arXiv:2504.16265v1 [math.CO] 22 Apr 2025

Term Coding for Extremal Combinatorics: Dispersion and Complexity Dichotomies

Søren Riis
Queen Mary University of London
s.riis@qmul.ac.uk

Abstract

We introduce *Term Coding*, a novel framework for analysing extremal problems in discrete mathematics by encoding them as finite systems of *term equations* (and, optionally, *non-equality constraints*). In its basic form, all variables range over a single domain, and we seek an interpretation of the function symbols that *maximises* the number of solutions to these constraints. This perspective unifies classical questions in extremal combinatorics, network/index coding, and finite model theory.

We further develop *multi-sorted Term Coding*, a more general approach in which variables may be of different sorts (e.g., points, lines, blocks, colours, labels), possibly supplemented by variable-inequality constraints to enforce distinctness. This extension captures sophisticated structures such as block designs, finite geometries, and mixed coding scenarios within a single logical formalism.

Our main result shows how to determine (up to a constant) the maximum number of solutions $\max_{\mathcal{I}}(\Gamma, n)$ for any system of term equations (possibly including non-equality constraints) by relating it to *graph guessing numbers* and *entropy measures*.

Finally, we focus on *dispersion problems*, an expressive subclass of these constraints. We discover a striking complexity dichotomy: deciding whether, for a given integer r, the maximum code size that reaches n^r is *undecidable*, while deciding whether it exceeds n^r is *polynomial-time decidable*.

1 Introduction

A recurring theme in complexity theory and combinatorics is that *small adjustments* to a problem's parameters can cause *dramatic* shifts in computational difficulty (see, e.g., [1, 2, 3]). This paper demonstrates a particularly dramatic version of such a phenomenon. Although full first-order logic with universal quantification can encode a broad range of combinatorial problems [4, 5, 6, 7], we restrict our attention to a quantifier-free, negation-free language of term equations—augmented, when needed, with non-equality constraints. (We refer to the resulting framework as *Term Coding*; a formal definition is given in Section 2.) Surprisingly, even this limited syntax captures many extremal problems and reveals rich structural properties. In particular, we show that a minimal increase in the solution threshold can flip an undecidable question into one decidable in polynomial time.

A striking single-sorted dichotomy

Within the framework of *single-sorted Term Coding* (see Section 2 for formal definitions), we consider *dispersion problems*—that is, systems of term equations over an n-element domain that

define a code (a set of tuples) whose size we wish to maximise. We prove that deciding whether, for a given dispersion problem Γ and an integer r, there exists some n for which the code size reaches n^r is undecidable; yet, if the target size is increased by one to $n^r + 1$, the problem becomes polynomial-time decidable.

A Specialised Framework with Broad Connections

Although *single–sorted Term Coding* is built on conjunctions of term equations (possibly with *non-equality constraints*, in the literature sometimes called *dis-equalities*), it resonates with several areas of discrete mathematics: extremal combinatorics, finite model theory, and network/index coding. By linking code sizes to *graph entropy* [8, 9] and *guessing numbers* [10, 11], we obtain tight bounds on the size of extremal solutions, highlighting the fine boundary between decidability and undecidability.

1.1 General Motivation

Many central combinatorial problems—such as block designs, finite geometries, and error-correcting codes—seek structures of size n that satisfy global constraints (see, e.g., [12, 13, 14, 15]). While such constraints are often expressible using first-order logic (FO) with quantifiers, investigating these problems through the lens of simpler logical fragments can yield new structural insights and connections.

In our *Term Coding* framework, we restrict attention to a quantifier-free language based primarily on conjunctions of *term equations*. In its basic single-sorted form, this resembles equational logic. However, the framework is significantly extended by permitting *multiple variable sorts* (each with its own domain) and incorporating *non-equality constraints* ($s \neq t$). As is standard in logic and detailed later (Theorem 4.1), any first-order sentence ψ can be transformed by standard logical transformations (Skolemisation followed by conversion to Conjunctive Normal Form (CNF)) into an equisatisfiable universal sentence, which in turn corresponds directly to a system of multi-sorted term equations and non-equalities. This means that the existence of *any* model (finite or infinite) for ψ is equivalent to the existence of a model for a corresponding Term Coding system Γ_ψ. Consequently, the framework captures the full model-theoretic satisfiability of first-order logic, despite its syntactic simplicity (lacking explicit quantifiers or full Boolean negation).

While possessing general expressive power, the emphasis in Term Coding shifts from merely asking *if* a model exists (a question whose complexity is tied to FO logic) to analysing the properties of the *code* over *finite* domains—the set of solution tuples $(a_1, \ldots, a_v)$—and, in particular, determining the *maximum possible size* of this code, $\max_{\mathcal{I}}(\Gamma, n)$. This quantitative focus on maximising solutions within finite structures allows us to connect model-theoretic questions to extremal combinatorics and information theory (via graph guessing numbers and entropy), leading to precise asymptotic bounds and revealing phenomena like the complexity dichotomy central to this work (which specifically concerns behaviour over finite domains).

By isolating and analysing this specific fragment—multi-sorted term equations with non-equalities, viewed through an optimisation lens over finite domains—Term Coding provides a unified algebraic and combinatorial framework for studying a broad spectrum of existence and extremal problems relevant to combinatorics.

1.2 Motivating Example: Steiner Triple Systems (t=2, k=3)

A *Steiner triple system* on a set P (with $|P| = n$) is equivalent to endowing P with a binary operation

$$f : P \times P \to P,$$

called a *Steiner quasigroup*. In our framework, this structure is completely characterised by the following universally quantified term equations:

Idempotence: $\forall x \ : \ f(x, x) = x$.

Commutativity: $\forall x, y \ : \ f(x, y) = f(y, x)$.

Inversion: $\forall x, y \ : \ f\big(x, f(x, y)\big) = y$.

These equations force that for any two distinct points x and y, the element $f(x, y)$ is the unique third point so that the triple $\{x, y, f(x, y)\}$ forms a block. Every Steiner triple system ($t = 2$ and $k = 3$) gives rise to such an operation f, and conversely, every binary operation satisfying the term equations defines a Steiner triple system. It is well known that Steiner triple systems with $t = 2$ and $k = 3$ exist if and only if $n \equiv 1$ or $3 \pmod 6$.

This basic example involves only a single sort and three simple term equations. This is a well-known algebraic reformulation for Steiner triple systems for $t = 2$ and $k = 3$.

In Term Coding, we consider problems that otherwise can be stated in universal algebra as optimisation problems. The focus is on codes, and each set of solutions naturally defines a code where each codeword (x, y) satisfies the given constraints. In this example, a computer search provided the following values for the first few values of n.

n	Maximum	Ideal n^2	Ratio
1	1	1	1.000
2	3	4	0.750
3	9	9	1.000
4	13	16	0.812
5	21	25	0.840
6	33	36	0.917
7	49	49	1.000
8	60	64	0.938
9	81	81	1.000

Table 1: Maximum values for $n \leq 9$. Note that the ideal value n^2 is attained if and only if $n \equiv 1$ or 3 modulo 6.

1.3 Term Coding: Purely Equational Single-Sorted Case

A particularly natural special case of Term Coding emerges when the constraints are reduced to *term equations* alone. Concretely, a *term equation* has two sides, each built from variables and function symbols. Formally, a system of term equations is given by a set

$$\Gamma \ = \ \{\, s_1 = t_1, \ \ldots, \ s_m = t_m\},$$

where each s_i and t_i is a term in variables $x_1, \ldots, x_v$. In Term Coding, such equations define *fixed-length codes* over an n-element alphabet $A = \{0, \ldots, n-1\}$. For each interpretation $\mathcal{I}$ of the function symbols into A (i.e., each choice of how the functions act on A), the solution set

$$S_{\mathcal{I}}(\Gamma) \;=\; \big\{(a_1, \ldots, a_v) \in A^v : \; \forall i \; \mathcal{I} \models s_i(a_1, \ldots, a_v) = t_i(a_1, \ldots, a_v)\big\}$$

is a code (a collection of v-tuples). Naturally, one would like to determine:

$$\max_{\mathcal{I}}(\Gamma, n) \;=\; \max_{\mathcal{I}} \big|S_{\mathcal{I}}(\Gamma)\big|,$$

where the maximum is taken over all possible interpretations $\mathcal{I}$ of Γ into A.

1.4 Illustrative Examples

To demonstrate the expressive power of Term Coding, we present a few more illuminating examples.

A Network Coding Example

Suppose we have two messages $x, y \in A$ together with an encoded value $z = f(x, y)$ that we wish to store at three locations, so that x and y can be recovered from any two locations. Specifically, let x, y, z range over an alphabet $A = \{0, 1, \ldots, n-1\}$, and consider the function symbols

$$f, \; h_1, \; h_2 : A \times A \;\to\; A.$$

We impose the following system of term equations:

$$z \;=\; f(x, y), \quad h_1\big(x, z\big) \;=\; y, \quad h_2\big(y, z\big) \;=\; x.$$

Under an interpretation $\mathcal{I}$, these constraints encode:

- z stores an encoded combination of x and y, via $z = f^{\mathcal{I}}(x, y)$.
- From (x, z) alone, we can recover y using $y = h_1^{\mathcal{I}}(x, z)$.
- From (y, z) alone, we can recover x using $x = h_2^{\mathcal{I}}(y, z)$.

A concrete interpretation setting $f(x, y) = x + y \pmod n$ with suitable decoding functions h_1, h_2 yields a code of size n^2 from the n^3 possible triples (x, y, z).

Self-Orthogonal Latin Squares

Term Coding can also capture classical combinatorial structures. Consider self-orthogonal Latin squares—$n \times n$ arrays filled with n symbols, each appearing exactly once per row and column. Define functions

$$f, h_1, h_2, h_3, h_4 : A \times A \;\to\; A$$

and consider the system of term equations:

$$h_1\big(f(x, y), y\big) = x, \quad h_2\big(x, f(x, y)\big) = y, \quad h_3\big(f(x, y), f(y, x)\big) = x, \quad h_4\big(f(x, y), f(y, x)\big) = y.$$

Here, $f(x, y)$ represents the symbol in row x, column y of the square. The constraints

$$h_1\big(f(x, y),\, y\big) = x \;\text{ and }\; h_2\big(x,\, f(x, y)\big) = y$$

enforce that each row and column is a permutation of the n symbols, ensuring the "Latin" property. Meanwhile,

$$h_3\big(f(x,y),\, f(y,x)\big) = x \;\text{ and }\; h_4\big(f(x,y),\, f(y,x)\big) = y$$

impose the "self-orthogonality" requirement.

Concretely, there is a *one-to-one correspondence* between solutions to this system (with n^2 codewords (x,y)) and self-orthogonal Latin squares of order n. Such squares exist for all n except $\{2,3,6\}$ (see [16, 17]), illustrating how term equations can encode fundamental Latin square properties within the Term Coding formalism.

1.5 An Unsolvable Variant

To demonstrate the Term Coding setup, consider a stricter version of the self-orthogonal Latin square constraints, where the "decoding" must be done by the square itself:

$$f\big(f(x,y),\, y\big) = x, \quad f\big(x,\, f(y,x)\big) = y, \quad f\big(f(x,y),\, f(y,x)\big) = x, \quad f\big(f(y,x),\, f(x,y)\big) = y.$$

A short argument shows that no such *self-orthogonal self-decoding Latin squares* exist—i.e. solutions that satisfy the term equations for $n > 1$. To see this, note:

- From the first two equations, f must be a Latin square.
- Comparing equations (1) and (3) forces $y = f(y,x)$ (by injectivity).
- Consequently, every row of f is constant, contradicting the Latin square property.

However, although there are no global solutions, there do exist partial solutions.

In several examples, we will rely on a key *normalisation step*, where *auxiliary variables* are introduced to replace nested or repeated subterms. As an illustration, we can introduce fresh variables $f(x,y) = z$ and $f(y,x) = w$, rewriting the four original equations as follows:

$$f\big(x,y\big) = z, \quad f\big(y,x\big) = w, \quad f\big(z,\, y\big) = x, \quad f\big(x,\, w\big) = y, \quad f\big(z,\, w\big) = x, \quad f\big(w,\, z\big) = y.$$

Any set of solutions $(x,y) \in M^2$ to the original system corresponds to a set of solutions $(x,y,z,w) \in M^4$ to these normalised equations, and vice versa (up to identifying the added variables).

1.6 Investigating Non-Solvability

It is important to understand that logically equivalent problems can lead to different optimisation problems. In many cases, a combinatorial problem can be stated in logically equivalent ways—one formulation may use k_1 free variables, while another uses k_2 free variables. In a formulation with k free variables, the ideal maximal code size is n^k. Thus, if one formulation has k_1 free variables and another has k_2 free variables, then the ideal maximal codes are n^{k_1} and n^{k_2}, respectively.

For instance, consider the following two logically equivalent formulations:

Formulation 1 (natural version $k = 2$): Let Γ_1 be the system

$$f\big(f(x,y),\, y\big) = x, \quad f\big(x,\, f(y,x)\big) = y, \quad f\big(f(x,y),\, f(y,x)\big) = x, \quad f\big(f(y,x),\, f(x,y)\big) = y.$$

In this formulation, the same variables x and y appear repeatedly. Consequently, the number of free variables is two, and, ideally, a maximal code would have size n^2.

Formulation 2 (refined version $k = 8$): Let Γ_2 be the system

$$f\big(f(x_1, y_1),\, y_1\big) = x_1, \quad f\big(x_2,\, f(y_2, x_2)\big) = y_2, \quad f\big(f(x_3, y_3),\, f(y_3, x_3)\big) = x_3, \quad f\big(f(y_4, x_4),\, f(x_4, y_4)\big) = y_4.$$

Here, we have renamed variables so that every occurrence is distinct; the system now has eight free variables. Consequently, the ideal maximal code in this formulation is n^8.

Although both formulations are logically equivalent (since one can identify x_1 with x_2, y_1 with y_2, etc.), they lead to different optimisation problems. In practice, the maximum number of solutions may be strictly less than the ideal n^k. To investigate this, we performed a computational search for interpretations $\mathcal{I}$ that maximise the number of solutions for small n. The best results obtained, providing lower bounds on the true theoretical maximum $\max_{\mathcal{I}}$, are shown below:

Table 2: Best values found via computer search for Formulation 1 (Γ_1, $k = 2$). These are lower bounds on the true maximum size $\max_{\mathcal{I}}(\Gamma_1, n)$.

n	Max Size Found	Max Possible (n^2)	$\frac{\text{Max Size Found}}{n^2}$
2	2	4	0.50
3	4	9	0.44
4	8	16	0.50
5	9	25	0.36
6	14	36	0.389

Table 3: Best values found via computer search for Formulation 2 (Γ_2, $k = 8$). These are lower bounds on the true maximum size $\max_{\mathcal{I}}(\Gamma_2, n)$.

n	Max Size Found	Max Possible (n^8)	$\frac{\text{Max Size Found}}{n^8}$
2	128	256	0.50
3	2205	6561	0.336
4	24576	65536	0.375
5	138125	390625	0.353
6	559872	1679616	0.333

These tables illustrate that, although both formulations are logically equivalent, the corresponding optimisation problems have different ideal bounds, and the maximum sizes found computationally (and the resulting ratios to the ideal) differ. Furthermore, the functions f yielding these best-known results for formulation 1 were not identical to those for formulation 2. This phenomenon opens up avenues for further research into the interplay between different optimisation problems related to Term Coding problems. By comparing equivalent formulations, one can analyse how close the maximal code is to the ideal bound of n^k for each formulation.

Moreover, our general Theorem 3.8 shows that the true theoretical maximum sizes, $\max_{\mathcal{I}}(\Gamma_1, n)$ and $\max_{\mathcal{I}}(\Gamma_2, n)$, are bounded below by constants times n^2 and n^8 respectively. Thus, the true ratios

$$c_2(n) = \frac{\max_{\mathcal{I}}(\Gamma_1, n)}{n^2} \quad \text{and} \quad C_8(n) = \frac{\max_{\mathcal{I}}(\Gamma_2, n)}{n^8}$$

remain bounded from below by a positive constant independent of n. The computationally obtained results presented in the tables, while potentially suboptimal lower bounds, are consistent with this theoretical guarantee.

1.6.1 Diversification of Function Symbols

A key technique for analysing these systems, particularly for obtaining asymptotic bounds, is to *diversify* the function symbols. Concretely, if the same symbol f appears with different tuples of variables in the normalised system, we introduce a distinct function symbol for each occurrence. Applying this to the normalised term equations derived from Formulation 1 yields the *diversified* system Γ'':

$$f_1(x,y) = z, \quad f_2(y,x) = w, \quad f_3(z,y) = x, \quad f_4(x,w) = y, \quad f_5(z,w) = x, \quad f_6(w,z) = y.$$

Let $S_n(\Gamma')$ and $S_n(\Gamma'')$ be the maximum solution set sizes for the normalised and diversified systems, respectively, over a domain of size n. The diversified system Γ'', having potentially more freedom by using distinct functions, provides an upper bound: $S_n(\Gamma') \leq S_n(\Gamma'')$. Conversely, standard domain partitioning arguments show that interpreting the diversified symbols over a suitably scaled alphabet of size $\approx n/c$ (for some constant c) yields a lower bound on the original system's solution set size. Combining these bounds shows that the systems are asymptotically equivalent: $S_n(\Gamma') = \Theta(S_n(\Gamma''))$. This is crucial because the dependency structure of the diversified system Γ'' is often simpler and directly amenable to analysis using techniques like graph guessing games.

1.6.2 Graph Guessing Game Reformulation and Our New Method

Guessing games on directed graphs were originally introduced by Riis [10, 18] to investigate Valiant's open question on information-flow bottlenecks in circuit complexity and to connect network coding with a combinatorial notion of graph guessing numbers. Subsequent work related these games to graph entropy and reversible versus irreversible information flows [9, 19] and examined special families such as shift graphs [20], triangle-free graphs [21], and undirected graphs [22, 11]. Further advances tied the guessing number to extremal graph theory: Martin and Rombach [23] explored how bounding the guessing number is equivalent to forbidding a finite set of subgraphs and studied classic Turán-type questions in this context. On another front, Gadouleau, Richard, and Riis [24] investigated fixed-point counts in (Boolean or multi-valued) network update functions, highlighting how the structure of "signed digraphs" links to guessing graphs and coding for asymmetric channels. These results also underscore the relevance of guessing games to Boolean networks, commonly employed for modelling gene regulation, neural interactions, and social dynamics, where local update rules translate into combinatorial constraints on the system's global state evolution. See also [25, 24] for additional applications to non-Shannon information inequalities, Boolean networks, and coding theory.

In a standard graph guessing game, each node (or "player") is assigned a hat colour from a finite alphabet, sees only the hat colours of its in-neighbours, and must guess its own colour via a deterministic function. The "guessing number" then measures how many hat assignments can be simultaneously guessed correctly by a suitably designed strategy.

We introduce *Term Coding*, which provides a broader approach to extremal combinatorics by encoding complex constraints (e.g. d-designs, coding theory, or finite model theory) as systems of *term equations* (with optional non-equality constraints). We further develop an *extended* multi-sorted version of the guessing game model, allowing each node to have its own "alphabet" and labelling beyond the classical single-sorted directed graph setting. Our main theorems show that this extended guessing-game viewpoint and the Term Coding formalism are tightly linked: specifically, the maximum number of solutions in a Term Coding system (i.e. code size) can be approximated up to a constant factor by the guessing number of the associated multi-sorted graph. In this way,

the new Term Coding method subsumes earlier guessing-game ideas while also generalising them to richer combinatorial settings and multiple sorts.

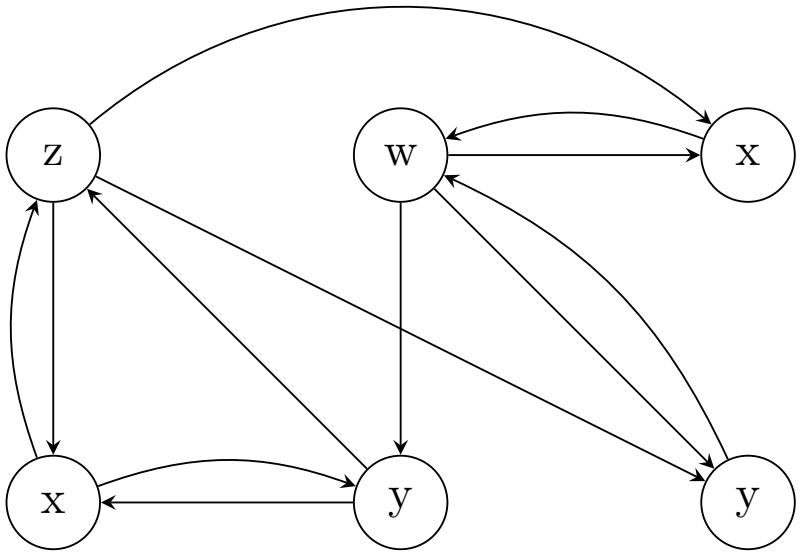

Figure 1: Directed graph showing the functional dependencies for the normalised and diversified term equations derived from the *unsolvable* self-decoding Latin square variant (introduced in Section 1.5). The graph has six nodes, labelled according to the variables $\{x, x, y, y, z, w\}$ on the right-hand side of these six equations. An edge $u \to v$ indicates that variable v depends functionally on variable u in the corresponding equation. The next section analyses this dependency structure using guessing number/entropy techniques.

