Efficiency Competition through Representation Changes: Pigeonhole Principle vs. Integer Programming Methods

Yury V. Smirnov
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213-3891, USA
smir@cs.cmu.edu

Manuela M. Veloso
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213-3891, USA
mmv@cs.cmu.edu

Abstract

The Pigeonhole Principle (PHP) has been one of the most appealing methods of solving combinatorial optimization problems. Variations of the Pigeonhole Principle, sometimes called the "Hidden" Pigeonhole Principle (HPHP), are even more powerful and often produce the most elegant solutions to nontrivial problems. However, some Operations Research approaches, such as the Linear Programming Relaxation (LPR), are strong competitors to PHP and HPHP. They can also be applied to combinatorial optimization problems to derive upper bounds. It has been an open question whether PHP or LPR establish tighter upper bounds and how efficiently, when applied to the same problem. Challenged by this open question, we identify that the main reason for the lack of ability to compare the efficiency of PHP and LPR is the fact that different problem representations are required by the two methods. We introduce a problem representation change into an Integer Programming form which allows for an alternative way of solving combinatorial problems. We also introduce several combinatorial optimization problems, and show how to perform representation changes to convert the original problems into the Integer Programming form. Using the new problem model, we re-define the Pigeonhole Principle as a method of solving Integer Programming problems, determine the difference between PHP and HPHP, prove that PHP has the same bounding power as LPR, and demonstrate that HPHP and Integer cuts are actually similar representation changes of the problem domains.

1 INTRODUCTION

The Pigeonhole Principle (PHP) [Thucker, 1980] is traditionally considered the simplest and most elegant method of deriving tight upper bounds for a class of combinatorial optimization problems. In its original formulation, the Pigeonhole Principle (also known as the Dirichlet drawer principle) states that "it is impossible to place $N + 1$ pigeons in N holes so that there is at most one pigeon in each hole." In this simple form, PHP looks like a naïve kindergarten-level rule. However, if taken to a higher level of different number of objects (pigeons) allowed in abstract units (holes), or used as a part of a multi-step logical proof, it quickly loses the nuance of obviousness. Though PHP has no clear-cut mathematical origin, its applications involve some of the most profound and difficult results in combinatorics of great relevance to Artificial Intelligence.

To solve a combinatorial optimization problem by PHP, one needs to represent a problem in such a way that the principle may be applied, i.e. to perform what we call the representation change. For example, to identify what in the problem should be mapped to objects (pigeons) and units (holes). For simple problems, the applications of PHP are almost straightforward and are obtained directly from the nature of the problems. However, often the proof by PHP requires additional heuristic knowledge that allows the effective representation change. If this representation change is found, then PHP easily produces the tight upper bound and an optimal solution. For some combinatorial optimization problems it is a very challenging task to find such a representation change. Hardness of this task encourages to look for alternative methods.

The introduction of polynomial-time algorithms solving Linear Programming problems allows some Operations Research methods to be applied also to combinatorial optimization problems to establish upper bounds for such problems. For example, the Linear Programming Relaxation (LPR) method in conjunction with Integer cuts has been applied effectively to solve some Integer Programming (IP) problems. LPR can be seen as requiring less effort to apply than PHP, because in general, unlike PHP, it does not need any additional knowledge or representation changes to provide upper bounds for IP problems.

The two approaches, namely PHP and LPR-based methods can be seen as "competitors" for solving combinatorial

problems. It has been an open question[1] whether which method, if any, can provide better upper bounds and which method, if any, is more efficient, when applied to the same combinatorial optimization problems. This paper reports on our work in solving this open question. As hinted so far, our work analyzes carefully the problem representation issues involved in the two approaches. We formally introduce an appropriate representation change that makes the comparison and analysis possible.

There has been attempts in comparing the two mentioned approaches, see [Ginsberg, 1996]. We chose an Empirical way of re-defining the Pigeonhole Principle, because it allowed us to apply it to a series of problems of a great importance for Artificial Intelligence. We also found an alternative meaning of representation changes that in conjunction with the original PHP constitute the "Hidden" Pigeonhole Principle.

For some problems, like the "Mutilated" Checkerboard problem [Newell, 1965], immediate applications of the Pigeonhole Principle fail to provide the tight upper bound. Additional knowledge (heuristics) or representation changes are needed to accomplish a successful application of PHP that would establish the best feasible value for a given problem. Had this knowledge been provided in advance or, equivalently, had the problem been stated in a more suitable (for PHP) form, such an application would be easy and almost straightforward. However, the process of obtaining relevant knowledge or finding appropriate representation changes is problem-dependent, and often is a challenging task itself. Successful applications that combine acquiring relevant knowledge or performing representation changes and applications of the Pigeonhole Principle are sometimes called the "Hidden" Pigeonhole Principle (HPHP), though the difference between PHP and HPHP has never been clearly defined. Problems solved by HPHP are usually very non-trivial, and HPHP is often the most elegant and simple way of solving such problems. In Section 4 we discuss the "Mutilated" Checkerboard and Firm Tiling problems in detail and demonstrate the similarity between the Hidden Pigeonhole Principle and the combination of LPR with Integer cuts.

Our work includes:

1. Re-defining PHP as a method of solving Integer Programming problems.
2. Proving that both PHP and LPR obtain the same upper bounds.
3. Providing an alternative approach to solving combinatorial optimization problems for which the effective representation change required by PHP is hard to find. The alternative approach that can be implemented in automated logical proof, AI planning, or scheduling systems, consists of the following steps:
   (a) Convert a combinatorial optimization problem into an Integer Programming form.
   (b) Apply LPR to obtain an upper bound B which, as we proved, is the same as the upper bound obtained by PHP.
   (c) Construct an optimal solution of value B.

In this work, we also draw a clear splitting line between the original and the "Hidden" Pigeonhole Principles, analyze the latter one and compare its bounding power with that of Integer cuts.

2 OUR REPRESENTATION CHANGE APPROACH

The Pigeonhole Principle is the most simpleminded ideas imaginable, yet its generalizations involve some of the most profound results in combinatorics[Thucker, 1980]. A more sophisticated fashion of applying PHP is sometimes called the "Hidden" Pigeonhole Principle. In such cases, an optimal solution is usually nontrivial and the application of the Hidden Pigeonhole Principle (HPHP) is probably the simplest way of proving the optimality of a known solution.

Linear Programming Relaxation (LPR) is one of the most popular Operations Research methods of establishing upper bounds for Integer Programming problems[Nemhauser and Wolsey, 1985]. Its main calculation engine is based on the discovery of polynomial methods of solving Linear Programming problems, such as the Ellipsoid algorithm[Khachian, 1979]. Actually, LPR is a two-step procedure consisting of relaxing integer requirements and solving the corresponding Linear Programming problem.

