1}(\Gamma')] = \sigma_{0,1}(z')) \Big| \le \frac{1}{k}$$

and

$$\Big| \mathrm{Prob}(Y[\Gamma''] = z'') - \mathrm{Prob}(Y[\sigma_{1,-t}(\Gamma'')] = \sigma_{1,-t}(z'')) \Big| \le \frac{t+1}{k} \ ,$$

respectively.

Denote the minimum and maximum of the resulting linear program as $\mathrm{lp}^{(k)}_{\min}$ and $\mathrm{lp}^{(k)}_{\max}$, respectively. Then,

$$\mathrm{lp}^{(k)}_{\min} \le R(\mathcal{E}) \le \mathrm{lp}^{(k)}_{\max} \ .$$

Proof. Consider $\mathcal{E}^{(k)}$ (as defined by (3.6)). For given M and N, define the index sets

$$\mathsf{D} = \partial(\mathsf{B}_{M,N}, \Lambda) \ , \quad \mathsf{I} = \bar{\partial}(\mathsf{B}_{M,N}, \Lambda) \ .$$

Obviously,

$$\lim_{M,N \to \infty} \frac{|\mathsf{I}|}{M \cdot N} = 1 \ . \tag{3.11}$$

Minimize (Maximize)

$$-\sum_{z \in \mathbb{S}[\Gamma]} \pi(z) \sum_{w \in \Sigma} \mu(w|z[\Psi]) \log_2 \mu(w|z[\Psi])$$

over the variables $(\pi(z) : z \in \mathbb{S}[\Gamma])$, subject to the following:

$$\sum_{z \in \mathbb{S}[\Gamma]} \pi(z) = 1 \; .$$

For all $z \in \mathbb{S}[\Gamma]$,

$$\pi(z) \geq 0 \; .$$

For all $z' \in \mathbb{S}[\Gamma']$,

$$\mathrm{Prob}(Y[\Gamma'] = z') = \mathrm{Prob}(Y[\sigma_{0,1}(\Gamma')] = \sigma_{0,1}(z')) \; . \tag{3.9}$$

For all $z'' \in \mathbb{S}[\Gamma'']$,

$$\mathrm{Prob}(Y[\Gamma''] = z'') = \mathrm{Prob}(Y[\sigma_{1,-t}(\Gamma'')] = \sigma_{1,-t}(z'')) \; . \tag{3.10}$$

Figure 3.5: Linear program. The minimum (maximum) value is denoted $\mathrm{lp}^*_{\min}$ ($\mathrm{lp}^*_{\max}$) and is a lower (upper) bound on $R(\mathcal{E})$.

Denote $A^{(k)} = A^{(k)}(\mathcal{E}, M, N)$. By (3.11) and Lemma 3.2,

$$R(\mathcal{E}^{(k)}) = \lim_{M,N \to \infty} \frac{H(A^{(k)}[\mathsf{I}]|A^{(k)}[\mathsf{D}])}{|\mathsf{I}|} \; .$$

Notice that $\Psi \subseteq \Lambda$. Thus, $\mathsf{I} \subseteq \bar{\partial}_{M,N}$, and we have

$$\begin{aligned} H(A^{(k)}[\mathsf{I}]|A^{(k)}[\mathsf{D}]) &= \sum_{(i,j) \in \mathsf{I}} H(A^{(k)}_{i,j}|A^{(k)}[\mathsf{T}_{i,j} \cap \mathsf{B}_{M,N}]) \\ &= \sum_{(i,j) \in \mathsf{I}} H(A^{(k)}_{i,j}|\tau_{i,j}(A^{(k)}, \Psi)) \; , \end{aligned}$$

where $\mathsf{T}_{i,j}$ is as defined in (3.2) and the last equality follows from (3.5).

We now prove the following claim: for all $(i,j) \in \mathsf{I}$, we have that

$$\mathrm{lp}^{(k)}_{\min} \leq H(A^{(k)}_{i,j}|\tau_{i,j}(A^{(k)}, \Psi)) \; . \tag{3.12}$$

To see this, fix some $(i, j) \in \mathsf{I}$, and define for all $z \in \mathbb{S}[\Gamma]$,

$$p^{(k)}(z) = \text{Prob}(\tau_{i,j}(A^{(k)}, \Gamma) = z) .$$

Substituting $\pi(z) = p^{(k)}(z)$, the objective function in Figure 3.5 is equal to $H(A_{i,j}^{(k)} | \tau_{i,j}(A^{(k)}, \Psi))$. Also, notice that the probability distribution of Y is equal to that of $\tau_{i,j}(A^{(k)}, \Lambda)$. By the fact that $A^{(k)}$ is quasi-stationary (and thus, so is every sub-configuration of it), all the linear requirements in the modified linear program are satisfied (i.e., the $p^{(k)}(z)$'s form a feasible solution). So, our claim (3.12) is proved.

We conclude that $\text{lp}_{\text{min}}^{(k)} \le R(\mathcal{E}^{(k)})$. Thus, by Lemma 3.3,

$$\text{lp}_{\text{min}}^{(k)} \le R(\mathcal{E}) .$$

A similar proof yields $R(\mathcal{E}) \le \text{lp}_{\text{max}}^{(k)}$. ■

Proof of Theorem 3.4. First, note that the modified linear program defined in Lemma 3.5 has at least one feasible solution, $p^{(k)}(z)$, whenever M and N are large enough so that I is non-empty.

For a given k, denote the minimizing variable values of the modified linear program by $\pi^{(k)}(z)$, $z \in \mathbb{S}[\Gamma]$. Think of these variable values as a vector

$$\boldsymbol{\pi}^{(k)} = (\pi^{(k)}(z))_{z \in \mathbb{S}[\Gamma]} .$$

By compactness, the series $\boldsymbol{\pi}^{(k)}$, $k = 1, 2, \ldots$, has a cluster point, which we denote by $\boldsymbol{\pi}^*$. Obviously, $\boldsymbol{\pi}^*$ implies a feasible solution for the linear program in Figure 3.5. More so, we must also have that the value of the objective function for this feasible solution is a lower bound on $R(\mathcal{E})$. So,

$$\text{lp}_{\text{min}}^* \le R(\mathcal{E}) .$$

Similarly, we deduce that

$$R(\mathcal{E}) \le \text{lp}_{\text{max}}^* .$$

■

Remark: While the definition of the encoder $\mathcal{E}$ includes (besides Ψ and μ) also the boundary distributions $\boldsymbol{\delta} = (\delta_{M,N})_{M,N>0}$, the bounds lp_{min}^* and lp_{max}^* do *not* depend on $\boldsymbol{\delta}$.

Applying Theorem 3.4 to our running example, with $r = 4$, $s = 5$, $t = 1$, gives

$$0.42430953 \le R(\mathcal{E}) \le 0.42442765 .$$

Table 3.1: Bounds on the rates of encoders using a small number of coins.

Constraint	Coins	$\mathrm{lp}^*_{\mathrm{min}}$	$\mathrm{lp}^*_{\mathrm{max}}$	[25]
$(2,\infty)$-RLL	1	0.440722	0.444679	0.4267
$(3,\infty)$-RLL	1	0.349086	0.386584	0.3402
n.i.b.	2	0.91773	0.919395	0.91276
$(1,\infty)$-RLL	3	0.587776	0.587785	—

To the best of our knowledge, our running example is the highest rate bit-stuffing encoder known, given that we are allowed to use at most two coins (i.e., two probability transformers). For comparison, we have calculated by the method presented in [9] that

$$\mathsf{cap}(\mathbb{S}_{\mathrm{kgs}}) \leq 0.425078 \ .$$

Namely, with two coins we achieve a rate that is only 0.2% less than capacity.

Table 3.1 contains our results for a number of constraints. We abbreviate the "no isolated bits" constraints as "n.i.b.". In the first three rows, we compare ourselves to the results in [25] (Table 1 and Equation (12)). For the comparison to be fair, we restrict ourselves to the neighbor sets Ψ used in [25], and use the same number of coins.

3.6 A lower bound on capacity

The following is a straightforward corollary of Theorem 3.4.

Corollary 3.6 *For every bit-stuffing encoder $\mathcal{E}$,*

$$\mathrm{lp}^*_{\mathrm{min}}(\mathcal{E}) \leq \mathsf{cap}(\mathbb{S}) \ .$$

Thus, we can use the minimizing linear program of Figure 3.5 to bound $\mathsf{cap}(\mathbb{S})$ from below.

To obtain better lower bounds on $\mathsf{cap}(\mathbb{S})$, we can search for good Ψ and μ. For instance, for the set $\Psi = \Psi_{\mathrm{kgs}}$ in (3.1), the function μ_{kgs} in (3.4) was obtained by maximizing $\mathrm{lp}^*_{\mathrm{min}}$ over all μ that form with Ψ_{kgs} (and every $\boldsymbol{\delta}$) a bit-stuffing encoder for $\mathbb{S}_{\mathrm{kgs}}$. Better lower bounds can be obtained by looking at larger sets Ψ (at the price of higher computational complexity).

Table 3.2: Bounds on the rates of certain bit-stuffing encoders.

Constraint	Coins	$\mathrm{lp}^*_{\mathrm{min}}$	$\mathrm{lp}^*_{\mathrm{max}}$	[38]	Then Best
$(2, \infty)$-RLL	5	**0.444202**	0.444997	0.444172	0.4423
$(3, \infty)$-RLL	2	0.359735	0.368964	0.365623	0.3641
$(0, 2)$-RLL	66	0.815497	0.816821	0.816007	0.7736
	18	0.815013	0.816176		
	9	0.810738	0.819660		
n.i.b.	56	**0.922640**	0.923748	0.920862	0.9156

Table 3.2 summarizes our results for certain constraints. The penultimate column contains the lower bounds published in [38], which to the best of our knowledge are the tightest known, excluding ours. We have highlighted values of $\mathrm{lp}^*_{\mathrm{min}}$ which are an improvement of [38]. The results in [38] were obtained independently of ours, at around the same time. Thus, we have also included the lower bounds that were the best known at the time that [38] and our method were published. These results are taken from [35], [22], [4], and [21], respectively, and appear in the last column.

Chapter 4

Concave Programming Upper Bounds on the Capacity of 2-D Constraints

In this chapter, we derive an upper bound on the capacity of 2-D constraints. Recall from Subsection 1.1.3 that the capacity of *1-D* constraints is given by the entropy of a corresponding stationary maxentropic Markov chain. Our bound is the results of extending certain aspects of this characterization to 2-D constraints.

The key steps are: The stationary maxentropic probability distribution on square configurations is considered. A set of linear equalities and inequalities is derived from this stationarity. The result is a concave program, which can be easily solved numerically.

