ELEKTROTEHNIŠKI VESTNIK 78(4): 181–192, 2011
ENGLISH EDITION

Of n-dimensional Dice, Combinatorial Optimization, and Reproducible Research: An Introduction

Franc Brglez

Computer Science, NC State University, Raleigh, NC 27695, USA
E-mail: brglez@ncsu.edu

Abstract. When throwing a solid object like a hexahedron, an octahedron or a tetrakis-hexahedron on a flat surface, we expect it to roll onto any of the faces with probabilities of exactly 1/6, 1/8, or 1/24, respectively. Informally, we view such objects as instances from the n-dimensional dice family; formally, they are instances from a *hyperhedron* family $\mathcal{H}(\Theta, b, n)$. Each of the faces is assigned a label $\{\underline{\xi}; \Theta(\underline{\xi})\}$; $\underline{\xi}$ represents a unique n-dimensional coordinate string $\underline{\xi}$, $\Theta(\underline{\xi})$ represents the value of the function Θ for $\underline{\xi}$. The number of coordinates is defined as $|\mathcal{H}(\Theta, b, n)| = b^n(n!)$; each coordinate string is an *oriented permutation* with parameter b denoting the number of symbols that encode the unique orientation of each permutation. Special cases include the combinational family $\mathcal{C}(\Theta, b, n)$ with $|\mathcal{C}(\Theta, b, n)| = b^n$ (each coordinate string is a unique n-tuple) and the (single orientation) permutation family $\mathcal{P}(\Theta, b, n)$ with $|\mathcal{P}(\Theta, b, n)| = n!$ (each coordinate string is a unique permutation). The paper introduces the hyperhedron not only as a model for instances that arise in the context of combinatorial optimization but also as a metaphor to illustrate a number of combinatorial search algorithms whose meta-structures do not change when instance coordinates change from an n-tuple to a simple or an oriented permutation of length n. The dice metaphor is applied to statistical performance experiments with instances whose dimension increases monotonically while the number of "best faces" remains constant. All results are archived as an integral part of *reproducible research* environment, controlled by components encapsulated in *tcl*, *R*, and LaTeX.

Keywords: combinatorial algorithms and optimization, dice metaphor, statistical performance experiments, reproducible research, literate programming

1 INTRODUCTION

The ultimate goal of combinatorial optimization is to find an optimal configuration from a finite set of configurations. Instances of combinatorial problems arise in many contexts such as operations research, automated reasoning, computer-aided design, computer-aided manufacturing, machine vision, databases, robotics, scheduling, integrated circuit design, computer architecture design, computer networking, etc. Many problems are NP-hard and, due to the number of feasible configurations, an explicit exhaustive search for an optimum configuration is not possible for most problem instances.

Dice such as a hexahedron, an octahedron or a tetrakis-hexahedron are instances of combinatorial objects with 6, 8, and 24 configurations, with *faces* numbered from 0-to-5, 0-to-7, and 0-to-23, respectively. Assume that any of these dice is thrown repeatedly until the first appearance of *best face* such as "0". Then, the probability distribution of the number of times the dice is thrown is a geometric distribution with $p = 1/6$, $p = 1/8$, and $p = 1/24$, respectively [1]. A larger family of polyhedra is described in [2]. A comprehensive collection of applets, on pages that not only render polyhedra dynamically in three dimensions but also catalogue their parameters, is accessible in [3].

Received August 24, 2011
Accepted November 8, 2011

The probability model associated with the dice examples above suggests a metaphor to view dice as instances of combinatorial problems, generalized as the n-dimensional *hyperhedron* family: $\mathcal{H}(\Theta, b, n)$. Each face of the hyperhedron is assigned a label $\{\underline{\xi}; \Theta(\underline{\xi})\}$; $\underline{\xi}$ represents a unique n-dimensional coordinate string $\underline{\xi}$, $\Theta(\underline{\xi})$ represents the value of the function Θ for $\underline{\xi}$.

A comprehensive introduction to the hyperhedron family of dice is illustrated with examples in Section 2. Section 3 introduces classes of combinatorial problem instances, including a class of the number partitioning problems. In Section 4, the model of dice metaphor illustrates an example of a combinatorial search algorithm whose meta-structures does not change when instance coordinates change from an n-tuple to a simple or an oriented permutation of length n. In Section 5, the dice metaphor is applied to statistical performance experiments with instances whose dimension increases monotonically while the number of "best faces" remains constant. All results are being archived for open access on the Web as an integral part of *reproducible research* environment, controlled by components encapsulated in *tcl*, in *R*, and in LaTeX. Some of these components, combinatorial solvers in particular, are modules natively compiled in Fortran, C, C++, and Java.

The notion of *reproducible research*, discussed in [4], [5] and illustrated in [6], has its roots in *literate programming* [7], [8]. The approach taken in this paper is a pragmatic one: it combines elements of programming in *tcl* [9], [10], [11], *R* [12], [13], [14], and LaTeX [15], with *tcl* as being the top-level driver for each of the experiments. The environment is still evolving; if interested in participation, contact the author.

This paper is an introduction to the series of musings in progress on combinatorial optimization, supported by reproducible research. The dice metaphor introduced in the paper reveals the principal factors that contribute to the notorious performance variability of combinatorial solvers on specific problems. In contrast to Einstein's *God does not play dice*, a number of controlled performance experiments shows that *combinatorial solvers do play dice* on *hard instances*: e.g. large variability in solver performance is reported in [16], [17].

2 Background and Motivation

The subsections below introduce notation, key concepts, and motivation for the remainder of the paper:

- Hasse Graphs
- On Geometric and Exponential Distributions
- Walks in Hyperhedrons: The First Steps

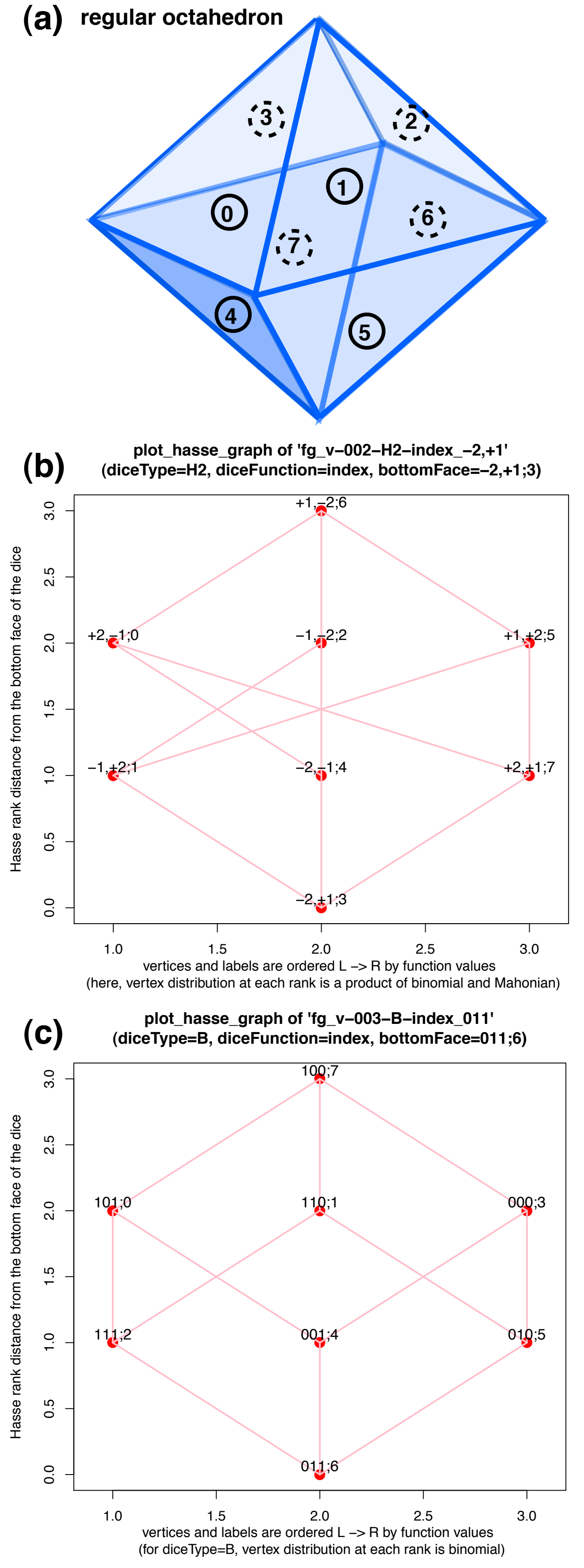
Hasse Graphs. Dice such as hexadron and octahedron introduced in Section 1 and the simple underlying probability model motivates the metaphor to view dice as instances of combinatorial problems, generalized as the n-dimensional *hyperhedron* family $\mathcal{H}(\Theta, b, n)$. Each face of the hyperhedron is assigned a label $\{\underline{\xi}; \Theta(\underline{\xi})\}$; $\underline{\xi}$ represents a unique n-dimensional coordinate string $\underline{\xi}$, $\Theta(\underline{\xi})$ represents the value of the function Θ for $\underline{\xi}$. The number of coordinates is defined as $|\mathcal{H}(\Theta, b, n)| = b^n(n!)$; each coordinate string is an *oriented permutation* with parameter b denoting the number of symbols that encode the unique orientation of each permutation. Special cases of interest include the combinational family $\mathcal{C}(\Theta, b, n)$ where $|\mathcal{C}(\Theta, b, n)| = b^n$ (each coordinate string is a unique n-tuple) and the permutation family with single orientation $\mathcal{P}(\Theta, n)$ where $|\mathcal{P}(\Theta, n)| = n!$ (each coordinate string is a unique permutation). We simplify the notation for $\mathcal{C}(\Theta, b, n)$ further: for $b = 2$ we denote $\mathcal{C}(\Theta, 2, n)$ as $\mathcal{B}(\Theta, n)$ so that $|\mathcal{B}(\Theta, n)| = 2^n$. Similarly, we denote $\mathcal{H}(\Theta, 2, n)$ as $\mathcal{H}2(\Theta, n)$, $\mathcal{C}(\Theta, 3, n)$ as $\mathcal{T}(\Theta, n)$, and $\mathcal{C}(\Theta, 4, n)$ as $\mathcal{Q}(\Theta, n)$. The common model for any n-dimensional *hyperhedron* is the *Hasse graph*: examples for $\mathcal{H}2(\text{index}, 2)$ and $\mathcal{B}(\text{index}, 3)$ are shown in Figure 1; examples for $\mathcal{H}2(\text{index}, 3)$, $\mathcal{B}(\text{npp0}, 5)$, and $\mathcal{P}(\text{jss}, 4)$ are shown in Figure 2.