Simplisticly, LPR can be viewed as a black-box with a particular instance of an Integer Programming problem as an input and a calculated upper bound as an output. From this point of view, LPR looks preferable to PHP, because it does not need any heuristic knowledge in deriving upper bounds. LPR can be applied to any instance of an Integer Programming problem without additional representation changes. Thus, the main questions of the competitive analysis between LPR and PHP can be stated as the *knowledge representation problem*: Which form of presenting combinatorial optimization problems allows to establish tighter bounds?

We start the following sections with the discussion of Linear Programming techniques relevant to our subject, then we re-define the Pigeonhole Principle for arbitrary instances of Integer Programming problems, determine the difference between PHP and HPHP, compare computational power of PHP and LPR, overview similarities between HPHP, PHP with heuristics, and Integer cuts for Integer Programming problems. We illustrate the discussion by a series of combinatorial optimization problems of a gradually increasing complexity.

[1] Identified at the First International Workshop on AI and OR, Portland, OR, June, 1995.

2.1 Important Facts from Linear Programming

In this section we introduce basic definitions and facts from the theory of Linear and Integer Programming. Readers familiar with this subject may skip this section.

A particular instance of a Linear Programming (LP) problem consists of the goal function and the set of inequalities (constraints). Both the goal function and the constraints depend linearly on each variable. Throughout this paper we consider variables to be real-valued, and their number is finite, the number of inequalities is also finite. Thus, a typical LP problem is of the following type:

$$\text{goal function}: \quad max(\sum_{i=1}^{N} c_i x_i)$$

$$\text{constraint set } J: \quad \sum_{i=1}^{N} a_{ij} x_i \le b_j \quad j = 1, \ldots, M$$

$$x_i \in R \quad i = 1, \ldots, N$$

Usually coefficients a_{ij}, b_j and c_i are rational, it allows to limit our consideration to rational-valued variables $x_i \in Q \quad i = 1, \ldots, N$. To simplify further discussion, we introduce a particular LP problem which will help to illustrate the discussion:

$$\text{goal function}: \quad max(y) \tag{1}$$

$$\text{constraint set } J: \quad -x \le 0 \tag{2}$$

$$-y \le 0 \tag{3}$$

$$y - x \le 1 \tag{4}$$

$$x + y \le 3 \tag{5}$$

$$x + 2y \le 8 \tag{6}$$

$$x \in Q \tag{7}$$

$$y \in Q \tag{8}$$

Figure 1 shows the feasible region, optimal solution $x^* = (1, 2)$, and the goal vector corresponding to the goal function (1). Theory of Linear Programming states that for any optimal solution x^* there exists a subset of constraints, called active constraints, such that:

- Inequalities representing active constraints are actually equalities for x^*, or, equivalently, there is no slack for active constraints with respect to x^*.
- The number of active constraints can vary from 1 to M.
- Goal function is a positive combination of the left-hand sides of active constraints.
- It is always possible to represent the goal function as a positive combination of at most N active constraints.

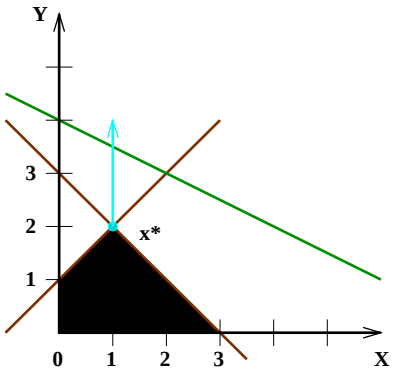


Figure 1: A Particular Instance of a LP Problem

In our example (1-8) inequalities (4) and (5) form the set of active constraints with respect to the optimal solution $x^* = (1, 2)$. Thus, the goal function (1) can be presented as a positive combination of (4) and (5): $y = \frac{1}{2}(y-x)+\frac{1}{2}(x+y)$.

If we vary inequality (6), it may also become an active constraint. For example, inequality $x + 2y \le 5$ is an active constraint with respect to the optimal solution $x^* = (1, 2)$. If added to the existing set of active constraints (4) and (5), it would constitute a redundancy: Left-hand sides of any two out of three inequalities can be used in a positive weighted sum to obtain the goal function. On the other hand, the goal function can also vary within a certain range to preserve the same optimal solution and the set of active constraints. In problem (1-8) for any goal function of the form $y + \alpha x$ with $\alpha \in [-1, 1]$, the set of active constraints consists of (4) and (5). For the extreme values of $\alpha \in \{-1, 1\}$ one of the active constraints becomes redundant, since the goal function coincides with the left-hand side of the other active constraint. Moreover, in both extreme cases there exists an infinite number of feasible solutions attaining the same optimal value. These optimal solutions form a face of a polyhedron of feasible solutions defined by the IP problem's constraints. For example, in problem (1-8) the set of feasible solutions forms a 2-D tetragon (see Figure 1), and the set of optimal solutions is either a zero-dimensional face (a single 2-D point) or a one-dimensional face (one of the tetragon's sides).

Thus, on one hand, a positive weighted sum of left-hand sides of active constraints establishes an upper bound: Since $\sum_j \sum_i \alpha_j a_{ij} x_i = \sum_i c_i x_i$, where $\alpha_j \ge 0$, and $\sum_i a_{ij} x_i \le b_j$, then $\sum_i c_i x_i \le \sum_j \alpha_j b_j$. On the other hand, any feasible solution $\bar{x}$ sets a lower bound for the goal function $\sum_i c_i \bar{x}_i$. Furthermore, active constraints have no slack with repeat to optimal solution x^*, therefore, the lower bound provided by x^* coincides with the upper bound established by the positive weighted sum of active constraints and both are equal to the optimal value $\sum_i c_i x_i^*$.

Integer Programming (IP) problems have an additional requirement that some of its variable are integer-valued. Throughout this paper we consider only those IP problems

that have linear goal functions and linear constraints. In general, the addition of integer-valued variables makes an IP problem NP-hard. However, in some particular cases, it is possible to solve IP problems efficiently, by applying Linear Programming Relaxation (LPR) or a combination of Integer Cuts and LPR[Nemhauser and Wolsey, 1985]. LPR is a simple procedure that consists of two steps:

1. Drop (relax) integrality requirements for all variables.
2. Solve the correspondent LP problem.

For some IP problems, like problem (1-8) introduced earlier, LPR outputs an integer feasible solution, thus, solving such IP problem. In other cases, when LPR outputs a fractional optimal solution x^*, it establishes an upper bound $\sum c_i x_i^*$ for the goal function. In such cases, LPR is usually not very helpful in finding integer solutions or indicating the tightness of the upper bound.