4.1 Introduction

Let Σ be a finite alphabet, and let $(G_{\text{row}}, G_{\text{col}})$ be a pair of vertex-labeled graphs over Σ, as defined in Section 1.2. Fix $\mathbb{S} = \mathbb{S}(G_{\text{row}}, G_{\text{col}})$, and recall that

$$\mathsf{cap}(\mathbb{S}) = \lim_{M \to \infty} \frac{1}{M^2} \cdot \log_2 |\mathbb{S}_M| \ . \tag{4.1}$$

In this chapter, we show a method for calculating an upper bound on $\mathsf{cap}(\mathbb{S})$.

Recall from Subsection 1.1.3 that the capacity of a given 1-D constraint is equal to the value of an optimization program, where the optimization is on the entropy of a certain stationary Markov chain, and is carried out over the conditional probabilities of that chain. We try to extend certain aspects of this characterization of capacity to 2-D constraints. What results is a (generally non-tight) upper bound on $\mathsf{cap}(\mathbb{S})$.

The structure of this chapter is as follows. In Section 4.2, we show the existence of a certain stationary random variable taking values on $\mathbb{S}_M$ and having entropy approaching the capacity of $\mathbb{S}$, as $M \to \infty$. We then consider a relatively small sub-configuration of that random variable, and denote it by $X^{(M)}$. The section concludes with an upper bound on the capacity of $\mathbb{S}$, which is a function of the probability distribution of $X^{(M)}$. In Section 4.3, we derive a set of linear equations which hold on the probability distribution of $X^{(M)}$. In Section 4.4, we argue as follows: The bound derived in Section 4.2 is a function of the probability distribution of $X^{(M)}$, which we do not know how to calculate; however, by Section 4.3 we know that this probability distribution is subject to a set of linear requirements. Thus, we formalize an optimization problem, where the unknown probability distribution is replaced by a set of variables, subject to the above-mentioned linear requirements. The maximum of this optimization problem is an upper bound on the capacity of $\mathbb{S}$. We then show that this optimization problem is easily solved, since it is an instance of convex programming. In Section 4.5, we show our computational results. Finally, in Section 4.6 we present an asymptotic analysis of our method.

We note at this point that although this chapter deals with 2-D constraints, our method can be easily generalized to higher dimensions as well.

4.2 A preliminary upper bound on $\mathsf{cap}(\mathbb{S})$

Let M be a positive integer and let W be a random variable taking values on $\mathbb{S}_M = \mathbb{S}[\mathsf{B}_M]$. We say that W is *stationary* if for all $\mathsf{U} \subseteq \mathsf{B}_M$, all $\alpha, \beta \in \mathbb{Z}$ such that $\sigma_{\alpha,\beta}(\mathsf{U}) \subseteq \mathsf{B}_M$, and all $w' \in \mathbb{S}[\mathsf{U}]$, we have that

$$\mathrm{Prob}(W[\mathsf{U}] = w') = \mathrm{Prob}(W[\sigma_{\alpha,\beta}(\mathsf{U})] = \sigma_{\alpha,\beta}(w')) \ .$$

The following is a corollary of [8, Theorem 1.4]. The proof is given in the Appendix.

Theorem 4.1 *There exists a series of random variables* $(W^{(M)})_{M=1}^{\infty}$ *with the following properties: (i) Each* $W^{(M)}$ *takes values on* $\mathbb{S}_M$*. (ii) The* *probability distribution of $W^{(M)}$ is stationary. (iii) The normalized entropy of $W^{(M)}$ approaches* $\mathsf{cap}(\mathbb{S})$,

$$\mathsf{cap}(\mathbb{S}) = \lim_{M\to\infty} \frac{1}{M^2} \cdot H(W^{(M)}) \ . \tag{4.2}$$

We now proceed towards deriving Lemma 4.2 below, which gives an upper bound on $\mathsf{cap}(\mathbb{S})$, and makes use of the stationarity property. We note in advance that this bound is not actually meant to be calculated. Thus, its utility will be made clear in the following sections. In order to enhance the exposition, we accompany the derivation with two running examples.

A *strict total order* $\prec$ is a relation on $\mathbb{Z}^2 \times \mathbb{Z}^2$, satisfying the following conditions for all $(i_1, j_1), (i_2, j_2), (i_3, j_3) \in \mathbb{Z}^2$.

- If $(i_1, j_1) \neq (i_2, j_2)$, then either $(i_1, j_1) \prec (i_2, j_2)$ or $(i_2, j_2) \prec (i_1, j_1)$, but not both.
- If $(i_1, j_1) = (i_2, j_2)$, then neither $(i_1, j_1) \prec (i_2, j_2)$ nor $(i_2, j_2) \prec (i_1, j_1)$.
- If $(i_1, j_1) \prec (i_2, j_2)$ and $(i_2, j_2) \prec (i_3, j_3)$, then $(i_1, j_1) \prec (i_3, j_3)$.

For $(i, j) \in \mathbb{Z}^2$, define $\mathsf{T}^{(\prec)}_{i,j}$ as the set of all the indexes preceding (i, j). Namely,

$$\mathsf{T}^{(\prec)}_{i,j} = \left\{ (i', j') \in \mathbb{Z}^2 : (i', j') \prec (i, j) \right\} \ .$$

Running Example I: Define the lexicographic order $\prec_{\text{lex}}$ as in Chapter 3. Namely, we have that $(i_1, j_1) \prec_{\text{lex}} (i_2, j_2)$ iff

- $i_1 < i_2$, or
- $(i_1 = i_2$ and $j_1 < j_2)$

Running Example II: Define the "interleaved raster scan" order $\prec_{\text{irs}}$ as follows. We have that $(i_1, j_1) \prec_{\text{irs}} (i_2, j_2)$ iff

- $i_1 \equiv 0 \pmod 2$ and $i_2 \equiv 1 \pmod 2$, or
- $i_1 \equiv i_2 \pmod 2$ and $i_1 < i_2$, or
- $i_1 = i_2$ and $j_1 < j_2$.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

$\prec_{\text{lex}}$

1	2	3	4	5
16	17	18	19	20
6	7	8	9	10
21	22	23	24	25
11	12	13	14	15

$\prec_{\text{irs}}$

Figure 4.1: An entry labeled i in the left (right) configuration precedes an entry labeled j according to $\prec_{\text{lex}}$ ($\prec_{\text{irs}}$) iff $i < j$.

(See Figure 4.1 for both examples.)

For the rest of this section, fix positive integers r and s, and define the index set

$$\Lambda = \mathsf{B}_{r,s} \; .$$

We will refer to Λ as "the patch." The bound we derive in Lemma 4.2 will be a function of the following:

- the strict total order $\prec$,
- the integers r and s, which determine the order $r \times s$ of the patch Λ,
- an integer c, which will denote the number of "colors" we encounter,
- a coloring function $f : \mathbb{Z}^2 \to \{1, 2, \ldots, c\}$, mapping each point in $\mathbb{Z}^2$ to one of c colors,
- c indexes, $(a_\gamma, b_\gamma)_{\gamma=1}^{c}$, such that for all $1 \le \gamma \le c$,

  $$(a_\gamma, b_\gamma) \in \Lambda$$

  (namely, each color γ has a designated point in the patch, *which may or may not be of color* γ).

The function f must satisfy two requirements, which we now elaborate on. Our first requirement is: for all $1 \le \gamma \le c$,

$$\lim_{M \to \infty} \frac{\{(i,j) \in \mathsf{B}_M : f(i,j) = \gamma\}}{M^2} = \frac{1}{c} \; . \tag{4.3}$$

Namely, as the orders of $W^{(M)}$ tend to infinity, each color is equally[1] likely. Our second requirement is: there exist index sets $\Psi_1, \Psi_2, \ldots, \Psi_c \subseteq \Lambda$ such that for all indexes $(i,j) \in \mathbb{Z}^2$,

$$\sigma_{i',j'}(\Psi_\gamma) = \mathsf{T}_{i,j}^{(\prec)} \cap \sigma_{i',j'}(\Lambda) \ , \tag{4.4}$$

where $\gamma = f(i,j)$, $i' = a_\gamma - i$, and $j' = b_\gamma - j$. Namely, let (i,j) be such that $f(i,j) = \gamma$, and shift Λ such that (a_γ, b_γ) is shifted to (i,j). Now, consider the set of all indexes in the shifted Λ which precede (i,j): this set must be equal to the correspondingly shifted Ψ_γ.

Running Example I: Take $r = 4$ and $s = 7$ as the patch orders. Let the number of colors be $c = 1$. Thus, we must define $f = f_{\text{lex}}$ as follows: for all $(i,j) \in \mathbb{Z}^2$, $f_{\text{lex}}(i,j) = 1$. Take the point corresponding to the single color as $(a_1 = 3, b_1 = 5)$. See also Figure 4.2(a).

Running Example II: As in the previous example, take $r = 3$ and $s = 5$ as the patch orders. Let the number of colors be $c = 2$. Define $f = f_{\text{irs}}$ as follows:

$$f_{\text{irs}}(i,j) = \begin{cases} 1 & i \equiv 0 \pmod 2 \\ 2 & i \equiv 1 \pmod 2 \end{cases} .$$

Take $(a_1 = 3, b_1 = 5)$ and $(a_2 = 2, b_2 = 4)$. See also Figure 4.2(b).

Lemma 4.2 *Let $(W^{(M)})_{M=1}^{\infty}$ be as in Theorem 4.1 and define*

$$X^{(M)} = W^{(M)}[\Lambda] \ .$$

Let $\prec$, r, s, c, f, $(\Psi_\gamma)_{\gamma=1}^c$, and $(a_\gamma, b_\gamma)_{\gamma=1}^c$ be given. For $1 \le \gamma \le c$, define

$$\Upsilon_\gamma = \{(a_\gamma, b_\gamma)\} \cup \Psi_\gamma \ .$$

Let

$$Y_\gamma = X^{(M)}[\Upsilon_\gamma] \ \text{ and } \ Z_\gamma = X^{(M)}[\Psi_\gamma]$$

(note that Y_γ and Z_γ are functions of M). Then,

$$\mathsf{cap}(\mathbb{S}) \le \limsup_{M \to \infty} \frac{1}{c} \sum_{\gamma=1}^{c} H(Y_\gamma | Z_\gamma) \ .$$

[1]In fact, it is possible to generalize (4.3), and require only that the limit exists for all γ. We have not found this generalization useful.