The *Hasse graph* represents an extension of the *Hasse diagram* [18]: it is a connected, sparse *polar graph* with *labeled vertices* representing faces, and edges representing face adjacencies. In addition to its label $\underline{\xi}; \Theta(\underline{\xi})$, each vertex can also be implicitly assigned a unique Cartesian coordinate (x_i, y_i); the unit of vertex spacing in the x direction is computed from the *maximum Hasse graph width* w_{max}, the unit of the vertex spacing in the y direction is determined by the *maximum Hasse graph height* h_{max}. The unit distance between vertices in the y direction is computed from the *Hasse rank distance* from the *bottom face* represented by the *bottom vertex*. At first glance, the notion of a *hypercube graph* [19] may appear similar since it relates to the special case of $|\mathcal{B}(\Theta, n)| = 2^n$, but the similarity stops here: in a hypercube graph, vertex labels are binary strings only, and, *vertices do not represent the faces of the cube*.

For instances from family $\mathcal{B}(\Theta, n)$, the Hasse rank distance is also known as *Hamming distance*, thus $h_{max} = n$; for instances from family $\mathcal{P}(\Theta, n)$ the Hasse rank distance is the *permutation inversion distance*, thus $h_{max} = n(n-1)/2$; for instances from family $\mathcal{H}2(\Theta, n)$ the Hasse rank distance combines the notions of Hamming and permutation inversion distances and thus $h_{max} = n(n+1)/2$.

Vertices in the x direction are ordered by the function value associated with each vertex label. As the *Hasse rank distance* increases, so does the number of vertices at each distance until the mid-point distance; it decreases afterwards. The vertex distribution is symmetrical with respect to the midpoint distance: for $\mathcal{B}(\Theta, n)$, it is binomial [20], [21] with the Hasse graph width $w_{max} = \binom{n}{\lfloor n/2 \rfloor}$. For $\mathcal{P}(\Theta, n)$, the distribution is mahonian [22] with width $w_{max} = \mathrm{T}(n, \lfloor n(n-1)/4 \rfloor)$. For $\mathcal{H}2(\Theta, n)$, the distribution is hasonian with width $w_{max} = \mathrm{h2}(n, \lfloor n(n+1)/4 \rfloor))$. While binomial, multinomial, and mahonian coefficient distributions are readily available [20], [21], [22], the author is unaware of the distribution named here whimsically as *hasonian* (h2). The name is derived from *HASA*, an acronym to be explained later in this section. However, it is possible to compute vertex distributions in the Hasse graph of $\mathcal{H}2(\Theta, n)$ as a function of Hasse rank distance by combining coefficients from binomial and mahonian distributions as illustrated for $\mathcal{H}2(\text{index}, 3)$ in the example below:

```
(1  +  3  +  3  +  1)(1 + 2 + 2 + 1)
------------------------------------
1  +  3  +  3  +  1
      2  +  6  +  6  +  2
            2  +  6  +  6  +  2
                  1  +  3  +  3  +  1
------------------------------------
1  +  5  + 11  + 14  + 11  +  5  +  1
```


(a) regular octahedron
3
2
1
0
6
7
4
5
(b) plot_hasse_graph of 'fg_v-002-H2-index_-2,+1'
(diceType=H2, diceFunction=index, bottomFace=-2,+1;3)
+1,-2;6
+2,-1;0
-1,-2;2
+1,+2;5
-1,+2;1
-2,-1;4
+2,+1;7
-2,+1;3
Hasse rank distance from the bottom face of the dice
vertices and labels are ordered L -> R by function values
(here, vertex distribution at each rank is a product of binomial and Mahonian)
(c) plot_hasse_graph of 'fg_v-003-B-index_011'
(diceType=B, diceFunction=index, bottomFace=011;6)
100;7
101;0
110;1
000;3
111;2
001;4
010;5
011;6
Hasse rank distance from the bottom face of the dice
vertices and labels are ordered L -> R by function values
(for diceType=B, vertex distribution at each rank is binomial)

(a) The octahedron on the left is a dice, with faces labeled as *numbers* (the singular form of *die* is increasingly uncommon). When thrown, this dice rolls onto any of its faces with the probability p of exactly 1/8. The throw is declared as *success* if the *bottom face* reveals a number, agreed on in advance, as having the *best value*. Suppose one throws this dice repeatedly until the first time "0" appears. Given $t = 1, 2, 3, ...$ as the number of throws, the probability distribution is geometric with $\text{pmf} = (1-p)^{t-1}p$ and the cumulative distribution function is $\text{cdf} = 1 - (1-p)^t$.

(b) The structure on the left is a model of a 2-dimensional dice as a hyperhedron $\mathcal{H}2(\text{index}, 2)$ where each face is assigned a coordinate-value pair $\{\underline{\xi}; \text{index}(\underline{\xi})\}$, here in lex order:

+1,+2;5 +1,-2;6 +2,+1;7 +2,-1;0 -1,+2;1 -1,-2;2 -2,+1;3 -2,-1;4

The function *index*, with range of $[0, 7]$, has a piecewise-linear relationship with coordinates when ordered lexographically, here the *breakpoint* is at +2,-1;0. We use this function for performance testing: the *breakpoint* "hides" the function minimum value from its "expected" coordinate of "+1,+2".

(c) The structure on the left is a model of a 3-dimensional dice as a hyperhedron $\mathcal{B}(\text{index}, 3)$ where each face is assigned a a coordinate-value pair $\{\underline{\xi}; \text{index}(\underline{\xi})\}$, here in lex order:

000;3 001;4 010;5 011;6 100;7 101;0 110;1 111;2

The function *index*, with range of $[0, 7]$, has a piecewise-linear relationship with coordinates when ordered lexographically, here the *breakpoint* is at 101;0.

(b,c) We model the hyperhedron as a *Hasse graph*: a connected, sparse polar graph with *labeled vertices* representing faces, and edges representing face adjacencies – this model extends the *Hasse diagram* [18]. In addition to its label $\underline{\xi}; \Theta(\underline{\xi})$, each vertex is implicitly assigned a unique Cartesian coordinate (x_i, y_i); units of vertex spacing in the x, y directions are computed from the *Hasse graph width* w_H and *Hasse graph height* h_H, respectively. The unit distance between vertices in the y direction also represents the *Hasse rank distance*. For instances from family $\mathcal{B}(\Theta, n)$, the Hasse rank distance is known as *Hamming distance*, with maximum value $h_{max} = n$; for instances from family $\mathcal{P}(\Theta, n)$ the Hasse rank distance is known as the *permutation inversion distance*, with maximum value $h_{max} = n(n-1)/2$; for instances from family $\mathcal{H}2(\Theta, n)$ the Hasse rank distance combines the notions of Hamming and permutation inversion distances, with maximum value $h_{max} = n(n+1)/2$.