Integer cuts produce additional constraints that utilize the integral nature of a subset of variables. Additional constraints can be derived in many different ways, Integer cuts produce ones that cannot be obtained by taking positive weighted sums of the existing inequalities. In conjunction with LPR, Integer cuts are capable of solving IP problems, whereas the efficiency of this combination in obtaining tight upper bounds relies on the "quality" of newly created constraints constructed by Integer cuts techniques. Examples of the Hidden Pigeonhole Principle's applications, namely the Mutilated Checkerboard problem and the Firm Tiling problem discussed in Section 4, illustrate an alternative way of solving these problems through Chvatal-Gomory's (Integer) cuts and LPR.

2.2 Converting Problems into the Integer Programming Form

We identified that the main reason of the lack of ability to compare the efficiency of PHP and LPR is that different problem representations are required by the two methods. Our work consisted of developing appropriate changes of representations that made possible a competitive analysis of the efficiency of these two methods.

To make both PHP and LPR applicable to the same combinatorial optimization problems, we perform representation changes for each problem discussed in this paper to convert them into Integer Programming problems of the following form:

$$\text{goal function :} \quad max(\sum_{i=1}^{N} c_i x_i)$$

$$\text{constraint set } J : \quad \sum_{i=1}^{N} a_{ij} x_i \leq b_j \quad j = 1, \ldots, M$$

$$\text{integrality requirements :} \quad x_i \in Z \quad i = 1, \ldots, N$$

A combinatorial optimization problem is defined as finding a solution $x^* = (x_1^*, \ldots, x_N^*)$ that is feasible, i.e. satisfies all the constraints from J and the integrality requirements, and also attains the best value of the goal function among all feasible solutions.

We considered a series of known combinatorial optimization problems, to which the Pigeonhole Principle is applicable. The empirical evidence obtained through this study suggested that the following definition reflects correctly the combinatorial nature of the Pigeonhole Principle. We confirm the correctness of this definition through a set of problems that illustrate applications of PHP for the original combinatorial form and the Integer Programming reformulation.

Definition 1 *Suppose that an Integer Programming problem of the above type has a known feasible solution x^*. We say that the problem admits a proof by the Pigeonhole Principle if there exists a subset of constraints $\hat{J} \subseteq J$, such that the sum of the left-hand sides of the inequalities from $\hat{J}$ (repetitions are allowed in $\hat{J}$) is a multiple of the goal function*

$$\sum_{j \in \hat{J}} \sum_{i=1}^{N} a_{ij} x_i = k \sum_{i=1}^{N} c_i x_i$$

and the value of x^* equals the same multiple of the sum of the right-hand side constants

$$\sum_{j \in \hat{J}} b_j = k \sum_{i=1}^{N} c_i x_i^* \ .$$

If such a solution x^* is not known, then the best value

$$\min_{\hat{J} \subseteq J} \frac{1}{k} \sum_{j \in \hat{J}} b_j \ , \quad such \ \ that \ \sum_{j \in \hat{J}} \sum_{i=1}^{N} a_{ij} x_i = k \sum_{i=1}^{N} c_i x_i$$

holds for some positive $k > 0$, is called an upper bound obtained by PHP.

In general, the procedure of considering a subset of constraints and summing them up (possibly with positive weights) results in setting upper bounds for the value of an optimal solution. Any feasible solution provides a lower bound for the optimal value of the problem. We identify applications of PHP with finding a subset of constraints and obtaining the optimal value of the goal function. In this case, the established upper bound matches the lower bound provided by the found optimal solution.

Definition 2 *If a set of constraints added to an Integer Programming problem changes the original IP problem into an extended IP problem that admits a proof by the Pigeonhole Principle, we say that the original problem admits a proof by the Hidden Pigeonhole Principle (HPHP).*

Additional constraints can be obtained in many different ways. Weighted positive sums, for example, can simplify some of the existing constraints, but they would not contribute any restrictions on fractional optimal solutions obtained by Linear Programming Relaxation. In Section 4 we consider an Integer cuts technique that allows to derive tighter valid inequalities and to remain all integer solutions feasible. Constraints obtained from Branch-and-Bound methods can also be sought as an addition to the existing set of constraints. We consider the way of constructing additional constraints and the sanity check that an optimal integer solution has not been cut off to be the responsibility of the problem solver.

The introduced representation change allows to reformulate the original PHP statement with 11 pigeons and 10 holes as the following IP problem:

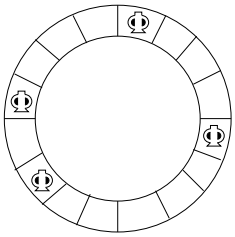
$$max(\sum_{i=1}^{11}\sum_{j=1}^{10} x_{ij})$$
$$\sum_{i=1}^{11} x_{ij} \leq 1 \quad j = 1, \ldots, 10$$
$$x_{ij} \in \{0, 1\},$$

where x_{ij} represents the amount of the ith pigeon in the jth hole. Thus, if we drop integer requirements and sum up all the constraints, we obtain an upper bound: $\sum_{i=1}^{11}\sum_{j=1}^{10} x_{ij} = \sum_{j=1}^{10}\sum_{i=1}^{11} x_{ij} \leq 10$. Since an obvious solution $\{x_{ii} = 1 \text{ for } i = 1, \ldots, 10;\ x_{ij} = 0 \text{ for } i \neq j\}$ provides the same value, we proved it to be optimal. For this simple problem, PHP and LRP approaches are identical.

In most obvious cases, the desired subset of constraints is the whole set of original inequalities. Consider, for example, the problem of placing chess kings on a circular chessboard with even number of squares (see Figure 2). One is supposed to find the maximal number of kings that can be placed on such a board so that no king is attacking another. Recall that a chess king attacks all its adjacent squares. For the board presented in Figure 2, we get the following IP problem:

If we sum up all the inequalities, we get $2\sum_{i=1}^{N} x_i \leq N$, which is equivalent to $\sum_{i=1}^{N} x_i \leq N/2$. The same upper bound can be obtained from applying a combination of two-coloring and PHP: if we color the circular board with even number of squares in alternating black and white colors, we can place kings on either color attaining the optimum value of $N/2$ derived this way by PHP.

