New randomized strategies for the color coding algorithm

Lucie Pansart, Hadrien Cambazard, Nicolas Catusse

24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

New randomized strategies for the color coding algorithm

Lucie Pansart and **Hadrien Cambazard** and **Nicolas Catusse** [1]

Abstract. The color coding technique is used to solve subgraph isomorphism problems, in particular path problems. One color among C is randomly assigned to each vertex of the graph and if distinct colors are given to the vertices of the desired subgraph, it can be found efficiently by dynamic programming. These two phases are repeated until the subgraph is found with a high probability, which can require a large number of iterations. We propose new coloring strategies that take advantage of the graph structure to increase this probability and thus reduce the number of iterations. They provide a guaranteed improvement over the original color coding technique based on a particular structural parameter related to the bandwidth. When this parameter is smaller than the number C of colors, we prove that only C calls to the dynamic program are needed to find the subgraph.

1 Introduction

Color coding is a randomized algorithm proposed by *Alon et al.* [3] to address NP-complete subgraph isomorphism problems. The method was initially introduced to find simple paths or cycles of limited length within a given input graph $G = (V, E)$. To briefly illustrate the technique, consider the L-path problem, *i.e.,* the problem of finding a simple path of L vertices in a given directed graph. The key idea is to randomly color the vertices of the graph with $C \geq L$ colors and to seek a colorful path, *i.e.,* a path where all vertices have distinct colors. On the one hand, the search for a colorful path is significantly more efficient than the search for a simple path as there are C distinct vertex identifiers instead of $|V|$. On the other hand, the random coloring might be unlucky and give the same color to at least two vertices of the path. The two steps are therefore repeated to ensure that the path is found with a high enough probability.

We encountered a weighted variant of the L-path problem in the context of the kidney exchange problem (KEP), which packs a given input graph with K-cycles and L-paths representing exchanges of kidneys. One of the best integer programming formulations solving the KEP is exponential and handled with column generation [22]. The pricing subproblem requires to solve a minimum-weighted L-path problem in a directed and arc-weighted graph. In this situation, which motivates the present work, many of these L-path problems must be solved very quickly.

This pricing subproblem is a special case of the Elementary Shortest Path Problem with Resource Constraints where the resource is the number of vertices and the constraint is to use exactly L of them. It is NP-hard [12, 24] but small instances can be solved in practice (with a dynamic program of complexity $O(2^{|V|} \cdot |V|^2)$ [13]). Many approaches to solve simple path problems consist in relaxing or constraining the problem to obtain a smaller dynamic program. The color coding algorithm is one of them as it constrains the path to have L distinct colors instead of L distinct vertices. Even then, the dynamic program, and thus a color coding iteration, can be very costly as it runs in $O(2^C \cdot |V|^2)$ time. For this reason, we aim at reducing the number of iterations needed to find an optimal weighted path.

The color coding technique is a two-phase algorithm: the first phase colors the graph, the second phase solves the dynamic program in order to find a best L-path with distinct colors. This second phase can miss optimal solutions if every best L-path contains two vertices of the same color. Both from theoretical and practical perspectives, in all prior works related to color coding, the graph is colored according to a discrete uniform distribution: all the colors are equally likely to be selected for each vertex. To the best of our knowledge, only one paper addresses the choice of the coloring for a derandomization purpose [18]. Yet the random distribution used to color the graph is an efficiency factor. Although it does not influence the runtime complexity of the dynamic program, it can highly reduce the number of its executions by increasing the chance for an optimal solution to be colorful.

Our key idea is to bias the coloring in the first phase to increase the probability that an optimal solution has distinct colors and thus reduce the number of calls to the dynamic program. These new coloring strategies take advantage of the graph structure and provide a guaranteed improvement over the original color coding technique on graphs with a particular structural property related to the bandwidth. In practice, a significant improvement is observed on instances from our application and from an online benchmark.

We propose to decompose the coloring phase into two steps: ordering and coloring, detailed in Section 3. Each step is randomized with different strategies. A new general coloring strategy that can be applied for any subgraph isomorphism problem is studied in Section 4. Section 5 introduces an ordering strategy for the special case of connected subgraph isomorphism problems such as path problems. These two techniques are combined together in an algorithm described in Section 6 and evaluated experimentally in Section 7.

2 Literature review

The color coding approach was proposed in 1995 by Alon *et al.* [3]. It has been studied in several papers in which the graph is colored either with a discrete uniform distribution or with a derandomized procedure and no other coloring strategy has been proposed, except in [18]. This is surprising as this algorithm is studied in various domains.

When combined with derandomization [21], color coding can be used to design deterministic, fixed-parameter tractable (FPT) algorithms. A number of state-of-the-art FPT algorithms rely on color coding for several fundamental problems related to packing [19], matching [14, 19], vertex cover [18] or L-path [3, 28]. Several ran-

[1] Univ. Grenoble Alpes, CNRS, G-SCOP, 38000 Grenoble, France, email: {firstname.lastname}@g-scop.grenoble-inp.fr

domized algorithmic frameworks [30] such as randomized divide-and-conquer [7], the random separation technique [6] or parallel fixed-parameter algorithms [4] are also based on color coding.

This technique has also been successfully used in practice, and in particular in the bioinformatics field (see *e.g.* [2, 11, 20, 26, 27]). A typical example is the detection of signaling pathways in protein interaction networks [17]. The problem is cast as a minimum-weighted L-path problem and practical experiments regarding the implementation of color coding is reported by Hüffner *et al.* [17]. In particular, they analyze the best trade-off between the number of colors (which controls the number of trials required) and the complexity of the dynamic programming step (the runtime of one trial).