Vertices in the x direction are ordered by the function value associated with each vertex label. The Hasse graph width ranges from $\binom{n}{\lfloor n/2 \rfloor}$ for instances from family $\mathcal{B}(\Theta, n)$ to $h2(n, \lfloor n(n+1)/4 \rfloor)$ for instances from family $\mathcal{H}2(\Theta, n)$, which represents the center coefficient from the *hasonian distribution* defined in Section 2. For examples of Hasse graphs in higher dimensions, see Figure 2.

Figure 1. A regular octahedron and two of its dice models: $\mathcal{H}2(\text{index}, 2)$ and $\mathcal{B}(\text{index}, 3)$.

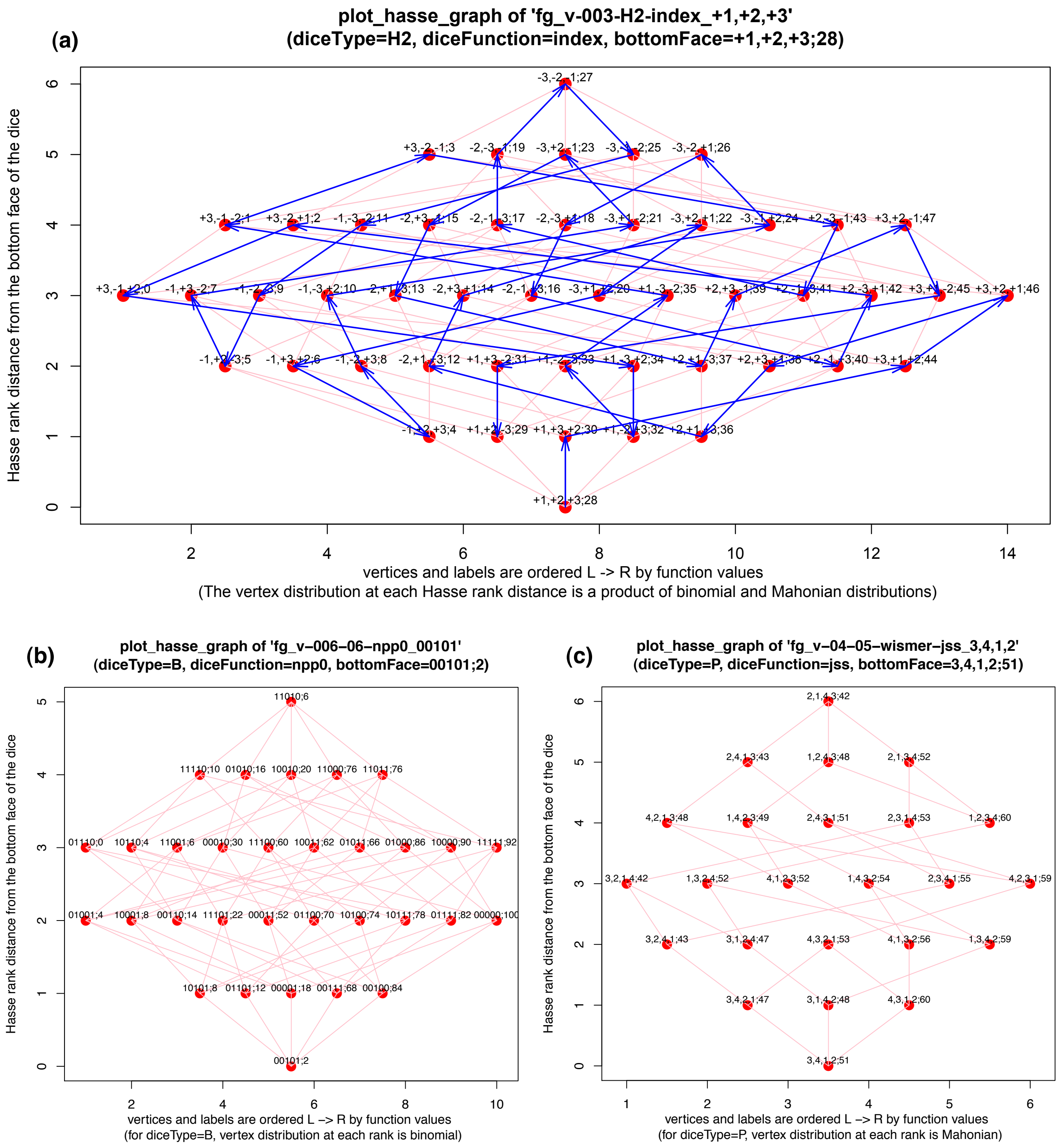

(a)
plot_hasse_graph of 'fg_v-003-H2-index_+1,+2,+3'
(diceType=H2, diceFunction=index, bottomFace=+1,+2,+3;28)
Hasse rank distance from the bottom face of the dice
vertices and labels are ordered L -> R by function values
(The vertex distribution at each Hasse rank distance is a product of binomial and Mahonian distributions)
(b)
plot_hasse_graph of 'fg_v–006–06–npp0_00101'
(diceType=B, diceFunction=npp0, bottomFace=00101;2)
Hasse rank distance from the bottom face of the dice
vertices and labels are ordered L –> R by function values
(for diceType=B, vertex distribution at each rank is binomial)
(c)
plot_hasse_graph of 'fg_v–04–05–wismer–jss_3,4,1,2'
(diceType=P, diceFunction=jss, bottomFace=3,4,1,2;51)
Hasse rank distance from the bottom face of the dice
vertices and labels are ordered L –> R by function values
(for diceType=P, vertex distribution at each rank is Mahonian)

Figure 2. Examples of three dice models: (a) represents a 3-dimensional dice $\mathcal{H}2(\text{index}, 3)$ with 48 faces, Hasse graph height $h_{max} = 6$, Hasse graph width $w_{max} = 14$, with coordinates as oriented permutations of length 3 in two orientations, and the integer function *index* computing values on an *index* problem instance with a single minimum value of "0" at +3,-1,+2; (b) represents a 5-dimensional dice $\mathcal{B}(\text{npp0}, 5)$ with 32 faces, $h_{max} = 5$, $w_{max} = 10$, with coordinates as binary strings of length of 5, and the integer function *npp0* computing values on a *number-partitioning-problem* instance with a single minimum value of "0" at 01110; (c) represents a 4-dimensional dice $\mathcal{P}(\text{jss}, 4)$ with 24 faces, $h_{max} = 6$, $w_{max} = 6$, with coordinates as permutations of length 4, and the integer function *jss* computing values on a *job-shop-scheduling* instance with two minima of "42" at 3,2,1,4 and 2,1,4,3. A *Hamiltonian path* such as shown in (a) also exists for instances shown in (b) and (c).

Table 1. Hasse graph parameters for three families of n-dimensional dice.

Family	dimension	degree	$\|V\|$	$\|E\|$	density	densApprox	height	width
$\mathcal{H}2(\Theta, n)$	n	$2n-1$	$2^n(n!)$	$(2n-1)*$ $2^{n-1}(n!)$	$d(\|E\|, (\|V\|)$	$1/$ $(2^{n-1}(n-1)!)$	$n(n+1)/2$	$h2(n, ...)$
	2	3	8	12	0.4285714	0.5000	3	3
	3	5	48	120	0.1063830	0.1250	6	14
	4	7	384	1344	0.0182768	0.0208	10	82
	...	...	...	...	...	...	...	...
$\mathcal{B}(\Theta, n)$	n	n	2^n	$n * 2^{(n-1)}$	$d(\|E\|, (\|V\|)$	$n/2^n$	n	$\binom{n}{\lfloor n/2 \rfloor}$
	3	3	8	12	0.4285714	0.375	3	3
	4	4	16	32	0.2666667	0.25	4	6
	5	5	32	80	0.1612903	0.15625	5	10
	10	10	1024	5120	0.0097751	0.0097656	10	252
	20	20	1048576	10485760	1.907e-05	1.907e-05	20	184756
	...	...	...	...	...	...	...	...
$\mathcal{P}(\Theta, n)$	n	$n-1$	$n!$	$(n-1)(n!)/2$	$d(\|E\|, (\|V\|)$	$1/(n-1)!$	$n(n-1)/2$	$\mathrm{T}(n,$ $\lfloor n(n-1)/4 \rfloor)$
	3	2	6	6	0.4000000	0.5	3	3
	4	3	24	36	0.1304348	0.1666667	6	6
	5	4	120	240	0.0336134	0.0416667	10	22
	6	5	720	1800	0.0069541	0.0083333	15	101
	7	6	5040	15120	0.0011907	0.0013889	21	573
	...	...	...	...	...	...	...	...