1.6.3 Labelling of Nodes and Distinctness Constraints

In our term-coding context, each node/player is *labelled* by a variable x_i. If two nodes share the same label, they must necessarily have the *same* hat colour (as they represent the same variable). Conversely, *non-equality* constraints ($x_i \neq x_j$) forbid those two labelled nodes from ever sharing a colour. Thus, in any random assignment of hats, if **a** violates these constraints, that assignment is not considered valid. The players can agree on their guessing strategy (choice of functions) in advance. The players know which labels must match or differ when they agree on a guessing strategy.

1.6.4 Deterministic Strategies and Winning Configurations.

A *guessing strategy* specifies, for each node, how to guess its own hat colour based on the observed colours of its in-neighbors. The "win" condition is that *all* nodes guess correctly on a given hat-color assignment. We measure the quality of a strategy by the total number of hat assignments (*configurations*) on which every guess is correct. In a term-coding analogy, each node's guess function enforces a local functional dependency (variables with edges from in-neighbors).

1.6.5 Maximal Solutions vs. Correct Guesses

In a *diversified* term-coding system, the largest set of hat assignments for which every node guesses its own hat colour correctly is *identical* to the maximum set of solutions for the term equations. Concretely, each valid solution corresponds to a winning hat assignment that respects the node labels (i.e. each label (variable) is assigned the same colour) and any distinctness constraints while aligning each node's guess function with its actual hat. Conversely, every winning assignment in the hat-guessing game (i.e. one where all guesses are correct) yields a consistent solution to the diversified equations. Thus, determining the largest set of winning configurations is the same as finding the diversified system's maximum code/solution size.

1.6.6 Computing the Guessing Number

We measure entropy in *bits* (base 2). For a random variable X over an n-element set,

$$\widetilde{H}(X) \;=\; -\sum_{x} p(x)\,\log_2\big(p(x)\big).$$

A uniform variable X has $\widetilde{H}(X) = \log_2(n)$ bits. To simplify, we define the *normalised* entropy function

$$H(X) \;=\; \frac{\widetilde{H}(X)}{\log_2(n)},$$

so that a uniform variable has $H(X) = 1$. Equivalently,

$$H(X) \;\le\; 1 \quad\Longleftrightarrow\quad \widetilde{H}(X) \;\le\; \log_2(n).$$

In the guessing game for the above graph, each variable must be *functionally determined* by its in-neighbours. For instance,

$$H(w \mid x, y) = 0, \quad H(x \mid z, y) = 0, \quad \ldots$$

We also impose $H(x) \le 1$, $H(y) \le 1$, $H(z) \le 1$, $H(w) \le 1$, ensuring no variable exceeds uniform randomness over $\{0, \ldots, n-1\}$. Using subadditivity and these functional constraints, we obtain

$$H(x, y, z, w) \;=\; H(x, y, z) \;=\; H(y, z) \;\le\; H(z) \;+\; H(w) \;\le\; 2.$$

Hence, $H(x, y, z, w) \le 2$. Moreover, this bound is *achievable* by letting $z = x$ and $w = y$ while choosing x, y uniformly and independently, so $H(x, y, z, w) = 2$ is optimal.

From the guessing-game viewpoint, $H(x, y, z, w) = 2$ corresponds to a *guessing number* of 2, implying $\Theta(n^2)$ codewords/solutions. Thus, the guessing number for this graph is 2, and there are $\Omega(n^2)$ solutions—matching the intuition that two free variables can each take n values. When applied to the impossible self-orthogonal self-decoding Latin squares, this shows (when combined with Theorem **??**) that $C_2(n)$ is indeed bounded below by a constant. A similar argument can be applied to the refined term equations for self-orthogonal self-decoding Latin squares, showing that $C_8(n)$ is also bounded below by a constant.

For further background on these entropy-based methods in guessing games and their links to graph entropy, see, for example, [9, 18, 8]. Interestingly, as discovered in [25], it turns out that classical Shannon-type information inequalities are, in general, insufficient to determine the exact

upper bounds of certain guessing games, and specific graphs require *non-Shannon* inequalities [26] for tighter bounds.

In summary, this example illustrates several important techniques: *normalising* term equations, *diversifying* repeated function symbols, and interpreting the resulting system as a *graph guessing game* whose *guessing number* (equivalently, *graph entropy*) determines asymptotic solution counts.

1.6.7 Multi-Sorted Term Coding and Non-Equality Constraints

Although we initially focus on the case where *all variables range over a single n-element domain*, it is often useful to allow *multiple sorts* of variables (e.g. points, blocks, labels) in more advanced applications such as block designs, finite geometries, or complex coding scenarios. Moreover, one may naturally include non-equality constraints (e.g. $x \neq y$) to enforce distinctness between variables. In later sections (§4 and beyond), we show how the same bounding principles extend naturally to the *multi-sorted* setting and how optional *non-equality constraints* can be incorporated without disrupting the main arguments on guessing numbers and entropy.

1.6.8 Assumption on Consistency

When non-equality constraints are added, they may remove some solutions; however, as long as the overall system is consistent, such distinctness constraints can as shown by the techniques in Section 3 reduce the number of solutions by, at most, a constant (i.e., independent of the domain size) multiplicative factor. In other words, if the system (without non-equality constraints) has $\Theta(n^r)$ solutions, then adding consistent non-equality constraints also yields $\Theta(n^r)$ solutions (that is, the asymptotic exponent remains unchanged). For example, even if a non-equality constraint rules out a subset of assignments, it does so by a factor that does not grow with the domain size. Hence, we assume that our systems are chosen to be consistent; that is, we assume that at least one interpretation (for sufficiently large domain sizes) exists that satisfies all term equations and all non-equality constraints. Pathological cases, such as

$$s = t \quad \text{together with} \quad s \neq t,$$

which trivially yield no solutions, are excluded.

1.6.9 Overview and Contributions

Beyond the single-sorted case, we develop a *multi-sorted* version of Term Coding, which permits variables from different sorts (e.g. points, lines, blocks) and optional non-equality constraints. This broader approach enables us to formulate classic design–theoretic configurations, finite geometries, and mixed coding scenarios as optimisation problems. We also define *multi-sorted dispersion problems* – a particularly natural subclass that focuses on the number of distinct s-tuples that can be realised as the image of a map. We demonstrate that the dispersion framework is surprisingly expressive (indeed, the multi-sorted setup is capable of encoding full finite model theory), while admitting precise *entropy–based* bounds and exhibiting the same striking *undecidable–to–polynomial–time complexity jump* when moving from n^r to $n^r + 1$.

Moreover, the maximum code or dispersion size can be determined up to a constant factor by relating term equations to *graph guessing numbers*, thereby unifying the analysis of combinatorial code sizes. In the single-sorted case, the corresponding guessing number is always an integer with a value that can be determined in polynomial time in the size of the term equations.

1.7 Outline of the Paper

We begin in Section 2 by introducing the single-sorted Term Coding setup. There, we define how to *normalise* an arbitrary system of term equations (by isolating nested subterms) and, optionally, *diversify* repeated function symbols.

In Section 3, we develop the tools that link code sizes to the maximal number of winning configurations in a *guessing-game* defined on an associated directed graph. We prove supermultiplicative inequalities, establish a limiting exponent for large alphabets, and formulate the *main bounding theorem* connecting the maximum code size to the guessing number. Section 3.7 provides a concrete single-sorted example (the 5-cycle C_5) to demonstrate normalisation, diversification, and the treatment of non-equality constraints.

Section 4 extends the discussion to *multi-sorted Term Coding*, where different sorts (e.g. points, lines, pairs, blocks) each have their own domains. We incorporate *non-equality constraints* (e.g. $x \neq y$) and illustrate how block designs can be encoded in this setting.

Next, Section 5 formalises the *multi-sorted guessing number* and shows that the same supermultiplicative and convergence arguments hold with multiple sorts and distinctness constraints.

In Section 6, we focus on *dispersion problems*, a subclass of Term Coding where we measure how many distinct s-tuples can be realised. We discuss historical motivations, show how dispersion can encode Boolean gates and Steiner-type designs, and explain why it remains a *proper* subclass of Term Coding.

Section 7 then presents the *core complexity dichotomy* for single-sorted dispersion: deciding whether $\max_{\mathcal{I}}(\Gamma, n)$ can ever reach n^k is *undecidable*, yet deciding if it eventually exceeds $n^k + 1$ is solvable in polynomial time.

Finally, in Section 8, we summarise our main findings, highlight open problems, and discuss future directions at the intersection of combinatorics, Term Coding, and guessing-number arguments.

1.8 Notation and Conventions

We describe the main terminology and notation used throughout this paper for the reader's convenience.

- *Term Equation:* A *term equation* is an equality between two terms built from variables and function symbols in our quantifier-free, negation-free language. When we write

  $$s = t,$$

  we mean that under any given interpretation, the evaluation of the term s equals that of the term t. In our basic (single-sorted) setting, all variables range over a common finite domain (of size n). In the more general (multi-sorted) setting, variables are assigned to different sorts (or types), and each sort has its own finite domain.

- *Non-Equality (Distinctness) Constraint:* A non-equality constraint is an assertion that two terms (or two variables) are not equal. For example, when we write

  $$x \neq y,$$

  we require that x and y be assigned distinct values. In our framework, such constraints enforce distinctness where necessary (e.g., to ensure that different variables or subterms take distinct values). We assume that the non-equality constraints are chosen to be consistent with the term equations. However, they may remove some assignments; they do so only by

a constant multiplicative factor (with the constant independent of the domain size) and thus do not affect the asymptotic exponent of the solution count.

- *Interpretation:* An *interpretation* assigns meanings to the symbols of our language. In the single-sorted setting, an interpretation assigns each variable an element of a finite set A (with $|A| = n$) and each function symbol a function $f^{\mathcal{I}} : A^k \to A$, where k is the arity of the symbol. In the multi-sorted setting, each sort is assigned its own finite domain (say, $A_1, A_2, \ldots$), and function symbols are interpreted as functions whose domain and codomain respect the specified sorts.

- *Code:* Given a system Γ of term equations, an interpretation $\mathcal{I}$ yields a *code* (i.e., a set of solutions) defined by

$$C_{\mathcal{I}}(\Gamma) = \{\, (a_1, \ldots, a_v) \in A^v : s^{\mathcal{I}}(a_1, \ldots, a_v) = t^{\mathcal{I}}(a_1, \ldots, a_v) \text{ for all } s = t \in \Gamma \,\}.$$

  Our primary object of study is the maximum size of such a code:

$$\max_{\mathcal{I}}(\Gamma, n) = \max_{\mathcal{I}} \big| C_{\mathcal{I}}(\Gamma) \big|.$$

Unless stated otherwise, we work in the *single-sorted* setting. When we extend our framework to the *multi-sorted* case, we explicitly indicate that variables belong to different sorts and denote the corresponding domain sizes by $n_1, n_2, \ldots$. This distinction is maintained in our definitions, theorems, and examples.

2 Normalising Term Equations and non-equalities: From Arbitrary Systems to a Normalised Form

In previous examples (e.g., the stricter Latin square constraints in Section 1.5) and in later examples (e.g., the cycle graph C_5 example in Section 3.7), we illustrate how rewriting nested or repeated function symbols simplifies the analysis of term equations. We now describe how to transform any arbitrary system of term constraints Γ (which may include both equations and non-equality constraints) into an equivalent normalised system Γ' whose structure is easier to analyse. This process preserves the exact solution set, so every solution of Γ corresponds uniquely to one of Γ'.

The normalised system Γ' paves the way for the *diversification* step (Section 2.3), which further modifies Γ' to facilitate asymptotic bounds without changing the solution exponent. Here, our primary focus is on flattening term equations by introducing auxiliary variables; we then briefly note that non-equalities are handled similarly.

Single- vs. Multi-Sorted Context. Although this section is presented for a single-sorted scenario (where all variables range over the same domain), the procedure applies equally well in the multi-sorted setting (see Section 4). In the latter, auxiliary variables are introduced so that each constraint (regardless of sort) is rewritten in a consistent normal form (e.g., ensuring that each $f(\ldots) = x_j$ is sorted correctly).

Normalisation Overview. The goal is to replace every nested subterm with a fresh auxiliary variable so all constraints become flat. For example, an equation

$$f\big(g(x, y), h(y, x)\big) = t$$

is rewritten by introducing

$$z = g(x, y), \quad w = h(y, x),$$

so that it becomes

$$f(z, w) = t.$$

Non-equality constraints are handled analogously. For instance, in the non-equality

$$f(x, f(x, y)) \neq f(y, f(y, x)),$$

we first introduce auxiliary variables to flatten the compound terms, obtaining

$$\begin{aligned} f(x, y) &= v_1, \\ f(y, x) &= v_2, \\ f(x, v_1) &= v_3, \\ f(y, v_2) &= v_4, \\ v_3 &\neq v_4. \end{aligned}$$

(Notice that this atomic non-equality may imply additional constraints among the original variables, for example, forcing $x \neq y$.)

2.1 Normalisation: Retaining Equivalence of Solutions

Let

$$\Gamma \;=\; \{\, s_1 = t_1,\; s_2 = t_2,\; \ldots,\; s_m = t_m\}$$

be an arbitrary system of term equations over a language L with variables $x_1, \ldots, x_v$ and function symbols $f_1, \ldots, f_r$ (including 0-ary symbols $c_1, \ldots, c_u$). Our goal is to transform Γ into an equivalent system Γ' where every constraint is in a "flat" form—that is, with exactly one function symbol on the left-hand side and a single variable on the right.

In other words, we want to achieve the following:

> *Transform Γ into Γ' so that every equation appears as $f_*(x_{i_1}, \ldots, x_{i_k}) = x_j$ or $c = x_j$, and every non-equality appears as $x_i \neq x_j$ without changing the set of solutions.*

Definition 2.1 (Normalised Equation)**.** A normalised term equation is one of the form

$$f_*(x_{i_1}, \ldots, x_{i_k}) = x_j \quad \text{or} \quad c = x_j,$$

where f_* denotes a single (possibly newly introduced) function symbol and the right-hand side is a lone variable. In a multi-sorted setting, the types of the variables must match the function's signature.

Procedure

To normalise a constraint ϕ (either an equation or a non-equality) in Γ, proceed as follows:

1. *Isolate Nested Subterms.* Replace every nested function application with a fresh auxiliary variable. For example, rewrite

   $$f\big(g(x, y), h(y, x)\big)$$

   by introducing $z = g(x, y)$ and $w = h(y, x)$, so that it becomes $f(z, w)$. Apply this recursively until every function application is "flat."

2. *Handle Non-Equalities.* If ϕ is a non-equality (e.g. $f(t_1, \ldots, t_k) \neq f(s_1, \ldots, s_k)$), first introduce auxiliary variables for each side so that the expression becomes an atomic non-equality (e.g. $v_1 \neq v_2$), then normalise the subterms.

3. *Merge Trivial Equalities.* If any constraint reduces to $x_j = x_l$, substitute x_j for x_l (or vice versa) throughout the system and remove the trivial equation.

4. *Rewrite in Normal Form.* At the end, every constraint appears either as

$$f(\ldots) = x_j \quad \text{or} \quad c = x_j,$$

or as an atomic non-equality (e.g. $x_i \neq x_j$).

Since these steps only introduce auxiliary variables or merge existing ones, the transformed system Γ' is logically equivalent to Γ.

Proposition 2.2 (Preservation of Solutions)**.** *For any interpretation $\mathcal{I}$ into a domain A, there is a one-to-one correspondence between the solutions of Γ and those of Γ'; in particular,*

$$\max_{\mathcal{I}}(\Gamma, n) = \max_{\mathcal{I}}(\Gamma', n).$$

Proof. Each transformation step (isolating nested terms, handling non-equalities, and merging trivial equalities) preserves the solution set; hence, the overall system remains equivalent. □

Remark 2.1 (Connection to Diversification)**.** Normalisation "flattens" the system so that every equation appears as $f(\ldots) = x_j$ and every non-equality is atomic. This clarifies the dependency structure among variables. Later, the diversification step (see Section 2.3) will replace repeated function symbols with distinct ones—altering the total number of solutions only by a constant factor and leaving the asymptotic behaviour unchanged.

2.2 Example: Characterizing Steiner Systems $S(t, t+1, n)$

This example illustrates the role of non-equality constraints. Let t and k be positive integers with $k = t + 1$, and let n be a positive integer representing the number of points. A *Steiner system* $S(t, t+1, n)$ is a set M with $|M| = n$ (the points) and a collection of subsets of M, each of size $t+1$ (the blocks), such that every t-subset of M is contained in exactly one block. We characterize this structure using a single t-ary function $f : M^t \to M$ with the following axioms:

2.2.1 Axioms

- *Symmetry Axiom*: For every permutation σ of $\{1, 2, \ldots, t\}$,

$$f(x_1, x_2, \ldots, x_t) = f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(t)})$$

  This ensures that f depends only on the set $\{x_1, x_2, \ldots, x_t\}$, not the order of the arguments.

- *Inversion Axiom*: For any distinct $x_1, x_2, \ldots, x_t \in M$, let $z = f(x_1, x_2, \ldots, x_t)$. Then, for each $i = 1, 2, \ldots, t$,

$$f(x_1, \ldots, x_{i-1}, z, x_{i+1}, \ldots, x_t) = x_i$$

  This axiom allows f to "invert" by recovering any input when the output z replaces that input.

- *Non-Equality Axioms (for distinct inputs)*: The core structure of the Steiner system relies on the behaviour of f when applied to t distinct points. We require that for any set of t pairwise distinct points $\{x_1, x_2, \ldots, x_t\} \subset M$:

$$f(x_1, x_2, \ldots, x_t) \neq x_i \quad \text{for each } i = 1, 2, \ldots, t.$$

  This condition, combined with the inherent distinctness $x_i \neq x_j$ for $i \neq j$ within the input set, ensures that the block $\{x_1, \ldots, x_t, f(x_1, \ldots, x_t)\}$ always contains exactly $t + 1$ distinct points.

  Crucially, these axioms, along with the Symmetry and Inversion axioms, apply only when $x_1, \ldots, x_t$ are pairwise distinct. The definition or value of f when two or more arguments are identical is not constrained by the fundamental Steiner system properties (unless additional algebraic axioms like idempotence are imposed, as in the $t = 2$ case).

Proposition 2.3. *There is a one-to-one correspondence between Steiner systems $S(t, t+1, n)$ and models of size n satisfying the symmetry axiom, inversion axiom, and non-equality axioms for the t-ary function f.*

Proof. Part 1: Steiner System to Model
Given a Steiner system $\mathcal{S} = (M, \mathcal{B})$ with $|M| = n$:

- *Define f*: For distinct $x_1, x_2, \ldots, x_t \in M$, there is a unique block $B \in \mathcal{B}$ containing $\{x_1, x_2, \ldots, x_t\}$. Since $|B| = t + 1$, there is exactly one point $z \in B$ not in $\{x_1, x_2, \ldots, x_t\}$. Set:

$$f(x_1, x_2, \ldots, x_t) = z$$

- *Symmetry*: Since B depends only on the set $\{x_1, x_2, \ldots, x_t\}$, $f(x_1, x_2, \ldots, x_t) = f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(t)})$ for any permutation σ.

- *Inversion*: Let $z = f(x_1, x_2, \ldots, x_t)$, so $B = \{x_1, x_2, \ldots, x_t, z\}$. The t-set $\{x_1, \ldots, x_{i-1}, z, x_{i+1}, \ldots, x_t\}$ is in B, and the unique point in B not in this set is x_i. Thus:

$$f(x_1, \ldots, x_{i-1}, z, x_{i+1}, \ldots, x_t) = x_i$$

- *Non-Equality*: By definition, $z \notin \{x_1, x_2, \ldots, x_t\}$, and the inputs are distinct, so the axioms hold.

Thus, $\mathcal{S}$ defines a model satisfying the axioms.
Part 2: Model to Steiner System
Given a model M of size n with f satisfying the axioms:

- *Define Blocks*: For distinct $x_1, x_2, \ldots, x_t$, form the block:

$$B = \{x_1, x_2, \ldots, x_t, f(x_1, x_2, \ldots, x_t)\}$$

  The non-equality axioms ensure $f(x_1, x_2, \ldots, x_t) \neq x_i$, so $|B| = t + 1$.

- *Uniqueness of Blocks*: For any t-subset $S = \{x_1, x_2, \ldots, x_t\}$, $B = S \cup \{f(x_1, x_2, \ldots, x_t)\}$ by the symmetry axiom. If $S \subseteq B'$ (another block), let $B' = \{x_1, \ldots, x_{t-1}, y, z\}$ where $z = f(x_1, \ldots, x_{t-1}, y)$. The inversion axiom on B' implies $B' = B$, so each t-subset is in exactly one block.

Thus, the model defines a Steiner system $S(t, t+1, n)$.