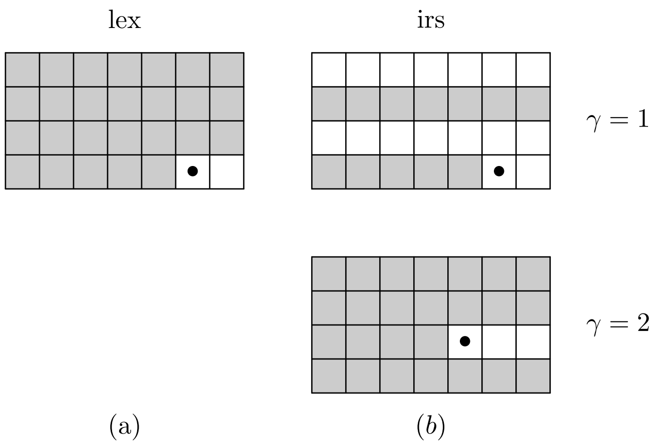


Figure 4.2: The left (right) column corresponds to Running Example I (II). The configurations are of order $r \times s$ and represent the index set Λ. The $\bullet$ symbol is in position (a_γ, b_γ). The shaded part is Ψ_γ.

Proof. Let X, W and $\mathsf{T}_{i,j}$ be shorthand for $X^{(M)}$, $W^{(M)}$ and $\mathsf{T}_{i,j}^{(\prec)}$, respectively. First note that

$$Y_\gamma = W[\Upsilon_\gamma] \text{ and } Z_\gamma = W[\Psi_\gamma] \; .$$

We show that

$$\lim_{M \to \infty} \frac{1}{M^2} H(W) \le \limsup_{M \to \infty} \frac{1}{c} \sum_{\gamma=1}^{c} H(Y_\gamma | Z_\gamma) \; .$$

Once this is proved, the claim follows from (4.2).

By the chain rule (Theorem 1.4), we have

$$H(W) = \sum_{(i,j) \in \mathsf{B}_M} H(W_{i,j} | W[\mathsf{T}_{i,j} \cap \mathsf{B}_M]) \; .$$

We now recall (4.4) and define the index set $\bar{\partial}$ to be the largest subset of B_M for which the following condition holds: for all $(i,j) \in \bar{\partial}$, we have that

$$\sigma_{i',j'}(\Psi_\gamma) \subseteq \mathsf{B}_M \; , \tag{4.5}$$

where hereafter in the proof, $\gamma = f(i,j)$, $i' = a_\gamma - i$, and $j' = b_\gamma - j$. Define $\partial = \mathsf{B}_M \setminus \bar{\partial}$. Note that since r and s are constant, and $\Psi_1, \Psi_2, \ldots, \Psi_c \subseteq \Lambda$,

then
$$\frac{|\partial|}{M^2} = O(1/M) \ .$$
Thus, on the one hand, we have
$$\frac{1}{M^2} \sum_{(i,j)\in\partial} H(W_{i,j}|W[\mathsf{T}_{i,j} \cap \mathsf{B}_M]) \le \log_2|\Sigma| \cdot O(1/M) \ .$$
On the other hand, from (4.4) and (4.5) we have that for all $(i,j) \in \bar{\partial}$,
$$\sigma_{i',j'}(\Psi_\gamma) \subseteq \mathsf{T}_{i,j} \cap \mathsf{B}_M \ .$$
Hence, since conditioning reduces entropy (Theorem 1.6),
$$\begin{aligned}
&\frac{1}{M^2} \sum_{(i,j)\in\bar{\partial}} H(W_{i,j}|W[\mathsf{T}_{i,j} \cap \mathsf{B}_M]) \\
\le &\frac{1}{M^2} \sum_{(i,j)\in\bar{\partial}} H(W_{i,j}|W[\sigma_{i',j'}(\Psi_\gamma)]) \\
= &\frac{1}{M^2} \sum_{(i,j)\in\bar{\partial}} H(W[\{(i,j)\} \cup \sigma_{i',j'}(\Psi_\gamma)]|W[\sigma_{i',j'}(\Psi_\gamma)]) \\
= &\frac{1}{M^2} \sum_{(i,j)\in\bar{\partial}} H(Y_\gamma|Z_\gamma) \ ,
\end{aligned}$$
where the last step follows from the stationarity of $W^{(M)}$. Recalling (4.3), the proof follows. ■

The following is a simple corollary of Lemma 4.2.

Corollary 4.3 *Let $(W^{(M)})_{M=1}^{\infty}$ be as in Theorem 4.1 and define*
$$X^{(M)} = W^{(M)}[\Lambda] \ .$$
Fix positive integers r and s. Let ℓ be a positive integer, and let $(\rho^{\langle k\rangle})_{k=1}^{\ell}$ be non-negative reals such that $\sum_{k=1}^{\ell} \rho^{\langle k\rangle} = 1$. For every $1 \le k \le \ell$, let $\prec^{\langle k\rangle}$, $c^{\langle k\rangle}$, $f^{\langle k\rangle}$, $(\Psi_\gamma^{\langle k\rangle})_{\gamma=1}^{c}$, and $(a_\gamma^{\langle k\rangle}, b_\gamma^{\langle k\rangle})_{\gamma=1}^{c}$ be given. Also, for $1 \le \gamma \le c^{\langle k\rangle}$, let
$$\Upsilon_\gamma^{\langle k\rangle} = \{(a_\gamma^{\langle k\rangle}, b_\gamma^{\langle k\rangle})\} \cup \Psi_\gamma^{\langle k\rangle} \ .$$
Define
$$Y_\gamma^{\langle k\rangle} = X^{(M)}[\Upsilon_\gamma^{\langle k\rangle}] \ \text{ and } \ Z_\gamma^{\langle k\rangle} = X^{(M)}[\Psi_\gamma^{\langle k\rangle}]$$

(note that $Y_\gamma^{\langle k\rangle}$ and $Z_\gamma^{\langle k\rangle}$ are functions of M). Then,

$$\mathsf{cap}(\mathbb{S}) \le \limsup_{M\to\infty} \sum_{k=1}^{\ell} \frac{\rho^{\langle k\rangle}}{c^{\langle k\rangle}} \sum_{\gamma=1}^{c^{\langle k\rangle}} H(Y_\gamma^{\langle k\rangle} | Z_\gamma^{\langle k\rangle}) \; .$$

Corollary 4.3 is the most general way we have found to state our results. This generality will indeed help us later on. However, almost none of the intuition is lost if the reader has in mind the much simpler case of

$$\begin{aligned} &\ell = 1 \; , \quad \rho^{\langle 1\rangle} = 1 \; , \quad c^{\langle 1\rangle} = 1 \; , \quad \prec^{\langle 1\rangle} = \prec_{\text{lex}} \; , \\ &\qquad (a_1^{\langle 1\rangle}, b_1^{\langle 1\rangle}) = (r-1, t) \; , \quad \text{and } \; \Psi_1^{\langle 1\rangle} = \Lambda \cap \mathsf{T}_{(a_1^{\langle 1\rangle}, b_1^{\langle 1\rangle})} \; , \end{aligned} \tag{4.6}$$

where $0 \le t < s$. This simpler case was demonstrated in Running Example I.

4.3 Linear requirements

Recall that $X^{(M)} = W^{(M)}[\Lambda]$ is an $r \times s$ sub-configuration of $W^{(M)}$, and thus stationary as well. In this section, we formulate a set of linear requirements (equalities and inequalities) on the probability distribution of $X^{(M)}$. For the rest of this section, let M be fixed and let X be shorthand for $X^{(M)}$.

4.3.1 Linear requirements from stationarity

In this subsection, we formulate a set of linear requirements that follow from the stationarity of $X^{(M)}$. Let $x \in \mathbb{S}[\Lambda]$ be a realization of X. Denote

$$p_x = \text{Prob}(X = x) \; .$$

We start with the trivial requirements. Obviously, we must have for all $x \in \mathbb{S}[\Lambda]$ that

$$p_x \ge 0 \; .$$

Also,

$$\sum_{x \in \mathbb{S}[\Lambda]} p_x = 1 \; .$$

Next, we show how we can use stationarity to get more linear equations on $(p_x)_{x\in\mathbb{S}[\Lambda]}$. Let

$$\Lambda' = \{(i,j) : 0 \le i < r-1 \; , \; 0 \le j < s\} \; .$$

For $x' \in \mathbb{S}[\Lambda']$ we must have by stationarity that

$$\mathrm{Prob}(X[\Lambda'] = x') = \mathrm{Prob}(X[\sigma_{1,0}(\Lambda')] = \sigma_{1,0}(x')) \ . \tag{4.7}$$

As a concrete example, suppose that $r = s = 3$. We claim that

$$\mathrm{Prob}\left(X = \begin{matrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ * & * & * \end{matrix} \right) = \mathrm{Prob}\left(X = \begin{matrix} * & * & * \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{matrix} \right) \ ,$$

where $*$ denotes "don't care".

Both the left-hand and right-hand sides of (4.7) are marginalizations of $(p_x)_x$. Thus, we get a set of linear equations on $(p_x)_x$, namely, for all $x' \in \mathbb{S}[\Lambda']$,

$$\sum_{x\,:\,x[\Lambda']=x'} p_x = \sum_{x\,:\,x[\sigma_{1,0}(\Lambda')]=\sigma_{1,0}(x')} p_x \ .$$

To get more equations, we now apply the same rational horizontally, instead of vertically. Let

$$\Lambda'' = \{(i,j) : 0 \le i < r \ , \ \ 0 \le j < s-1\} \ .$$

for all $x'' \in \mathbb{S}[\Lambda'']$,

$$\sum_{x\,:\,x[\Lambda'']=x''} p_x = \sum_{x\,:\,x[\sigma_{0,1}(\Lambda'')]=\sigma_{0,1}(x'')} p_x \ .$$

4.3.2 Linear equations from reflection, transposition, and complementation

We now show that if $\mathbb{S}$ is reflection, transposition, or complementation invariant (defined below), then we can derive yet more linear equations.

Define $v_M(\cdot)$ ($h_M(\cdot)$) as the vertical (horizontal) reflection of a rectangular configuration with M rows (columns). Namely,

$$(v_M(w))_{i,j} = w_{M-1-i,j} \ , \quad \text{and} \quad (h_M(w))_{i,j} = w_{i,M-1-j} \ .$$

Define τ as the transposition of a configuration. Namely,

$$\tau(w)_{i,j} = w_{j,i} \ .$$

For $\Sigma = \{0,1\}$, denote by $\mathrm{comp}(w)$ the bitwise complement of a configuration w. Namely,

$$\mathrm{comp}(w)_{i,j} = \begin{cases} 1 & \text{if } w_{i,j} = 0 \\ 0 & \text{otherwise} . \end{cases}$$

We state three similar lemmas, and prove the first. The proof of the other two is similar.