The generic formula we use for density is $d(|E|, (|V|) = 2 * |E|/(|V| * |V| - 1))$
The parameters height and width imply *maximum height*, *maximum width*.
The explicit formula for the *Hasse graph width* of the dice family $\mathcal{H}2(\Theta, n)$ is yet to be derived.

To verify this computation, see the vertex distribution in Figure 2(a). For a summary of Hasse graph parameters, including the rapidly decreasing graph density (or conversely, the increasing graph sparsity) as the graph dimension (number of vertices) increases, see Table 1. The most important notions about Hasse graphs, whether or not they are of size that can be plotted in full detail as shown here, include:

Hasse graph height, width, density: Maximum height of each graph in $\mathcal{B}(\Theta, n)$ is n, in $\mathcal{P}(\Theta, n)$ it is $n(n-1)/2$, and in $\mathcal{H}2(\Theta, n)$ it is $n(n+1)/2$. The maximum width of each graph in $\mathcal{B}(\Theta, n)$ is $\binom{n}{\lfloor n/2 \rfloor}$, in $\mathcal{P}(\Theta, n)$ it is $\mathrm{T}(n, \lfloor n(n-1)/4 \rfloor))$, and in $\mathcal{H}2(\Theta, n)$ it is still to be determined from $\binom{n}{\lfloor n/2 \rfloor}$ and $\mathrm{T}(n, \lfloor n(n-1)/4 \rfloor))$. Here $\mathrm{T}(n, \lfloor n(n-1)/4 \rfloor))$ represents coefficients in the Mahonian distribution [22]. As the dimension n increases, the density of each Hasse graph rapidly approaches 0, though it can never become 0.

Hasse graph degree, Hamiltonicity: The degree of each vertex in $\mathcal{B}(\Theta, n)$ is n, in $\mathcal{P}(\Theta, n)$ it is $n-1$, and in $\mathcal{H}2(\Theta, n)$ it is $2n-1$. Every Hasse graph in $\mathcal{B}(\Theta, n)$ is Hamiltonian for $n > 1$, so is every Hasse graph in $\mathcal{P}(\Theta, n)$ and in $\mathcal{H}2(\Theta, n)$. A Hamiltonian path traverses the vertices and edges by tracing vertex coordinates in the Gray order. Gray order coordinates in class $\mathcal{B}$ are generated with the algorithm in [23]. Gray order coordinates in class $\mathcal{P}$ are based on ideas from a bell-ringing scheme [24], dating from 1850, which are simpler to implement than the better known Steinhaus-Johnson-Trotter algorithm [25]. Gray order coordinates in class $\mathcal{H}2$ are implemented by combining the algorithms in [23] and [24].

top face, top vertex: a specific label $\{\underline{\xi}_t; \Theta(\underline{\xi}_t)\}$, a coordinate and function value pair, associated with the top vertex, given that the Hasse graph is a polar graph by definition.

bottom face, bottom vertex: a specific label $\{\underline{\xi}_b; \Theta(\underline{\xi}_b)\}$, a coordinate and function value pair, associated with the bottom vertex. Metaphorically, the bottom face represents the face on which the dice lands after a random throw.

Hasse rank distance: A measure of distance between two coordinates. Frequently but not always, one of the coordinates is the *bottom vertex* which induces the ordering and position in the Hasse graph vertices as shown in Figures 1 and 2. For instances from family $\mathcal{B}(\Theta, n)$ the Hasse rank distance is also known as *Hamming distance*. For instances from family $\mathcal{P}(\Theta, n)$ the Hasse rank distance is the *permutation inversion distance*. For instances from family $\mathcal{H}2(\Theta, n)$ the Hasse rank distance combines the definitions of Hamming and permutation inversion distances.

coordinate complement: implies a pairing of two coordinates: a reference coordinate and its complement coordinate with the property that the Hasse rank distance between the two is the maximum distance – equivalent to the height of the Hasse graph defined for $\mathcal{B}(\Theta, n)$, $\mathcal{P}(\Theta, n)$, and $\mathcal{H}2(\Theta, n)$

earlier. As also seen in Figures 1 and 2, coordinates associated with the bottom face and the top face are complements of each other.

function symmetry: We say that the function is symmetric with respect to its complement coordinate if for each coordinate and its complement the function evaluates to the same value. In practice, such cases do arise. For example, consider a list of integers $\{4\ 5\ 7\ 8\ 35\ 41\}$ which associates with a 6-dimensional dice $\mathcal{B}(\text{npp}, 6)$ where 'npp' is the name of the *number-partitioning function*. This function returns a 'discrepancy' of 0 for any coordinate that represents a perfect partition. For this example, there are two coordinates with perfect partitions: 001110 and 110001 which we can write either as

$$4 + 5 + 41 = 7 + 8 + 35$$

or

$$7 + 8 + 35 = 4 + 5 + 41$$

The coordinates 001110 and 110001 are complements of each other and we can reduce the dimensions of the Hasse graph from 6 to 5 by removing the first variable, making the modified function 'npp0' unsymmetrical in the remaining 5 variables. As shown in Figure 2(b) the Hasse graph $\mathcal{B}(\text{npp0}, 5)$ now has 01110 as the only coordinate with the function value of "0". When a function is known to be intrinsically symmetric, we can reduce the number of its configurations by making it unsymmetrical as shown here. On the other hand, consider the 4-dimensional dice $\mathcal{P}(\text{jss}, 4)$ in Figure 2(c): it has 24 faces with 'jss' defined as a *job-shop-scheduling* function [26] that exhibits two minima of "42" at 3,2,1,4 and 2,1,4,3. These two coordinates are not complements of each other, hence 'jss' is not symmetric. We could arrive at the same conclusion about symmetry just by observing the labels associated with the bottom and the top face of this dice: '3,4,1,2;51' and '2,1,4,3;42'. While the coordinates in these two label are complements of each other, the 'jss' function evaluates to 51 and 42 respectively.

HASA, Hyperhedron Association for Scientific Applications: Extrapolating from the examples in Figure 2, the height of the Hasse graph increases at most as $O(n^2)$ while the width increases as $O(\alpha^n(n!))$ where $\alpha < b$ and b denotes the number of symbols that encode the unique orientation of each permutation. In other words, as n increases, each hyperhedron may be seen as a huge, almost flat, galaxy of stars that form an extremely sparsely connected polar graph. Each of these stars is characterized by a unique coordinate and an internal temperature that can only be determined by probing in situ. In principle, the dice throwing metaphor for finding the planet with lowest (highest) temperature still applies; however, we argue for the application of scientific methods to finding solutions that will scale better as the dimensionality of the problem space increases.

Geometric and Exponential Distributions Primer. A cummulative distribution for dice has been introduced in Figure 1:

$$\text{cdf} = 1 - (1 - p)^t$$

where p implies the probability of success and $t = 1, 2, 3, ...$ is the number of throws required to observe the first success. Nominally, dice are expected to have only a single designated face that is considered "success". We proceed with this analogy in this paper, and, without loss of generality, consider only hyperhedrons whose combinatorial functions have an optimum value at a single coordinate. Generalizations to functions with multiple occurrences of optima are straightforward.

We consider problems where $p << 1$ so we can approximate the discrete cdf above with the continuous exponential function $\text{cdf} = 1 - exp(-pt)$ where $\text{mean} = 1/p$, $\text{median} = ln(2)/p$, and standard deviation $\text{sd} = 1/p$. Furthermore, we rewrite this cdf as a *solvability function* [27] which in the current context takes the form

$$\mathcal{S}(\tau) = 1 - exp((-pb_f)\tau))$$

where τ refers to a standardized unit of cost to find the optimal solution and b_f is the *boost factor* associated with the characteristics of the solver. The cost that dominates finding an optimum solution to combinatorial problem is the cost of evaluating the combinatorial function; it may take a significant fraction of CPU time. In this paper, we refer to it as the cost of *probing* and we measure it strictly by incrementing the *probe counter* each time we evaluate the function. Compared to the cost of probing, we consider the cost of generating and comparing coordinate values as negligible.