Part 3: One-to-One Correspondence

The map from Steiner systems to models (via f) and from models to Steiner systems (via blocks) are inverses, as f is uniquely determined by the blocks and vice versa. Hence, there is a one-to-one correspondence. □

2.3 Diversification: An Approximation Technique

Once Γ' is normalised, the *diversification* step renames repeated occurrences of the same function symbol appearing in different contexts, assigning each occurrence a distinct symbol. By doing so, the dependency structure is decoupled, simplifying the analysis of asymptotic solution counts via guessing numbers (see Theorem 3.8). Importantly, diversification alters the solution set by at most a constant factor, preserving the asymptotic growth rate.

Illustrative Example

Suppose Γ' contains the equations

$$f(x, y) = z \quad \text{and} \quad f(y, x) = w.$$

In the diversified system Γ'', these equations become

$$f_1(x, y) = z \quad \text{and} \quad f_2(y, x) = w.$$

where f_1 and f_2 are distinct function symbols replacing the original f in these two different contexts.

The associated dependency graph $G_{\Gamma'}$ reflects that both z and w depend on the same function symbol f. In contrast, in $G_{\Gamma''}$, the two function symbols f_1 and f_2 are independent. This decoupling permits separate guessing strategies for z and w, typically increasing the overall number of solutions. Theorem 5.2, there exists a constant $c > 0$ such that for all n,

$$c \cdot \max_{\mathcal{I}}(\Gamma'', n) \leq \max_{\mathcal{I}}(\Gamma', n) \leq \max_{\mathcal{I}}(\Gamma'', n).$$

In summary:

- *Normalisation* transforms Γ into a flat system Γ' (see Section 2) without altering the solution set.
- *Diversification* renames function symbols to decouple dependencies, enabling tractable asymptotic bounds using guessing numbers, while preserving the solution count up to a constant multiplicative factor.

3 Foundations and Main Result

Having introduced the normalisation process in Section 2 and the fundamental notions of *term languages*, *interpretations*, and *codes*, we now present our key asymptotic bound (Theorem 3.8). Specifically, we connect the maximum code size of a system of term equations to the *guessing number* of an associated directed graph. While our proofs often assume a single-sorted setting (all variables share one domain), the same framework and main theorem naturally extend to *multi-sorted* Term Coding (see Section 4).

3.1 Associated Directed Graph G_Γ

To connect the number of solutions of a Term Coding system to guessing games and entropy measures, we associate a directed graph G_Γ to the system. This graph captures the functional dependencies between variables imposed by the equations. The construction relies on the *normalised* form of the system, as described in Section 2.

Definition 3.1 (Variable Dependency Graph)**.** Let Γ' be a system of term equations in variables $V = \{x_1, \ldots, x_v\}$ (including any auxiliary variables introduced during normalisation), where each equation is in the normalised form $f(x_{i_1}, \ldots, x_{i_k}) = x_j$ or $c = x_j$. The *variable dependency graph* is $G_\Gamma = (V, E)$ where:

- The vertex set is V, the set of all variables in the normalised system Γ'.
- For each equation of the form $f(x_{i_1}, \ldots, x_{i_k}) = x_j$ in Γ', we add the directed edges $(x_{i_p} \to x_j)$ for all $p \in \{1, \ldots, k\}$.
- Equations of the form $c = x_j$ mean x_j is determined by a constant, corresponding to a vertex with in-degree 0 in terms of variable dependencies.

Thus, in this graph, each variable x_j appearing on the right-hand side of a non-constant equation is functionally determined by its set of in-neighbours $N^-(x_j)$.

Note on Multi-Sorted Systems: In multi-sorted Term Coding (see Section 4), the vertex set V can be partitioned according to the sorts of the variables. The edge construction remains the same, respecting the function signatures. The fundamental idea that a variable is determined by its in-neighbours still holds.

Example: Illustration of G_Γ

Consider the system (before normalisation):

$$f(x, f(x, y)) \;=\; y, \quad f\big(f(x, y),\, y\big) \;=\; x.$$

During *normalisation*, we introduce an auxiliary variable $z = f(x, y)$, obtaining the equivalent system Γ':

$$f(x, z) = y, \quad f(z, y) = x, \quad z = f(x, y).$$

The variables in Γ' are $V = \{x, y, z\}$. Applying Definition 3.1:

- The equation $z = f(x, y)$ adds edges $(x \to z)$ and $(y \to z)$.
- The equation $f(x, z) = y$ adds edges $(x \to y)$ and $(z \to y)$.
- The equation $f(z, y) = x$ adds edges $(z \to x)$ and $(y \to x)$.

The resulting graph G_Γ is depicted in Figure 2.
As established by Theorem 3.8, the maximum code size $\max_{\mathcal{I}}(\Gamma, n)$ is asymptotically determined by the *guessing number* of this graph G_Γ, denoted $\mathrm{Guess}(G_\Gamma)$. This number reflects the maximum achievable exponent in the $\Theta(n^{\mathrm{Guess}(G_\Gamma)})$ bound (conditional on Conjecture 3.9).

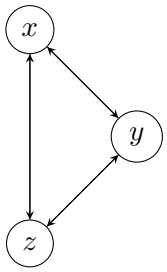

Figure 2: Variable dependency graph G_Γ for the normalised system derived from $f(x, f(x, y)) = y$ and $f(f(x, y), y) = x$, where $z = f(x, y)$.

3.2 Interpretations and Codes

Definition 3.2 (Interpretation)**.** Let A be a non-empty finite set (alphabet) with $|A| = n$. An *interpretation* $\mathcal{I}$ of L in A assigns:

- To each constant symbol c (0-ary function symbol) an element $c^{\mathcal{I}} \in A$.
- To each k-ary function symbol f a function $f^{\mathcal{I}} : A^k \to A$.

Given an assignment $(a_1, \ldots, a_v) \in A^v$ to the variables, any term t evaluates to an element $t^{\mathcal{I}}(a_1, \ldots, a_v) \in A$.

Multi-sorted Variant. In the multi-sorted case, $\mathcal{I}$ must provide a finite domain A_s for each sort s (so the total domain size might be distributed across sorts), and interpret each function symbol f consistently with its specified input and output sorts.

Definition 3.3 (Code of an Interpretation)**.** For Γ and an interpretation $\mathcal{I}$, the *code* defined by $\mathcal{I}$ is

$$C_{\mathcal{I}}(\Gamma) = \big\{(a_1, \ldots, a_v) \in A^v : s_i^{\mathcal{I}}(a_1, \ldots, a_v) = t_i^{\mathcal{I}}(a_1, \ldots, a_v) \text{ for all } s_i = t_i \in \Gamma\big\}.$$

We define $S_n = \max_{\mathcal{I}}(\Gamma, n) = \max_{\mathcal{I}}\big|C_{\mathcal{I}}(\Gamma)\big|$, the maximum code size over all interpretations into an alphabet of size n. In a multi-sorted scenario, we consider all interpretations where each sort's domain D has size n_D, and $S_{(n_D)_D}$ denotes the maximum size.

3.3 Guessing Numbers for Labelled Directed Graphs

The notion of a guessing game on a directed graph, together with its *guessing number*, provides a combinatorial measure closely related to the maximum code size S_n. This concept has been studied in various forms ([9, 8, 11]). Here, we adapt it to the variable dependency graph G_Γ, which is effectively a *labelled* directed graph, meaning that multiple vertices may correspond to the same underlying conceptual variable (though often normalisation gives distinct auxiliary variables), or constraints might force equality. The key connection is between solutions and correctly guessed configurations.

Definition 3.4 (Guessing Game)**.** Let $G = (V, E)$ be the variable dependency graph G_Γ, and let A be an alphabet (or "colour set") of size n.

- A *configuration* is an assignment $(a_v)_{v \in V} \in A^{|V|}$ of a value from A to each variable (vertex).
- A *deterministic guessing strategy* is a collection of local functions $\{\, G_v : A^{|N^-(v)|} \to A : v \in V\}$, where each G_v "guesses" the value a_v based only on the values $(a_u)_{u \in N^-(v)}$ of its in-neighbours. (For v with $N^-(v) = \emptyset$, G_v is a constant).

- A configuration $(a_v)_{v\in V}$ is *guessed correctly* by strategy $\{G_v\}$ if $a_v = G_v\big((a_u)_{u\in N^-(v)}\big)$ for all $v \in V$.
- The *set of correctly guessed configurations* for a strategy $\{G_v\}$ is $S_{\{G_v\}} = \{(a_v) \in A^{|V|} \mid \forall v \in V, a_v = G_v(\dots)\}$.

There is a direct correspondence between interpretations $\mathcal{I}$ of the *diversified* normalised system Γ'' (see Section 2.3) and guessing strategies $\{G_v\}$ for the associated dependency graph G_Γ. The set of solutions $C_\mathcal{I}(\Gamma'')$ corresponds precisely to the set of correctly guessed configurations $S_{\{G_v\}}$ for the strategy derived from $\mathcal{I}$. Let

$$S''_n = \max_{\mathcal{I}} |C_\mathcal{I}(\Gamma'')| = \max_{\{G_v\}} |S_{\{G_v\}}|$$

denote the maximum number of correctly guessable configurations (which equals the maximum code size for the diversified system Γ'') over an alphabet of size n.

We define the *normalised guessing number* of the graph G_Γ for alphabet size n as:

Definition 3.5 (Normalised Guessing Number)**.**

$$\mathrm{Guess}(G_\Gamma, n) \;=\; \log_n(S''_n).$$

This quantity measures the exponent achieved by the maximum number of winning configurations relative to the alphabet size. By definition, we have $S''_n = n^{\mathrm{Guess}(G_\Gamma,n)}$.

Recall that the maximum code size S_n for the original (potentially undiversified) system Γ relates to S''_n via $S_n = \Theta(S''_n)$. Therefore, the guessing number $\mathrm{Guess}(G_\Gamma, n)$ also determines the asymptotic behaviour of S_n, as its limit $L = \lim_{n\to\infty} \mathrm{Guess}(G_\Gamma, n)$ (whose existence we establish later) governs the growth $S_n = \Theta(n^L)$.

Example 3.1 (Bidirected Two-Vertex Guessing Game)**.** Consider the directed graph $G = (V, E)$ with two vertices v_1, v_2 (representing variables x_1, x_2) and edges $(v_1 \to v_2)$ and $(v_2 \to v_1)$. This corresponds to normalised equations like $f_1(x_2) = x_1$ and $f_2(x_1) = x_2$ in the diversified system Γ''. Each variable takes a value from A, $|A| = n \geq 2$.

A Good Strategy (Optimal): Let the guessing functions be $G_{v_1}(a) = a$ and $G_{v_2}(a) = a$. A configuration (a_1, a_2) is guessed correctly if $a_1 = G_{v_1}(a_2) = a_2$ and $a_2 = G_{v_2}(a_1) = a_1$. The set of correctly guessed configurations is $S_{\{G_v\}} = \{(a, a) \mid a \in A\}$, with $|S_{\{G_v\}}| = n$. This strategy yields the maximum possible size, so $S''_n = n$. Thus, the normalised guessing number is $\mathrm{Guess}(G, n) = \log_n(S''_n) = \log_n(n) = 1$.

A Poor Strategy (Yielding No Solutions): Since an optimal strategy exists that performs better than average or random guessing, there must also be strategies that perform worse. To illustrate, consider a fundamentally inconsistent strategy. Let $G_{v_1}(a_2) = a_2$ (identity). Let $G_{v_2}(a_1) = \sigma(a_1)$, where $\sigma : A \to A$ is a fixed-point-free permutation (such permutations exist for $|A| \geq 2$; e.g., a cyclic shift $a \mapsto a + 1 \pmod n$). For a configuration (a_1, a_2) to be guessed correctly, we require: 1. $a_1 = G_{v_1}(a_2) = a_2$. 2. $a_2 = G_{v_2}(a_1) = \sigma(a_1)$. Substituting (1) into (2) yields $a_1 = \sigma(a_1)$. However, since σ has no fixed points, this equation has no solution in A. Therefore, the set of correctly guessed configurations for this strategy is empty: $S_{\{G_v\}} = \emptyset$, and $|S_{\{G_v\}}| = 0$.

The guessing number uses the strategy that maximises $|S_{\{G_v\}}|$, which is the "good" strategy yielding $S''_n = n$.

The guessing number uses the strategy that maximises $|S_{\{G_v\}}|$, which is $S''_n = n$ in this case.

Limit and Convergence. We are interested in the limit $L = \lim_{n\to\infty} \mathrm{Guess}(G_\Gamma, n)$. We will show this limit exists and state a conjecture about its rate of convergence.

3.4 Existence of the Limit via Fekete's Lemma

The existence of the limit $L = \lim_{n\to\infty} \text{Guess}(G_\Gamma, n)$ follows from a standard product construction and Fekete's Lemma. First, we establish a supermultiplicativity property.

Proposition 3.6 (Supermultiplicativity of Solution Counts)**.** *Let $S_n = \max_{\mathcal{I}} |C_{\mathcal{I}}(\Gamma)|$ be the maximum code size for the original system Γ, and let $S''_n = \max_{\mathcal{I}} |C_{\mathcal{I}}(\Gamma'')|$ be the maximum code size for the diversified system Γ''. For any integers $n_1, n_2 \geq 1$, both quantities are supermultiplicative:*

$$S_{n_1 n_2} \geq S_{n_1} \times S_{n_2} \quad \textit{and} \quad S''_{n_1 n_2} \geq S''_{n_1} \times S''_{n_2}.$$

Proof. We prove it for S_n; the proof for S''_n is identical, just replacing Γ with Γ''. Let $\mathcal{I}_1$ be an interpretation over A_1 ($|A_1| = n_1$) achieving $|C_{\mathcal{I}_1}(\Gamma)| = S_{n_1}$, and $\mathcal{I}_2$ be an interpretation over A_2 ($|A_2| = n_2$) achieving $|C_{\mathcal{I}_2}(\Gamma)| = S_{n_2}$. Define a product interpretation $\mathcal{I}_{12}$ over the alphabet $A = A_1 \times A_2$ (of size $n_1 n_2$). For a function symbol f (of arity k), define $f^{\mathcal{I}_{12}}\big((a_1^{(1)}, a_1^{(2)}), \ldots, (a_k^{(1)}, a_k^{(2)})\big) = (f^{\mathcal{I}_1}(a_1^{(1)}, \ldots, a_k^{(1)}), f^{\mathcal{I}_2}(a_1^{(2)}, \ldots, a_k^{(2)}))$. If $(a_1^{(1)}, \ldots, a_v^{(1)})$ is a solution for $\mathcal{I}_1$ and $(a_1^{(2)}, \ldots, a_v^{(2)})$ is a solution for $\mathcal{I}_2$, consider the combined tuple $\mathbf{a}_{12} = \big((a_1^{(1)}, a_1^{(2)}), \ldots, (a_v^{(1)}, a_v^{(2)})\big) \in A^v$. For any term equation $s = t$ in Γ, evaluating s and t under $\mathcal{I}_{12}$ at $\mathbf{a}_{12}$ yields: $s^{\mathcal{I}_{12}}(\mathbf{a}_{12}) = (s^{\mathcal{I}_1}(\mathbf{a}_1), s^{\mathcal{I}_2}(\mathbf{a}_2))$ and $t^{\mathcal{I}_{12}}(\mathbf{a}_{12}) = (t^{\mathcal{I}_1}(\mathbf{a}_1), t^{\mathcal{I}_2}(\mathbf{a}_2))$. Since $\mathbf{a}_1$ and $\mathbf{a}_2$ are solutions, $s^{\mathcal{I}_1}(\mathbf{a}_1) = t^{\mathcal{I}_1}(\mathbf{a}_1)$ and $s^{\mathcal{I}_2}(\mathbf{a}_2) = t^{\mathcal{I}_2}(\mathbf{a}_2)$. Thus $s^{\mathcal{I}_{12}}(\mathbf{a}_{12}) = t^{\mathcal{I}_{12}}(\mathbf{a}_{12})$, meaning $\mathbf{a}_{12}$ is a solution for $\mathcal{I}_{12}$. This construction yields $S_{n_1} \times S_{n_2}$ distinct solutions over A. Thus, $S_{n_1 n_2} = \max_{\mathcal{I}} |C_{\mathcal{I}}(\Gamma)| \geq |C_{\mathcal{I}_{12}}(\Gamma)| \geq S_{n_1} \times S_{n_2}$. □

Now we apply Fekete's Lemma to S''_n, the quantity used to define the guessing number. The supermultiplicativity $S''_{nm} \geq S''_n S''_m$ implies $\log S''_{nm} \geq \log S''_n + \log S''_m$.

Define the sequence $a_k = \log S''_{n_0^k}$ for some integer base $n_0 \geq 2$. The supermultiplicativity implies $a_{k+l} = \log S''_{n_0^{k+l}} \geq \log(S''_{n_0^k} S''_{n_0^l}) = a_k + a_l$. Thus, $\{a_k\}$ is a superadditive sequence. By Fekete's Lemma for superadditive sequences, the limit $\lim_{k\to\infty} a_k/k$ exists and equals $\sup_k a_k/k$. Let this limit be L'.

Recall our definition $\text{Guess}(G_\Gamma, n) = \log_n(S''_n) = \frac{\log S''_n}{\log n}$. For the subsequence $n = n_0^k$, we have

$$\text{Guess}(G_\Gamma, n_0^k) = \frac{\log S''_{n_0^k}}{k \log n_0} = \frac{a_k}{k \log n_0}.$$

Therefore, the limit along this subsequence exists:

$$\lim_{k\to\infty} \text{Guess}(G_\Gamma, n_0^k) = \frac{L'}{\log n_0}.$$

Let this limit be L. Standard arguments involving the relationship between S''_n and $S''_{\lfloor n^{1/k} \rfloor^k}$ show that the limit exists not just along subsequences n_0^k but for $n \to \infty$ through integers. Thus, the limit

$$L = \lim_{n\to\infty} \text{Guess}(G_\Gamma, n) = \lim_{n\to\infty} \frac{\log S''_n}{\log n}$$

exists. This limit L represents the asymptotic exponent governing the growth of the maximum number of solutions for the diversified system, $S''_n = n^{L+o(1)}$, and consequently (since $S_n = \Theta(S''_n)$), also for the original system Γ, $S_n = \Theta(n^L)$.

Historical Note. The existence of the limit for the normalised guessing number,

$$L = \lim_{n\to\infty} \text{Guess}(G_\Gamma, n)$$

, was established previously. It was stated without proof in early work [10] and later rigorously proved using arguments based on conflict graphs [8]. The derivation presented here, using Fekete's Lemma applied directly to the supermultiplicativity of the maximum solution count S''_n (established via a product interpretation construction), offers a more direct alternative proof.

3.5 Relating Original and Diversified Systems

Before presenting the main bounds in terms of the guessing number, we formalise the relationship between the maximum code size $S_n = \max_{\mathcal{I}} |C_{\mathcal{I}}(\Gamma)|$ of the original system Γ and the maximum code size $S''_n = \max_{\mathcal{I}} |C_{\mathcal{I}}(\Gamma'')|$ of its normalised and diversified version Γ''. Recall that normalisation (Section 2) produces an equivalent system Γ' with $S_n(\Gamma) = S_n(\Gamma')$. Diversification transforms Γ' into Γ'' by assigning a unique function symbol f_i to each distinct function application (equation) present in Γ'. Let k be the number of distinct variables in the normalised system Γ'.

The relationship relies on domain partitioning arguments. We illustrate the core idea of the lower bound construction with the unsolvable self-decoding Latin square example from Section 1.5.

Example: Lower Bound for the Unsolvable SOLS Variant

Recall the normalised system Γ' for this example, involving variables x, y, z, w ($k = 4$) and a single function symbol f:

$$f(x,y) = z, \quad f(y,x) = w, \quad f(z,y) = x, \quad f(x,w) = y, \quad f(z,w) = x, \quad f(w,z) = y.$$

The diversified system Γ'' uses distinct functions $f_1, \ldots, f_6$ for each equation. Let $S''_m = \max_{\mathcal{I}} |C_{\mathcal{I}}(\Gamma'')|$ over an alphabet $\tilde{A}$ of size m.

To obtain a lower bound for $S'_n = \max_{\mathcal{I}} |C_{\mathcal{I}}(\Gamma')|$ over alphabet A of size n, we partition A into $k = 4$ disjoint subsets A_x, A_y, A_z, A_w, each of size $m \approx \lfloor n/4 \rfloor$. Let $\mathcal{I}''^*$ be an optimal interpretation for Γ'' over $\tilde{A}$. We construct an interpretation $\mathcal{I}'$ for Γ' over A by defining the single function $f^{\mathcal{I}'}$ piece-wise, mimicking the behaviour of $\mathcal{I}''^*$. For instance, for inputs $(a, b) \in A_x \times A_y$, we define $f^{\mathcal{I}'}(a, b)$ to be the element in A_z corresponding (via a fixed bijection $\phi_z : \tilde{A} \to A_z$) to $f_1^{\mathcal{I}''^*}(\phi_x^{-1}(a), \phi_y^{-1}(b))$. Similarly, for $(a, b) \in A_y \times A_x$, $f^{\mathcal{I}'}(a, b)$ lands in A_w mimicking $f_2^{\mathcal{I}''^*}$, and so on for all 6 equations.