The case when a proper subset of constraints is involved in obtaining a tight upper bound is more complicated. To apply the Pigeonhole Principle in the original combinatorial form, one has to find which problem's objects should be matched with pigeons and which - with holes. In the Integer Programming form it means that one has to come up with a rule (heuristic) of finding a desired subset of constraints to sum them up. For some problems, it is easy to find such a subset,



$$max(\sum_{i=1}^{N} x_i)$$
$$x_1 + x_2 \leq 1$$
$$x_2 + x_3 \leq 1$$
$$\ldots$$
$$x_{N-1} + x_N \leq 1$$
$$x_N + x_1 \leq 1$$
$$x_i \in \{0, 1\}$$

Figure 2: Chess kings on a circular chessboard with N positions and corresponding IP representation. Kings cannot be in adjacent positions, i.e. attack each other.

whereas for others it is not obvious. For example, the popular N-Queen problem[Hoffman et al., 1969, Nauck, 1850], which is the problem of placing the maximal number of chess queens on NxN-chessboard, so that no queen is attacking another, can be represented as the following IP problem:

$$max(\sum_{i=1}^{N}\sum_{j=1}^{N} x_{ij})$$
$$\text{for each } i \in \{1, \ldots, N\} \text{ and } j \in \{1, \ldots, N\} \begin{cases} \sum_{k=1}^{N} x_{ik} \leq 1 \\ \sum_{k=1}^{N} x_{kj} \leq 1 \\ \sum_{(k,l)\in I(i,j)} x_{kl} \leq 1 \\ \sum_{(k,l)\in J(i,j)} x_{kl} \leq 1 \\ x_{ij} \in \{0, 1\} \end{cases}$$

where $I(i,j)$ and $J(i,j)$ are the sets of squares which a chess queen threatens along two diagonals from the square (i,j), including the square (i,j). In this form we have four groups of constraints: row, column, and two diagonal constraints, one of each type per square. For this problem it is easy to find a subset of constraints that provides the tight upper bound. If we sum up N inequalities corresponding only to row constraints for squares from different rows, we get N as the upper bound: $\sum_{i=1}^{N}\sum_{j=1}^{N} x_{ij} \leq N$. Beginning with $N = 4$, the problem has an N-Queen solution[Hoffman et al., 1969] (see Figure 3).

Though PHP provides the tight upper bound, it is not always immediately clear how to construct an optimal solution that

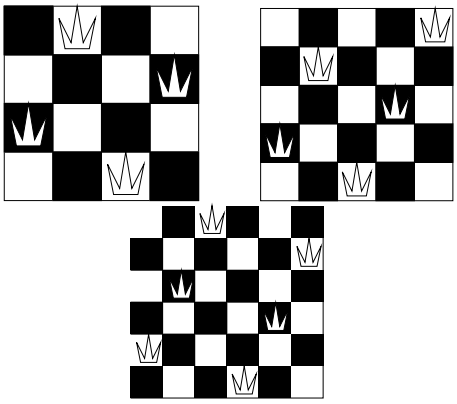


Figure 3: N-Queen Problem Solutions for N = 4, 5, 6

will attain the obtained bound. The *N*-Queen problem stimulated the development of a generation of backtracking algorithms constructing optimal solutions for the *N*-Queen problem, which is a hard problem itself. The Chess Knight problem[Bennett and Potts, 1967] is one of the jewels of PHP applications; it is the one for which the selection of a constraint subset is a challenging task. We discuss this problem in the following section in detail.

3 PHP vs. LPR

In this section, we discuss relations between the Pigeonhole Principle and the Linear Programming Relaxation. For some problems, it is tempting to apply LPR in a brute-force manner. Instead of deriving an additional heuristic that will suggest how to select the desired subset of constraints, one can apply already developed methods of Linear Programming with the hope of deriving the same or even better bound. This brings us back to the main question of this paper: Is it true that PHP and LPR always provide the same upper bound? The following theorem answers this question:

Theorem 1 *If an Integer Programming problem admits a proof by the Pigeonhole Principle, then LPR provides the same optimal value. Conversely, a bound derived by LPR can be matched by PHP.*

Proof: If an IP problem admits PHP in deriving the tight upper bound, then there exists a feasible integer solution x^* attaining the derived bound. On the other hand, there exists a subset of constraints $\hat{J}$, sum of the left-hand sides of which is an integer multiple of the goal function: $\sum_{j\in\hat{J}}\sum_i a_{ij}x_i = k\sum_i c_i x_i$, and the value of x^* is the same multiple of bounding constants: $\sum_{j\in\hat{J}} b_j = k\sum_i c_i x_i^*$.

If we apply LPR to this IP problem, it will provide an integer or fractional optimal solution x^{**} for a relaxed problem. Since PHP has derived the tight upper bound, and the relaxed problem is obtained from the original IP problem by dropping integrality requirements, x^* was one of the candidates for LPR's optimal solution, and the value of LPR's solution x^{**} is equal to PHP's tight upper bound: $\sum_{i=1} c_i x_i^{**} = \sum_{i=1} c_i x_i^*$.

If an optimal integer feasible solution x^* is not known, PHP establishes an upper bound. We show that this bound is equal to the value of the LPR's solution $\sum_{i=1} c_i x_i^{**}$. According to Linear Programming theory, there exists a subset of the constraint set, called "active" constraints, such that the goal function is a positive weighted sum of the left-hand sides of the constraints from this subset $\sum_{j\in\hat{J}}\sum_i \alpha_j a_{ij}x_i = \sum_i c_i x_i$. Moreover, optimal solution x^{**} has no slack for each of the constraints from $\hat{J}$, that is, satisfies them as equality $\sum_i a_{ij}x_i^{**} = b_j$ for $j \in \hat{J}$. Since all the coefficients in the constraints J and the goal function are integer, all the weights $\alpha_j \geq 0$ (positive coefficients) of the weighted sum are rational. We can find integer k to scale rational coefficients up $\beta_j = k\alpha_j$ and make all of β_j integer. Integer k now plays the role of a scaling coefficient, integer β_j tells how many times should the "active" constraint $j \in \hat{J}$ be used in an "unweighted" sum of left-hand sides $\sum_{j\in\hat{J}}\beta_j\sum_i a_{ij}x_i = k\sum_i c_i x_i$. Hence, the upper bound $1/k\sum_{j\in\hat{J}}\beta_j b_j = \sum_i c_i x^{**}$ can be matched by PHP.

Thus, the set of "active" constraints forms the desired subset $\hat{J}$ and positive integer weights determine the number of repetitions in $\hat{J}$. Therefore, the value of the solution derived by LPR does not improve the upper bound provided by PHP and, conversely, it can be mimicked by PHP to establish the same upper bound. ■

Theorem 1 provides the following answer to the main question of the paper: *If the optimal value is obtained by PHP for a combinatorial optimization problem stated in the Integer Programming form, LPR provides the same bound as PHP*. We demonstrate that this result is nontrivial by solving the Chess Knight problem[Bennett and Potts, 1967]: "What is the maximal number of chess knights that can be placed on the 8x8 chessboard in such a way that they do not attack each other?"