For example, if we employ a gambler as the solver with a strategy of *probing* for best value by simply tossing a dice and evaluating the function value for each toss counted as t, then $\tau = t$, $b_f = 1$ and the mean value to reach the optimum is $1/p$. However, as we show shortly in this section, if the gambler changes the strategy by following each toss (and a probe) with a segment of random walk that probes coordinates without repetition, this strategy rewards the gambler with a boost factor of $b_f = 2$. Not surprisingly, the value of about $b_f = 2$ is expected also if we are to perform a Hamiltonian walk which, by definition, visits each vertex only once.

Choosing a gambler to solve combinatorial problems optimally is not expected to scale as the dimensions of the problem increase (unless there is an proportionate increase in the number of optima). However, the model does provide a useful metaphor for walk strategies we

Table 2. Experiments with $\mathcal{B}(\text{index}, 5)$: gambler's approaches to finding the face with minimum value.

$\mathcal{B}(\text{index}, 5)$	Throws only		Walks With Repetitions					Walks With No Repetitions					
instance:	all	uniq	throws	probes	throws	walk	dist.	throws	probes	throws	walk	tries	dist.
firstSeed					+probes	length				+probes	length		
0:1776	11	10	1	29	30	79	3	1	20	21	19	63	3
1:61110	74	29	1	23	24	46	2	1	5	6	4	17	2
2:27423	4	4	1	14	15	18	4	1	23	24	22	72	4
3:16076	1	1	1	1	1	0	0	1	1	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
127:68095	49	27	1	27	28	72	4	3	26	29	25	76	2
128:6568	11	8	1	28	29	71	3	1	4	5	3	14	1
median	21.5	17.0	1	16.0	17.0	22.5	3.0	1.0	15.5	16.5	14.5	52.0	2.0
mean	32.9	16.8	1	16.1	17.0	33.1	2.4	1.3	15.1	16.3	14.1	47.1	2.4
stdDev	31.7	9.3	0	9.0	9.1	31.2	1.2	1.0	9.0	9.5	9.0	25.3	1.3
stdErr	2.8	0.8	0	0.8	0.8	2.8	0.1	0.1	0.8	0.8	0.8	2.2	0.1

(1) The count of operations stops on the first observance of the face with minimum value, computed by the function 'index'.
(2) The value of 'dist.' measures Hasse rank distance, here Hamming distance, from the coordinate with minimum value to the *dice bottom*, determined by first throw (associated with *firstSeed*).
(3) All statistics are based on 128 instances of $\mathcal{B}(\text{index}, 5)$, each associated with a randomly selected *firstSeed*. Operation counts for the first instance, instance '0', are not included.

introduce later on, strategies that are also based on rules and not chance alone.

Walks in Hyperhedrons: The First Steps. The initial group of experiments that sets the tone is summarized in Table 2. Here, the gambler engages in three strategies using the hyperhedron $\mathcal{B}(\text{index}, 5)$. In the experiment, gambler uses 129 hyperhedron instances for each strategy: the seed for the first instance is chosen by the gambler and is used to generate the random seeds for the next 128 instances. Only the last 128 instances are used to evaluate the reported statistics.

The first strategy is based on random throws only: we observe the mean number of throws as 32.9 and the standard deviation as 31.7. The second strategy is based on a random walk following the first throw where coordinate repetitions are allowed: we observe the mean value of walk length as 33.1 and the standard deviation as 31.2. The third strategy is based on a random walk where coordinate repetitions are not allowed. This restriction may terminate the walk before reaching the optimum, but the gambler is allowed to repeat throws and walks until the optimum value is found. Here, we observe the mean value of walk length as 14.1 and the standard deviation as 9.0.

We use Hasse graphs to illustrate the two walking strategies employed by the gambler, see Figure 3(a-c). The statistics such as shown in Table 2 are computed for hyperhedrons with dimensions of $n = 5, 6, 7, 8, 11, 13, 15$ and are plotted in Figure 3(d). The parameter "cntOps" implies the mean values of random throws (xRT), the mean values of random walk lengths for random walks with coordinate replication (xRW1), and the mean values of random walk lengths for random walks *without* coordinate replication (xRW2). Clearly, both xRT and xRW1 have the boost factor of $b_f \approx 1$ as a statistician would readily predict, and for xRW2 we observe the boost factor of $b_f \approx 2$.

3 Problem Instance Selection

Instances of problems that arise in practice have typically more than one optimal solution; the more such solutions, the easier are they to solve in principle. In this paper, we normalize all instances to having a single optimum: we achieve this property either by direct instance construction (functions in the *index* family) or by iterated sieving methods that also include removing the symmetries in the original problem (generating functions in the *npp0* family).

Instances in the *index* family. We provide two sets of instances based on related functions: *index* and *index-perm*. The function *index* is piewise-linear as described in Figure 1, function values of *index-perm* are created by randomly permuting values of *index* under a lexographical coordinate order for both. Scatter plots differences between these two functions under lexographical order are significant as shown for a 5-variable example in Figures 5-(b) and 5-(d) (fully introduced in Section 5). The purpose of the function *index* in this form is for performance testing when n is increasing: the *breakpoint* "hides" the function minimum value from its "expected" coordinate of "00000".

The purpose of the function *index-perm* is to show that *order matters*: compared to *index*, instances based on *index-perm* are significantly harder to solve optimally with the method we use in this paper. Three families of dice instances can be tested with the function *index*: $\mathcal{H}2(\text{index}, n)$, $\mathcal{B}(\text{index}, n)$, $\mathcal{P}(\text{index}, n)$. Similar