Crucially, the input domains for these 6 required mappings ($A_x \times A_y$, $A_y \times A_x$, $A_z \times A_y$, $A_x \times A_w$, $A_z \times A_w$, $A_w \times A_z$) are disjoint subsets of $A \times A$ because the partitions $A_x, \ldots, A_w$ are disjoint. Thus, the piece-wise definition of $f^{\mathcal{I}'}$ is consistent. Any solution $(\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w})$ to Γ'' under $\mathcal{I}''^*$ maps via $\phi_x, \ldots, \phi_w$ to a solution (x, y, z, w) for Γ' under $\mathcal{I}'$ with $x \in A_x$, etc. This yields at least S''_m solutions for Γ'. Therefore, $S'_n \geq S''_{\lfloor n/4 \rfloor}$. This illustrates how partitioning by the $k = 4$ variables provides a lower bound with scaling constant $c = k$.

General Equivalence Lemma

This construction generalises. Let k be the number of distinct variables in the normalised system Γ'.

Lemma 3.7 (Asymptotic Equivalence of $S_n(\Gamma')$ and S''_n)**.** *Let Γ' be a normalised system of term equations with k distinct variables, and let Γ'' be its diversification. Let $S'_n = \max_{\mathcal{I}} |C_{\mathcal{I}}(\Gamma')|$ and*

$S''_n = \max_{\mathcal{I}} |C_{\mathcal{I}}(\Gamma'')|$. *Then* $S'_n = \Theta(S''_n)$. *More precisely, there exist constants* $K_1, K_2 > 0$ *such that for sufficiently large* n*:*

(a) $S'_n \leq S''_n$

(b) $S'_n \geq K_2 \cdot S''_{\lfloor n/k \rfloor}$

(c) $S''_n \leq K_1 \cdot S'_{\lceil n/N_{occ} \rceil}$

Consequently, $\lim_{n\to\infty} \frac{\log S'_n}{\log n} = \lim_{n\to\infty} \frac{\log S''_n}{\log n}$*, if the limits exist.*

Proof. (a) *Upper bound ($S'_n \leq S''_n$):* As before, any interpretation for Γ' yields one for Γ''. Maximising over interpretations gives $S'_n \leq S''_n$.

(b) *Lower bound ($S'_n \geq K_2 \cdot S''_{\lfloor n/k \rfloor}$):* Let k be the number of distinct variables in Γ'. Let $m = \lfloor n/k \rfloor$. Consider an optimal interpretation $\mathcal{I}''^*$ for Γ'' over $\tilde{A}$ (size m), achieving S''_m solutions. Partition A (size n) into k disjoint subsets $A_1, \ldots, A_k$, each corresponding to a variable x_i and having size $\geq m$. Define bijections $\phi_i : \tilde{A} \to A_i$. Construct $\mathcal{I}'$ for Γ' over A. Let f be a function symbol in Γ'. For each equation $f^{(j)}(x_{p_1}, \ldots, x_{p_r}) = x_q$ in Γ' (corresponding to $f_j(x_{p_1}, \ldots) = x_q$ in Γ''), define the action of $f^{\mathcal{I}'}$ on inputs $(a_1 \in A_{p_1}, \ldots, a_r \in A_{p_r})$ as $f^{\mathcal{I}'}(a_1, \ldots, a_r) = \phi_q(f_j^{\mathcal{I}''^*}(\phi_{p_1}^{-1}(a_1), \ldots, \phi_{p_r}^{-1}(a_r)))$. This ensures the output lands in the correct partition A_q. Is this consistent if f appears in multiple equations, say $f^{(j)}$ and $f^{(l)}$? The definition depends on the index j or l of the equation it appears in. Since f is a single symbol in Γ', we define its action based on the *context* (which variables are in which positions, corresponding to a specific equation index j). As the domains $A_{p_1} \times \cdots \times A_{p_r}$ are effectively separated by the partitioning (even if some A_p repeats), we can define $f^{\mathcal{I}'}$ piece-wise based on the structure of the input tuple relative to the equations in Γ'. This construction ensures solutions to Γ'' map to solutions to Γ', giving $S'_n \geq K_2 \cdot S''_{\lfloor n/k \rfloor}$ (where K_2 handles floors, etc.).

(c) *Upper bound ($S''_n \leq K_1 \cdot S'_{\lceil n/N_{occ} \rceil}$):* The argument here typically involves partitioning based on the N_{occ} function occurrences, rather than variables, to show that the freedom of Γ'' doesn't add too much. We omit the details (see e.g., [8]).

Limit Equivalence: Follows from bounds (a) and (b) as argued previously (using S''_n in the denominator). □

Remark 3.1 (Handling Non-Equalities)**.** The arguments above assume only term equations. If the original system Γ also contains non-equality constraints $x_i \neq x_j$, these are carried through to Γ' and Γ''. The supermultiplicativity (Prop. 3.6) still holds. The domain partitioning construction in Lemma 3.7(b) needs slight modification: the mapping of solutions from Γ'' (on $\tilde{A}$) to Γ' (on A) must ensure that if x_i and x_j are required to be distinct in Γ', their images $a_i \in A_i$ and $a_j \in A_j$ are distinct. Since A_i and A_j are disjoint by construction for $i \neq j$, this is automatically satisfied. If a constraint involves terms, e.g., $f(x_1) \neq x_2$, the construction must ensure the corresponding outputs differ. This can typically be accommodated, potentially by refining the partitions or adjusting constants, without changing the Θ relationship. Thus, $S_n(\Gamma) = \Theta(S''_n(\Gamma''))$ holds even with consistent non-equality constraints.

3.6 Main Theorem: Finite Bounds for Code Size

We now connect the maximal code size of the diversified system S''_n to the guessing number defined in Section 3.3. Combined with Lemma 3.7, this provides bounds for the original system S_n.

Theorem 3.8 (Finite-n Bounds via Guessing Number)**.** *Let Γ be the original system (possibly with non-equalities), let Γ'' be its normalised and diversified version, let $G_{\Gamma''}$ be its dependency graph, and let $S_n = \max_{\mathcal{I}} |C_{\mathcal{I}}(\Gamma)|$ and $S''_n = \max_{\mathcal{I}} |C_{\mathcal{I}}(\Gamma'')|$. Let $\mathrm{Guess}(G_{\Gamma''}, n) = \log_n(S''_n)$.*

(a) (Definition) By definition, $S''_n = n^{\mathrm{Guess}(G_{\Gamma''},n)}$.

(b) (Bounds for Original System) There exist constants $K_2 > 0$ and $k \geq 1$ (number of variables in Γ') such that for sufficiently large n:

$$K_2 \cdot (\lfloor n/k \rfloor)^{\mathrm{Guess}(G_{\Gamma''},\lfloor n/k \rfloor)} \;\leq\; S_n \;\leq\; n^{\mathrm{Guess}(G_{\Gamma''},n)}.$$

Proof. (a) This is Definition 3.5.

(b) The upper bound $S_n \leq S''_n$ follows from Lemma 3.7(a). Substituting the definition from (a) gives $S_n \leq n^{\mathrm{Guess}(G_{\Gamma''},n)}$. The lower bound $S_n \geq K_2 \cdot S''_{\lfloor n/k \rfloor}$ follows from Lemma 3.7(b) (with $c = k$). Substituting $S''_m = m^{\mathrm{Guess}(G_{\Gamma''},m)}$ with $m = \lfloor n/k \rfloor$ gives the stated lower bound $S_n \geq K_2 \cdot (\lfloor n/k \rfloor)^{\mathrm{Guess}(G_{\Gamma''},\lfloor n/k \rfloor)}$. The remark above notes these bounds extend to systems with non-equalities. □

Convergence Rate. While Fekete's Lemma guarantees the existence of the limit $L = \lim_{n\to\infty} \mathrm{Guess}(G_{\Gamma''}, n)$, it provides no information on the speed of convergence. Obtaining the precise asymptotic behaviour $S_n = \Theta(n^L)$ requires understanding this rate. Based on analyses of related graph parameters and entropy convergence (e.g., [Relevant Citation(s) if available]), it is often observed or conjectured that the convergence is relatively fast. We adopt this as a working hypothesis:

Conjecture 3.9 (Fast Convergence Rate)**.** *Let $L = \lim_{n\to\infty} \mathrm{Guess}(G_{\Gamma''}, n)$. The convergence to the limit satisfies:*

$$L - \mathrm{Guess}(G_{\Gamma''}, n) = L - \log_n(S''_n) = O\left(\frac{1}{\log n}\right).$$

If this convergence rate holds, we can combine it with the bounds in Theorem 3.8 to establish:

Corollary 3.10 (Asymptotic behaviour (Conditional))**.** *Under the assumptions of Conjecture 3.9, the maximum code size for the original system Γ satisfies:*

$$\max_{\mathcal{I}}(\Gamma, n) = S_n = \Theta\Big(n^L\Big).$$

3.7 A Single-Sorted Example: The Cycle C_5 with non-equality Constraints

Remark on Single-Sorted Scope. Although our main theorems accommodate *multi-sorted* frameworks (and allow non-equality constraints among same–sort variables), the core steps—normalisation, diversification, and the guessing–game interpretation—are nicely illustrated by a small single–sorted example. Below, we show how a system of three main term equations (plus two non-equality constraints for illustration) yields the 5–node cycle graph C_5 under a guessing interpretation. (It is known that the guessing number $\mathrm{Guess}(C_5, n)$, and hence the graph entropy $E(C_5, n)$, approaches 2.5 as $n \to \infty$.)

Example 3.2 (The C_5 System with non-equality Constraints)**.** Consider the following system over variables (x, y, z, w, v), all ranging over the same single–sorted domain A of size n. Let $f : A\times A \to A$ be our function symbol, and impose

$$\Gamma \;=\; \Big\{ f\big(f(z,x),\, y\big) = x, \quad f\big(x, f(y,z)\big) = y, \quad f\big(f(y,z),\, f(z,x)\big) = z \Big\},$$

and

$$\Delta = \Big\{ x \neq z, \quad f(x,y) \neq f(y,x) \Big\}.$$

Formally, we seek an interpretation $\mathcal{I}$ of f (plus assignments $(x,y,z) \in A^3$) and we wish to estimate

$$\max_{\mathcal{I}} \Big| \{ (x,y,z) \in A^3 : \Gamma \cup \Delta \quad \text{holds} \} \Big|.$$

It will turn out that this number grows on the order of $n^{2.5}$.

3.7.1 Step 1: Normalising the System

First, isolate every nested appearance of $f(\cdot,\cdot)$. For instance, in the equation

$$f\big(f(z,x),\, y\big) = x,$$

introduce a fresh variable $\alpha = f(z,x)$ so that it becomes $f(\alpha, y) = x$. Repeating similarly for the other equations yields a normalised system:

$$\begin{aligned} \Gamma' = \Big\{ & f(z,x) = \alpha, \quad f(y,z) = \beta, \quad f(\alpha,y) = x, \\ & f(x,\beta) = y, \quad f(\beta,\alpha) = z, \quad f(x,y) = \gamma, \\ & f(y,x) = \delta \Big\}. \end{aligned}$$

and

$$\Delta' = \Big\{ \gamma \neq \delta, \quad x \neq z, \quad x \neq y \Big\}.$$

Here, the non-equality $f(x,y) \neq f(y,x)$ has been normalised into $\gamma \neq \delta$, and we retain the explicit constraint $x \neq z$. (The implicit $x \neq y$ arising from $f(x,y) \neq f(y,x)$ is displayed for clarity.)

3.7.2 Step 2: Diversifying f

Next, we diversify each occurrence of f so that each distinct pair of input variables gets a unique function symbol. Concretely, we rename:

$$\begin{aligned} & f_1(z,x) = \alpha, \quad f_2(y,z) = \beta, \quad f_3(\alpha,y) = x, \\ & f_4(x,\beta) = y, \quad f_5(\beta,\alpha) = z, \quad f_6(x,y) = \gamma, \quad f_7(y,x) = \delta. \end{aligned}$$

Also, we include the original non-equality constraints $x \neq z$ and $f(x,y) \neq f(y,x)$ (the latter now becomes $\gamma \neq \delta$). The resulting system

$$\Gamma'' = \left\{ \begin{array}{ll} f_1(z,x) = \alpha, & f_2(y,z) = \beta, \quad f_3(\alpha,y) = x, \\ f_4(x,\beta) = y, & f_5(\beta,\alpha) = z, \\ f_6(x,y) = \gamma, & f_7(y,x) = \delta, \\ \gamma \neq \delta, \quad x \neq z, \quad x \neq y & \end{array} \right\}$$

changes the solution set only by a constant factor asymptotically. Its key advantage is that the dependency graph now consists of the original 5–node cycle on $\{x,y,z,\alpha,\beta\}$ (see Figure 3 below) together with four additional nodes labelled x, y, γ, and δ (with directed edges from x and y to γ and δ) representing the diversified output constraints.

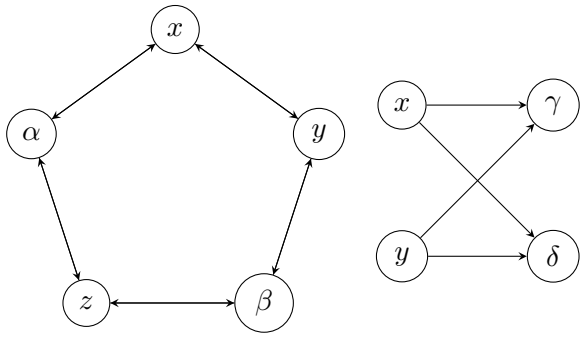

Figure 3: Left: A bidirected pentagon representing the dependency graph on $\{x, y, z, \alpha, \beta\}$. Right: Four additional nodes x, y, γ and δ with directed edges from x and y to γ and δ.

3.7.3 Step 3: Evaluating the Graph Guessing Number

A guessing–game viewpoint on this cycle shows that the size of any code consistent with Γ'' grows as $\Theta(n^{2.5})$ for large n. (For details, see Theorem 3.8 and related references.)

3.8 The 5-Cycle C_5 and Its Entropy of 2.5

The upper and lower bound arguments presented here were first proved in [10]. Consider the directed 5–cycle C_5 with vertices $\{1, 2, 3, 4, 5\}$, where each vertex i observes the two neighbouring variables x_{i-1} and x_{i+1} (indices modulo 5). Define the normalised entropy of a code $\mathcal{C} \subseteq \{1, \ldots, n\}^5$ as

$$E(C_5, n) = \log_n |\mathcal{C}|,$$

so that a uniform assignment yields an entropy of 1 per variable. One shows that

$$E(C_5) = \sup_n \{\, E(C_5, n)\} = 2.5.$$

Upper Bound.

Standard Shannon inequalities and elimination of dependent variables yield

$$H(1, 2, 3, 4, 5) \leq 2.5.$$

Achievability.

Assume $n = m^2$. Identify each variable x_i with an ordered pair (x'_i, x''_i) where $x'_i, x''_i \in \{1, \ldots, m\}$. Label the vertices of the 5–cycle cyclically by $1, \ldots, 5$. At vertex i, define the local guessing function by

$$G_i\Big((x'_{i-1}, x''_{i-1}), (x'_{i+1}, x''_{i+1})\Big) \;=\; \big(x''_{i-1},\, x'_{i+1}\big).$$

A straightforward verification shows that if all vertices follow this rule, the resulting configuration is correct if and only if the cyclic consistency conditions hold, yielding exactly m^5 valid codewords. Since $m^5 = (m^2)^{2.5} = n^{2.5}$, the upper bound is achieved.

General Lower Bound.

For general n, our convergence results guarantee that

$$\max_{\mathcal{I}}(\Gamma'', n) = \Omega(n^{2.5}),$$

so the asymptotic behaviour is indeed governed by an entropy of 2.5.

In this example, the normalised and diversified system yields a dependency graph that is essentially a 5–node cycle (see Figure 3 in Section 3.7.2). This structure underlies the calculation of the guessing number, which, as shown, corresponds to a normalised entropy of 2.5.

4 Multi-Sorted Term Coding with Non-Equality Constraints

In many combinatorial problems, it is natural to partition variables into *multiple sorts*, each ranging over a distinct finite domain (e.g. points, blocks, or colours). Additionally, one often requires not only *term equations* but also *non-equality constraints* (for instance, $x \neq y$) to ensure that certain variables or terms differ. Such a *multi-sorted* language with non-equalities extends Term Coding beyond the purely single-sorted, equation-based setting, enabling us to encode more complex structures such as Steiner designs, finite geometries, or colour-labelling schemes.

4.1 Multi-Sorted Term Languages and Non-Equalities

Sorts.

A *multi-sorted term language* partitions the variable set $x_1, \ldots, x_v$ into *sorts*. For example, one might define:

- Points-sort (for elements in a geometry or block design),
- Blocks-sort (for k-element subsets or lines),
- $\mathsf{Colours}$-sort (for label or colour sets),
- etc.

Each *function symbol* f is typed to map from a *tuple of sorts* to a single *output sort*. For instance,

$$f : \mathsf{Points} \times \mathsf{Points} \longrightarrow \mathsf{Blocks},$$

or a Boolean function

$$f : \mathsf{Points} \times \mathsf{Points} \longrightarrow \{\text{True}, \text{False}\}.$$

An *interpretation* $\mathcal{I}$ for this multi-sorted language must provide a finite domain A_s for each sort s (e.g. $|\mathsf{Points}| = n$, $|\mathsf{Blocks}| = q$), and interpret every function symbol f by a function $f^{\mathcal{I}}$ whose domain and codomain match the declared sorts.

Terms and Term Equations.

A *term* in the multi-sorted context is constructed by applying function symbols to variables (or subterms), ensuring that the input sorts match each function's signature so that the term itself acquires the function's output sort. A *term equation* is a constraint $s = t$ stipulating that for every assignment of variables

$$s^{\mathcal{I}}(a_1, \ldots, a_v) \;=\; t^{\mathcal{I}}(a_1, \ldots, a_v),$$

where s and t are of the same sort. This is analogous to the single-sorted case, but with careful tracking of sorts.

Non-Equality Constraints.

In addition, we now allow *non-equality constraints*. Specifically, for variables (or terms) x_i and x_j of the same sort, one may write

$$x_i \;\neq\; x_j \quad \text{or} \quad f(\ldots) \;\neq\; g(\ldots),$$

demanding that these two terms (of the same sort) evaluate to different domain elements under any valid interpretation. This is common in, for instance, Steiner-type problems (to ensure distinct points in a block) or colourability contexts (forcing different vertices not to share the same colour).

Henceforth, a *multi-sorted system of term constraints* refers to a finite set of *term equations* ($s = t$) and optional *non-equality constraints* ($s \neq t$), all quantifier-free and negation-free. In subsequent subsections we shall illustrate how block designs or finite geometries can be naturally encoded in this multi-sorted setting, analogous to the single-sorted case in Section **??**, but now with multiple domains and additional non-equality constraints.

4.2 Expressive Power: Encoding First-Order Finite Satisfiability

Having defined multi-sorted Term Coding systems involving both term equations and non-equality constraints, we now establish their significant expressive power. Specifically, we show that the problem of determining whether any first-order sentence has a finite model can be reduced to determining if an effectively constructible multi-sorted Term Coding system has a model. This result connects our framework directly to the foundations of finite model theory [27] and highlights its capability to capture complex logical properties.

Theorem 4.1 (Encoding FO Finite Satisfiability)**.** *For any first-order sentence ψ over a finite relational signature Σ, one can effectively construct a multi-sorted Term Coding system Γ_ψ, consisting of term equations and non-equality constraints over a signature Σ' (derived from Σ) and using auxiliary sorts (including* Bool*), such that: ψ has a finite model M with domain D ($|D| = n \geq 2$) if and only if Γ_ψ has a model $\mathcal{I}$ where the interpretation of the primary sort corresponding to D has size n, and the auxiliary sorts have fixed, standard interpretations (e.g., $|\mathsf{Bool}| = 2$).*

Proof. The proof proceeds via standard logical transformations, translating the first-order sentence into a set of quantifier-free term constraints.

Step 1: Skolemisation. Given the FO sentence ψ over signature Σ, we first transform it into a sentence ψ_{Sk} in Skolem normal form. This sentence is purely universal, $\psi_{Sk} \equiv \forall x_1 \ldots \forall x_m\, \phi(x_1, \ldots, x_m)$, where ϕ is quantifier-free. The signature Σ_{Sk} may contain new function symbols (Skolem functions) compared to Σ. Crucially, ψ has a finite model if and only if ψ_{Sk} has a finite model

[**hodges1993model**]. The satisfiability problem is thus reduced to finding a model for the universal sentence ψ_{Sk}. Such a model interprets the original sorts from Σ and provides interpretations for the Skolem functions over the domain(s).

Step 2: Conjunctive Normal Form (CNF). The quantifier-free matrix $\phi(x_1, \ldots, x_m)$ can be converted into an equivalent formula ϕ_{CNF} in conjunctive normal form using standard logical equivalences. ϕ_{CNF} is a conjunction of clauses $C_1 \wedge C_2 \wedge \cdots \wedge C_p$, where each clause C_i is a disjunction of literals $L_{i1} \vee L_{i2} \vee \cdots \vee L_{iq_i}$. Each literal L_{ij} is either an atomic formula (like $R(t_1, \ldots, t_a)$ or $t_1 = t_2$) or a negated atomic formula.