Lemma 4.4 *Suppose that $\mathbb{S}$ is such that for all $M > 0$ and $w \in \Sigma^{M\times M}$,*

$$w \in \mathbb{S} \iff h_M(w) \in \mathbb{S} \iff v_M(w) \in \mathbb{S} .$$

Then, w.l.o.g., the probability distribution of W is such that for all $w \in \mathbb{S}_M$,

$$\mathrm{Prob}(W = w) = \mathrm{Prob}(W = h_M(w)) = \mathrm{Prob}(W = v_M(w)) . \tag{4.8}$$

Lemma 4.5 *Suppose that $\mathbb{S}$ is such that for all $M > 0$ and $w \in \Sigma^{M\times M}$,*

$$w \in \mathbb{S} \iff \tau(w) \in \mathbb{S} .$$

Then, w.l.o.g., W is such that for all $w \in \mathbb{S}_M$,

$$\mathrm{Prob}(W = w) = \mathrm{Prob}(W = \tau(w)) . \tag{4.9}$$

Lemma 4.6 *Suppose that $\Sigma = \{0,1\}$ and $\mathbb{S}$ is such that for all $M > 0$ and $w \in \Sigma^{M\times M}$,*

$$w \in \mathbb{S} \iff \mathrm{comp}(w) \in \mathbb{S} .$$

Then, w.l.o.g., W is such that for all $w \in \mathbb{S}_M$,

$$\mathrm{Prob}(W = w) = \mathrm{Prob}(W = \mathrm{comp}(w)) . \tag{4.10}$$

Proof of Lemma 4.5. Let h and v be shorthand for h_M and v_M, respectively. For M fixed, we define a new random variable W^{new} taking values on $\mathbb{S}_M$, with the following distribution: for all $w \in \mathbb{S}_M$,

$$\mathrm{Prob}(W^{\mathrm{new}}{=}w) = \frac{1}{4} \sum_{\substack{w' \in \\ \{w,h(w),v(w),h(v(w))\}}} \mathrm{Prob}(W{=}w') .$$

Since $h(h(w)) = v(v(w)) = w$ and $h(v(w)) = v(h(w))$ we get that (4.8) holds for W^{new}. Moreover, by the concavity of the entropy function,

$$H(W) \leq H(W^{\text{new}}) \; .$$

Thus, the properties defined in Theorem 4.1 hold for W^{new}. ■

If the condition of Lemma 4.4 holds, then we get the following equations by stationarity. For all $x \in \mathbb{S}[\Lambda]$,

$$p_x = p_{v_r(x)} = p_{h_s(x)} \; .$$

If the condition of Lemma 4.5 holds, then the following holds by stationarity. Assume w.l.o.g. that $r \leq s$, and let

$$\tilde{\Lambda} = \{(i,j) : 0 \leq i, j < r\} \; .$$

For all $\chi \in \mathbb{S}[\tilde{\Lambda}]$,

$$\sum_{x \,:\, x[\tilde{\Lambda}]=\chi} p_x = \sum_{x \,:\, x[\tilde{\Lambda}]=\tau(\chi)} p_x \; .$$

If the condition of Lemma 4.6 holds, then we get the following equations by stationarity. For all $x \in \mathbb{S}[\Lambda]$,

$$p_x = p_{\mathrm{comp}(x)} \; .$$

4.4 An upper bound on $\mathsf{cap}(\mathbb{S})$

For the rest of this section, let r, s, ℓ, $\rho^{\langle k \rangle}$, $\prec^{\langle k \rangle}$, $c^{\langle k \rangle}$, $f^{\langle k \rangle}$, $\Psi_\gamma^{\langle k \rangle}$, and $(a_\gamma^{\langle k \rangle}, b_\gamma^{\langle k \rangle})$ be given as in Corollary 4.3. Recall from Corollary 4.3 that we are interested in $H(Y_\gamma^{\langle k \rangle} | Z_\gamma^{\langle k \rangle})$, in order to bound $\mathsf{cap}(\mathbb{S})$ from above.

As a first step, we fix M and express $H(Y_\gamma^{\langle k \rangle} | Z_\gamma^{\langle k \rangle})$ in terms of the probabilities $(p_x)_x$ of the random variable $X^{(M)}$. For given $1 \leq k \leq \ell$ and $1 \leq \gamma \leq c^{\langle k \rangle}$, let

$$y \in \mathbb{S}[\Upsilon_\gamma^{\langle k \rangle}] \;\text{ and }\; z \in \mathbb{S}[\Psi_\gamma^{\langle k \rangle}]$$

be realizations of $Y_\gamma^{\langle k \rangle}$ and $Z_\gamma^{\langle k \rangle}$, respectively. Let

$$p_{\gamma,y}^{\langle k \rangle} = \mathrm{Prob}(Y_\gamma^{\langle k \rangle} = y) \quad \text{and} \quad p_{\gamma,z}^{\langle k \rangle} = \mathrm{Prob}(Z_\gamma^{\langle k \rangle} = z)$$

($p^{\langle k \rangle}_{\gamma,y}$ and $p^{\langle k \rangle}_{\gamma,z}$ are functions of M). From here onward, let p_y and p_z be shorthand for $p^{\langle k \rangle}_{\gamma,y}$ and $p^{\langle k \rangle}_{\gamma,z}$, respectively. Both p_y and p_z are marginalizations of $(p_x)_x$, namely,

$$p_y = \sum_{x \in \mathbb{S}[\Lambda] \,:\, x[\Upsilon^{\langle k \rangle}_\gamma] = y} p_x \ , \quad p_z = \sum_{x \in \mathbb{S}[\Lambda] \,:\, x[\Psi^{\langle k \rangle}_\gamma] = z} p_x \ .$$

Thus, for given γ and k,

$$H(Y^{\langle k \rangle}_\gamma | Z^{\langle k \rangle}_\gamma) = \sum_{y \in \mathbb{S}[\Upsilon^{\langle k \rangle}_\gamma]} -p_y \log_2 p_y + \sum_{z \in \mathbb{S}[\Psi^{\langle k \rangle}_\gamma]} p_z \log_2 p_z$$

is a function of the probabilities $(p_x)_x$ of $X^{(M)}$.

Our next step will be to reason as follows: We have found linear requirements that the p_x's satisfy and expressed $H(Y^{\langle k \rangle}_\gamma | Z^{\langle k \rangle}_\gamma)$ as a function of $(p_x)_x$. However, *we do not know of a way to actually calculate* $(p_x)_x$. So, instead of the probabilities $(p_x)_x$, consider the *variables* $(\bar{p}_x)_x$. From this line of thought we get our main theorem.

Theorem 4.7 *The value of the optimization program given in Figure 4.3 is an upper bound on* $\mathsf{cap}(\mathbb{S})$.

Proof. First, notice that if we take $\bar{p}_x = p_x$, then (by Section 4.3) all the requirements which the $\bar{p}_x$'s are subject to indeed hold, and the objective function is equal to

$$\sum_{k=1}^{\ell} \frac{\rho^{\langle k \rangle}}{c^{\langle k \rangle}} \sum_{\gamma=1}^{c^{\langle k \rangle}} H(Y^{\langle k \rangle}_\gamma | Z^{\langle k \rangle}_\gamma) \ .$$

So, the maximum is an upper bound on the above equation. Next, by compactness, a maximum indeed exists. Since the maximum is not a function of M, the claim now follows from Corollary 4.3. ■

We now proceed to show that the optimization problem in Figure 4.3 is an instance of concave programming [7, p. 137], and thus easily calculated. Since the requirements that the variables $(\bar{p}_x)_x$ are subject to are linear, this reduces to showing that the objective function is concave in $(\bar{p}_x)_x$.

Lemma 4.8 *The objective function in Figure 4.3 is concave in the variables* $(\bar{p}_x)_{x \in \mathbb{S}[\Lambda]}$*, subject to them being non-negative.*

maximize

$$\sum_{k=1}^{\ell} \frac{\rho^{\langle k \rangle}}{c^{\langle k \rangle}} \sum_{\gamma=1}^{c^{\langle k \rangle}} \Xi(k,\gamma)$$

over the variables $(\bar{p}_x)_{x \in \mathbb{S}[\Lambda]}$, where for

$$1 \le k \le \ell \; , \quad 1 \le \gamma \le c^{\langle k \rangle} \; , \quad y \in \mathbb{S}[\Upsilon_\gamma^{\langle k \rangle}] \; , \quad z \in \mathbb{S}[\Psi_\gamma^{\langle k \rangle}] \; ,$$

we define

$$\bar{p}_{\gamma,y}^{\langle k \rangle} \triangleq \sum_{x \in \mathbb{S}[\Lambda] \,:\, x[\Upsilon_\gamma^{\langle k \rangle}]=y} \bar{p}_x \; , \quad \bar{p}_{\gamma,z}^{\langle k \rangle} \triangleq \sum_{x \in \mathbb{S}[\Lambda] \,:\, x[\Psi_\gamma^{\langle k \rangle}]=z} \bar{p}_x \; ,$$

$$\Xi(k,\gamma) \triangleq -\sum_{y \in \mathbb{S}[\Upsilon_\gamma^{\langle k \rangle}]} \bar{p}_{\gamma,y}^{\langle k \rangle} \log_2 \bar{p}_{\gamma,y}^{\langle k \rangle} + \sum_{z \in \mathbb{S}[\Psi_\gamma^{\langle k \rangle}]} \bar{p}_{\gamma,z}^{\langle k \rangle} \log_2 \bar{p}_{\gamma,z}^{\langle k \rangle} \; ,$$

and the variables $\bar{p}_x$ are subject to the following requirements:

(i)

$$\sum_{x \in \mathbb{S}[\Lambda]} \bar{p}_x = 1 \; .$$

(ii) For all $x \in \mathbb{S}[\Lambda]$,

$$\bar{p}_x \ge 0 \; .$$

(iii) For all $x' \in \mathbb{S}[\Lambda']$,

$$\sum_{x \,:\, x[\Lambda']=x'} \bar{p}_x = \sum_{x \,:\, x[\sigma_{1,0}(\Lambda')]=\sigma_{1,0}(x')} \bar{p}_x \; .$$

(iv) For all $x'' \in \mathbb{S}[\Lambda'']$,

$$\sum_{x \,:\, x[\Lambda'']=x''} \bar{p}_x = \sum_{x \,:\, x[\sigma_{0,1}(\Lambda'')]=\sigma_{0,1}(x'')} \bar{p}_x \; .$$

(v) (If $\mathbb{S}$ is reflection (resp. complementation) invariant) For all $x \in \mathbb{S}[\Lambda]$,

$$\bar{p}_x = \bar{p}_{v_r(x)} = \bar{p}_{h_s(x)} \quad (\text{resp. } \bar{p}_x = \bar{p}_{\mathrm{comp}(x)}) \; .$$

(vi) (If $\mathbb{S}$ is transposition invariant) For all $\chi \in \mathbb{S}[\tilde{\Lambda}]$,

$$\sum_{x \,:\, x[\tilde{\Lambda}]=\chi} \bar{p}_x = \sum_{x \,:\, x[\tilde{\Lambda}]=\tau(\chi)} \bar{p}_x \; .$$

Figure 4.3: Optimization program over the variables $\bar{p}_x$ (assuming w.l.o.g. that $r \le s$). The optimum is an upper bound on $\mathsf{cap}(\mathbb{S})$.