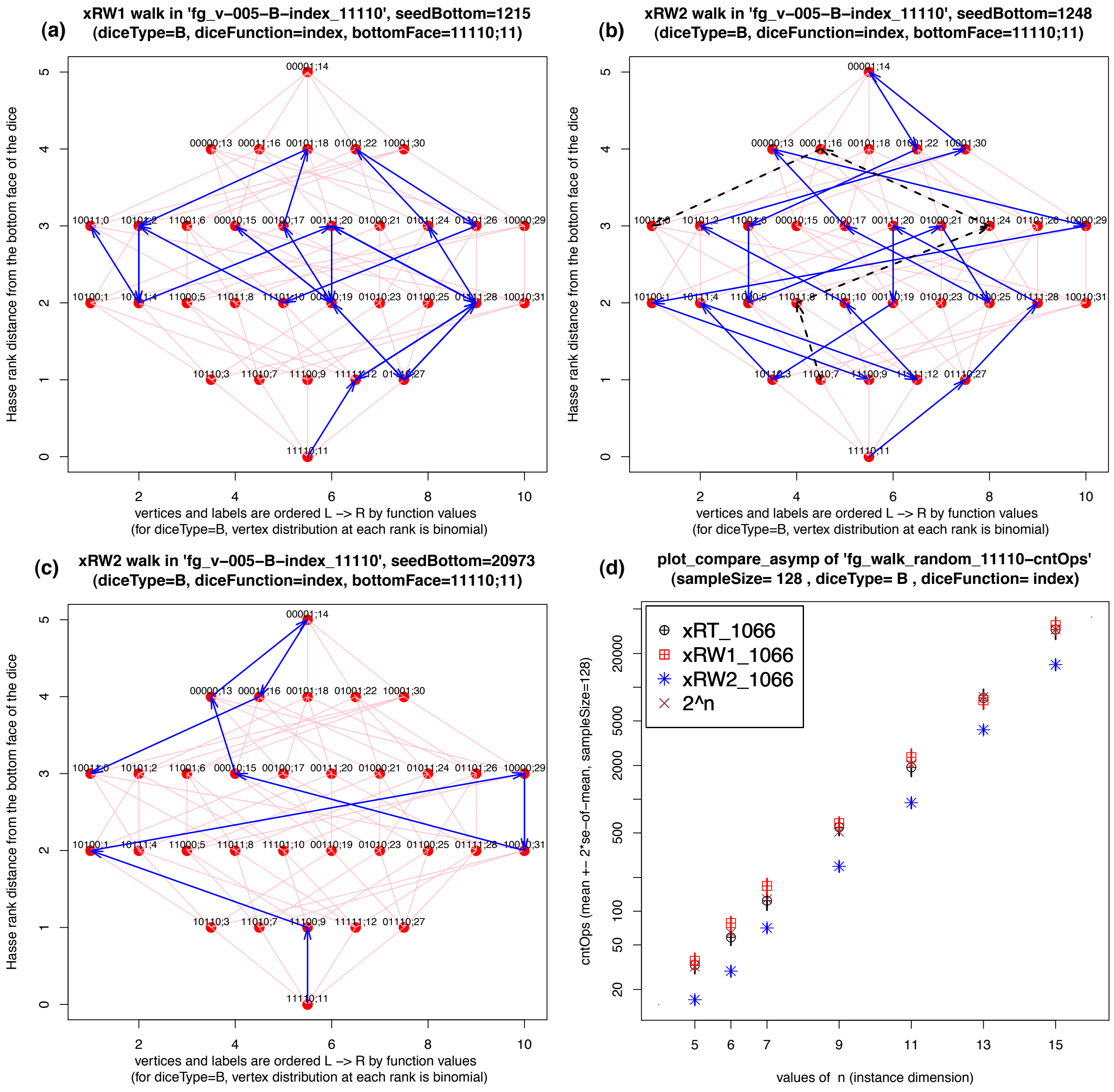


Figure 3. Hasse graphs illustrate the walk strategies summarized in Table 2. In (a), a single throw (bottom coordinate 11110) is followed by a random walk *with* coordinate replication until the minimum value of "0" is found at 10011. In (b), a random walk *without* coordinate replication, is blocked at the coordinate 11100 (immediately above the bottom coordinate) and another throw (which landed a 11010) is needed before reaching the minimum value of "0". In (c), a walk *without* coordinate replication finds the minimum value of "0" in just 9 steps after the first throw. In (d), we compare statistics for strategies outlined in Table 2 as dice increase in size.

choices are available for the the function *index-perm*.

Instances in the *npp0* family. We provide a set of instances based on functions that evaluate the *number partitioning problem*, npp [28], [29], [30], [31]. All instances are normalized to have a single optimum by an iterated sieving method (to be described elsewhere). The method also removes the *function symmetry*, as already explained in the preceding section for the *npp0* family of functions. Properties of these instances include:

- the number of bits used to construct each integer in the list and the total number of integers in the list to be partitioned is the same (a conjectured prerequisite for the hardest instance with perfect partition);
- under the function *npp0*, each instance has one and only one perfect partition with discrepancy = 0;

- when solved by the Largest Differencing Method (LDM) algorithm [31], each instance reports discrepancy > 0 (i.e. the LDM heuristic fails to find the perfect partition).

An example of a *npp0* function where 5-variable binary coordinates are listed in *lexographical order* is shown in Figure 5-(c) (fully introduced in Section 5). Clearly, there is no obvious correlation of function values to this variable order; the scatter plot is similar when binary coordinates are listed in *gray order*. Scatter plots such as this one may indicate that the problem will be hard to solve for its optimum value with any search method; this conjecture is confirmed with experimental results in this paper as well as anticipated earlier in [32].

4 Stochastic Search Strategies

Neither the tossing nor the combinations of tossing and walking strategies introduced by the *gambler* in Section 2 scale adequately for sizes of problems encountered in practice. Today, all stochastic search methods rely on strategies that are based not only on a combination of random choices but also on rules where decisions depend on dynamic probing the function values of the instance being solved.

Historical milestones on stochastic search methods include the Metropolis-Hastings algorithm [33], [34], simulated annealing, based on a model of a cauldron of hot liquid where liquid is crystalizing under a controlled cooling schedule [35], [36], Gibbs sampling [37], and tabu search that uses memory structures to describe the visited solutions [38], [39]. Entries on these topics are continually updated on Wikipedia, and expanded with new metaphors such as ant colonies [40], bird flocks and particle swarms [41], natural disasters [42], genetics and biological processes [43], etc.

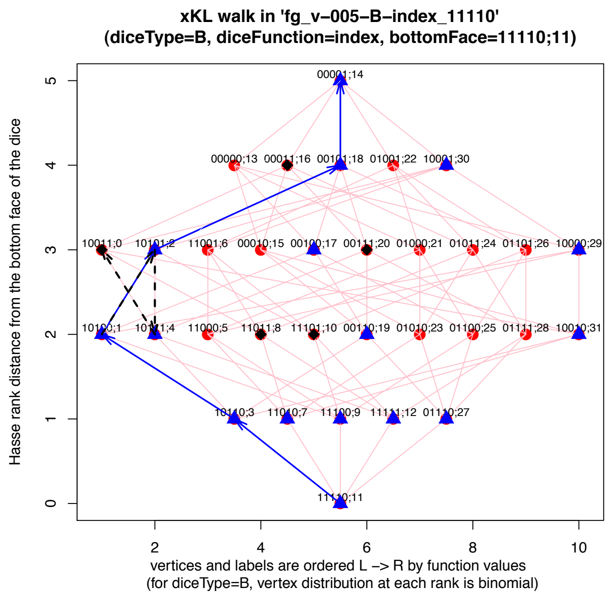
The approach described briefly in this paper has roots in a deterministic hill-climbing procedure with a well-defined stopping criterion, known as the Kernighan-Lin (KL) algorithm [44], [45] which can also be applied in contexts other than graph partitioning [46], [47]. A version of *randomized KL* has been introduced in [47] by repeated application of a random toss, followed by an invocation of the KL algorithm until the specified objective has been met. In the follow-up experiment, the performance of the randomized KL algorithm could be compared fairly with the best known version of the *Evolutionary Search (ES)* algorithm, applied to the notoriously hard *low autocorrelation binary sequence (labs)* problem [48]. Surprisingly, the randomized KL not only held ground against ES in the operations count (number of function evaluations), its implementation was also significantly faster, so that larger problems could be solved with KL for the (then) known exact values provided by the branch-and-bound method [49].

The brief description and the context of the *randomized KL* algorithm in this paper is different from the one in [47]. Foremost, the flow of execution is mapped onto a Hasse graph and proceeds from the bottom face: the first vertex whose label (coordinate;value pair) is also known as the first *pivot pair*. Before making a step (or walk) to a new pivot pair, all adjacent vertices at the Hasse rank distance of 1 *above* the current pivot are probed for function value. A new pivot pair is chosen from the list of most recently probed vertices with the 'best' (minimum) function value. The vertex probing, the pivot selection, and the walk continues upwards until reaching the top face (the last pivot pair in the first walk). At this point, the first walk is terminated, and the next bottom face is selected from the list of current pivots as the pivot with the 'best' (minimum) function value. A second walk is now initiated from the new bottom face, and the process repeats. The probing and the walk is terminated either when the value associated with the best pivot pair is repeated or when all vertices adjacent to the current pivot pair have been visited already.

The walking process just described is illustrated for the 5-variable binary function *index* with a Hasse graph in Figure 4: the first walk is depicted with the solid lines (blue), the second walk is depicted with dashed lines (black). While it is not immediately obvious from Figure 4 that the second walk is again from new bottom face; this is indeed the case and can be illustrated by drawing a new Hasse graph, now with '10100;1' as the new bottom face in place of '11110;11' shown in the current graph.

Vertices probed during the first walk are marked with triangles (blue), vertices that were probed during the second walk are marked with solid black circles. The spreadsheet adjacent to the Hasse graph provides additional details, including explicit markings for the neighborhood vertices that were probed before making the selection about the choice of the next best pivot. It should also be noted that pivot pairs are *never marked*, even if they were marked as 'neighbors' earlier. Pivot pairs can also re-appear in the follow-up walk.

As shown in in Figure 4, the number of probes taken by the first KL walk for hyperhedrons in the family $\mathcal{B}(\Theta, n)$ is $O(0.5n^2)$ since the height of the Hasse graph is is n. However, the description of the walk would not change if we were considering hyperhedron under different coordinate systems such as $\mathcal{P}(\Theta, n)$ or the family $\mathcal{H}2(\Theta, n)$. What would change is the number of probes during the walk. In either case, the number of probes taken by the first KL walk would be $O(n^4)$, primarily due to the increased height in each Hasse graph; details will be provided elsewhere.