Step 3: Handling Negation and Introducing Boolean Sort. We introduce an auxiliary sort Bool with two designated distinct constants, T (True) and F (False). We add the non-equality constraint $EQ_{Bool}(T, F) = F$, where EQ_{Bool} is the equality function for the Bool sort. We also introduce standard Boolean function symbols $AND : \mathsf{Bool}^2 \to \mathsf{Bool}$, $OR : \mathsf{Bool}^2 \to \mathsf{Bool}$, $NOT : \mathsf{Bool} \to \mathsf{Bool}$, along with term equations defining their standard truth tables (e.g., $NOT(T) = F$, $NOT(F) = T$, $AND(T, T) = T$, $AND(T, F) = F$, etc.).

For every atomic formula A (e.g., $R(\ldots)$ or $t_1 = t_2$) appearing in ϕ_{CNF}, we introduce a corresponding function symbol f_A that maps the sorts of the arguments of A to Bool. For example, if R has signature $S_1 \times S_2 \to \mathit{Relation}$, we introduce $f_R : S_1 \times S_2 \to \mathsf{Bool}$. If t_1, t_2 have sort S, we introduce $f_{eq,S} : S \times S \to \mathsf{Bool}$ (often written just EQ_S). A literal L_{ij} in a clause C_i is then represented by a Boolean term:

- If L_{ij} is a positive atom A, it is represented by the term $f_A(\ldots)$.
- If L_{ij} is a negated atom $\neg A$, it is represented by the term $NOT(f_A(\ldots))$.

Step 4: Encoding Clauses as Term Equations. The universal sentence ψ_{Sk} is satisfied if and only if every clause C_i evaluates to true under all assignments to the variables $x_1, \ldots, x_m$. Let $Term(L_{ij})$ be the Boolean term representing the literal L_{ij} as constructed in Step 3. Each clause $C_i = L_{i1} \vee \cdots \vee L_{iq_i}$ is equivalent to the condition that the corresponding Boolean term evaluates to T. Using the OR function symbol (which can be built from AND and NOT if needed, or taken as primitive), we express this condition as a term equation:

$$OR(Term(L_{i1}), OR(Term(L_{i2}), \ldots, OR(Term(L_{i,q_i-1}), Term(L_{iq_i}))\ldots)) = T$$

This equation must hold for all assignments to the free variables $x_1, \ldots, x_m$ appearing in the terms. Since our framework implicitly assumes universal quantification over solutions, this single term equation enforces the clause C_i.

Step 5: Constructing Γ_ψ. The final multi-sorted Term Coding system Γ_ψ comprises:

- The term equations defining the standard behaviour of the Boolean functions (AND, OR, NOT) over $\{T, F\}$.
- The non-equality constraint ensuring $T \neq F$ (e.g., $EQ_{Bool}(T, F) = F$).
- For each clause C_i in ϕ_{CNF}, the term equation derived in Step 4 ensuring the clause evaluates to T.

The signature Σ' for Γ_ψ includes the sorts and function symbols from Σ_{Sk} plus the Bool sort and the associated logical function symbols (f_A for atoms A, EQ_{Sort}, AND, OR, NOT).

Equivalence: An interpretation $\mathcal{I}$ over appropriate domains (including a domain D of size n for the primary sort(s) and a domain $\{True, False\}$ for Bool) satisfies all equations in Γ_ψ if and only if:

- The Boolean operations behave standardly.
- Every clause C_i evaluates to True under the interpretation induced for the atomic formulas (via the f_A functions) for all assignments to $x_1, \ldots, x_m$.

This is precisely the condition for the interpretation $\mathcal{I}$ (restricted to the signature Σ_{Sk}) to be a model of the universal sentence ψ_{Sk}. Since ψ_{Sk} is equivalent to ψ regarding finite satisfiability, Γ_ψ has a model with the primary sort of size n if and only if ψ has a finite model of size n. The construction is effective as Skolemisation, CNF conversion, and term generation are algorithmic. □

Corollary 4.2. *The problem of determining whether a given multi-sorted Term Coding system has a finite model is undecidable.*

Proof. This follows immediately from Theorem 4.1 and Trakhtenbrot's theorem, which states that the set of first-order sentences having a finite model is undecidable [28]. □

Indeed, as we will explore later in the context of complexity dichotomies (Section 7, relying on appendix A), even more specific questions related to achieving certain solution thresholds within the simpler single-sorted Term Coding framework (particularly for dispersion problems) remain undecidable.

This theorem firmly establishes that multi-sorted Term Coding possesses significant logical strength, capable of capturing the full complexity of first-order finite model theory. Consequently, the framework provides a unified lens for studying a wide range of combinatorial existence problems that can be specified in first-order logic.

4.3 Scope and Expressiveness of Multi-Sorted Term Coding

The introduction of multiple sorts and non-equality constraints in Section 4 significantly broadens the range of problems addressable by the Term Coding framework. While the single-sorted, purely equational version already captures interesting combinatorial optimisation problems, the multi-sorted extension with distinctness constraints allows for the direct encoding of structures involving different types of objects and necessary separation properties.

As formally established in Theorem 4.1, this extended framework is powerful enough to capture the full expressiveness of first-order logic concerning finite satisfiability. This has several important consequences:

- **Combinatorial Designs:** The existence problem for a wide variety of combinatorial designs can be formulated as a satisfiability problem within multi-sorted Term Coding. This includes structures like general $t-(v,k,\lambda)$ designs, projective and affine planes, transversal designs, orthogonal arrays, and specific types of Latin squares (e.g., those avoiding certain substructures), where distinct points, blocks, or symbols must be handled.
- **Graph Properties:** Many fundamental graph-theoretic problems concerning the existence of specific subgraphs or properties in finite graphs are expressible in first-order logic. Consequently, questions like the existence of a k-clique, the satisfiability of k-colorability (for fixed k), or the existence of certain cycle structures can be translated into Term Coding satisfiability problems, typically using sorts for vertices, edges, and potentially colors or indices.
- **Finite Model Theory and Beyond:** The framework inherently encompasses any property checkable by a fixed first-order sentence on finite relational structures. This connects Term Coding to core questions in finite model theory, database theory (e.g., query containment

under constraints), and verification (e.g., model checking finite-state systems against FO properties).

Therefore, the existence problem for any finite structure definable by a set of first-order axioms can, in principle, be represented as finding a satisfying interpretation for a corresponding multi-sorted Term Coding system Γ. Although our primary focus remains on the *optimisation* aspect – determining the maximum size $\max_{\mathcal{I}}(\Gamma, \mathbf{n})$ and its relation to guessing numbers – the ability to encode satisfiability underlines the fundamental nature and broad applicability of the framework. The undecidability results for Term Coding satisfiability (Corollary 4.2) are a direct consequence of this expressive power, mirroring Trakhtenbrot's theorem for first-order logic.

4.4 Extremal and Existence Questions Revisited

When moving to *multi-sorted Term Coding* with non–equality constraints, we still define

$$\max_{\mathcal{I}}(\Gamma; n_1, n_2, \dots)$$

as the largest number of solutions $(x_1, \dots, x_v)$ that are consistent with the given *term equations* (possibly in normal form) and *non–equality constraints* (i.e. $x \neq y$) under an interpretation $\mathcal{I}$. Here, each sort S_i has a finite domain of size n_i; any constraint of the form $f(\dots) \neq g(\dots)$ functions analogously to a functional constraint that forbids $f(\dots) = g(\dots)$ from simultaneously holding. Hence, from an information–theoretic perspective, non–equality constraints alter the permissible solution sets by at most constant factors, assuming the system remains consistent. This is because each consistent non-equality constraint rules out a fixed pattern of value coincidences (e.g., $x = y$), the relative frequency of which vanishes as the domain sizes n_i grow, while the overall structure determining the growth exponent remains unchanged.

Bounding Results Still Hold. Since adding or removing consistent non–equality constraints only impacts $\max_{\mathcal{I}}(\Gamma; n_1, n_2, \dots)$ by constant multiplicative factors (dependent on Γ but not on the n_i), all the key bounding arguments (cf. Theorem **??** for the single-sorted case and the principles extended in Section 5 for the multi-sorted case) carry over with little modification. In particular, the asymptotic behaviour of the maximum number of solutions is still governed by the limiting guessing number $L = \mathrm{Guess}(G_\Gamma)$ of the associated variable dependency graph G_Γ. Specifically,

$$\max_{\mathcal{I}}(\Gamma; n_1, n_2, \dots) = \Theta(M_{\max}^L) \quad \text{as } n_i \to \infty,$$

where $L = \lim_{n\to\infty} \mathrm{Guess}(G_\Gamma; (n, m_1), \dots, (n, m_r))$ is the limiting exponent (assuming common scaling n), and $M_{\max}$ is the effective geometric mean size of the domains, weighted by the number of variables of each sort (see Corollary 5.3 for the precise definition). In essence, the asymptotic exponent L remains the determining factor.

Complexity Aspects. As in the single–sorted case, determining whether a multi–sorted system (with non–equality constraints) can ever exceed a threshold corresponding to an integer exponent (like $M_{\max}^k$ for integer k) is *undecidable*, stemming from the framework's ability to encode first-order satisfiability. In contrast, verifying whether the system eventually attains more than this threshold (e.g., exceeding $M_{\max}^k$ asymptotically) becomes decidable in polynomial time, particularly for dispersion problems where L is known to be an integer (or efficiently computable rational in some settings). This reflects the same sharp complexity dichotomy between reaching an integer threshold and exceeding it. Similarly, *dispersion* arguments—i.e. counting how many distinct s-tuples can appear—extend naturally to the multi–sorted setting, inheriting these complexity properties.

Broader Impact and Examples. Block designs, coding problems, and finite geometries often require enforcing the distinctness of points, lines, blocks, or symbols; hence, non–equality constraints arise naturally. By allowing multiple sorts (e.g. *points* versus *blocks*, etc.) and imposing $x \neq y$–type requirements, we capture these classical structures within the Term Coding framework. In subsequent sections (like Section 5), we demonstrate how the bounding principles (i.e. normalisation, diversification, guessing–number arguments) adapt seamlessly.

Summary. Thus, *multi–sorted Term Coding with non–equality constraints* naturally generalises the single–sorted, purely equation–based approach. Apart from constant–factor differences in solution counts (provided consistency is maintained), all the bounding techniques, finite–n theorems based on guessing numbers, and complexity dichotomies remain valid. This broader viewpoint accommodates a richer class of combinatorial designs and existence problems (such as Steiner systems, projective planes, colourings, etc.) without altering the essential extremal and complexity insights derived from the framework.

5 Guessing Number in the Multi-Sorted Setting (with non-equality Constraints)

In Section 3, we defined a *guessing number* $\mathrm{Guess}(G, n)$ for a *single-sorted* scenario, in which each node of $G = (V, E)$ chooses its hat colour from one alphabet of size n. We now generalise to a *multi-sorted* setup, allowing different nodes (each assigned a variable) to have different alphabet sizes, possibly with additional *non-equality* constraints (i.e. $v \neq w$) among certain nodes of the same sort. (Note that in our framework each node is associated with a variable, which has both a name and a sort. If two nodes are assigned the same variable, then in any valid assignment they must receive the same hat colour.) Our goal is to define a multi-sorted guessing number that generalises the single-sorted concept consistently, particularly regarding the extraction of an asymptotic exponent governing the growth of the maximum number of guessable configurations.

5.1 Multi-Sorted Hat Assignments and Distinctness

Multiple sorts and alphabets.

Suppose we have r distinct *sorts* $S_1, \ldots, S_r$, each with an associated alphabet (hat-colour set) $\mathcal{A}_j$ of size s_j. A directed graph $G = (V, E)$ is now *labelled* (or *multi-sorted*), meaning each node $v \in V$ carries a label $\mathrm{sort}(v) \in \{S_1, \ldots, S_r\}$. Thus:

- Node v's hat colour is chosen from $\mathcal{A}_{\mathrm{sort}(v)}$, whose size we denote by $s_{\mathrm{sort}(v)}$.
- If n_j is the number of nodes labelled by the sort S_j, then each of those n_j nodes draws its colour from $\mathcal{A}_j$ of size s_j.
- Hence, the total *raw* configuration space has size

$$\mathrm{TotalConfigurations} = \prod_{j=1}^{r} s_j^{n_j},$$

  where $\sum_{j=1}^{r} n_j = |V|$ (the total number of nodes).

Non-equality (Distinctness) Constraints.

In many applications (e.g. certain block designs, Steiner systems, or colourings), some pairs of nodes v, w *of the same sort* must be forced to have distinct colours (i.e. $v \neq w$). Concretely:

- We forbid any configuration in which two same-sort nodes v, w share the same colour if $v \neq w$ is declared as a non-equality constraint.
- This reduces the permissible configuration space to a subset of the raw configuration space.

5.2 Multi-Sorted Guessing Game with Non-Equalities

Just as in the single-sorted game, each node v sees its in-neighbours' colours and tries to guess its own colour. However:

- *Sort-Specific Alphabets:*
  v's guess function must output a colour in $\mathcal{A}_{\mathrm{sort}(v)}$ of size $s_{\mathrm{sort}(v)}$.
- *Distinctness Constraints:*
  If $v \neq w$ is declared (i.e. two nodes of the same sort must differ), no valid configuration can assign them the same colour. Any strategy that attempts to guess "all nodes" correctly on a set S of assignments must include only those assignments satisfying $v \neq w$.

A *deterministic guessing strategy* is thus a collection $\{G_v\}$ of local functions

$$G_v \colon \Big(\prod_{u \in N^-(v)} \mathcal{A}_{\mathrm{sort}(u)} \Big) \to \mathcal{A}_{\mathrm{sort}(v)}.$$

We denote the maximum size of a set S of valid assignments (respecting all constraints) on which a single strategy $\{G_v\}$ is correct for all $v \in V$ by $\max |S|$.

5.3 Defining the Multi-Sorted Guessing Number Consistently

While $\max |S|$ is the fundamental quantity representing the largest number of correctly guessable configurations, our goal is often to understand its asymptotic behaviour, particularly the exponent that governs its growth, analogous to L in the single-sorted n^L scaling. To extract such an exponent consistently across different distributions of sort sizes, we need a suitable normalisation base for the logarithm.

The geometric mean of the alphabet sizes, weighted by the number of nodes of each sort, provides a natural normalisation. It represents the "average alphabet size per node" in a logarithmic sense ($\log M = \frac{1}{|V|} \sum n_j \log s_j$). Using this base ensures that if all sort sizes are equal ($s_j = n$), the base becomes n, recovering the single-sorted definition. Furthermore, this choice allows us to define a limiting exponent $L = \lim \mathrm{Guess}(G; \dots)$ which governs the asymptotic growth as $\max |S| = \Theta(M^L)$, facilitating direct comparisons between different multi-sorted systems and connecting to the integer thresholds relevant for the complexity dichotomy.

Definition 5.1 (Multi-Sorted Guessing Number with Distinctness)**.** Let $G = (V, E)$ be a *multi-sorted* directed graph with sorts $S_1, \dots, S_r$, each sort S_j having alphabet size s_j, and suppose n_j nodes are labelled by S_j. Let $\max |S|$ be the maximum size of a set of valid configurations correctly guessed by a single deterministic strategy.

We define the *multi-sorted guessing number* as

$$\mathrm{Guess}(G;\,(s_1,n_1),\dots,(s_r,n_r)) \;=\; \log_M\Big(\max_{\text{strategy}}\;|S|\Big),$$

where the base M is the weighted geometric mean of the alphabet sizes:

$$M = \left(\prod_{j=1}^{r} s_j^{\,n_j}\right)^{\frac{1}{\sum_{j=1}^{r} n_j}} = \left(\prod_{j=1}^{r} s_j^{\,n_j}\right)^{1/|V|}.$$

5.4 Multi-Sorted Guessing Strategies and their Relation to Term Coding

The multi-sorted guessing game setup provides a combinatorial interpretation for the behaviour of multi-sorted Term Coding systems.

One way to relate the multi-sorted setup to the single-sorted one is through an intuitive "blowing-up" argument. If we consider the information content of assigning a value to a node v of sort S_j (alphabet size s_j) relative to some base alphabet size n, it corresponds roughly to $\log_n s_j$. We can imagine replacing node v with approximately $\log_n s_j$ conceptual nodes, each using alphabet size n. An edge $(u \to v)$ in the original graph would then correspond to connections between all conceptual nodes derived from u and all conceptual nodes derived from v. While this is heuristic (especially if $\log_n s_j$ is not an integer), it suggests that the overall structure and information flow are preserved. A deterministic guessing strategy on the original multi-sorted graph corresponds to a strategy on this conceptual single-sorted graph, leading to the same maximum number of globally consistent assignments (relative to their respective total configuration spaces). When all $s_j = n$, this trivially recovers the single-sorted game.

In many combinatorial applications, the different sort sizes s_j are often functions of a single underlying parameter, typically the size n of a primary sort (e.g., the number of points in a design). If $s_j = s_j(n)$, we can view the multi-sorted guessing number as a function of n:

$$\mathrm{Guess}\big(G;\{s_j(n)\},\{n_j\}\big) \;=\; \log_{M(n)}\Big(\max|S|\Big), \quad \text{where} \quad M(n) \;=\; \left(\prod_j (s_j(n))^{n_j}\right)^{1/|V|}.$$

This naturally connects the abstract definition to concrete problems where parameters scale together.

Finally, consider the multi-sorted Term Coding framework (cf. Section 4). Variables correspond to nodes in the dependency graph G_Γ, inheriting their sorts. An equation $f(x_{i_1},\dots) = x_j$ dictates that node x_j must guess its value based on its in-neighbours $x_{i_1},\dots$. A non-equality constraint $x_p \neq x_q$ translates directly to a distinctness constraint between nodes p and q (if they have the same sort). An interpretation $\mathcal{I}$ for the Term Coding system Γ directly defines a guessing strategy $\{G_v\}$. The maximum size of the solution set for the corresponding *diversified* system Γ'' (where distinct function symbol occurrences are treated independently) exactly equals $\max|S|$, the maximum number of configurations correctly guessable by a single strategy. Since $\max_{\mathcal{I}}(\Gamma;\mathbf{s}) = \Theta(\max_{\mathcal{I}}(\Gamma'';\mathbf{s}))$, we have $\max_{\mathcal{I}}(\Gamma;\mathbf{s}) = \Theta(\max|S|)$. Taking the logarithm with base M shows that $\mathrm{Guess}(G;\mathbf{s})$ captures the asymptotic exponent governing the original Term Coding problem's solution size. Thus, the core principles relating solutions, graph structure, and guessing numbers extend consistently to the multi-sorted setting.

5.5 Example: A Two-Node Graph with Alphabet Sizes n_1, n_2

Consider a directed graph G consisting of exactly two nodes $\{v_1, v_2\}$, representing variables X and Y. Node v_1 has sort S_1 with alphabet size $s_1 = n_1$, and node v_2 has sort S_2 with alphabet size $s_2 = n_2$. Assume edges $(v_1 \to v_2)$ and $(v_2 \to v_1)$, meaning each node must guess its value based on the other's value. This corresponds to term equations like $f_1(Y) = X$ and $f_2(X) = Y$.

The strongest possible correlation is imposed: any valid assignment (x, y) must satisfy both equations. This implies that the set of valid assignments S must satisfy $X = f_1(f_2(X))$ and $Y = f_2(f_1(Y))$. Let the maximum size of such a set be $\max|S|$. Information-theoretically, the dependencies imply $H(X|Y) = 0$ and $H(Y|X) = 0$. Standard entropy properties then give $H(X, Y) = H(X) = H(Y)$. Since $H(X) \leq \log|\mathcal{A}_1| = \log n_1$ and $H(Y) \leq \log|\mathcal{A}_2| = \log n_2$ (using an arbitrary logarithm base), we must have $H(X, Y) \leq \min(\log n_1, \log n_2)$. The number of possible assignments is maximized when this bound is achieved, giving $\max|S| = \text{base}^{H(X,Y)} \leq \min(n_1, n_2)$. This bound is achievable by defining f_1, f_2 appropriately (e.g., if $n_1 \leq n_2$, map A_1 injectively into A_2 via f_2 and define f_1 as its partial inverse). Thus, $\max|S| = \min(n_1, n_2)$.

Now we apply Definition 5.1. We have $r = 2$, $n_1 = 1$ node of sort 1, $n_2 = 1$ node of sort 2. The geometric mean base is

$$M = (s_1^{n_1} s_2^{n_2})^{1/(n_1+n_2)} = (n_1^1 n_2^1)^{1/2} = \sqrt{n_1 n_2}.$$

The multi-sorted guessing number is:

$$\text{Guess}\big(G;\,(n_1,1),(n_2,1)\big) \;=\; \log_{\sqrt{n_1 n_2}}\Big(\min\{n_1,\,n_2\}\Big).$$

If $n_1 = n_2 = n$, the base is n and $\max|S| = n$, so $\text{Guess}(G; (n, 1), (n, 1)) = \log_n(n) = 1$, correctly matching the single-sorted result for a bidirected edge.

5.6 A Multi-Sorted Analogue of Theorem ?? (with non-equality Constraints)

We now extend Theorem 3.8—the finite-n bounds in the single-sorted case—to a *multi-sorted* Term Coding framework, possibly including non-equality (distinctness) constraints among variables of the same sort. Instead of one domain of size n, we allow each sort S_j to have its own domain of size n_j, and we may require that certain variables of that sort be distinct. (Recall that in our framework, each node is associated with a variable that has both a name and a sort; if two nodes share the same variable name, they must receive the same hat colour.)