The classical elegant solution relies on the existence of a Hamiltonian tour of length 64 following the knight moves. One of such tours is presented in Figure 4.

One can split the tour into 32 pairs of chess squares that are adjacent in the sense of the knight move, and apply PHP: Each pair can contain at most one knight, otherwise two knights would attack each other. For example, consider pairs of squares $(1,2)$, $(3,4)$,...,$(63,64)$. None of them can accommodate more than one knight (see Figures 4 and 5). Thus, 32 pairs of adjacent squares can accommodate at most 32 knights. Although this simple proof does not provide us with a solution to the problem, (for example, it allows to place knights on squares 4 and 5), it gives an upper bound.

Moreover, this proof gives an impression of using a "hidden" application of the Pigeonhole Principle, whereas the Hamiltonian tour is just a heuristic for finding a subset of the constraint set. The Chess Knight problem for the stan-

8	43	24	53	6	41	22	51
25	54	7	42	23	52	5	40
44	9	62	15	46	31	50	21
55	26	45	32	63	14	39	4
10	33	16	61	30	47	20	49
27	56	29	64	13	60	3	38
34	11	58	17	36	1	48	19
57	28	35	12	59	18	37	2

Figure 4: Hamiltonian Tour on a Chessboard for the Knight

dard chessboard can be presented as the following Integer Programming problem:

$$max(\sum_{i=1}^{8}\sum_{j=1}^{8} x_{ij})$$

$$x_{ij} + x_{kl} \leq 1 \quad i = 1, \ldots, 8 \;\; j = 1, \ldots, 8 \;\; (k, l) \in U(i, j)$$

$$x_{ij} \in \{0, 1\}$$

where x_{ij} represents the amount of knights in the square (i, j); $U(i, j)$ is the set of squares on the chessboard which a chess knight threatens from the cell (i, j) (see Figure 5).

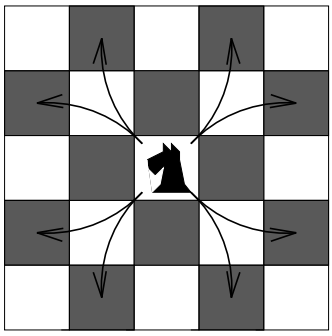


Figure 5: The Set of Adjacent Squares for the Knight on a Chessboard

If applied to the Chess Knight problem, the Linear Programming Relaxation method provides the same upper bound of 32. Unlike PHP, LPR does not require heuristics to identify any subset of constrains, its calculational routine considers "active" constraints inside the solving process. However, LPR is likely to produce fractional solutions, say $x_i = 1/2$ for $i = 1, 2, \ldots, 64$. Theorem 1 shows that, if applied, the original Pigeonhole Principle will provide the same value. So, if the Hamiltonian tour heuristic were not known, the application of LPR would tell that 32 is the best upper bound that can be obtained by the original PHP. Since a chess knight alternate colors (see Figure 5), one can place 32 knights on the chess squares of the same color, thus attaining the optimal value and constructing the optimal solution for the Chess Knight problem.

We showed that, if an optimal solution of value B for a combinatorial optimization problem is derived by the Pigeonhole Principle, an optimal solution for the last problem in the following chain of representation changes has the same value B:

> Original combinatorial problem→ IP problem → Linear Programming problem.

Examples discussed in this section showed that an additional effort is needed to apply the Pigeonhole Principle. For some problems it is easy to match problem's objects with pigeons and holes, in some cases it is a state of art. Often PHP applications hint on how to construct optimal solutions, though for some problems it is an independent difficult problem.

In its turn, LPR can be applied to any instance of an IP problem without need in additional knowledge. Unfortunately, LPR often outputs a fractional optimal solution, which does not shed any light on how to transform it into an integer one of the same value. We continue the discussion on benefits and disadvantages of PHP and LPR in the next section.

4 APPLICATIONS OF PHP THAT REQUIRE REPRESENTATION CHANGES

In the previous sections, we were able to apply PHP in a brute-force manner, because each Integer Programming problem contained a subset of constraints, sum of which provided the desired bounds for values of the goal functions. We call such a case a regular application of PHP, as opposed to the "Hidden" Pigeonhole Principle (HPHP) that requires additional representation changes and heuristic knowledge to fulfill a similar task.

This section is devoted to the discussion on HPHP and its relation to the original Pigeonhole Principle. To make it more intuitive, we illustrate the discussion by the classical Mutilated Checkerboard [Newell, 1965] and the Firm Tiling problems:

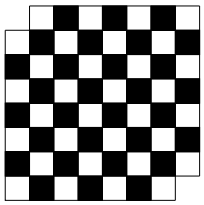


Figure 6: A Mutilated Checkerboard

Mutilated Checkerboard Problem: Consider an NxN checkerboard with two opposite corners removed (see Figure 6). Can one cover this "mutilated" checkerboard completely by non-overlapping domino pieces, each of the size of two squares of the checkerboard?

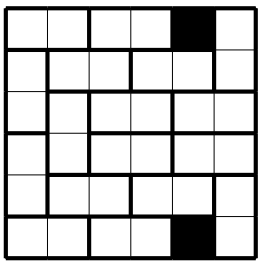


Figure 7: Firm 17-Tile Solution.

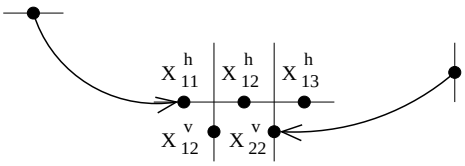


Figure 8: Modeling Domino Overlap as a Set of Inequalities

Firm Tiling Problem: Consider a checkerboard of size 6x6 made of a soft square cloth and 18 hard tiles of size 1x2. Can one glue all 18 tiles to such a checkerboard, so that the "middle-cut" requirement is satisfied, i.e. each splitting line inside the checkerboard goes through the middle line of at least one tile?

Figure 7 shows a firm 17-tile solution. The "middle-cut" requirement for a particular splitting line restricts the cloth to be folded along this splitting line: If the line crosses the middle of at least one tile, the cloth cannot be folded along this line unless the tile is broken. This is what we call a *"single-firm"* tiling. A more restrictive *"double-firm"* tiling requires each splitting line to cross middle lines of at least two tiles. In this section we show that a *"single-firm"* complete tiling implies a *"double-firm"* tiling in the Firm Tiling problem.