xKL walk in 'fg_v–005–B–index_11110'
(diceType=B, diceFunction=index, bottomFace=11110;11)
Hasse rank distance from the bottom face of the dice
vertices and labels are ordered L –> R by function values
(for diceType=B, vertex distribution at each rank is binomial)
00001;14
00000;13
00011;16
00101;18
01001;22
10001;30
10011;0
10101;2
11001;6
00010;15
00100;17
00111;20
01000;21
01011;24
01101;26
10000;29
10100;1
10111;4
11000;5
11011;8
11101;10
00110;19
01010;23
01100;25
01111;28
10010;31
10110;3
11010;7
11100;9
11111;12
01110;27
11110;11

probes	probes Cum	pivotPair	neighbor 1	neighbor 2	neighbor 3	neighbor 4	neighbor 5
5	5	11110;11	10110;3	11010;7	11100;9	11111;12	01110;27
4	9	10110;3	10100;1	10111;4	00110;19	10010;31	NA
3	12	10100;1	10101;2	00100;17	10000;29	NA	NA
2	14	10101;2	00101;18	10001;30	NA	NA	NA
1	15	00101;18	00001;14	NA	NA	NA	NA
0	15	00001;14	NA	NA	NA	NA	NA
0	0	10100;1	10101;2*	10110;3*	11100;9*	00100;17*	10000;29*
1	1	10101;2	10111;4*	11101;10	00101;18*	10001;30*	NA
2	3	10111;4	10011;0	11111;12*	00111;20	NA	NA
2	5	10011;0	11011;8	00011;16	NA	NA	NA
1	6	11011;8	01011;24	NA	NA	NA	NA
0	6	01011;24	NA	NA	NA	NA	NA
0	0	10011;0	10111;4*	11011;8*	00011;16*	10001;30*	10010;31*
0	0	10111;4	10101;2*	10110;3*	11111;12*	00111;20*	NA

A * attached to the value of the function denotes that the coordinate associated with this value has been visited already, for example 10101;2 is marked as 10101;2* -- however, only vertices in neighbor columns are so marked, not vertices found in the pivotPair column!!

Figure 4. The Hasse graph and the spreadsheet next to it illustrate a rule-based walk; in the original context, these rules are named as *Kerninghan-Lin* or KL. The walk starts at the bottom vertex (*pivot pair*) and before choosing the next vertex, all adjacent vertices *above the current vertex*, are *probed* and *marked*, the vertex with the " best value" is chosen as the next *pivot pair*. After the walk of n steps, we reach the top vertex and choose the *best pivot pair* as the new bottom vertex to continue the walk. Traditionally, the walk is terminated when the value associated with *best pivot pair* is repeated. However, as shown in Figure 3, termination criteria can also consider whether vertices adjacent to the current pivot pair have been visited already.

5 Experimental Results

Experiments were performed with a version of randomized KL algorithm (xKL) described in Section 4 on instances from families *index*, *index-perm*, and *npp0* introduced in Section 3. Asymptotic results of experiments, as summarized for values of n ranging from 5 to 19 in Figure 5, are not entirely surprising. The exponential cost of solving npp0 instances has already been predicted in [32] and is not likely to change with the change of the (stochastic search) algorithm. What may be surprising is the dramatic effect of random permutation on the piecewise linear function that produces *index-perm*. While the performance of the solver xKL is $O(n^{2.25})$ on instances from *index*, the same solver performance on instances from *index-perm* is $O(2^n)$ – comparable to the performance of the random walk with repetition (xRW1) introduced in Section 2. Applying the random walk *without* repetition, xRW2 from Section 2, on instances from *index-perm* improves on the performance of xKL by a factor of 2.

6 Summary and Conclusions

We introduce and argue for the merits of the hyperhedron not only as a model for instances that arise in the context of combinatorial optimization but also as a metaphor to illustrate a combinatorial search algorithm, xKL in this paper, whose meta-structure does not change when instance coordinates change from an n-tuple to a simple or an oriented permutation of length n.

In this paper, we introduce problem instances where the cardinality of dice coordinates for each instance maps directly to b^n, $n!$, or $b^n(n!)$, respectively. However, some problem instance classes are expressed with only subsets of these coordinates or coordinates may not even be discrete. In each case, most suitable mapping to dice coordinates such as defined here needs to be decided by the user. One example is the class of Golomb Rulers which continues to consume considerable computational resources: the current search for the optimal GR is running at $n = 27$ since 2009 [50]. Another example is the class of instances defined in terms cartesian coordinates in k-dimensional space [51]: the objectives here include the highly multimodal Rastrigin function with a single global optimum [52]. Computational experiments with these and similar problem instances are in progress and will be reported elsewhere.

Contents for all figures in this paper, including automatic generation of figure labels and captions for inclusion in the LaTex source files, can be reproduced by re-running a few scripts in tcl and R – for any instance parameters and choices of (encapsulated) solvers. If interested in getting an early version of this (xBed) environment, contact the author.

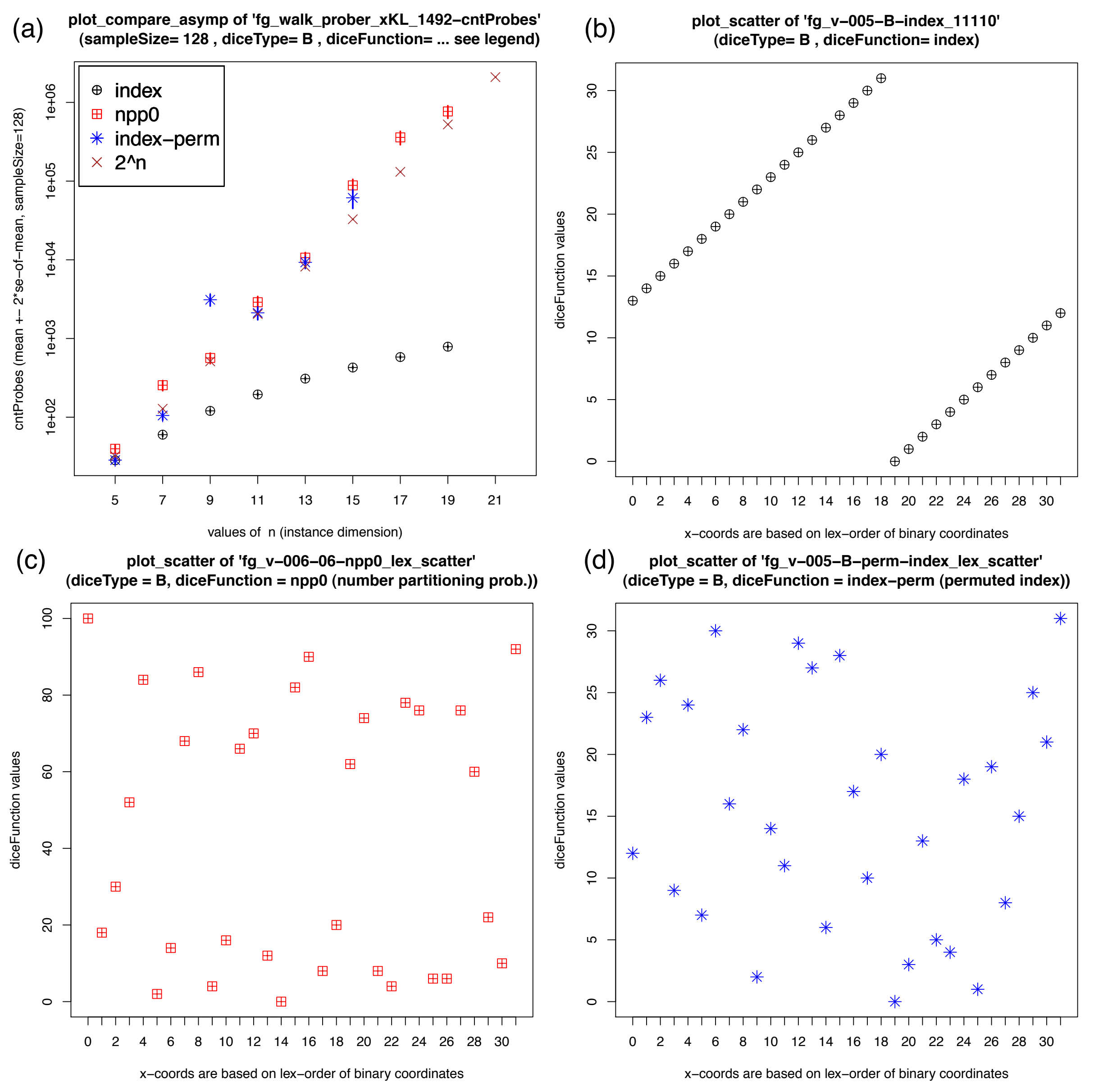


Figure 5. In (a) we measure *probe counts* with an identical xKL solver on instances of three dice functions and dimensions ranging from $n = 5$ to $n = 19$. Functions *index* and *index-perm* are related, their distributions of 2^n function values are the same. However, there are significant differences: *index* is piewise-linear as described in Figure 1, function values of *index-perm* are created by randomly permuting values of *index* under a lexographical coordinate order for both. Scatter plots differences between these two functions under lexographical order are significant as shown in (b) and (d) – and, it should also be noted that a scatter plot of *index* under *gray order* would look comparable to (d). Now, a scatter plot of function *npp0* in (c) looks random under both lexographical and gray order. Results of asymptotic experiments in (a) demonstrate that problem instances with the same number of optima but different levels of correlation between function values and their coordinates make the problems with better correlation also easier to solve (for this solver).