The key result below shows that

$$\max_{\mathcal{I}}(\Gamma; n_1, \ldots, n_r)$$

can be sandwiched by multi-sorted guessing-number bounds, much like the single-sorted sandwich in Theorem **??**. We use the *multi-sorted guessing number*

$$\text{Guess}\Big(G_\Gamma;\,(s_1, m_1), \ldots, (s_r, m_r)\Big)$$

as defined in Definition 5.1, where G_Γ is the variable dependency graph of the Term Coding system Γ, m_j is the number of nodes in G_Γ corresponding to variables of sort S_j, and s_j is the alphabet size for sort S_j.

Theorem 5.2 (Multi-Sorted Sandwich Bounds (non-equality Included))**.** *Let Γ be a multi-sorted Term Coding system with r sorts, where sort S_j has domain size $s_j = n_j$ and corresponds to m_j*

nodes in the dependency graph $G_\Gamma = (V, E)$. Assume Γ includes term equations and consistent non-equality constraints. For sufficiently large domain sizes $\mathbf{n} = (n_1, \ldots, n_r)$, the maximum size of any solution set $\max_{\mathcal{I}}(\Gamma; \mathbf{n})$ satisfies:

$$K_1 \cdot M_{lower}^{\mathrm{Guess}(G_\Gamma;\, \mathbf{n}_{lower})} \;\leq\; \max_{\mathcal{I}}(\Gamma; \mathbf{n}) \;\leq\; M^{\mathrm{Guess}(G_\Gamma;\, \mathbf{n})},$$

where:

- $M = (\prod_{j=1}^{r} n_j^{m_j})^{1/|V|}$ *is the geometric mean base for the upper bound, with* $\mathbf{n}$ *in the Guess function representing the tuple* $((n_1, m_1), \ldots, (n_r, m_r))$.
- $M_{lower} = (\prod_{j=1}^{r} (\lfloor n_j/w_j \rfloor)^{m_j})^{1/|V|}$ *is the base for the lower bound, using reduced alphabet sizes* $\mathbf{n}_{lower} = ((\lfloor n_1/w_1 \rfloor, m_1), \ldots, (\lfloor n_r/w_r \rfloor, m_r))$ *in the Guess function.*
- w_j *is approximately the number of distinct variables of sort* S_j *used in* Γ*, required for the domain partitioning in the lower bound proof.*
- $K_1 > 0$ *is a constant depending on* Γ *but not on* $\mathbf{n}$*, absorbing effects of non-equalities, diversification adjustments (relating* Γ *to* Γ''*), and the floor function in the lower bound construction. The notation* X^Y *uses the exponent* $Y = \mathrm{Guess}(\ldots)$.

In particular, if all $n_j = n$, these bounds simplify to $K_1' \cdot n^{\mathrm{Guess}(G_\Gamma;(\lfloor n/w \rfloor, \ldots))} \leq \max_{\mathcal{I}}(\Gamma; n, \ldots, n) \leq n^{\mathrm{Guess}(G_\Gamma;(n, \ldots))}$.

Proof Sketch. The proof adapts the single-sorted arguments.

Upper bound: Any solution set $C_{\mathcal{I}}(\Gamma)$ under interpretation $\mathcal{I}$ is a subset of the correctly guessed configurations $S_{\{G_v\}}$ for the strategy $\{G_v\}$ derived from $\mathcal{I}$. Thus $\max_{\mathcal{I}} |C_{\mathcal{I}}(\Gamma)| \leq \max_{\{G_v\}} |S_{\{G_v\}}| = \max |S|$. By Definition 5.1, $\max |S| = M^{\mathrm{Guess}(G_\Gamma;\mathbf{n})}$. The upper bound follows with effective constant $K_2 = 1$.

Lower bound: Adapt the domain partitioning technique (as used in proofs related to graph capacity or single-sorted guessing numbers) [10, 8]. For each sort S_j, partition the domain A_j of size n_j into w_j blocks (where w_j is roughly the number of distinct variables of sort j), each of size $\approx \lfloor n_j/w_j \rfloor$. Construct an interpretation $\mathcal{I}_{\text{lower}}$ based on an optimal strategy for the smaller alphabet sizes $(\lfloor n_j/w_j \rfloor)_j$. Use distinct blocks for distinct variables to satisfy non-equality constraints. The number of solutions generated is related to the maximum guessable set S_{lower} for the smaller alphabets. Relating the solutions of the original Γ to this constructed solution involves factors related to diversification (comparing Γ' to $\Gamma'' \approx S_{\text{lower}}$) and the floor function, absorbed into K_1. This yields $|S_{\text{lower}}| \approx M_{\text{lower}}^{\mathrm{Guess}(G_\Gamma;\mathbf{n}_{\text{lower}})}$, giving the lower bound $\max_{\mathcal{I}}(\Gamma; \mathbf{n}) \geq K_1 \cdot M_{\text{lower}}^{\mathrm{Guess}(G_\Gamma;\mathbf{n}_{\text{lower}})}$.

Combining bounds yields the result. The specialization to $n_j = n$ follows directly. □

Corollary 5.3 (Asymptotic Multi-Sorted Bounds (Conditional))**.** *Fix a multi-sorted system Γ with sorts of sizes $(n_1, \ldots, n_r)$, and let $G_\Gamma = (V, E)$ be the associated labelled (multi-sorted) graph with m_j nodes of sort S_j. Let* $\mathrm{Guess}(G_\Gamma)$ *be the limiting multi-sorted guessing number, i.e.,* $\mathrm{Guess}(G_\Gamma) = \lim_{n\to\infty} \mathrm{Guess}(G_\Gamma; (n, m_1), \ldots, (n, m_r))$. *Assume the convergence to this limit is sufficiently fast, satisfying*

$$\mathrm{Guess}(G_\Gamma; (n, m_1), \ldots, (n, m_r)) = \mathrm{Guess}(G_\Gamma) + O\left(\frac{1}{\log n}\right)$$

(analogous to the rate implied by Conjecture 3.9 in the single-sorted case). Then, as $n_j \to \infty$ for each j (e.g., proportionally to some parameter n),

$$\max_{\mathcal{I}}\big(\Gamma; n_1, \ldots, n_r\big) \;=\; \Theta\Big(M_{\max}^{\mathrm{Guess}(G_\Gamma)}\Big),$$

where

$$M_{\max} = \left(\prod_{j=1}^{r} n_j^{m_j} \right)^{1/|V|}$$

is the weighted geometric mean base evaluated at the current domain sizes $(n_1, \ldots, n_r)$*. Hence, if* $n_1 = \cdots = n_r = n$*, we obtain*

$$\max_{\mathcal{I}}(\Gamma; n, \ldots, n) = \Theta\big(n^{\mathrm{Guess}(G_\Gamma)}\big).$$

Proof. The existence of the limit $\mathrm{Guess}(G_\Gamma)$ follows from a multi-sorted version of the super-multiplicative argument and Fekete's Lemma, as sketched in the proof of Theorem 5.2 and detailed for the single-sorted case in Section 3.4. The Θ-asymptotic result then follows from the sandwich bounds in Theorem 5.2. Specifically, the upper bound $\max_{\mathcal{I}} \leq M^{\mathrm{Guess}(G_\Gamma;\mathbf{n})}$ and the lower bound $\max_{\mathcal{I}} \geq K_1 \cdot M_{\mathrm{lower}}^{\mathrm{Guess}(G_\Gamma;\mathbf{n}_{\mathrm{lower}})}$, combined with the assumed fast convergence rate $\mathrm{Guess}(G_\Gamma; \mathbf{s}) = L + O(1/\log(\min s_j))$, allow us to conclude $\max_{\mathcal{I}} = \Theta(M_{\max}^L)$. The crucial step requires that terms like $M^{O(1/\log M)}$ are bounded by constants, which holds under the assumed convergence rate. The specialization to $n_1 = \cdots = n_r = n$ follows directly, yielding $\Theta(n^L)$. □

Thus, in a multi-sorted scenario with distinctness constraints, we can still sandwich $\max_{\mathcal{I}}(\Gamma; n_1, \ldots, n_r)$ using the multi-sorted guessing number. The geometric mean normalisation provides a consistent way to define this number and relate it to the asymptotic behavior, generalizing the classic single-sorted result $\Theta(n^{\mathrm{Guess}(G_\Gamma)})$ while maintaining consistency with standard bounding techniques.

6 Dispersion in Multi-Sorted Settings

In earlier sections, we discussed how *term equations* and *code sizes* (maximal solution sets) capture extremal phenomena via local functional dependencies, often related to guessing numbers and graph entropy. We now turn to the related notion of *dispersion*, originally introduced by Riis and coauthors in a single-sorted context *without* non-equality constraints [29], which measures how many distinct s-tuples a system of terms can generate. We present here a generalised definition that incorporates:

- Multiple sorts, each with its own finite domain.
- Distinctness constraints among *variables* ($x_p \neq x_q$) within the same sort.
- Distinctness constraints among *terms* ($t_i \neq t_j$) produced by the system, provided they belong to the same sort.

As shown in Section 6.3, any dispersion problem can be viewed as a special case of Term Coding. The primary goal shifts from maximizing the number of input assignments $(x_1, \ldots, x_k)$ satisfying certain implicit constraints, to maximizing the size of the *output set* $\{(t_1, \ldots, t_s)\}$. A key feature, explored in Section 6.6, is that the asymptotic growth exponent for standard (unweighted, single-sorted) dispersion problems is always an integer. This property arises because the underlying structure often relates to network flow capacities or matching problems, guaranteeing integral solutions [29]. This contrasts with general Term Coding problems where the exponent (guessing number) can be non-integer.

After defining dispersion precisely, we illustrate its scope, including its connection to network coding, Boolean logic, and its ability to encode finite satisfiability problems. Unlike some earlier

treatments, our formulation does not assume built-in logical connectives like "OR", relying instead on term equations and non-equalities, potentially augmented with a dedicated Boolean sort if complex logical conditions need to be simulated.

6.1 Historical Context and the Riis–Gadouleau Example

The notion of *dispersion* was introduced by Riis and Gadouleau [29] as a flexible tool for analyzing information flow, particularly in network coding scenarios subject to dynamic changes or failures. In their original work, the focus was primarily on the *single-sorted* setting where all variables share a common alphabet A of size n. We illustrate their core ideas and then show how the concept fits naturally into our multi-sorted framework.

A Single-Function Relay: The Case Study.

In [29, §VI], a central example involves a *single coding function*

$$f : A^2 \longrightarrow A,$$

and four variables $x, y, z, w \in A$. The term set under consideration is

$$\Gamma \;=\; \{\, f(x,y),\; f(x,z),\; f(w,y),\; f(w,z)\}.$$

The dispersion problem asks for the maximum number of distinct 4-tuples $(f(x,y), f(x,z), f(w,y), f(w,z))$ that can be generated by choosing an optimal interpretation for $f\colon A^2 \to A$. While the ideal outcome might seem to be n^4 (perfect separation), the shared function f creates dependencies. Riis and Gadouleau showed that collisions are unavoidable, proving an upper bound on the *dispersion exponent* $\gamma(\Gamma, |A|) = \log_{|A|}(\max|\text{Image}|)$ strictly less than 4:

$$\gamma(\Gamma, |A|) \;\le\; 4 \;-\; \log_{|A|}\Big(1 \;-\; 2|A|^{-1} + 3|A|^{-2} - |A|^{-3}\Big).$$

(Note: The original bound might involve slightly different terms depending on exact assumptions, but the principle is a non-trivial upper bound ¡ 4.)

Multi-Sorted Generalisation.

This example readily extends to a multi-sorted scenario. Suppose we assign sorts:

$$x, w \in \mathsf{Sort1} \quad (\text{size } n_1), \quad y, z \in \mathsf{Sort2} \quad (\text{size } n_2),$$

and the function maps to a third sort:

$$f : \mathsf{Sort1} \times \mathsf{Sort2} \longrightarrow \mathsf{Sort3} \quad (\text{size } n_3).$$

The set of terms $\Gamma = \{\, f(x,y),\; f(x,z),\; f(w,y),\; f(w,z)\}$ now defines a multi-sorted dispersion problem. The quantity of interest is $\mathrm{Disp}(\Gamma; n_1, n_2, n_3)$, the maximum size of the image set. Proposition 6.1 provides simple bounds based on elementary counting arguments.

Proposition 6.1 (Bounds for the Single-Relay Example in Multi-Sorted Form)**.** *Let $x, w \in \mathsf{Sort1}$ (size n_1), $y, z \in \mathsf{Sort2}$ (size n_2), and $f\colon \mathsf{Sort1} \times \mathsf{Sort2} \to \mathsf{Sort3}$ (size n_3). Let*

$$\Gamma \;=\; \{\, f(x,y),\; f(x,z),\; f(w,y),\; f(w,z)\}.$$

Its dispersion (maximum image size), denoted $\mathrm{Disp}(\Gamma; n_1, n_2, n_3)$*, satisfies:*

$$\min\Big(n_1(n_1-1)n_2(n_2-1),\ n_3(n_3-1)(n_3-2)(n_3-3)\Big) \ \le\ \mathrm{Disp}(\Gamma; n_1, n_2, n_3) \ \le\ \min\Big(n_3^4,\ U_{\mathrm{part}}\Big),$$

where the partition-based upper bound U_{part} *accounts for potential equalities among input variables:*

$$U_{\mathrm{part}} = n_1(n_1-1)n_2(n_2-1) + n_1n_2(n_2-1) + n_1(n_1-1)n_2 + n_1n_2.$$

Proof. Upper bound. The image consists of 4-tuples from Sort3, so the size is at most n_3^4. Alternatively, consider the four cases based on equalities among x, w and y, z. The number of distinct input tuples (x, y, z, w) in these cases are $n_1(n_1-1)n_2(n_2-1)$, $n_1n_2(n_2-1)$, $n_1(n_1-1)n_2$, and n_1n_2, respectively. Summing these gives U_{part}. Even if f maps each distinct input tuple type to a unique output tuple, the total image size cannot exceed U_{part}. Thus, $\mathrm{Disp}(\Gamma; n_1, n_2, n_3) \le \min(n_3^4, U_{\mathrm{part}})$.

Lower bound. Assume $n_1, n_2 \ge 2$. If $n_3 < 4$, the term $n_3(n_3-1)(n_3-2)(n_3-3)$ is non-positive, making the lower bound trivial. If $n_3 \ge 4$, we can construct an interpretation. Focus on inputs where $x \ne w$ and $y \ne z$. There are $n_1(n_1-1)n_2(n_2-1)$ such input combinations. We can choose f to map these inputs injectively to distinct elements in Sort3. Furthermore, we can ensure that for such inputs, the four outputs $f(x, y), f(x, z), f(w, y), f(w, z)$ are all distinct, requiring at least 4 distinct values in Sort3. This construction generates at least $n_1(n_1-1)n_2(n_2-1)$ distinct image tuples, provided the target sort Sort3 is large enough to accommodate the necessary distinctness (at least 4 elements needed just for one output tuple). A simple lower bound reflecting this is $\min(n_1(n_1-1)n_2(n_2-1), n_3(n_3-1)(n_3-2)(n_3-3))$. □

Remark on Exponent vs. Direct Count Definitions

As noted, some literature defines dispersion via an exponent D, where the maximum image size scales like n^D (for single-sort size n). This exponent $D = \lim_{n\to\infty} \frac{\log(\max|\mathrm{Image}|)}{\log n}$ is particularly useful for asymptotics and is known to be an integer for standard dispersion problems (see Section 6.6). In this paper, we primarily define $\mathrm{Disp}(\Gamma; n_1, \ldots, n_r)$ as the direct *maximum cardinality* of the image set, $\max|\mathrm{Image}_{\mathcal{I}}|$. This avoids logarithms in the definitions and theorems. The two perspectives are equivalent for asymptotic analysis: our results on the count $\max|\mathrm{Image}|$ directly imply results on the exponent D, and vice versa. We choose the direct count for definitional clarity, especially in the multi-sorted setting.

6.2 Defining Dispersion with Distinctness Constraints

We now formalize the definition for the multi-sorted case with constraints. A multi-sorted dispersion problem is specified by:

- A set of sorts $S_1, \ldots, S_r$.
- Variables $x_1, \ldots, x_k$, each x_i belonging to some sort $S_{(x_i)}$.
- A set of variable distinctness constraints $x_p \ne x_q$, where x_p, x_q must have the same sort.
- A finite family of function symbols $f_1, \ldots, f_m$, with specified sort signatures (e.g., $f_j : S_{j_1} \times \cdots \times S_{j_a} \to S_{j_0}$).
- A list of output terms $t_1, \ldots, t_s$, each well-typed using the variables and function symbols. Let $S_{(t_i)}$ be the sort of term t_i.

- A set of term distinctness constraints $t_i \neq t_j$, where t_i, t_j must have the same sort $S_{(t_i)} = S_{(t_j)}$.

Definition 6.2 (Multi-Sorted Dispersion under Distinctness)**.** Let Γ be a multi-sorted dispersion problem specified as above. An interpretation $\mathcal{I}$ assigns each sort S_i a finite domain A_i (of size n_i) and interprets each function symbol f as a function $f^{\mathcal{I}}$ respecting the sort signatures. An assignment $\mathbf{a} = (a_1, \ldots, a_k)$ of values from the corresponding domains to the variables $x_1, \ldots, x_k$ is *input-valid* if $a_p \neq a_q$ whenever the constraint $x_p \neq x_q$ is given. An input-valid assignment $\mathbf{a}$ is *output-valid* under $\mathcal{I}$ if $t_i^{\mathcal{I}}(\mathbf{a}) \neq t_j^{\mathcal{I}}(\mathbf{a})$ whenever the constraint $t_i \neq t_j$ is given.

The *image set* under $\mathcal{I}$ is the set of output tuples generated by valid assignments:

$$\mathrm{Image}_{\mathcal{I}}(\Gamma) \;=\; \big\{ \big(t_1^{\mathcal{I}}(\mathbf{a}), \ldots, t_s^{\mathcal{I}}(\mathbf{a})\big) \;:\; \mathbf{a} \text{ is input-valid and output-valid under } \mathcal{I} \big\}.$$

The *dispersion* of Γ for domain sizes $\mathbf{n} = (n_1, \ldots, n_r)$ is the maximum possible size of this image set over all interpretations $\mathcal{I}$:

$$\mathrm{Disp}(\Gamma; \mathbf{n}) \;=\; \max_{\mathcal{I}} \big|\mathrm{Image}_{\mathcal{I}}(\Gamma)\big|.$$

If all sorts have the same domain size n, we write $\mathrm{Disp}(\Gamma; n)$.

Remark on Encoding Logic. While the definition only includes atomic non-equalities ($x_p \neq x_q$, $t_i \neq t_j$), more complex logical conditions (like coverage axioms in designs, e.g., "point p must be on one of lines L_1, L_2, L_3") can often be simulated. This typically involves introducing auxiliary Boolean sorts and function symbols, translating the logic into term equations and non-equalities within that extended system, similar to the process outlined in Section 6.5. Demanding maximal dispersion in such encoded systems can then enforce the original combinatorial requirements.

6.3 From Multi-Sorted Dispersion to Multi-Sorted Term Coding

We demonstrate that optimizing dispersion is equivalent to optimizing the code size in a specific Term Coding problem. Given a dispersion problem Γ_{disp} with variables $x_1, \ldots, x_k$ and output terms $t_1, \ldots, t_s$, subject to constraints X_{neq} (on variables) and T_{neq} (on terms).

Constructing the Equivalent Term Coding Problem.

Define a Term Coding system Γ_{TC} as follows:

- Variables: Keep the original variables $x_1, \ldots, x_k$. Introduce s new variables $y_1, \ldots, y_s$, where the sort of y_i matches the sort of t_i.
- Function Symbols: Keep the original function symbols used in $t_1, \ldots, t_s$. Introduce k new function symbols $h_1, \ldots, h_k$, where h_i maps the tuple of sorts $(S_{(t_1)}, \ldots, S_{(t_s)})$ to the sort $S_{(x_i)}$.
- Term Equations: Add $s + k$ equations:

$$y_i = t_i \quad \text{for } i = 1, \ldots, s$$

$$x_i = h_i(y_1, \ldots, y_s) \quad \text{for } i = 1, \ldots, k$$

- Non-Equality Constraints: Keep the original variable constraints X_{neq}. Translate the term constraints T_{neq} into constraints on the corresponding y-variables (e.g., $t_i \neq t_j$ becomes $y_i \neq y_j$).

Equivalence Argument.

An interpretation $\mathcal{I}_{\text{TC}}$ for Γ_{TC} includes interpretations for the original functions and the new h_i functions. Consider the projection map π from the solutions of Γ_{TC} onto the y-coordinates: $\pi(\text{solution}) = (y_1, \ldots, y_s)$. The set of all possible projected tuples $(y_1, \ldots, y_s)$ obtained from solutions of Γ_{TC} under an optimal interpretation $\mathcal{I}^*_{\text{TC}}$ is precisely the maximum image set $\text{Image}_{\mathcal{I}^*_{\text{disp}}}(\Gamma_{\text{disp}})$ for an optimal interpretation $\mathcal{I}^*_{\text{disp}}$ of the original dispersion problem. Specifically, $\max_{\mathcal{I}_{\text{TC}}} |\{(y_1, \ldots, y_s) \text{ from solutio}$ $\text{Disp}(\Gamma_{\text{disp}}; \mathbf{n})$. The decoding functions h_i in Γ_{TC} serve only to ensure that each distinct image tuple $(y_1, \ldots, y_s)$ corresponds to at least one valid input $(x_1, \ldots, x_k)$. Maximizing the number of solutions for Γ_{TC} essentially corresponds to maximizing the number of distinct $(y_1, \ldots, y_s)$ tuples that can be generated and then decoded back to a valid input.