Following the proposed approach, the first step needed is the representation change converting the above problems into Integer Programming problems. To accomplish this step, one needs to model the fact that domino pieces do not overlap in the Integer Programming form. We assign variables x^v_{ij} to vertical splits, where i is the row number and j is the split in ith row between squares (i, j) and $(i, j + 1)$. In the same way, we define variables x^h_{ij} for horizontal splits between squares (i, j) and $(i + 1, j)$. One-valued variables $x^v_{ij} = 1$ model the horizontal placement of a domino piece (tile) in such a way so that its middle line is located at ith vertical splitting line and jth row; one-valued variables $x^h_{kl} = 1$ model the vertical placement of a domino piece (tile) with its middle line at kth horizontal split in lth column. In these terms, non-overlapping can be represented by the following set of constraints:

$$x^v_{ij} + x^v_{i-1j} \le 1 \quad x^v_{ij} + x^v_{i+1j} \le 1 \quad x^v_{ij} + x^h_{i-1j-1} \le 1 \qquad (9)$$

$$x^v_{ij} + x^h_{i-1j} \le 1 \quad x^v_{ij} + x^h_{ij-1} \le 1 \quad x^v_{ij} + x^h_{ij} \le 1 \qquad (10)$$

Figure 8 demonstrates a part of a checkerboard and the correspondence of domino pieces (tiles) placings to 0/1-variables and splits.

In the Mutilated Checkerboard problem the goal function is the unweighted sum of all domino-piece variables: $\sum_{ij} x^v_{ij} + \sum_{ij} x^h_{ij}$. It is easy to guess one of the fractional optimal solutions produced by LPR: $x^*_{**} = 1/2$. It tells that if applied, PHP will provide the same bound of 31 (which is not tight). Furthermore, an optimal fractional solution does not help to find an optimal integer one, even when the upper bound is tight.

If the original checkerboard is of size NxN with N odd, a simple PHP application shows that one can use at most $\frac{N^2-3}{2}$ domino pieces for a non-overlapping covering of the mutilated checkerboard: Each piece contains two squares, and there are $N^2 - 2$ squares in the mutilated checkerboard. Therefore, one can put at most $\lfloor \frac{N^2-2}{2} \rfloor = \frac{N^2-3}{2}$ non-overlapping domino pieces. Actual attempts to cover the mutilated checkerboard with odd sizes readily give an optimal solution of the above value.

If N is even, the original PHP does not put any additional restrictions on the number of pieces. However, none of the attempts to cover the mutilated checkerboard by domino pieces achieves the desired bound of $\frac{N^2-2}{2}$, all constructed solutions provide at most $\frac{N^2-4}{2}$ pieces. Neither PHP nor LRP provide a tight upper bound for even-sized mutilated checkerboards. Nonetheless, the heuristic of two-coloring the mutilated checkerboard and applying PHP to the monochrome set of squares of smaller size completes the proof of the fact that $\frac{N^2-4}{2}$ is actually the optimal value for covering the mutilated checkerboard with even sizes. If we color the original checkerboard in usual black-and-white chess colors (see Figure 6) and then cut off 2 opposite corner squares (of the same color), the remaining mutilated checkerboard contains unevenly colored square sets. The checkerboard presented in Figure 6, for example, has 30 black squares and 32 white squares. Since each domino piece contains 2 squares of the opposite colors, applying PHP to a smaller set of black or white squares, we obtain the tight upper bound of $\frac{N^2-4}{2}$. This argument completes the proof of the Mutilated Checkerboard problem by HPHP. Such a modification of PHP is favorable in comparison with LPR and the original PHP, because it is capable of deriving the optimal value of the goal function. As we mentioned before, Linear Programming Relaxation provides $\frac{N^2-2}{2}$ as an upper bound. According to Theorem 1, if applied, PHP outputs exactly the same upper bound.

The idea of two-coloring is a simple elegant heuristic that allows to apply the Hidden Pigeonhole Principle and derive the tight upper bound. After being known for several dozen years, this application of HPHP might seem to be too

simple to stimulate the development of alternative methods. We introduce the Firm Tiling problem as an example of a problem which is hard to solve without prior knowledge of the appropriate heuristic. For example, multi-coloring does not help to find the desirable matching between tiles, checkerboard squares, pigeons and holes.

Theory of Integer Programming suggests to apply Integer cuts, for example, Chvatal-Gomory's cuts, to extend the set of valid inequalities. The main idea of the Chvatal-Gomory's cut (CG-cut) is to use the integrality of variables in feasible solutions. If a weighted sum of the inequalities derived so far contains the left-hand side with integer coefficients, the right-hand side can be rounded down to the nearest integer: $\sum_{i=1}^{N} a_i x_i \leq b$ implies $\sum_{i=1}^{N} a_i x_i \leq \lfloor b \rfloor$, for integer a_i and $x_i \quad i = 1, \ldots, N$. In a certain sense CG-cuts look as simple as the original Pigeonhole Principle. However, CG-cuts allow to derive constraints that cannot be obtained from the initial set of inequalities by taking weighted positive sums.

We, first, demonstrate the correctness of the Integer Programming formulation of the Mutilated Checkerboard problem and then derive an upper bound for it by means of Chvatal-Gomory's cuts and the Hidden Pigeonhole Principle.

Since we are about to apply some of the techniques of Integer cuts to the Mutilated Checkerboard problem, we first demonstrate simple reductions from Integer Programming theory. For example, Lemma 1 builds a reduction from the set of pairwise constraints with 0/1-vertex variables to the Exclusive Rule for a clique K_N, where a clique is a complete graph with $N \geq 2$ vertices.

Lemma 1 *If a clique K_N admits exclusively 0/1-assignments to its vertices $v_1, \ldots, v_N$ and, for each pair of vertices (v_i, v_j), at most one vertex can be assigned to 1, then there is at most one vertex assigned to 1 in the whole clique K_N.*

Proof: There are $\frac{N(N-1)}{2}$ inequalities of the type $x_i + x_j \leq 1$. We would like to prove that this collection of constraints implies a single clique inequality $\sum_{i=1}^{N} x_i \leq 1$ (Exclusive Rule) for 0/1-variables $x_i \in \{0, 1\} \quad i = 1, \ldots, N$. We prove it by induction on the number of vertices.
Induction Base: If $N = 2$, the given inequality and the clique inequality $x_1 + x_2 \leq 1$ coincide.
Induction Step: Suppose that we can derive all N clique inequalities for each of the N sub-cliques of size $N - 1$. If we sum them up, we get:

$$\sum_{j=1}^{N} \sum_{i \neq j} x_i \leq \sum_{j=1}^{N} 1,$$

which implies that $(N-1)\sum x_i \leq N$ or, equivalently, $\sum x_i \leq N/(N-1)$. Since $N > 2$, $\lfloor \frac{N}{N-1} \rfloor = \lfloor 1 + \frac{1}{N-1} \rfloor = 1$. If we apply CG-cut to the last inequality, we get the desired N-clique inequality:

$$\sum_{i=1}^{N} x_i \leq 1. \quad \blacksquare \qquad (11)$$

Four (or less) splits that form the border of the square (i, j) correspond to four variables that model placing of domino pieces (tiles). Corner or side squares border with only two or three internal splits. According to Lemma 1, if we consider all six (or less) pairwise inequalities (9-10) involving four variables bordering a checkerboard square, CG-cuts provide the Exclusive Rule for the clique associated with the square. This Exclusive Rule corresponds to the requirement of non-overlapping of domino pieces over the square (i, j). The addition of the clique inequalities derived by CG-cuts to the initial set of constraints constitutes a representation change of the IP problem.