Proof. Recall that for all $1 \le k \le \ell$ we have that $\frac{\rho^{\langle k \rangle}}{c^{\langle k \rangle}}$ is non-negative. Thus, it suffices to prove that for all $1 \le k \le \ell$ and $1 \le \gamma \le c^{\langle k \rangle}$, the function $\Xi(k, \gamma)$ is concave in the variables $(\bar{p}_x)_x$. So, let k and γ be fixed, and let $\bar{p}_y$ and $\bar{p}_z$ be shorthand for $\bar{p}^{\langle k \rangle}_{\gamma,y}$ and $\bar{p}^{\langle k \rangle}_{\gamma,z}$, respectively.

Recalling the definition of $\bar{p}^{\langle k \rangle}_{\gamma,y}$ and $\bar{p}^{\langle k \rangle}_{\gamma,z}$ in Figure 4.3 and the fact that $\Psi^{\langle k \rangle}_{\gamma} \subseteq \Upsilon^{\langle k \rangle}_{\gamma}$, we get that

$$\Xi(k, \gamma) = \sum_{\substack{y \in \mathbb{S}[\Upsilon^{\langle k \rangle}_{\gamma}] \\ z = y[\Psi^{\langle k \rangle}_{\gamma}]}} -\bar{p}_y \log_2 \frac{\bar{p}_y}{\bar{p}_z} \; .$$

Thus, it suffices to show that each summand is concave in $(\bar{p}_x)_x$. This is indeed the case: let $(\bar{p}^{(1)}_x)_{x \in \mathbb{S}[\Lambda]}$ and $(\bar{p}^{(2)}_x)_{x \in \mathbb{S}[\Lambda]}$ be non-negative. Let $0 \le \xi \le 1$ be given, and define $(\bar{p}^{(3)}_x)_{x \in \mathbb{S}[\Lambda]}$ as

$$\bar{p}^{(3)}_x = \xi \bar{p}^{(1)}_x + (1 - \xi)\bar{p}^{(2)}_x \; , \quad x \in \mathbb{S}[\Lambda] \; .$$

For $t = 1, 2, 3$, denote by $\bar{p}^{(t)}_y$ and $\bar{p}^{(t)}_z$ the marginalizations corresponding to $(\bar{p}^{(t)}_x)_x$. Obviously,

$$\bar{p}^{(3)}_y = \xi \bar{p}^{(1)}_y + (1 - \xi)\bar{p}^{(2)}_y \; , \quad y \in \mathbb{S}[\Upsilon^{\langle k \rangle}_{\gamma}] \; .$$

and

$$\bar{p}^{(3)}_z = \xi \bar{p}^{(1)}_z + (1 - \xi)\bar{p}^{(2)}_z \; , \quad z \in \mathbb{S}[\Psi^{\langle k \rangle}_{\gamma}] \; .$$

We must show that for all $y \in \mathbb{S}[\Upsilon^{\langle k \rangle}_{\gamma}]$, $z = y[\Psi^{\langle k \rangle}_{\gamma}]$

$$\bar{p}^{(3)}_y \log_2 \frac{\bar{p}^{(3)}_y}{\bar{p}^{(3)}_z} \le \xi \bar{p}^{(1)}_y \log_2 \frac{\bar{p}^{(1)}_y}{\bar{p}^{(1)}_z} + (1 - \xi)\bar{p}^{(2)}_y \log_2 \frac{\bar{p}^{(2)}_y}{\bar{p}^{(2)}_z} \; .$$

This is indeed the case, by the log sum inequality [13, p. 29]. ■

4.5 Computational results

At this point, we have formulated a concave optimization problem, and wish to solve it. There are quite a few programs, termed *solvers*, that enable one to do so. Many such solvers — most of them proprietary — are hosted on the servers of the NEOS project [14][24][16], and the public may submit moderately sized optimization problems to them. We have coded our optimization problems in the AMPL modeling language [23], and submitted them to NEOS.

Essentially, a solver starts with some initial guess as to the optimizing value of $(\bar{p}_x)_{x \in \mathbb{S}[\Lambda]}$, and then iteratively improves the value of the objective function. This process is terminated when the solver decides that it is "close enough" to the optimum. Denote by $\widetilde{\boldsymbol{p}} = (\widetilde{p}_x)_{x \in \mathbb{S}[\Lambda]}$ this "close enough" assignment to the variables. Of course, we must supply an upper bound on $\mathsf{cap}(\mathbb{S})$, not an approximation to one. Thus, let $\widetilde{f}$ and

$$\widetilde{\boldsymbol{g}} = (\widetilde{g}_x)_x \ , \quad x \in \mathbb{S}[\Lambda] \ ,$$

be the value of the objective function and its gradient at $\widetilde{\boldsymbol{p}}$, respectively. Obviously, $\widetilde{f}$ is a lower bound on the value of our optimization problem. For an upper bound, we replace the objective function in Figure 4.3 by

$$\text{maximize} \quad \left(\widetilde{f} + \sum_{x \in \mathbb{S}[\Lambda]} \widetilde{g}_x \cdot (\bar{p}_x - \widetilde{p}_x) \right) \ ,$$

and get a *linear program* (the value of which can be calculated exactly). By concavity, the value of this linear program is indeed an upper bound. So, we use NEOS yet again to solve it. For the sake of double-checking, we submitted the above optimization problems to two solvers: IPOPT [44] and MOSEK.

Before stating our computational results, let us first define one more strict total order, which we have termed the "skip" order, $\prec_{\text{skip}}$ (see Figure 4.4). We have that $(i_1, j_1) \prec_{\text{skip}} (i_2, j_2)$ iff

- $i_1 < i_2$, or
- $(i_1 = i_2$ and $j_1 \equiv 0 \pmod 2$ and $j_2 \equiv 1 \pmod 2)$, or
- $(i_1 = i_2$ and $j_1 \equiv j_2 \pmod 2$ and $j_1 < j_2)$

Our computational results appear in Table 4.1. To the best of our knowledge, they are presently the tightest. We compare our results to those obtained by the method described in [20]. When available, these compared-to bounds are taken from previously published work, as indicated to the left of them. The rest are the result of our implementation of [20]. We note that

1	5	2	6	3	7	4
8	12	9	13	10	14	11
15	19	16	20	17	21	18

$\prec_{\text{skip}}$

Figure 4.4: An entry labeled i in the configuration precedes an entry labeled j according to $\prec_{\text{skip}}$ iff $i < j$.

Table 4.1: Upper-bounds on the capacity of some 2-D constraints.

Constraint	r	s	k	$\prec$ used	Upper bound	Comparison
$(2, \infty)$-RLL	3	8	7	$\prec_{\text{lex}}$, $\prec_{\text{skip}}$	0.4457	0.4459 [22]
$(3, \infty)$-RLL	4	8	5	$\prec_{\text{lex}}$	0.36821	0.3686 [22]
$(0, 2)$-RLL	3	5	2	$\prec_{\text{lex}}$	0.816731	0.817053
n.i.b.	3	4	1	$\prec_{\text{skip}}$	0.92472	0.927855

the indexes $(a_\gamma^{\langle k \rangle}, b_\gamma^{\langle k \rangle})$ and coefficients $\rho^{\langle k \rangle}$ used for each constraint were optimized by hand, through trial and error. Also, we note that when applying our method to the 2-D $(1, \infty)$-RLL constraint, our bound was inferior to the one presented in [45] (utilizing the method of [9]). Finally, recall that Table 3.2 contains corresponding lower bounds.

4.6 Asymptotic analysis

For a given constraint $\mathbb{S}$ and positive integers r and s, let t be an integer such that $0 \leq t < s$. Denote by $\mu(r, s, t)$ the value of the optimization program in Figure 4.3, where the parameters are as in (4.6). In this section, we show that even if we restrict ourselves to this simple case, we get an upper bound which is asymptotically tight, in the following sense.

Theorem 4.9 *For all $\epsilon > 0$, there exist*

$$r_0 > 0\,, \quad s_0 > 0\,, \quad 0 \leq t_0 < s_0$$

such that for all

$$r \geq r_0 \ , \quad s \geq s_0 \ , \quad t_0 \leq t \leq s - (s_0 - t_0) \ ,$$

we have that

$$\mu(r, s, t) - \mathsf{cap}(\mathbb{S}) \leq \epsilon \ .$$

In order to prove Theorem 4.9, we need the following lemma.

Lemma 4.10 *For all $\epsilon > 0$, there exist*

$$r_0 > 0 \ , \quad s_0 > 0 \ , \quad 0 \leq t_0 < s_0$$

such that

$$\mu(r_0, s_0, t_0) - \mathsf{cap}(\mathbb{S}) < \epsilon \ .$$

Proof. Another well known method for bounding $\mathsf{cap}(\mathbb{S})$ from above is the so called "stripe method." Namely, for some given θ, consider the 1-D constraint $S = S(\theta)$ defined as follows. The alphabet of the constraint is Σ^θ. A word of length r' satisfies S if and only if when we write its entries as rows of length θ, one below the other, we get an $r' \times \theta$ configuration which satisfies the 2-D constraint $\mathbb{S}$.