Acknowledgments

My recent plunge into R has been softened and rewarded by a number of user-friendly resources on the web and occasionally, even by email: Brian Ripley, Tony Plate, Duncan Murdoch, David Hiebeler, I thank you for the help and the patience. My skills in Tcl would not match my needs without the generous advice from Hemang Lavana and Clif Flynt. For prompting a timely delivery of this manuscript, credits are due to Baldomir Zajc, Andrej Žemva, and Andrej Trost from University of Ljubljana, Slovenia.

Finally, Wikipedia, and its universal accessibility, provides up-to-date information on a number of topics related to this paper – and as such is well deserving of continued support from its readers.

REFERENCES

[1] Wikipedia. http://en.wikipedia.org/wiki/Geometric_distribution, Nov 2011.
[2] Wikipedia. http://en.wikipedia.org/wiki/Polyhedron, Nov 2011.
[3] D. I. McCooey. Java Applets for Visualizing Polyhedra. http://homepage.mac.com/dmccooey/polyhedra/, Nov 2011.
[4] Wikipedia. Reproducible Research, Nov 2011.
[5] Patrick Vandewalle, Jelena Kovacevic, and Martin Vetterli. Reproducible Research in Signal Processing. *IEEE SIGNAL PROCESSING MAGAZINE*, 26(3):37–47, May 2009.
[6] C. Geyer. A Sweave Demo: Literate Programming in R. http://www.stat.umn.edu/˜charlie/Sweave/, Nov 2011.
[7] Literate programming. http://www.literateprogramming.com/, Nov 2011.
[8] D. E. Knuth. Literate Programming. *The Computer Journal*, 27(2):97–111, 1984.
[9] Tcl Developer Exchange. http://www.tcl.tk/, Nov 2011.
[10] J. K. Ousterhout and K. Jones. *Tcl and the Tk Toolkit (2nd Edition)*. Addison Wesley, 2009.
[11] Clif Flynt. *Tcl/Tk, Second Edition: A Developer's Guide*. Morgan Kaufmann Publishers, 2003.
[12] The R Project for Statistical Computing. http://www.r-project.org/, Nov 2011.
[13] P. Dalgaard. *Introductory Statistics with R*. Springer Verlag, 2008.
[14] Alastair Sanderson. Web pages on using the R statistical environment package. http://www.sr.bham.ac.uk/ ajrs/, Nov 2011.
[15] LaTeX – A document preparation system. http://www.latex-project.org/, Nov 2011.
[16] F. Brglez and J. A. Osborne. Performance Testing of Combinatorial Solvers With Isomorph Class Instances. In *ACM-FCRC*, 2007. http://doi.acm.org/10.1145/1281700.1281713.
[17] M. Stallmann and F. Brglez. High-Contrast Algorithm Behavior: Observation, Conjecture, and Experimental Design. In *ACM-FCRC*, 2007. http://doi.acm.org/10.1145/1281700.1281712.
[18] Wikipedia. Hasse Diagram, Nov 2011.
[19] Wikipedia. Hypercube Graph, Nov 2011.
[20] N. J. A. Sloane and M. Bernstein. Pascal's triangle read by rows. http://oeis.org/A007318, Nov 2011.
[21] N. J. A. Sloane. Triangle of multinomial coefficients. http://oeis.org/A036040, Nov 2011.
[22] N. J. A. Sloane. Triangle of Mahonian numbers T(n,k). http://oeis.org/A008302, Nov 2011.
[23] M. C. Er. On Generating the N-ary Reflected Gray Codes. *IEEE Trans. Computers*, C-33(8):739–741, August 1984.
[24] The Art of English Bell-ringing. http://www.chaddesley-corbett.co.uk/tower_history.htm, Nov 2011.
[25] Hale F. Trotter and Selmer M. Johnson. Generation of Permutations by Adjacent Transposition. *Mathematics of Computation*, 17(83):282–285, July 1963.
[26] D. A. Wismer. Solution of the Flowshop-Scheduling Problem with No Intermediate Queues. *OpRes*, 20(3):689–697, 1972.
[27] F. Brglez, X. Y. Li, and M. F. M. Stallmann. On SAT instance classes and a method for reliable performance experiments with SAT solvers. *Ann. Math. Art. Intell.,*, 43(1):1–34, 2005.
[28] S. Mertens. A physicist's approach to number partitioning. *Theoretical Computer Science*, 265(1-2):79–108, 2001.
[29] Brian Hayes. The Easiest Hard Problem. *American Scientist*, 90(2):113–117, 2002.
[30] Stephan Mertens. The Easiest Hard Problem: Number Partitioning. In A.G. Percus, G. Istrate, and C. Moore, editors, *Computational Complexity and Statistical Physics*, pages 125–139, New York, 2006. Oxford University Press.
[31] Stefan Boettcher and Stephan Mertens. Analysis of the Karmarkar-Karp differencing algorithm. *European Physics Journal B*, pages 131–140, 2008.
[32] Stephan Mertens. Random Costs in Combinatorial Optimization. *Phys. Rev. Lett.*, 84(6):1347–1350, February 2000.
[33] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21:1087–1092, June 1953.
[34] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.
[35] M. P. Vecchi S. Kirkpatrick, D. Gelatt Jr. Optimization by simulated annealing. *Science*, 220(5-6):671–680, 1983.
[36] Wikipedia. Simulated Annealing, Nov 2011.
[37] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, November 1984.
[38] Fred Glover. Tabu Search – Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.
[39] Fred Glover. Tabu Search – Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.
[40] Wikipedia. Ant Colony Optimization Algorithms, Nov 2011.
[41] Wikipedia. Particle Swarm Optimization, Nov 2011.
[42] Wikipedia. Great Deluge Algorithm, Nov 2011.
[43] Wikipedia. Evolutionary Computation, Nov 2011.
[44] B.W.Kernighan and S.Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Tech. J.*, pages 291–307, 1970.
[45] Wikipedia. Kernighan–Lin algorithm, Nov 2011.
[46] W.-K. Chen, M. Stallmann, and E.F. Gehringer. Hypercube embedding heuristics: An evaluation. *International Journal on Parallel Programming*, 18(6):505 – 549, 1989.
[47] F. Brglez, X. Y. Li, M. Stallmann, and B. Militzer. Reliable Cost Predictions for Finding Optimal Solutions to LABS Problem: Evolutionary and Alternative Algorithms. In *Proc. of The Fifth Int. Workshop on Frontiers in Evolutionary Algorithms (FEA2003), Cary, NC, USA*, September 2003. http://militzer.berkeley.edu/papers/2003-FEA-Brglez-posted.pdf.
[48] B. Militzer, M. Zamparelli, and D. Beule. Evolutionary search for low autocorrelated binary sequences. *IEEE Transactions on Evolutionary Computation*, 2(1):34–39, April 1998.
[49] S. Mertens. Exhaustive search for low-autocorrelation binary sequences. *Journal of Physics A: Mathematical and General*, 29:473–481, 1996.
[50] Golomb Rulers Project. http://www.distributed.net/OGR, Nov 2011.
[51] H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel Comput.*, 17:619–632, September 1991.
[52] Wikipedia. Rastrigin function, Nov 2011.

Franc Brglez received a Dipl-Ing degree in Electrical Engineering from the University of Ljubljana in 1965, served a year in the Yugoslav Army, and then continued with graduate studies in Electrical Engineering at the University of Colorado in Boulder, where he received a PhD degree in 1970. He spent the next 16 years with Bell-Northern Research (BNR) in Ottawa, Canada, where he was engaged in computer-aided design and research of communication circuits. In 1986, he moved to Microelectronics Center of North Carolina (MCNC) in Research Triangle Park (RTP) of North Carolina, first as BNR research scientist, where he initiated a series of trend-setting yearly workshops in Layout Synthesis and Logic Synthesis. Since 1995, he has been a Research Professor with the Computer Science Department of North Carolina State University (NCSU) in Raleigh. For some time, his primary activities have been in experimental design of test pattern generators, combinatorial optimization, and performance evaluation of combinatorial solvers, in particular with instances from the area of CAD of electronic circuits, starting with the ISCAS'85 and ISCAS'89 benchmarks. This paper is an attempt at closure for a subset of these activities; more musings are in progress.