Lemma 2 *If clique inequalities for all squares of the NxN mutilated checkerboard (N is even) are added to the set of constraints, the optimal value of the relaxed Linear Programming problem is $\frac{N^2-4}{2}$.*

Proof: Consider the following subset of clique constraints: Pick a monochrome subset of squares of smaller size and sum up all the clique constraints corresponding to these squares (of the same color). Each clique constraint is an unweighted sum of four (or less) clique variables $x_{i_1} + x_{i_2} + x_{i_3} + x_{i_4} \leq 1$. Since squares of the same color do not share sides, the sum of all clique constraints corresponding to squares of the same color is an unweighted sum of variables. On the other hand, all variables are presented in the final sum, because a legitimate placement of a domino piece covers squares of both colors. Since the number of clique constraints corresponds to the number of monochrome squares of smaller size, the sum of the constraints establishes a tighter bound for the goal function $\sum x_i \leq \frac{N^2-4}{2}$. Knowing this bound, it is easy to come up with a solution for an even-sized Mutilated Checkerboard problem that attains this value. $\blacksquare$

Thus, the proof that uses HPHP, relies on additional knowledge that each domino piece covers a bi-chromatic configuration, whereas the combination of LPR with CG-cuts takes into account this argument automatically. CG-cuts expand the set of constraints by adding clique inequalities for all checkerboard squares. After that LPR solves the new Integer Programming problem. We identify the expansion of the constraint set with the representation changes for the original Integer Programming problem. If the two-coloring heuristic were not known, then LPR with CG-cuts could be used to obtain the tight upper bound of $\frac{N^2-4}{2}$. After that, it is relatively easy to construct a solution attaining this value, which is proven to be optimal. The relations between HPHP and LPR with Integer cuts in solving the Mutilated Checkerboard problem are very similar to those between PHP and LPR.

The Firm Tiling problem does not admit the proof by the original PHP, because an obvious fractional solution $x^*_{**} = 1/2$ is feasible for a relaxed LP problem and the "middle-cut" requirement

$$\sum_{j=1}^{6} x^h_{ij} = \sum_{j=1}^{6} \frac{1}{2} = 3 \geq 1 \quad i = 1, 2, \ldots, 5$$

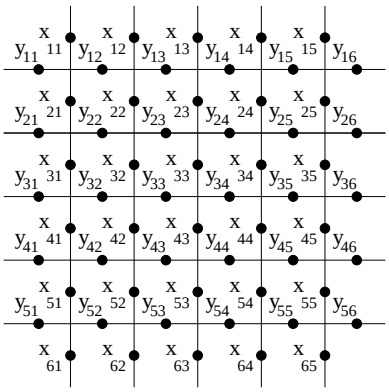


Figure 9: Modeling a Firm Tiling of a 6x6 Checkerboard.

$$\sum_{i=1}^{6} x_{ij}^{v} = \sum_{i=1}^{6} \tfrac{1}{2} = 3 \geq 1 \quad j = 1, 2, \ldots, 5$$

is satisfied for all internal splitting lines. However, brute-force attempts to construct a complete firm tiling fail to provide an optimal 18-tile solution. The best firm tiling consist of at most 17 non-overlapping tiles, Figure 7 shows one of such tilings.

Nonetheless, the Hidden Pigeonhole Principle is capable of setting the tight upper bound of 17 tiles for this problem. To preserve the beauty of the elegant solution by HPHP for now, we first establish the tight upper bound through the combination of LPR and CG-cuts. We show that if one more constraint

$$\sum x_{ij}^{h} + \sum x_{ij}^{v} > 17 \tag{12}$$

is added to the original IP problem, the set of feasible integer solutions becomes empty.

Figure 9 presents 6x6 checkerboard with 0/1-variables assigned to its internal splits. To avoid superscript notations we denote x_{ij}^{v} as x_{ij} and x_{ij}^{h} as y_{ij}. Since x_{ij} and y_{ij} are integer, inequality (12) implies

$$\sum x_{ij} + \sum y_{ij} \geq 18. \tag{13}$$

In its turn, the latter inequality implies that the whole checkerboard should be covered by tiles. Since clique inequalities prohibit tiles from overlapping, each square is covered exactly by half-a-tile[2]. This is not a surprising conclusion, as we are attempting to cover a 6x6 checkerboard by 18 non-overlapping tiles.

Lemma 3 *Inequality (13) and clique inequalities (11) for all squares of the 6x6 checkerboard imply* "double-firm" *tiling.*

Proof: We prove the statement of the Lemma by induction. In the induction base, we show it for the leftmost vertical splitting line. Due to the symmetry this proof remains unchanged for horizontal splitting lines.

Induction Base: If we consider the leftmost column of a checkerboard presented in Figure 9, inequalities (9,10) and (13) imply the following six equalities:

$$x_{11} + y_{11} = 1 \quad x_{21} + y_{11} + y_{21} = 1 \quad x_{31} + y_{21} + y_{31} = 1 \tag{14}$$

$$x_{61} + y_{51} = 1 \quad x_{41} + y_{31} + y_{41} = 1 \quad x_{51} + y_{41} + y_{51} = 1 \tag{15}$$

The "middle-cut" requirement for the leftmost vertical splitting line corresponds to the following inequality:

$$\sum_{i=1}^{6} x_{i1} \geq 1 \tag{16}$$

Sum of equalities (14) and (15) produces a new constraint:

$$\sum_{i=1}^{6} x_{i1} + 2\sum_{j=1}^{5} y_{j1} = 6 \tag{17}$$

Now we can subtract (16) from (17) and divide it by two:

$$2\sum_{j=1}^{5} y_{j1} \leq 5 \quad \textit{implies} \quad \sum_{j=1}^{5} y_{j1} \leq 2.5 \tag{18}$$