Define the normalized capacity of S as

$$\widehat{\mathsf{cap}}(S) = \frac{1}{\theta}\mathsf{cap}(S) \ .$$

By the definition of $\mathsf{cap}(\mathbb{S})$, the normalized capacity approaches $\mathsf{cap}(\mathbb{S})$ as $\theta \to \infty$. Thus, fix a θ such that

$$\widehat{\mathsf{cap}}(S) - \mathsf{cap}(\mathbb{S}) \leq \epsilon/2 \ .$$

We say that a 1-D constraint has memory m if there exists a graph representing it, and all paths in the graph of length m with the same labeling terminate in the same vertex. By [31, Theorem 3.17] and its proof, there exists a series of 1-D constraints $\{S_\mathsf{m}\}_{\mathsf{m}=1}^{\infty}$ such that $S \subseteq S_\mathsf{m}$, the memory of S_m is m, and $\lim_{\mathsf{m}\to\infty} \mathsf{cap}(S_\mathsf{m}) = \mathsf{cap}(S)$. Thus, fix m such that

$$\widehat{\mathsf{cap}}(S_\mathsf{m}) - \widehat{\mathsf{cap}}(S) \leq \epsilon/2 \ .$$

To finish the proof, we now show that

$$\mu(r_0, s_0, t_0) \leq \widehat{\mathsf{cap}}(S_{\mathsf{m}}) \ ,$$

where

$$r_0 = \mathsf{m}+1 \ , \quad s_0 = 2 \cdot \theta \ , \quad t_0 = \theta - 1 \ .$$

Note that $\mu(r_0, s_0, t_0)$ is the maximum of

$$H(\bar{X}_{\mathsf{m},\theta-1} | \bar{X}[\mathsf{T}^{(\prec_{\mathrm{lex}})}_{\mathsf{m},\theta-1} \cap \mathsf{B}_{\mathsf{m}+1,2\cdot\theta}]) \ , \tag{4.11}$$

over all random variables $\bar{X} \in \mathbb{S}_{\mathsf{m}+1,2\cdot\theta}$ with a probability distribution satisfying our linear requirements.

For all $0 \leq \phi < \theta$ we get by the (imposed) stationarity of $\bar{X}$ that (4.11) is bounded from above by

$$H_\phi = H(\bar{X}_{\mathsf{m},\phi} | \bar{X}[\mathsf{T}^{(\prec_{\mathrm{lex}})}_{\mathsf{m},\phi} \cap \mathsf{B}_{\mathsf{m}+1,\theta}]) \ .$$

So, (4.11) is also bounded from above by

$$\frac{1}{\theta} \sum_{\phi=0}^{\theta-1} H_\phi \ . \tag{4.12}$$

The first θ columns of $\bar{X}$ form a configuration with index set $\mathsf{B}_{\mathsf{m}+1,\theta}$. By our linear requirements, stationarity (specifically, vertical stationarity) holds for this configuration as well. So, we may define a stationary 1-D Markov chain [31, §3.2.3] on S_{m}, with entropy given by (4.12). That entropy, in turn, is at most $\widehat{\mathsf{cap}}(S_{\mathsf{m}})$. ■

Proof of Theorem 4.9. The following inequalities are easily verified:

$$\mu(r, s, t) \geq \mu(r+1, s, t) \ .$$

$$\mu(r, s, t) \geq \mu(r, s+1, t) \ .$$

$$\mu(r, s, t) \geq \mu(r, s+1, t+1) \ .$$

The proof follows from them and Lemma 4.10. ■

Appendix

Our goal in this appendix is to prove Theorem 4.1. Essentially, Theorem 4.1 will turn out to be a corollary of [8, Theorem 1.4]. However, [8, Theorem 1.4] deals with configurations in which the index set is $\mathbb{Z}^2$. So, some definitions and auxiliary lemmas are in order.

Recall that $(G_{\text{row}}, G_{\text{col}})$ is the pair of vertex-labeled graphs through which $\mathbb{S} = \mathbb{S}(G_{\text{row}}, G_{\text{col}})$ is defined. Also, recall that each member of $\mathbb{S}$ is a configuration with a rectangular index set. Namely, the index set of a configuration in $\mathbb{S}$ is $\sigma_{i,j}(\mathsf{B}_{M,N})$, for some i, j, M, and N. We now give a very similar definition to that of $\mathbb{S}$, only now we require that the index set of each configuration is $\mathbb{Z}^2$. Namely, define $\mathcal{S} = \mathcal{S}(G_{\text{row}}, G_{\text{col}})$ as follows: A configuration $(w_{i,j})_{(i,j)\in\mathbb{Z}^2}$ over Σ is in $\mathcal{S}(G_{\text{row}}, G_{\text{col}})$ iff there exists a configuration $(u_{i,j})_{(i,j)\in\mathbb{Z}^2}$ over the vertex set V with the following properties: for all $(i,j) \in \mathbb{Z}^2$, (a) the labeling of $u_{i,j}$ satisfies $L(u_{i,j}) = w_{i,j}$; (b) there exists an edge from $u_{i,j}$ to $u_{i,j+1}$ in G_{row}; (c) there exists an edge from $u_{i,j}$ to $u_{i+1,j}$ in G_{col}.

For positive integers $M, N > 0$, define $\mathcal{S}_{M,N}$ as the restriction of $\mathcal{S}$ to $\mathsf{B}_{M,N}$. Namely,

$$\mathcal{S}_{M,N} = \mathcal{S}[\mathsf{B}_{M,N}] \ ,$$

where the definition of the restriction operation is as in (1.2). Also, for M equal to N, define

$$\mathcal{S}_M = \mathcal{S}_{M,M} \ .$$

Note that for all $M, N > 0$ we have

$$\mathcal{S}_{M,N} \subseteq \mathbb{S}_{M,N} \ , \tag{4.13}$$

and there are cases in which the inclusion is strict. Next, define the capacity of $\mathcal{S}$ as

$$\mathsf{cap}(\mathcal{S}) = \lim_{M\to\infty} \frac{1}{M^2} \cdot \log_2 |\mathcal{S}_M| \ .$$

The limit indeed exists, by sub-additivity (see [28, Appendix], and references therein).

For integers $M, N > 0$ and $\delta \geq 0$, denote

$$\mathsf{C}_{M,N,\delta} = \sigma_{-\delta,-\delta}(\mathsf{B}_{M+2\delta,N+2\delta})$$

and let

$$\mathbb{S}_{M,N,\delta} = \mathbb{S}[\mathsf{C}_{M,N,\delta}] \ .$$

Note that the index set $\mathsf{C}_{M,N,\delta}$ of each element of $\mathbb{S}_{M,N,\delta}$ is simply $\mathsf{B}_{M,N}$, padded with δ columns to the right and left and δ rows to the top and bottom. The following lemma will help us bridge the gap between finite and infinite index sets.

Lemma 4.11 *Let w be a configuration over the finite alphabet Σ with index set $\mathsf{B}_{M,N}$. If for all $\delta \geq 0$ we have that*

$$w \in \mathbb{S}_{M,N,\delta}[\mathsf{B}_{M,N}] \ , \tag{4.14}$$

then we must have that

$$w \in \mathcal{S}_{M,N} \ .$$

Proof. Define the following auxiliary directed graph. The vertex set is

$$\bigcup_{\delta \geq 0} \{\hat{w} \in \mathbb{S}_{M,N,\delta} \ : \ \hat{w}[\mathsf{B}_{M,N}] = w\} \ .$$

For every $\delta \geq 0$, there is a directed edge from $w_1 \in \mathbb{S}_{M,N,\delta}$ to $w_2 \in \mathbb{S}_{M,N,\delta+1}$ iff $w_1 = w_2[\mathsf{C}_{M,N,\delta}]$. It is easily seen that this graph is a directed tree with root w, as defined in [19, §2.4]. Since (4.14) holds for all $\delta \geq 0$, the vertex set of the tree is infinite (and countable). On the other hand, since the alphabet size $|\Sigma|$ is finite, the out-degree of each vertex is finite. Thus, by König's Infinity Lemma [19, Theorem 2.8], we must have an infinite path in the tree starting from the root w.

Denote the vertices of the above-mentioned infinite path as

$$w = w^{[0]}, w^{[1]}, w^{[2]}, \ldots \ .$$

We now show how to find a configuration $(w'_{i,j})_{(i,j)\in\mathbb{Z}^2}$ such that $w' \in \mathcal{S}$ and $w = w'[\mathsf{B}_{M,N}]$. For each $(i,j) \in \mathbb{Z}^2$, define $w'_{i,j}$ as follows: let $\delta \geq 0$ be such that $(i,j) \in \mathsf{C}_{M,N,\delta}$, and take $w'_{i,j} = w^{[\delta]}_{i,j}$. It is easily seen that w' is well defined and contained in $\mathcal{S}$. ■

The following lemma states that although the inclusion in (4.13) may be strict, the capacities of $\mathbb{S}$ and $\mathcal{S}$ are equal.

Lemma 4.12 *Let $\mathbb{S}$ and $\mathcal{S}$ be as previously defined. Then,*

$$\mathsf{cap}(\mathcal{S}) = \mathsf{cap}(\mathbb{S}) \ . \tag{4.15}$$

Proof. By (4.13), we must have that $\mathsf{cap}(\mathcal{S}) \le \mathsf{cap}(\mathbb{S})$. For the other direction, it suffices to prove that for all $M > 0$,

$$\mathsf{cap}(\mathbb{S}) \le \frac{1}{M^2} \log_2 |\mathcal{S}_M| \ . \tag{4.16}$$

So, let us fix M and prove the above. By Lemma 4.11, there exists $\delta \ge 0$ such that for all $w \in \Sigma^{M \times M}$,

$$w \notin \mathcal{S}_M \quad \Longrightarrow \quad w \notin \mathbb{S}_{M,M,\delta}[\mathsf{B}_M] \ .$$

For $t > 0$, let M' be shorthand for

$$M' = t \cdot M \ .$$

By the definition of capacity, we have that

$$\mathsf{cap}(\mathbb{S}) = \lim_{t \to \infty} \frac{1}{(M')^2} \log_2 |\mathbb{S}_{M'}| \ . \tag{4.17}$$

Now, let us partition $\mathsf{B}_{M'}$ into the following disjoint sub-sets of indexes: for $0 \le i, j < t$, define the set

$$\mathsf{D}_{i,j} = \sigma_{i \cdot M, j \cdot M}(\mathsf{B}_M) \ .$$

Let $w' \in \mathbb{S}_{M'}$. Notice that for all $0 \le i, j < t$ for which

$$\sigma_{i \cdot M, j \cdot M}(\mathsf{C}_{M,M,\delta}) \subseteq \mathsf{B}_{M'} \ , \tag{4.18}$$

we must have that $w'[\mathsf{D}_{i,j}]$ is equal to some correspondingly shifted element of $\mathcal{S}_M$. On the other hand, for M and δ fixed, the number of pairs (i, j) for which (4.18) does not hold is $O(t)$. Thus, a simple calculation gives us that

$$\frac{1}{(M')^2} \log_2 |\mathbb{S}_{M'}| \le \frac{1}{M^2} \log_2 |\mathcal{S}_M| + O(1/t) \ .$$

This, together with (4.17), proves (4.16). ∎

For a given $M > 0$, define the set $\mathcal{F}(M)$ of configurations with index set $\mathbb{Z}^2$ as follows: a configuration $(w_{i,j})_{(i,j) \in \mathbb{Z}^2}$ is in $\mathcal{F}(M)$ iff for all $(i, j) \in \mathbb{Z}^2$,

$$w[\sigma_{i,j}(\mathsf{B}_M)] \in \mathcal{S}_M \ .$$

Namely, each $M \times M$ "patch" is a correspondingly shifted element of $\mathcal{S}_M$.