Thus, optimizing dispersion is equivalent to optimizing the size of the projection of the solution set of the corresponding Term Coding problem. Dispersion is effectively a special case focusing on the size of the output space rather than the input space.

Remark on Strict Expressiveness.

As highlighted previously (Section 3.7), general Term Coding problems can yield non-integer asymptotic exponents $L = \text{Guess}(G_\Gamma)$ (e.g., $L = 2.5$ for C_5). In contrast, standard dispersion problems yield integer exponents D (see Section 6.6). This implies that some structures encodable by Term Coding (like the dependencies in C_5) cannot be captured purely by optimizing dispersion, confirming that Term Coding is a strictly more expressive framework.

6.4 Boolean Functions via Dispersion

Dispersion maximization can surprisingly define basic logical operations. Consider a single sort Bool with domain $A_{\text{Bool}} = \{0, 1\}$, intended to represent Boolean values. Introduce:

- A constant $c \in \mathsf{Bool}$, intended to be 1.
- A binary function $S : \mathsf{Bool} \times \mathsf{Bool} \to \mathsf{Bool}$.
- A non-equality constraint $S(c, c) \neq c$. This forces $S(1, 1) \neq 1$, i.e., $S(1, 1) = 0$.

Now consider the dispersion problem for the map $T : \mathsf{Bool}^3 \to \mathsf{Bool}^3$ defined by

$$T(x, y, z) = \big(S(x, x),\ S(c, y),\ S(z, c)\big).$$

The total input space has size $2^3 = 8$. Maximizing the dispersion, $\text{Disp}(T; 2)$, means making the image size as large as possible. The maximum possible size is 8, which requires T to be a bijection. For T to be bijective, each component map must also induce a bijection on certain inputs. The constraint $S(1, 1) = 0$ is already imposed. Bijectivity forces the other values:

- $x \mapsto S(x, x)$ must be a permutation. Since $S(1, 1) = 0$, we must have $S(0, 0) = 1$.
- $y \mapsto S(c, y) = S(1, y)$ must be a permutation. Since $S(1, 1) = 0$, we must have $S(1, 0) = 1$.
- $z \mapsto S(z, c) = S(z, 1)$ must be a permutation. Since $S(1, 1) = 0$, we must have $S(0, 1) = 1$.

These conditions precisely define $S(p, q)$ as the NAND function: $S(p, q) = \neg(p \wedge q)$.

Similarly, defining $\texttt{neg} : \mathsf{Bool} \to \mathsf{Bool}$ with the constraint $\texttt{neg}(x) \neq x$ forces $\texttt{neg}$ to be the NOT function (the only fixed-point-free permutation). Since NAND and NOT are functionally complete, any Boolean function can be realized by maximizing the dispersion of an appropriately constructed term system.

6.5 Encoding First-Order Finite Satisfiability as a Single-Sort Dispersion Problem

The expressive power illustrated by the Boolean example extends much further: as outlined in appendix B, any first-order sentence ψ (over a finite relational signature) can be effectively translated into a *single-sort* dispersion problem Γ_ψ such that ψ has an n-element finite model if and only if Γ_ψ achieves maximal dispersion under an interpretation using domain size related to n.

The encoding, detailed in appendix **??**, follows standard logical transformations:

- *Skolemisation:* Convert ψ to an equisatisfiable universal sentence $\tilde{\psi} = \forall \mathbf{x}\, \phi(\mathbf{x})$, where ϕ is quantifier-free.
- *Conjunctive Normal Form (CNF):* Rewrite ϕ as a conjunction of clauses $C_1 \wedge \cdots \wedge C_m$.
- *Boolean Simulation:* Introduce a sort Bool (or BoolPadded) with values for True/False. Replace each atomic formula A in the clauses with a Boolean-valued term A^* (e.g., using function symbols R^*, EQ^*).
- *Clause Encoding:* Use Boolean function symbols (definable via dispersion as shown in 6.4) to represent the clauses. The entire formula ϕ becomes a single Boolean term $\phi^*(\mathbf{x})$.
- *Dispersion Formulation:* The problem becomes finding an interpretation $\mathcal{I}$ such that the term $\phi^*(\mathbf{x})$ evaluates to True for *all* assignments $\mathbf{a}$ to $\mathbf{x}$. This can be framed as a dispersion problem. For instance, consider the dispersion of the term $\phi^*(\mathbf{x})$ itself. If there exists an interpretation making $\phi^*(\mathbf{a})$ always True, the image set is just True, of size 1. If no such interpretation exists, the image set might contain False or True, False. Thus, checking finite satisfiability of ψ is equivalent to checking if the dispersion of ϕ^* can be restricted to True.

This process demonstrates that the framework of *single-sort* dispersion with non-equality constraints is logically powerful enough to capture the complexity of first-order finite model theory. Determining if a dispersion problem can achieve a certain image size (specifically, maximal dispersion in this encoding) can be equivalent to solving an arbitrary FO finite satisfiability problem.

For technical details, including the construction of Boolean gadgets using term equations and non-equalities within the dispersion framework, see appendix **??**. A concrete example demonstrating this encoding for the property of being a total linear order is provided in appendix **??**.

6.6 Asymptotic behaviour and Integer Exponents

A key property distinguishing dispersion problems within the broader Term Coding framework relates to their asymptotic behavior. Let Γ be a single-sorted dispersion problem involving terms $t_1, \ldots, t_s$ over variables $x_1, \ldots, x_k$ using an n-element alphabet, and let $\max_{\mathcal{I}} |\text{Image}_{\mathcal{I}}|$ be its dispersion (the maximum image size, as defined in Definition 6.2).

It is known [29] that this maximum image size grows asymptotically as:

$$\max_{\mathcal{I}} |\text{Image}_{\mathcal{I}}| = \Theta(n^D)$$

where the exponent D is always an integer. This integer D can be interpreted as the guessing number of the specific directed graph associated with the (normalized and diversified) dispersion problem Γ. The convergence is often monotonic, meaning $\frac{\log(\max |\text{Image}_{\mathcal{I}}|)}{\log n} \to D$ from below as $n \to \infty$, indicating that collisions vanish asymptotically.

This integrality of the exponent D for dispersion problems contrasts sharply with general Term Coding problems, where the limiting exponent $L = \text{Guess}(G_\Gamma)$ can be non-integer (as seen for C_5 where $L = 2.5$). This property is fundamental to the complexity dichotomy discussed in Section 7. In the multi-sorted case, analogous results often relate the maximum image size to integer values derived from network flow or matching formulations [29].

7 A Complexity Dichotomy in Single-Sorted Dispersion

We now focus on the complexity of analysing single-sorted dispersion problems, revealing a sharp dichotomy between undecidable and polynomial-time decidable questions based on the nature of the solution size threshold. Let Γ be a single-sorted dispersion problem defined over an n-element domain, involving terms $t_1, \ldots, t_s$. Let $\max_{\mathcal{I}} |\text{Image}_{\mathcal{I}}|$ denote its dispersion count (Definition 6.2).

A key property, discussed in Section 6.6 and established in [29], is the asymptotic behavior:

$$\max_{\mathcal{I}} |\text{Image}_{\mathcal{I}}| = \Theta(n^D)$$

where the limit exponent $D = \lim_{n\to\infty} \frac{\log(\max_{\mathcal{I}} |\text{Image}_{\mathcal{I}}|)}{\log n}$ is always an integer. This integer D is the guessing number of the graph associated with Γ and is often computable efficiently.

The complexity arises when we ask whether the dispersion count can meet or exceed specific thresholds. The undecidable side of the dichotomy stems from foundational results concerning the finite satisfiability of term equations.

7.1 Undecidability Background

It is well-established (e.g. [30, 31, 28], refined in [32]; see [33] for a survey) that determining the existence of finite models for term equations is undecidable. Specifically:

Problem 1 (Finite Satisfiability). Given a system of term equations

$$\Gamma_{eq} = \{t'_1 = s'_1, \ldots, t'_w = s'_w\},$$

decide whether there exists a finite non-trivial model (of size ≥ 2) in which all equations hold. This problem is **undecidable**, even for a single equation [31, 28].

This fundamental result leads to the undecidability of related problems central to dispersion thresholds:

Problem 2 (Finite Bijectivity). Given terms $t_1, \ldots, t_k$ (variables $x_1, \ldots, x_k$), decide if there exists a finite set A ($|A| \geq 2$) and an interpretation $\mathcal{I}$ such that the induced map

$$\Theta_{\mathcal{I}} : (x_1, \ldots, x_k) \mapsto (t_1^{\mathcal{I}}(x_1, \ldots, x_k), \ldots, t_k^{\mathcal{I}}(x_1, \ldots, x_k))$$

is bijective. This problem is also **undecidable**. Its undecidability can be shown via a reduction from the undecidable problem of Horn Clause Finite Satisfiability, as detailed in appendix A.

These undecidability results directly impact our ability to determine if a dispersion system can achieve certain exact performance thresholds, forming the basis for the difficult side of the dichotomy.

7.2 The Dichotomy: Threshold Cases

We now analyse the complexity of deciding if $\max_{\mathcal{I}} |\text{Image}_{\mathcal{I}}| \geq h_{\text{threshold}}(n)$ based on the threshold function $h_{\text{threshold}}(n)$.

Case 1: Threshold is Maximal Size ($h_{threshold}(n) = n^k$). (Assuming the number of output terms is $s = k$). The problem asks: Does there exist a finite domain size $n \geq 2$ such that $\max_{\mathcal{I}} |\text{Image}_{\mathcal{I}}| \geq n^k$? This requires the map $\Theta_{\mathcal{I}} : A^k \to A^k$ defining the dispersion (where $|A| = n$) to be injective (hence bijective, since the domain A is finite) for some interpretation $\mathcal{I}$ over a finite domain A of size n.

Theorem 7.1. *Assuming $s = k$, the dispersion problem with threshold $h_{threshold}(n) = n^k$ (i.e., deciding if $\max_{\mathcal{I}} |\text{Image}_{\mathcal{I}}| \geq n^k$ for some finite $n \geq 2$) is **undecidable**.*

Proof. This is equivalent to Problem 2 (Finite Bijectivity) applied to the terms $t_1, \ldots, t_k$ defining the dispersion map $\Theta_{\mathcal{I}}$. Since Problem 2 is undecidable (appendix A), this problem is undecidable. □

Case 2: Threshold is an Integer Power ($h_{threshold}(n) = n^r$, $1 \leq r \leq k$). The problem asks: Does there exist a finite $n \geq 2$ such that $\max_{\mathcal{I}} |\text{Image}_{\mathcal{I}}| \geq n^r$?

Theorem 7.2. *For any fixed integer r, the dispersion problem with threshold $h_{threshold}(n) = n^r$ (i.e., deciding if $\max_{\mathcal{I}} |\text{Image}_{\mathcal{I}}| \geq n^r$ for some finite $n \geq 2$) is **undecidable**, provided r is sufficiently large (depending on the complexity required for the reduction).*

Proof. This follows from the ability of *single-sort dispersion systems* (as demonstrated in Section 6.5 and detailed in appendix B) to encode arbitrary finite satisfiability problems (like Problem 1). Showing that the image size *can* reach n^r relates to finding specific finite models satisfying the encoded FO sentence ψ, which links back to the undecidability of Problem 1. A detailed reduction, adapting the techniques outlined in appendix B, would be needed to formally establish undecidability for a specific r. □

Case 3: Threshold Strictly Between Integer Powers ($n^d < h_{threshold}(n) \in o(n^{d+1})$). Here, the question naturally becomes asymptotic: Does $\max_{\mathcal{I}} |\text{Image}_{\mathcal{I}}| \geq h_{\text{threshold}}(n)$ hold for all sufficiently large n?

Theorem 7.3. *If the threshold function $h_{threshold}(n)$ satisfies $n^d < h_{threshold}(n)$ and $h_{threshold}(n) \in o(n^{d+1})$ for some fixed integer d, then deciding if*

$$\max_{\mathcal{I}} |\text{Image}_{\mathcal{I}}| \geq h_{threshold}(n) \quad \textit{holds for all sufficiently large } n$$

*is solvable in **polynomial time** in the size of the input dispersion problem Γ.*

Proof. The condition holds for all sufficiently large n if and only if the integer asymptotic exponent D satisfies $D \geq d + 1$. This integer D is the guessing number associated with Γ. As established in [29, 34], for dispersion problems, D is an integer and can be computed in polynomial time (e.g., via max-flow/min-cut algorithms on the dependency graph). Therefore, the decision procedure involves computing the integer D (in PTIME) and checking if $D \geq d+1$. This is a polynomial-time algorithm. For instance, checking if $\max_{\mathcal{I}} |\text{Image}_{\mathcal{I}}| \geq n^d + 1$ holds for large n is equivalent to checking if $D > d$. □

7.3 Summary of the Dichotomy

This analysis reveals the striking complexity dichotomy for single-sorted dispersion problems:

- **Undecidable:** Determining if the maximum image size *can ever reach* a specific integer power threshold (n^k or n^r) for *some finite* $n \geq 2$.
- **Polynomial-Time Decidable:** Determining if the maximum image size *asymptotically exceeds* an integer power threshold (e.g., by checking against $n^d + 1$, or any threshold $h(n)$ with $n^d < h(n) \in o(n^{d+1})$) *for all sufficiently large* n.

This sharp transition hinges on the difference between existential questions over specific finite domain sizes (linked to undecidable satisfiability problems) and universal questions about asymptotic limits (determined by the efficiently computable integer exponent D). Further details on the undecidability proofs are in appendix A.

8 Open Problems, Challenges and Conclusion

Given a Term Coding problem—a system of term equations (possibly with non–equality constraints) over fixed finite alphabets—two central computational tasks arise:

- *Model Finding:* Develop efficient algorithms to construct a solution (i.e. a finite structure or code) when one exists.
- *Non–Existence Certification:* Devise methods to efficiently certify that no solution exists.

While standard search procedures and SAT solvers perform well in some cases, our results indicate that these methods may face inherent combinatorial barriers near critical thresholds.

A key observation is that one can transform a Term Coding problem into a corresponding SAT instance. Let Γ denote the set of term equations and L the set of non-equality constraints. A satisfying assignment to the resulting SAT instance, $\{\text{SAT}\}_{\Gamma,L}$, corresponds one-to-one with a finite model of the combined system $\Gamma \cup L$ (equations Γ and constraints L). Moreover, if $\Gamma \cup L$ has no finite models, then results on the complexity of unsatisfiability proofs [35], particularly for tree–resolution, imply:

- If $\Gamma \cup L$ holds in some infinite model, then every tree–resolution proof certifying the non–existence of a finite model requires exponential size in n (the domain size).

Furthermore, Riis [35] demonstrated a related complexity gap (a dichotomy) concerning proof complexity:

- The existence of polynomial–size (in n) tree-resolution proofs certifying the non–existence of a finite solution for $\Gamma \cup L$ is equivalent to the system $\Gamma \cup L$ failing to hold in *any* infinite model.

Together with classical exponential lower bounds for resolution proofs of principles like the propositional pigeonhole principle [**haken85**, **Ajtai88**], these results motivate the following conjecture.

Conjecture 8.1. *Let $\Gamma \cup L$ be a single–sorted Term Coding problem (term equations Γ, non-equality constraints L) over an n-element alphabet, and define the set of solvable domain sizes*

$$S = \{m \in \mathbb{N} : \Gamma \cup L \text{ has a solution for domain size } m\}.$$

We conjecture that for an instance size n for which no solution exists ($n \notin S$), if n is close to a threshold of solvability (specifically, if its distance to the set of solvable sizes satisfies $dist(n, S) = \min_{m \in S} |n - m| \leq \log^{O(1)}(n)$), then any resolution-based proof certifying the non-existence of a solution for size n (via the corresponding SAT instance $\{\text{SAT}\}_{\Gamma,L}$) must have exponential size.

Other Open Problems:

- *Proof Complexity Lower Bounds:* Establish sharp lower bounds for resolution–based proofs for SAT instances derived from Term Coding problems. For instance, prove (or disprove) that for specific systems $\Gamma \cup L$ unsolvable at certain domain sizes n, every tree–resolution refutation requires exponential size in n.

- *Efficient Model Finding and Hybrid Methods:* Design specialised algorithms for Term Coding problems that exploit their algebraic and combinatorial structure (e.g., via the associated guessing number and dispersion invariants) to potentially outperform generic SAT solvers. Can hybrid methods that combine search tree exploration with structural insights (e.g., reinforcement learning or simulated annealing guided by evaluation networks) yield improved performance?

- *Quantitative Analysis of Near-Optimal Solutions:* For specific classes of Term Coding problems, determine how close the maximal solution size $\max_{\mathcal{I}}(\Gamma \cup L, n)$ can get to the ideal bound suggested by the guessing number (e.g., n^L). Identify structural conditions within $\Gamma \cup L$ that lead to a sharp increase in the solution size or prevent the ideal bound from being met.

- *Improved Term Equations and Non-Equalities for Designs:* A problem like finding $t-(v, k, \lambda)$ designs can be expressed using multi-sorted term equations and non-equalities. This encoding, however, is not unique. Different choices of defining equations and sorts can lead to multiple optimisation problems (associated Term Coding problems), each with its own ideal maximal code size that is attained if and only if the design exists. Are there particularly 'nice' or efficient axiomatisations to consider? Perhaps structures as simple as the algebraic formulation for Steiner triple systems ($t = 2, k = 3$) exist for more general cases.

- *Improved Asymptotic Bounds:* For concrete families of Term Coding problems (such as those encoding combinatorial designs or error–correcting codes), derive tighter lower and upper bounds on the maximum number of solutions. Can refined entropy– or guessing number–based techniques capture finer combinatorial properties of the encoded structures?

- *Extending Keevash's Construction:* Investigate whether Keevash's methods based on random greedy algorithms and absorption for the existence of t-designs can be adapted or applied to broader classes of Term Coding problems. In particular, can one relax the strict term equations and non–equality constraints in design encodings to allow for probabilistic constructions guaranteeing optimal or near-optimal solution counts?

- *Structured and Linear Algorithms:* When the alphabet possesses additional algebraic structure (e.g., a finite field or vector space), study Term Coding problems restricted to interpretations using linear functions. Can one develop efficient algorithms or obtain tighter bounds in this linear setting, paralleling classical results in algebraic coding theory?

- *Search Tree Complexity and Learning Methods:* Analyse the combinatorial complexity of the search space associated with finding models for Term Coding problems. Can reinforcement learning or classical search algorithms like simulated annealing be rigorously designed and

analysed for efficiently navigating this space, and what complexity bounds can be established for such algorithms?

- *Further Unification:* Investigate further connections among Term Coding, graph entropy, network/index coding, and potentially other areas like constraint satisfaction problems (CSPs). How can techniques from these areas be integrated to derive new extremal combinatorial results or improved algorithms?

In conclusion, the Term Coding framework recasts many classical extremal combinatorial problems into a unified algebraic setting, amenable to analysis using guessing number and entropy techniques. Our main results establish tight asymptotic bounds on maximum code sizes and reveal a striking complexity dichotomy related to decision thresholds: a minimal one–unit increase in the solution threshold can transform an undecidable problem into one polynomial–time decidable (for dispersion problems). This phase transition underscores the intricate interplay between algebra, logic, and combinatorics, and opens numerous avenues for future research.

Ultimately, we believe that Term Coding not only sheds new light on classical extremal problems but also offers potential for transformative advances in combinatorial design theory and algorithmic reasoning, by bridging the gap between logical specifications and algebraic structures.

References

[1] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[2] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[3] Thomas J. Schaefer. "The complexity of satisfiability problems". In: *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC)*. 1978, pp. 216–226.

[4] Ronald Fagin. "Generalized first-order spectra and polynomial-time recognizable sets". In: *Complexity of computation* 7 (1974), pp. 43–73.

[5] Neil D Jones and Alan L Selman. "Turing machines and the spectra of first-order formulas with equality". In: *Proceedings of the fourth annual ACM symposium on Theory of computing*. 1972, pp. 157–167.

[6] Etienne Grandjean. "The spectra of first-order sentences and computational complexity". In: *SIAM Journal on Computing* 13.2 (1984), pp. 356–373.

[7] Robert O Robson. "Model theory and spectra". In: *Journal of pure and applied algebra* 63.3 (1990), pp. 301–327.

[8] Maximilien Gadouleau and Søren Riis. "Graph-theoretical constructions for graph entropy and network coding based communications". In: *IEEE Transactions on Information Theory* 57.10 (2011), pp. 6703–6717.

[9] Soren Riis. "Graph entropy, network coding and guessing games". In: *arXiv preprint arXiv:0711.4175* (2007).

[10] Søren Riis. "Information flows, graphs and their guessing numbers". In: *2006 4th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*. IEEE. 2006, pp. 1–9.