Since all y_{j1} in the left-hand side of inequality (18) are integer variables, we can apply CG-cut to obtain a new (tighter) inequality:

$$\sum_{j=1}^{5} y_{j1} \leq 2 \tag{19}$$

In its turn, constraints (17) and (19) imply

$$\sum_{i=1}^{6} x_{i1} \geq 2 \tag{20}$$

Induction Step: Suppose that we have proved that inequalities (11) and (13) imply the *"double-firm"* tiling for $j_0 - 1$ leftmost columns of the checkerboard ($1 \leq j_0 - 1 \leq 4$). It implies that

$$\sum_{i=1}^{6} x_{ij} \geq 2 \quad j = 1, \ldots, j_0 - 1 \tag{21}$$

Consider the tiling of left j_0 columns. All tiles $y_{ij} = 1$ with $i \leq j_0$ cover two checkerboard squares in left j_0 columns, the same is true for tiles $x_{ij} = 1$ with $i < j_0$. These tiles contribute two to the amount of covered squares in the discussed portion of the checkerboard. Tiles $x_{ij_0} = 1$ contribute one to the number of tiled squares in left j_0 columns. If we count the number of tiled squares in left j_0 columns according to the above observation and take into account that each square should be covered by a tile, we get the following equation:

$$\sum_{j=1}^{j_0}\sum_{i=1}^{5} 2y_{ij} + \sum_{j=1}^{j_0-1}\sum_{i=1}^{6} 2x_{ij} + \sum_{i=1}^{6} x_{ij_0} = 6j_0 \tag{22}$$

Since a *"single-firm"* tiling requires $\sum_{i=1}^{6} x_{ij_0} \geq 1$, equality (22) and "firmness" imply

$$\sum_{j=1}^{j_0}\sum_{i=1}^{5} 2y_{ij} + \sum_{j=1}^{j_0-1}\sum_{i=1}^{6} 2x_{ij} \leq 6j_0 - 1 \tag{23}$$

[2]This fact can be proved by considering a subset of clique constraints corresponding to a monochrome set of checkerboard squares in the way similar to the Mutilated Checkerboard problem.

Inequality (23) can be divided by two:

$$\sum_{j=1}^{j_0}\sum_{i=1}^{5} y_{ij} + \sum_{j=1}^{j_0-1}\sum_{i=1}^{6} x_{ij} \leq 3j_0 - 1/2 \quad (24)$$

If we apply CG-cut to inequality (24), we get a tighter constraint:

$$\sum_{j=1}^{j_0}\sum_{i=1}^{5} y_{ij} + \sum_{j=1}^{j_0-1}\sum_{i=1}^{6} x_{ij} \leq 3j_0 - 1 \quad (25)$$

which together with equality (22) imply the *"double-firm"* tiling of j_0th vertical splitting line:

$$\sum_{i=1}^{6} x_{ij_0} \geq 2 \quad (26)$$

■

Lemma 3 constitutes the hardest part of the Firm Tiling problem. Were we given the *"double-firm"* tiling requirement as the part of the initial problem, a simple application of the Pigeonhole Principle would provide the negative (infeasible) answer: Since there are ten splitting lines, each passing the middle line of at least two tiles, and none of the tiles can be shared by splitting lines in such counting, one needs at least 20 tiles for a *"double-firm"* tiling, in which case they would overlap. The Integer cuts technique demonstrated in Lemma 3 proves infeasibility through deriving the *"double-firm"* tiling requirement from a required *"single-firmness"* and the completeness of tiling (13). Since inequality (13) is actually an equality for the 6x6 checkerboard, the *"double-firm"* tiling inequalities make the Firm Tiling problem infeasible.

From a glance, Integer cuts seem to be manipulating with halfs and other fractionals in a beneficial manner. However, it is not just a game with fractionals. Integer cuts perform methodological "squeezing" of the polygon of feasible solutions remaining all integer solutions feasible. Moreover, new constraints obtained through Integer cuts can not be derived by taking positive weighted sums of the existing constraints.

Although both problems presented in this section do not admit proofs by the original Pigeonhole Principle, it is possible to perform a representation change of the problem statements and transform both problems into ones that admit proofs by the Pigeonhole Principle. We call such method of solving Integer Programming problems as the Hidden Pigeonhole Principle (HPHP).

Such application of HPHP are useful mainly for deriving the tight upper bound. However, for some problems precise knowledge of the optimal value allows AI planning systems to construct an optimal solution. For the problems presented in this section this can be done, for example, by placing domino pieces randomly or greedily and applying the backtracking techniques when necessary. Success in constructing an optimal solution from the optimal value depends heavily on planning domain properties. Nonetheless, without a HPHP application the optimal value would be unknown, and any attempts to construct a solution attaining non-tight upper bound would fail.

5 CONCLUSIONS

In this paper, we showed how through an effective representation change we solved the problem of comparing the bounding power of the Pigeonhole Principle and the Linear Programming Relaxation method. We proved that both methods establish the same upper bounds, when applied to the same problems. The Pigeonhole Principle is more intuitive and often shows whether the upper bound is tight, whereas Linear Programming Relaxation can be applied to any instance of an Integer Programming problem without the use of additional knowledge or representation changes.

We illustrated our new representation change in several combinatorial optimization problems, and demonstrated how this representation change can be utilized to provide an alternative way of solving such problems, which is easier to implement in AI planning systems than PHP or HPHP.

Acknowledgements

This research is sponsored in part by the National Science Foundation under grant number IRI-9502548. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations or the U.S. government.

Thanks to Bart Selman and Matthew Ginsberg for fruitful discussions, to Eugene Fink for creative comments.

References

[Bennett and Potts, 1967] Bennett, B. and Potts, R. (1967). Arrays and brooks. *Journal for the Australian Mathematical Society*, pages 23–31.

[Ginsberg, 1996] Ginsberg, M. (1996). Personal Communications.

[Hoffman et al., 1969] Hoffman, E., Loessi, J., and Moore, R. (1969). Constructions for the solution of the *m* queens problem. *National Mathematics Magazine*, March-April:66–72.

[Khachian, 1979] Khachian, L. (1979). A polynomial algorithm in linear programming. *Soviet Math. Doklady*, 20(1).

[Nauck, 1850] Nauck (1850). Schach. *Illustrierte Zeitung*, 361:352.

[Nemhauser and Wolsey, 1985] Nemhauser, G. and Wolsey, L. (1985). *Integer and Combinatorial Optimization*. John Wiley & Sons.

[Newell, 1965] Newell, A. (1965). Limitations of the current stock of ideas about problem solving. In Kent, A. and Tualbee, O., editors, *Electronic Information Handling*. Spartan Books, Washington, DC.

[Thucker, 1980] Thucker, A. (1980). *Applied Combinatorics*. John Wiley & Sons.