Note that there exist vertex-labeled graphs $G_{\mathrm{row}}(M)$ and $G_{\mathrm{col}}(M)$ such that $\mathcal{F}(M) = \mathcal{S}(G_{\mathrm{row}}(M), G_{\mathrm{col}}(M))$. Specifically, the vertex set of both graphs is equal to $\mathcal{S}_M$; the label of each such vertex is its lower-left entry; there is an edge from $w_1 \in \mathcal{S}_M$ to $w_2 \in \mathcal{S}_M$ in $G_{\mathrm{row}}(M)$ ($G_{\mathrm{col}}(M)$) iff the first $M-1$ rows (columns) of w_1 are equal to the last $M-1$ (rows) columns of w_2. Thus, $\mathsf{cap}(\mathcal{F}(M))$ exists. Also, since $w \in \mathcal{S}$ implies $w \in \mathcal{F}(M)$, we have

$$\mathsf{cap}(\mathcal{S}) \le \mathsf{cap}(\mathcal{F}(M)) \; . \tag{4.19}$$

The following is a direct corollary of [8, Theorem 1.4].

Corollary 4.13 *For all $M > 0$, there exists a stationary random variable $W^{(M)}$ taking values on $\mathcal{F}(M)[\mathsf{B}_M]$ such that*

$$\mathsf{cap}(\mathcal{F}(M)) \le \frac{1}{M^2} H(W^{(M)}) \; . \tag{4.20}$$

Proof of Theorem 4.1. Notice that

$$\mathcal{F}(M)[\mathsf{B}_M] = \mathcal{S}_M \subseteq \mathbb{S}_M \; .$$

Thus, $W^{(M)}$ satisfies conditions (i) and (ii) in Theorem 4.1. From (4.15), (4.19), and (4.20) we get that

$$\mathsf{cap}(\mathbb{S}) \le \lim_{M\to\infty} \frac{1}{M^2} \cdot H(W^{(M)}) \; .$$

But since $W^{(M)}$ takes values on $\mathbb{S}_M$, we have by Theorem 1.5 that the above inequality is in fact an equality. Thus, condition (iii) is proved. ■

Bibliography

[1] R. L. Adler, D. Coppersmith, and M. Hassner. Algorithms for sliding block codes—an application of symbolic dynamics to information theory. *IEEE Trans. Inform. Theory*, 29:5–22, 1983.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Massachusetts, 1974.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows.* Prentice Hall, Engelwood Cliffs, New Jersey, 1993.

[4] J. J. Ashley and B. H. Marcus. Two-dimensional low-pass filtering codes. *IEEE Trans. Commmun.*, 46:724–727, 1998.

[5] P. Bender and J. K. Wolf. A universal algorithm for generating optimal and nearly optimal run-length-limited, charge constrained binary sequences. In *Proc. IEEE Int'l Symp. Inform. Theory (ISIT'1993)*, page 6, San Antonio, Texas, 1993.

[6] R. Berger. The undecidability of the domino problem. *Mem. Amer. Math. Soc.*, 66, 1966.

[7] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, Cambridge, UK, 2004.

[8] R. Burton and J. E. Steif. Non-uniqueness of measures of maximal entropy for subshifts of finite type. *Ergod. Th. Dynam. Sys.*, 14:213–235, 1994.

[9] N. Calkin and H. S. Wilf. The number of independent sets in a grid graph. *SIAM J. Discrete Math.*, 11:54–60, 1997.

[10] M. Cohen. On the channel capacity of read/write isolated memory. *Discrete Applied Math.*, 56:1–8, 1995.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, Cambridge, Massachusetts, second edition, 2001.

[12] T. M. Cover. Enumerative source coding. *IEEE Trans. Inform. Theory*, 19:73–77, 1973.

[13] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley, 1991.

[14] J. Czyzyk, M. P. Mesnier, and J. J. Moré. The NEOS server. *IEEE Computational Science & Engineering*, 5(3):68–75, 1998.

[15] J. Campello de Souza, B. H. Marcus, R. New, and B. A. Wilson. Constrained systems with unconstrained positions. *IEEE Trans. Inform. Theory*, 48:866–879, 2002.

[16] E. D. Dolan, R. Fourer, J. J. Moré, and T. S. Munson. The neos server for optimization: Version 4 and beyond. Technical report, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 2002.

[17] K. Engel. On the Fibonacci number of an $m \times n$ lattice. *Fibonacci Quarterly*, 28:72–78, 1990.

[18] T. Etzion. Cascading methods for runlength-limited arrays. *IEEE Trans. Inform. Theory*, 43:319–324, 1997.

[19] S. Even. *Graph Algorithms.* Computer Science Press, 1979.

[20] S. Forchhammer and J. Justesen. Bounds on the capacity of constrained two-dimensional codes. *IEEE Trans. Inform. Theory*, 46:2659–2666, 2000.

[21] S. Forchhammer and T. V. Laursen. A model for the two-dimensional no isolated bits constraint. In *Proc. IEEE Int'l Symp. Inform. Theory (ISIT'2006)*, pages 1189–1193, Seattle, Washington, 2006.

[22] S. Forchhammer and T. V. Laursen. Entropy of bit-stuffing-induced measures for two-dimensional checkerboard constraints. *IEEE Trans. Inform. Theory*, 53:1537–1546, 2007.

[23] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Duxbury Press, second edition, 2002.

[24] W. Gropp and J. J. Moré. Optimization environments and the neos server. In *Approximation Theory and Optimization*, pages 167–182. Cambridge University Press, 1997.

[25] S. Halevy, J. Chen, R. M. Roth, P. H. Siegel, and J. K. Wolf. Improved bit-stuffing bounds on two-dimensional constraints. *IEEE Trans. Inform. Theory*, 50:824–838, 2004.

[26] S. Halevy and R. M. Roth. Parallel constrained coding with application to two-dimensional constraints. *IEEE Trans. Inform. Theory*, 48:1009–1020, 2002.

[27] K. A. S. Immink. A practical method for approaching the channel capacity of constrained channels. *IEEE Trans. Inform. Theory*, 43:1389–1399, 1997.

[28] A. Kato and K. Zeger. On the capacity of two-dimensional run-length constrained code. *IEEE Trans. Inform. Theory*, 45:1527–1540, 1999.

[29] B. H. Marcus and R. M. Roth. Bounds on the number of states in encoder graphs for input-constrained channels. *IEEE Trans. Inform. Theory*, 37:742–758, 1991.

[30] B. H. Marcus and R. M. Roth. Improved Gilbert–Varshamov bound for constrained systems. *IEEE Trans. Inform. Theory*, 38:1213–1221, 1992.

[31] B. H. Marcus, R. M. Roth, and P. H. Siegel. Constrained systems and coding for recording channels. In V.S. Pless and W.C. Huffman, editors, *Handbook of Coding Theory*, pages 1635–1764. Elsevier, Amsterdam, 1998.

[32] A. V. Oppenheim and A. S. Willsky. *Signals and Systems.* Prentice Hall, second edition, 1997.

[33] E. K. Orcutt and M. W. Marcellin. Enumerable multi-track (d, k) block codes. *IEEE Trans. Inform. Theory*, 39:1738–1744, 1994.

[34] E. K. Orcutt and M. W. Marcellin. Redundent multi-track (d, k) codes. *IEEE Trans. Inform. Theory*, 39:1744–1750, 1994.

[35] E. Ordentlich and R. M. Roth. Capacity lower bounds and approximate enumerative coding for 2-D constraints. In *Proc. IEEE Int'l Symp. Inform. Theory (ISIT'2007)*, pages 1681–1685, Nice, France, 2007.

[36] R. M. Robinson. Undecidability and nonperiodicity for tiling the plane. *Inventions Math.*, 12:177–209, 1971.

[37] R. M. Roth, P. H. Siegel, and J. K. Wolf. Efficient coding schemes for the Hard-Square model. *IEEE Trans. Inform. Theory*, 47:1166–1176, 2001.

[38] A. Sharov and R. M. Roth. Two-dimensional constrained coding based on tiling. In *Proc. IEEE Int'l Symp. Inform. Theory (ISIT'2008)*, pages 1468–1472, Toronto, Ontario, 2008.

[39] P. H. Siegel and J. K. Wolf. Bit-stuffing bounds on the capacity of 2-dimensional constrained arrays. In *Proc. IEEE Int'l Symp. Inform. Theory (ISIT'1998)*, page 323, Cambridge, Massachusettes, 1998.

[40] R. P. Stanley. *Enumerative Combinatorics*, volume 2. Cambridge University Press, Cambridge, UK, 1999.

[41] R. Talyansky, T. Etzion, and R. M. Roth. Efficient code constructions for certain two-dimensional constraints. *IEEE Trans. Inform. Theory*, 45:794–799, 1999.

[42] T. van Aardenne-Ehrenfest and N. G. de Bruijn. Circuits and trees in oriented linear graphs. *Simon Stevin*, 28:203–217, 1951.

[43] A. Vardy, M. Blaum, P. H. Siegel, and G. T. Sincerbox. Conservative arrays: Multidimensional modulation codes for holographic recording. *IEEE Trans. Inform. Theory*, 42:227–230, 1996.

[44] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.*, 106(1):25–57, 2006.

[45] W. Weeks and R. E. Blahut. The capacity and coding gain of certain checkerboard codes. *IEEE Trans. Inform. Theory*, 44:1193–1203, 1998.

[46] H. S. Wilf. The problem of the kings. *Electr. J. Combinatorics*, 2:#R3, 1995.

צפינה וחסמים לאילוצים דו-ממדיים

עדו טל

צפינה וחסמים לאילוצים דו-ממדיים

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
דוקטור לפילוסופיה

עדו טל

הוגש לסנט הטכניון – מכון טכנולוגי לישראל
טבת ה'תשס"ט חיפה ינואר 2009

המחקר נעשה בהנחיית פרופ׳ רוני רוט ופרופ׳ טובי עציון בפקולטה למדעי המחשב.

ברצוני להודות מקרב למנחי, פרופ׳ רוני רוט ופרופ׳ טובי עציון, על עזרתם ותמיכתם הרבה.

רוני, תודה שנתת לי את הזכות והעונג להיות המונחה שלך. הנחייתך היתה מסורה מאוד. הדברים הרבים שלימדת אותי — ואני מקווה שלמדתי — אין ערוך להם. כמו כן, תודה לך על סבלנותך ושלוותך במהלך כל ההתחלות השגויות והמבואות הסתומים שנתקלנו בהם.