[11] Noga Alon et al. "The hat guessing number of graphs". In: *Journal of Combinatorial Theory, Series B* 144 (2020), pp. 119–149.

[12] J. H. Van Lint and R. M. Wilson. *A Course in Combinatorics*. 2nd. Cambridge University Press, 2001.

[13] Peter Keevash. "The existence of designs". In: *arXiv preprint arXiv:1401.3665* (2014).

[14] W Cary Huffman, Jon-Lark Kim, and Patrick Solé. *Concise encyclopedia of coding theory*. Chapman and Hall/CRC, 2021.

[15] Takeuchi & Yahagi. Kobayashi Tsunoda. *Mathematics of information and coding*. Vol. 203. American Mathematical Soc., 2002.

[16] RK Brayton, Donald Coppersmith, and AJ Hoffman. "Self-orthogonal latin squares of all orders n≠2,3,6". In: *Bull. Amer. Math. Soc.* (1974), 80 (1): 116–118.

[17] A Hedayat. "386: Self Orthogonal Latin Square Designs and Their Importance, II". In: *Biometrics* (1975), pp. 755–759.

[18] S Riis. "Information flows, graphs and their guessing numbers. the electronic journal of combinatorics". In: *R44–R44* (2007).

[19] Søren Riis. "Reversible and irreversible information networks". In: *IEEE Transactions on Information Theory* 53.11 (2007), pp. 4339–4349.

[20] Taoyang Wu, Peter Cameron, and Søren Riis. "On the guessing number of shift graphs". In: *Journal of Discrete Algorithms* 7.2 (2009), pp. 220–226.

[21] Peter J Cameron, Anh N Dang, and Soren Riis. "Guessing games on triangle-free graphs". In: *arXiv preprint arXiv:1410.2405* (2014).

[22] Demetres Christofides and Klas Markström. "The guessing number of undirected graphs". In: *the electronic journal of combinatorics* (2011), P192–P192.

[23] Jo Martin and Puck Rombach. "Guessing numbers and extremal graph theory". In: *arXiv preprint arXiv:2009.04529* (2020).

[24] Maximilien Gadouleau, Adrien Richard, and Søren Riis. "Fixed points of Boolean networks, guessing graphs, and coding theory". In: *SIAM Journal on Discrete Mathematics* 29.4 (2015), pp. 2312–2335.

[25] Rahil Baber et al. "Graph guessing games and non-Shannon information inequalities". In: *IEEE Transactions on Information Theory* 63.7 (2016), pp. 4257–4267.

[26] Zhen Zhang and Raymond W Yeung. "A non-Shannon-type conditional inequality of information quantities". In: *IEEE Transactions on Information Theory* 43.6 (1997), pp. 1982–1986.

[27] Leonid Libkin. *Elements of finite model theory*. Vol. 41. Springer, 2004.

[28] Boris A Trakhtenbrot. "Impossibility of an algorithm for the decision problem for finite classes". In: *Doklady Akademiia Nauk SSSR*. Vol. 70. 1950, p. 569.

[29] Søren Riis and Maximilien Gadouleau. "Max-flow min-cut theorems on dispersion and entropy measures for communication networks". In: *Information and Computation* 267 (2019), pp. 49–73.

[30] A Markov. "On certain insoluble problems concerning matrices". In: *Doklady Akad. Nauk SSSR*. Vol. 57. 6. 1947, pp. 539–542.

[31] Emil L Post. "A variant of a recursively unsolvable problem". In: *Bull. Amer. Math. Soc.* 52.4 (1946), pp. 264–269.

[32] Ralph McKenzie. "On spectra, and the negative solution of the decision problem for identities having a finite nontrivial model1". In: *The Journal of Symbolic Logic* 40.2 (1975), pp. 186–196.

[33] Egon Börger, Erich Grädel, and Yuri Gurevich. *The classical decision problem*. Springer Science & Business Media, 2001.

[34] Maximilien Gadouleau and Søren Riis. "Max-flow min-cut theorem for Rényi entropy in communication networks". In: *2011 IEEE International Symposium on Information Theory Proceedings*. IEEE. 2011, pp. 603–607.

[35] Søren Riis. "A complexity gap for tree resolution". In: *Computational Complexity* 10.3 (2001), pp. 179–209.

A Proof of Undecidability of Finite Bijectivity (Problem 2)

We reduce the *Horn–clause finite satisfiability problem* (HFS), known to be undecidable by a version of Trakhtenbrot's theorem, to the following decision problem.

Problem 2 (Finite Bijectivity) Given terms $t_1, \dots, t_k$ over variables $x_1, \dots, x_k$ and a signature Σ, decide whether there is a finite structure (A, I) with $|A| \geq 2$ such that the map

$$F : A^k \longrightarrow A^k, \qquad F(a_1, \dots, a_k) = \big(I(t_1)(\bar{a}), \dots, I(t_k)(\bar{a})\big)$$

is a bijection.

Horn–clause finite satisfiability (HFS) Given a finite Horn theory $\mathcal{H}$ over a relational signature Σ_{rel} (we assume — w.l.o.g. — that every predicate is binary), decide whether there is a finite model $\mathcal{A} \models \mathcal{H}$ with $|\mathcal{A}| \geq 2$.

The reduction proceeds in eight steps.

Step 1: The expanded signature $\Sigma^\star$: For an instance $\mathcal{H}$ over Σ_{rel} we build the signature $\Sigma^\star$ used by the term map.

- **Constants.** c_T, c_F, c_E ("true", "false", "error/padding").
- **Relation encoders.** For every $R \in \Sigma_{\text{rel}}$ add $h^\star_R : A^2 \to A$. Add a designated equality simulator $h^\star_{EQ} : A^2 \to A$.
- **Boolean core.** $\neg^\star : A \to A$ (intended NOT) and $\mathsf{nand}^\star : A^2 \to A$ (intended NAND).
- **Conditional–collapse gadget.** For every $d \geq 1$ the symbol $\mathrm{cond}^\star_{(d)} : A^{1+d} \to A^d$ is interpreted by

$$I\big(\mathrm{cond}^\star_{(d)}(g, x_1, \dots, x_d)\big) = \begin{cases} (I(c_E), \dots, I(c_E)) & \text{if } g = I(c_T), \\ (x_1, \dots, x_d) & \text{otherwise.} \end{cases}$$

  Only the case $d = 1$ is actually used below.
- **Auxiliary symbol.** A ternary $f : A^3 \to A$ (used once, in Step A).

Step 2: Forcing $c_T \neq c_F$: Take fresh variables x', y' and add the two output components

$$f(c_T, c_F, x') \quad \text{and} \quad f(c_F, c_T, y')$$

to the eventual map F'. Should an interpretation I satisfy $I(c_T) = I(c_F)$, both components collapse to the same unary function $(x', y') \mapsto f(a, a, x', y')$, which cannot be surjective on A^2 when $|A| \geq 2$; hence bijectivity of F' enforces $I(c_T) \neq I(c_F)$.

Step 3: Enforcing Boolean behaviour: **NOT.** Include the single-variable component $\mathrm{cond}^\star_{(1)}\big(\neg^\star(\neg^\star(w_{not})), w_{not}\big)$ with fresh input w_{not}. Bijectivity forces $\neg^\star$ to be an involution. Add two further components. $\mathrm{cond}^\star_{(1)}\big(h^\star_{EQ}(\neg^\star(c_T), c_T), w_T\big)$ and $\mathrm{cond}^\star_{(1)}\big(h^\star_{EQ}(\neg^\star(c_F), c_F), w_F\big)$. Together, they ensure that $I(\neg^\star)$ swaps $I(c_T)$ and $I(c_F)$.

NAND. Take three fresh variables w_{n1}, w_{n2}, w_{n3} and insert the triple

$$\big(\mathsf{nand}^\star(w_{n1}, w_{n1}),\ \mathsf{nand}^\star(c_T, w_{n2}),\ \mathsf{nand}^\star(w_{n3}, c_T)\big)$$

as outputs corresponding to the same inputs. This tuple of unary maps forms a permutation of A^3 iff $I(\mathsf{nand}^\star)$ realises true NAND on $\{T, F\}$.

Step 4: Making $h^\star_{EQ}$ real equality:

- *Reflexivity, symmetry, transitivity.* Add output components $\mathrm{cond}^\star_{(1)}\big(\neg^\star(h^\star_{EQ}(x, x)), w_{ref}\big)$, $\mathrm{cond}^\star_{(1)}\big(\neg^\star(h^\star_{EQ}(x, y) \leftrightarrow h^\star_{EQ}(y, x)), w_{sym}\big)$ and an analogous triple-variable component for transitivity, each time with fresh w–inputs.
- *Distinguishing non-equals.* For *every* pair (v_i, v_j) of variables in the original Horn instance introduce
$$\mathrm{cond}^\star_{(1)}\big(h^\star_{EQ}(v_i, v_j), w_{ij}\big).$$
Should $I(h^\star_{EQ})(a, b) = T$ for some $a \neq b$, the map collapses on w_{ij}, contradicting bijectivity.

These gadgets force $I(h^\star_{EQ})$ to agree with actual equality on the subset of A that is reachable through variables occurring in $\mathcal{H}$; call this subset D.

Step 5: Range restriction for relation encoders: For each encoder $h^\star_R$ and every (ordered) pair of variables (u, v) occurring in $\mathcal{H}$ introduce

$$\mathrm{cond}^\star_{(1)}\big(\mathrm{guard}_{\mathrm{range}}\big(h^\star_R(u, v)\big), w_{R,(u,v)}\big), \quad \text{where} \quad \mathrm{guard}_{\mathrm{range}}(x) := \neg^\star\Big(\mathsf{or}^\star\big(h^\star_{EQ}(x, c_T),\ h^\star_{EQ}(x, c_F)\big)\Big).$$

The guard evaluates to T *iff* the value of $h^\star_R(u, v)$ is *neither* T nor F, so the gadget collapses precisely on out-of-range outputs, preventing bijectivity. Hence $I(h^\star_R)$ maps A^2 into $\{T, F\}$.

Step 6: Clause gadgets: For every clause $C \equiv R_1(v_1, v_2) \wedge \cdots \wedge R_m(v_{2m-1}, v_{2m}) \to T(v_{2m+1}, v_{2m+2})$ let $\mathrm{Viol}_C(\bar{v})$ be the term

$$\mathsf{nand}^\star\Big(\mathsf{nand}^\star\big(h^\star_{R_1}(v_1, v_2), \ldots, h^\star_{R_m}(v_{2m-1}, v_{2m})\big), h^\star_T(v_{2m+1}, v_{2m+2})\Big),$$

which yields T exactly when the antecedent of C is satisfied but its consequent is not. Insert the component $\mathrm{cond}^\star_{(1)}\big(\mathrm{Viol}_C(\bar{v}), w_C\big)$ with fresh input w_C. Any violation of C collapses this component and destroys bijectivity.

Step 7: Assembling the global map F': Let $\mathbf{v}$ be the variables of $\mathcal{H}$ and let $\mathbf{w}$ collect *all* auxiliary variables introduced above. The map $F' : A^{\mathbf{v} \cup \mathbf{w}} \to A^{\mathbf{v} \cup \mathbf{w}}$ is obtained by concatenating, in order,

- the two "distinct T/F" outputs of Step A;
- all components from Steps 3–6 (each aligned with its own input);
- identity components $v_i \mapsto v_i$ for every original variable $v_i \in \mathbf{v}$;
- identity components for those auxiliary inputs that have not yet been used as an output.

Step 8: Correctness of the reduction:

($\Rightarrow$) Suppose $\mathcal{H}$ has a finite model $\mathcal{A} = (A, \{R^{\mathcal{A}}\})$, $|A| \geq 2$. Extend $\mathcal{A}$ to an interpretation I by

- picking distinct $I(c_T), I(c_F) \in A$ and an arbitrary $I(c_E) \in A$;
- setting $I(h^{\star}_R)(a, b) = I(c_T)$ iff $(a, b) \in R^{\mathcal{A}}$, else $I(c_F)$;
- interpreting $h^{\star}_{EQ}$ as true equality on A;
- interpreting $\neg^{\star}$ and $\mathsf{nand}^{\star}$ as Boolean NOT and NAND on $\{T, F\}$ and arbitrarily elsewhere;
- choosing any functions for the remaining symbols, respecting the clauses above.

Every guard in the construction evaluates to F, so every $\text{cond}^{\star}$ returns its live input; F' is a variable permutation and thus bijective.

($\Leftarrow$) Conversely, let (A, I) make F' bijective. Steps 2–6 force

$$I(c_T) \neq I(c_F), \qquad I(\neg^{\star}), I(\mathsf{nand}^{\star}) \text{ act as Boolean NOT/NAND},$$

$I(h^{\star}_{EQ})$ to coincide with equality on the set $D \subseteq A$ reached by the variables of $\mathcal{H}$, and every $h^{\star}_R$ to take only T or F. Finally every clause-gadget's guard must be F, hence every clause of $\mathcal{H}$ is satisfied by the relations $R^{\mathcal{A}} := \{(a, b) \mid I(h^{\star}_R)(a, b) = I(c_T)\}$ on the finite domain D.

Conclusion. Because HFS is undecidable and we have provided a computable reduction to Problem 2, Finite Bijectivity is likewise undecidable whenever the underlying signature is allowed to contain the constants and function symbols listed in Step 1. "'

B Encoding First-Order Finite Satisfiability as a Single-Sort Dispersion Problem

This appendix outlines how the problem of determining if a first-order sentence ψ has a finite model can be reduced to a dispersion problem Γ_ψ operating over a single sort. This demonstrates the significant expressive power of the dispersion framework, which relies on maximising the size of an image set subject only to non-equality constraints, rather than using explicit term equations to define function behaviour. The construction shows that dispersion can capture the complexity of first-order logic on finite structures.

Step 1: Logical Preliminaries. First, we convert the given FO sentence ψ (over a signature Σ) into an equisatisfiable sentence $\tilde{\psi}$ in Skolem Normal Form. This $\tilde{\psi}$ has the form

$$\forall \mathbf{x}\, \phi(\mathbf{x}),$$

where ϕ is quantifier-free and uses an expanded signature Σ_{Sk}. Second, we convert $\phi(\mathbf{x})$ into Conjunctive Normal Form (CNF), resulting in

$$\forall \mathbf{x}\, \phi_{CNF}(\mathbf{x}), \quad \text{where} \quad \phi_{CNF} = \bigwedge_i C_i.$$

Each clause C_i is a disjunction of literals (atomic formulas A or their negations $\neg A$), where atoms are typically $R(t_1, \ldots, t_k)$ or $t_1 = t_k$. Let $\mathcal{A}toms$ be the set of atoms in ϕ_{CNF}.

Step 2: The Single Sort and Signature Σ^*. We define a single sort, UniversalSort. An interpretation $\mathcal{I}$ assigns a finite set A to this sort.

- **The Domain A:** For an intended FO model size n, A must contain at least $n+2$ elements. We conceptually partition A into $D \cup \{T, F\} \cup P$, where $|D| = n$ (the FO domain), T, F represent 'True'/'False', and P are padding elements.
- **Signature Σ^*:** Operates on UniversalSort. Includes:
  - **Constants:** c_T, c_F (interpreted as T, F), and $c_E \in P$ (padding/error).
  - **Original/Skolem Functions:** Symbols f^* corresponding to $f \in \Sigma_{Sk}$.
  - **Atomic Predicate Functions:** Symbols A^* for each $A \in \mathcal{A}toms$.
  - **Boolean Simulation Functions:** S^* (NAND), neg^* (NOT), plus auxiliary symbols for their defining 'gadget' maps (e.g., the map T_S used to define S^*, see appendix **??**).
  - **Conditional Function:** $COND^*$ implementing IF-THEN-ELSE on k-tuples, where $k = |\mathbf{x}|$. Built from S^*, neg^*.

Step 3: Constraints: Simulating Logic and Restricting Predicates. The correct behaviour is enforced by non-equality constraints and dispersion maximisation requirements on interpretations $\mathcal{I}$.

- **Simulating Boolean Logic ($\mathcal{C}_{logic}$ and Gadget Maximisation):** We impose non-equalities $\mathcal{C}_{logic}$, including $c_T \neq c_F$, $\text{neg}^*(c_T) \neq c_T$, $\text{neg}^*(c_F) \neq c_F$, $S^*(c_T, c_T) \neq c_T$. Critically, a valid interpretation $\mathcal{I}$ must also *maximise the dispersion of the auxiliary gadget maps* (such as the map $T_S(x, y, z) = (S^*(x, x), S^*(c_T, y), S^*(z, c_T))$ used to define S^* in appendix **??**), subject only to the constraints in $\mathcal{C}_{logic}$. This combined requirement forces $I(S^*)$ and $I(\text{neg}^*)$ to simulate NAND and NOT correctly on the $\{T, F\}$ subset.
- **Predicate Range Restriction ($\mathcal{C}_{range}$):** For each predicate function A^*, we add non-equalities $A^*(x_1, \ldots, x_k) \neq p$ for every padding element $p \in P$ (including $p = c_E$). This ensures the output, for inputs from D, lies within $\{T, F\}$. *This restriction is necessary because the simulated Boolean logic functions built from S^*, neg^* require their inputs to be effectively Boolean (T or F) to guarantee correct logical output.*
- **Interpretation Consistency (Implicit Requirement):** Dispersion maximisation does not force A^* to correctly represent the truth of atom A. The reduction seeks an interpretation $\mathcal{I}$ that simultaneously satisfies the constraints above *and* is consistent with some underlying n-element FO model $\mathcal{M}$ of ψ. *Specifically, 'consistent interpretation' means that for $a_1, \ldots, a_k \in D$, $I(f^*)(a_1, \ldots, a_k) = f^{\mathcal{M}}(a_1, \ldots, a_k)$, and $I(A^*)(a_1, \ldots, a_k)$ equals T if atom A holds for $(a_1, \ldots, a_k)$ in $\mathcal{M}$, and F otherwise.*

Step 4: Constructing the Final Dispersion Map. Translate $\phi_{CNF}(\mathbf{x})$ into a term $\Phi^*(\mathbf{x})$ using A^* and derived Boolean connectives ($AND^*, OR^*, \ldots$). Define the final map F ($k = |\mathbf{x}|$):

$$F(\mathbf{x}) = COND^*(\Phi^*(\mathbf{x}), \mathbf{x}, (c_E, \ldots, c_E)).$$

The conditional function $COND^*$ is essential here. Simply evaluating the dispersion of $\Phi^*(\mathbf{x})$ would yield an image size of 1 or 2. To connect FO satisfaction to maximal dispersion over the n^k inputs in D^k, the map F must output k-tuples. $COND^*$ achieves this: if $\Phi^*(\mathbf{x})$ is T, F outputs the input $\mathbf{x}$; if $\Phi^*(\mathbf{x})$ is F, F outputs the collapse tuple $(c_E, \ldots, c_E)$. Thus, F acts as the identity on D^k (achieving dispersion n^k) only if $\Phi^*(\mathbf{x})$ is always T.*

Step 5: The Dispersion Problem Γ_ψ and Equivalence. Γ_ψ Given Σ^*, the logic gadgets, and $F(\mathbf{x})$, does there exist a finite interpretation $\mathcal{I}$ over A (containing D of size n) such that:

1. $\mathcal{I}$ satisfies $\mathcal{C}_{logic}$, $\mathcal{C}_{range}$, $I(c_T) \neq I(c_F)$, and $\mathcal{I}$ *maximises the dispersion of the logic-defining gadget maps* (subject to $\mathcal{C}_{logic}$).
2. $\mathcal{I}$ is *consistent* (as defined in Step 3) with some n-element structure $\mathcal{M}$ for Σ.
3. The dispersion problem asks if the maximum possible image size for F under such interpretations equals n^k:
$$\max_{\mathcal{I} \text{ satisfying (1, 2)}} \Big|\{F(\mathbf{a}) \mid \mathbf{a} \in D^k\}\Big| \stackrel{?}{=} n^k.$$

Equivalence: The maximum possible dispersion $|\text{Image}_{\mathcal{I}}(F|_{D^k})|$ is n^k. This value is achieved if and only if $F(\mathbf{a}) = \mathbf{a}$ for all $\mathbf{a} \in D^k$, which, by the definition of $COND^*$, occurs precisely when $\Phi^*(\mathbf{a}) = I(c_T)$ for all $\mathbf{a} \in D^k$. Given that condition (1) ensures correct Boolean simulation and condition (2) ensures consistency with a structure $\mathcal{M}$, $\Phi^*(\mathbf{a}) = I(c_T)$ for all $\mathbf{a}$ if and only if $\mathcal{M} \models \phi_{CNF}(\mathbf{a})$ for all $\mathbf{a}$. Therefore, maximal dispersion n^k is achievable if and only if there exists an n-element model $\mathcal{M}$ of $\forall \mathbf{x}\, \phi_{CNF}(\mathbf{x})$, and thus of ψ.

Conclusion. We have outlined how to reduce the FO-FinSat problem for ψ to deciding whether a specific dispersion problem Γ_ψ can achieve maximal dispersion (n^k). This relies on interpretations simultaneously satisfying implicit consistency conditions and explicit requirements to maximise the dispersion of auxiliary logic-defining gadgets while obeying non-equality constraints. This confirms the high expressive power of the single-sort dispersion framework.