טובי, תודה על כך שהיית המנחה שלי כאשר רוני היה בשנת שבתון. במהלך תקופה זו, השגנו שלושתנו את התוצאה המרכזית הראשונה בתזה זו, במידה רבה מאוד הודות לשיחות שניהלת איתי. בנוסף לדברים הרבים שלמדתי ממך, תמיד אזכור בהערכה רבה שהסכמת להנחות אותי, בזמן שהייתי למעשה ״תקוע״.

ברצוני להודות לאשתי סמדר ולבני יואב על אהבתם ועידודם.

אני מודה לטכניון ולחברת HP על התמיכה הכספית הנדיבה בהשתלמותי.

תוכן עניינים

רשימת איורים

תקציר

להתקני זיכרון רבים (כגון כוננים קשיחים, תקליטורים, וכו') יש קלטים "רעים", במובן זה שתהליך כתיבתם, ואחר כך קריאתם, מועד לשגיאות. הדרך להתגבר על בעיה זו היא, ראשית, לאפיין את קבוצת הקלטים הרעים, ולאחר מכן להרשות רק לקלטים "טובים" להיכתב להתקן הזיכרון. קבוצה זו של קלטים טובים נקראת "מערכת אילוצים" או "אילוץ".

באילוץ חד-ממדי הקלטים הם מילים. האילוץ מוגדר על-ידי גרף (קבוצת צמ-תים וקבוצת קשתות מכוונות בין הצמתים), שקשתותיו מסומנות באותיות מעל הא"ב הרלוונטי. קיבול האילוץ מוגדר כיחס האסימפטוטי בין log מספר המילים הטובות באורך מסוים, חלקי האורך. עבור גרפים בעלי גודל סביר, קיבול האילוץ הוא מספר קל לחישוב. יתר על כן, בעזרת אלגוריתם פיצול המצבים ניתן לבנות ביעילות צמד מצפין/מפענח לאילוץ. דהיינו, מצפין הוא אלגוריתם הממפה באו-פן חד-חד ערכי את האינפורמציה אותה אנו רוצים לקודד למילה המקיימת את האילוץ, ומפענח הוא אלגוריתם המבצע את הפעולה ההפוכה (אולם, יתכן שה-מילה שהמפענח קורא איננה המילה שהמצפין כתב, בגלל התווספות של רעש). אם נניח כי לא קיים רעש במערכת (כלומר, מילה טובה תמיד תיקרא נכון), אז ניתן לבנות בעזרת אלגוריתם פיצול המצבים צמד מצפין/מפענח בעלי קצב קרוב כרצוננו לקיבול. בנוסף, למפענח יש לרוב "חלון פענוח" סופי. תכונה זו מאפשרת (לאחר נקיטת צעדים נוספים) חסינות מסוימת לרעש.

באילוץ דו-ממדי הקלטים הם מערכים דו-ממדיים, בניגוד למילים חד-ממדיות. כעת, קיבול האילוץ מוגדר כיחס האסימפטוטי בין log מספר המערכים הריבועי-ים בגודל מסוים, חלקי מספר הכניסות. בהשוואה לאילוצים חד-ממדיים, הידע שלנו לגבי אילוצים דו-ממדיים לוקה בחסר. בפרט, לא ידועה מקבילה דו-ממדית לאלגוריתם פיצול המצבים. יתר על כן, יש קושי אינהרנטי לחשב את הקיבול של אילוץ דו-ממדי כללי: השאלה "האם לאילוץ הנתון יש קיבול אי-שלילי?" היא בעיה אי-כריעה, עבור אילוצים מסוימים. בשל כך, אלגוריתמי ההצפנה/פענוח הידועים

הם ספציפיים לאילוץ, ובדרך כלל תת-אופטימליים. אלגוריתמים רבים מניחים כי לא קיים רעש במערכת (הנחה שאינה מתקיימת בפועל), ולא ברור כיצד ניתן להפוך אותם לחסינים לרעש.

בעבודה זו נציג שלוש תוצאות עיקריות. התוצאה הראשונה היא צמד מצפין/מפענח בעל קצב קבוע למשפחה גדולה של אילוצים דו-ממדיים. הצפנה ופענוח מתבצעים שורה אחר שורה, והפענוח הוא sliding-block (כלומר, פענוח השורה הנוכחית תלוי רק במספר מוגבל של שורות סביבה). פענוח מסוג זה הוא טבעי ומעניין כאשר, בנוסף, כל שורה מוגנת מפני רעש על-ידי קוד לתיקון שגיאות.

בתוצאה הראשונה, אנו למעשה הופכים את האילוץ הדו-ממדי לקבוצת אילוצים חד-ממדיים פשוטים ובלתי תלויים. דהיינו, אנו מבצעים מעין רדוקציה מן המקרה הדו-ממדי למקרה החד-ממדי. צעד זה נעשה על-ידי חלוקת המערך הדו-ממדי לרצועות אנכיות בעלות רוחב קבוע והתייחסות לכל שורה ברצועה כאל אות בא"ב מסוים. לאחר מכן, אנו מוצאים את הגרף המתאר את האילוץ החד-ממדי המתקבל. אנו מחשבים את שרשרת המרקוב המקסאנטרופית ומבצעים לה פרטורבציה. בעז-רת ההסתברויות המותנות של הקשתות בפרטורבציה זו אנו בונים מקודד הפועל במקביל על הרצועות. לצורך מציאת הפרטורבציה אנו מגדירים רשת זרימה בעלת חסמים עליונים ותחתונים על הזרימה בקשתות. יצוין, כי תרומתנו היא השיטה לחישוב הפרטורבציה; השלבים הקודמים שתוארו הופיעו בעבודה קודמת שאינ-נה שלנו. לבסוף, נציין כי למרות שהמוטיבציה העיקרית לשיטתנו הם אילוצים דו-ממדיים, ניתן להשתמש בה גם עבור הצפנה ופענוח עבור אילוצים חד-ממדיים. בפרט, מקרה מעניין, בו שיטתנו שימושית, הוא המקרה בו נדרוש כי המשקל היחסי של כל מילה אותה מייצר המקודד יהיה קבוע נתון (בנוסף לדרישה שכל מילה כנ"ל תקיים את האילוץ החד-ממדי).

תוצאתנו השנייה היא שיטה לחישוב חסמים עליונים ותחתונים על קצב של מקודד מסוג bit-stuffing. דהיינו, מקודד בו הערך הנכתב לכניסה נתונה של המערך הוא תוצאה של הטלת מטבע, כאשר הטיית המטבע האמור היא פונקציה של קבוצת "שכנים" שנכתבו כבר למערך. בנוסף, תוצאות הטלת "המטבע" הנ"ל הן למעשה פונקציה הפיכה של הקלט אותו אנו רוצים לקודד.

במקום לנתח ישירות את המקודד המקורי, אנו מראים כי ניתן, בלי הגבלת הכל-ליות, לנתח מקודד אחר. מקודד זה הוא במובן מסוים "מיצוע" של המקודד המקורי, ולכן קוואזי-סטציונארי. דהיינו, ההתפלגות של שני מקטעים במערך הקר-ובים זה לזה וכן בעלי צורה זהה היא שווה בקירוב. מאפיינים נוספים של תכונת הקוואזי-סטציונאריות מאפשרים לנו להניח, בלי הגבלת הכלליות, כי כל מקטע

סופי של המערך הוא סטציונארי (ללא הסתייגות ה"קוואזי"). הנחה זו מאפש-
רת לנו לנסח מערכת משוואות לינאריות (שוויונים ואי-שוויונים) המתקיימות על
פילוג ההסתברויות של כל מקטע סופי של המערך הנוצר על-ידי המקודד. מערכת
משוואות זו משמשת אותנו לבניית תכנית לינארית. תכנית זו למעשה "מאלצת"
סטציונאריות במקטע של המערך, ופונקציית המטרה שלה היא האנטרופיה של
כניסה מסוימת, בהינתן השכנים שלה. המינימום והמקסימום של התכנית הלי-
נארית חוסמים את קצב המקודד מלמעלה ומלמטה, בהתאמה. ההפרש בין שני
חסמים אלו הולך ומצטמצם ככל שמגדילים את המקטע הנ"ל. אולם, גם גודלה של
התכנית הלינארית הולך וגדל ככל שמגדילים את גודל המקטע (זמן פתירת תכנית
לינארית עומד ביחס ישר לגודלה).

חסם תחתון על קצב של מקודד הוא גם חסם תחתון על קיבול האילוץ המתאים.
לאילוצים מסוימים, תוצאתנו השנייה מובילה לחסמים תחתונים הדוקים יותר
ממה שהיה ידוע עד כה על קיבול האילוץ.

כפי שציינו, הקיבול של אילוץ חד-ממדי שווה לאנטרופיה של שרשרת מרקוב
סטציונארית מתאימה. כלומר, אנו ממקסמים את האנטרופיה על פני קבוצה סו-
פית של הסתברויות המקיימות תכונות מסוימות. בתוצאתנו השלישית, אנו מנסים
להכליל היבטים מסוימים של אפיון זה לאילוצים דו-ממדיים. התוצאה היא שיטה
לחישוב חסם עליון על קיבולים של אילוצים דו-ממדיים.

הצעדים העיקריים הם כדלקמן: אנו מוכיחים כי קיימת התפלגות מקסאנטרופית
על מערכים ריבועיים שהיא גם סטציונארית. דהיינו, התפלגות של שני מקטעים
בעלי צורה שווה במערך היא זהה. מעתה ואילך, אנו מתבוננים בהתפלגות זו. בדומה
לתוצאתנו השנייה, תכונת הסטציונאריות מאפשרת לנו לנסח מערכת משוואות
לינאריות (שוויונים ואי-שוויונים) על פילוג ההסתברויות של מערכים ריבועיים.
מערכת משוואות זו משמשת אותנו לבניית תכנית קמורה. כמקודם, אנו מאלצים
סטציונאריות — ולעתים גם תכונות סימטריה — על מקטע של המערך. פונקציית
המטרה היא מיצוע האנטרופיות של כניסות מסוימות, בהינתן כניסות קודמות להן
במקטע. פתרונה של תכנית זו ניתן לחישוב ביעילות, והוא מהווה חסם עליון על
קיבול האילוץ. כמקודם, החסם מתהדק והתכנית גדלה ככל שמגדילים את גודל
המקטע. לבסוף, נעיר כי ניתן להגדיר את המושג "כניסות קודמות" בדרכים שונות.
אנו אכן עושים זאת, והדבר מועיל לנו.