Another relevant application for the present paper is the work of Borndörfer *et al.* [5] which uses the color coding to solve a path problem, encountered as a pricing subproblem in the context of line planning in public transport. The kidney exchange problem can also be solved with an exponential formulation whose pricing step requires to solve a weighted elementary length constrained path problem [22]. Actually, exponential integer programming formulations based on paths are common in the field of transport and planning. For these formulations, the color coding turns out to be of major interest to solve the pricing problems. Surprisingly though, this algorithm has received little attention in this context and, to the best of our knowledge, only [5] uses it in such a situation.

3 The color coding algorithm

In the following, let $G = (V, E)$ be a simple graph, directed or not, and $n = |V|$. The color coding method is meant to find any subgraph $H = (V_H, E_H)$ with $L = \left|V_H\right|$ in G. The set V is colored with C colors and if by chance the vertices of H all have distinct colors then the problem is solved.

A coloring of a subset S of vertices is a tuple $c \in \{1, ..., C\}^{|S|}$. There are $C^{|S|}$ colorings of S. Let c_i bet the color of vertex $i \in S$. A coloring c is said to be colorful for the subset $S \subseteq V$, if $c_i \neq c_j \; \forall i \neq j \in S$. A graph is colored when each vertex of V is assigned to a color. It is a randomized procedure and the color assigned to a vertex follows a probability distribution. Let $C_i \in \{1, ..., C\}$ be the random variable giving the color of vertex i. A coloring strategy defines $\mathbb{P}(C_i = c), \forall i \in V$ and $\forall c \in \{1, ..., C\}$.

The standard version of the color coding algorithm draws each color according to a discrete uniform distribution. In this case, denoted by `unif`, the probability for a vertex to get a color is the same for each color and each vertex: $\forall i \in V, \; \forall c \in \{1, ..., C\}$, $\mathbb{P}(C_i = c) = \frac{1}{C}$. They are $\frac{C!}{(C-L)!}$ colorful colorings for H and every coloring is equally likely to happen. The probability that H is colorful can be easily computed and was reported by Alon *et al.* [3] in the case $L = C$.

Property 1. *If the graph is colored following the* `unif` *strategy, then* $\mathbb{P}(H \text{ is colorful}) = \frac{C!}{(C-L)!C^L}$

This probability does not depend on H, nor on the structure of the graph or the problem. Our idea is to introduce a bias in the graph coloring with the purpose to increase the chance that H is colorful. To address this question, we decompose the coloring of G into two randomized steps:

- The ordering step, which randomly orders the vertices according to an ordering strategy.
- The coloring step, which, given the ordering, randomly colors the vertices according to a coloring strategy.

Ordering step. An ordering x is a permutation of the set $\{0, ..., n-1\}$. Let x_i be the position of vertex i in the ordering. Note that the ordering begins at position 0. We also refer to x as a coloring sequence. We propose to randomize the vertex ordering and define, $\forall i \in V$, $X_i \in \{0, ..., n-1\}$ the random variable giving the position of i in the sequence. An ordering strategy defines a probability distribution for the variable $X = (X_1, ..., X_n)$. When the random variable X follows a discrete uniform distribution each coloring sequence has the same probability to occur than the others. A new strategy using the structure of the graph will be presented in Section 5.

Coloring step We assume that a coloring sequence x has been already drawn, *i.e.*, $X_i = x_i \; \forall i \in V$. We propose to make the choice of a color for each vertex i dependent on the previously colored vertices, $\eta(i) = \{j \in V : X_j < X_i\}$. Note that $|\eta(i)| = X_i$, meaning that when i has to be colored, X_i vertices are already colored. We now define a coloring strategy by $\mathbb{P}(C_i = c | C_j \; \forall j \in \eta(i))$.

Many choices can be made for this distribution. Our idea is to force the colors to be equally spread in the graph. The next section details this new strategy and how it affects the probability that H is colorful.

4 Coloring by intervals

In this section we present a new coloring strategy that aims at spreading the colors uniformly. The idea is to color the ordered vertices by intervals of size C so that each interval is colorful. For sake of simplicity assume in this section that $n = qC$ with $q \in \mathbb{N}$ (the following results directly adapt when $n = qC + r$). We divide the coloring sequence $[\![0; n-1]\!]$ into q intervals $I_1, I_2, ..., I_q$ of size C: $I_i = [\![(i-1)C; iC - 1]\!]$. $I(X_i)$ denotes the interval containing the position of i, *i.e.*, $X_i \in I_k \Leftrightarrow I(X_i) = I_k$. By slightly abusing the notation we also write $i \in I_k$.

We propose that the color of a vertex depends only of the colors taken by the other vertices in the same interval. We refer to this strategy as `spread` and intentionally eclipse the mathematical definition of $\mathbb{P}(C_i = c | C_j \; \forall j \in \eta(i))$ to give a clearer description: $\forall I \in \{I_1, ..., I_n\}$, $\forall (i_1, ..., i_C) \in I$, $(c_{i_1}, ..., c_{i_C})$ is a permutation of $\{1, ..., C\}$ chosen with a uniform probability over all the permutations. Two vertices in the same interval thus cannot have the same color. As a result, since the coloring of two intervals is independent, the probability that two vertices take distinct colors can be easily computed.

Property 2. *If the coloring step follows the* `spread` *strategy, then* $\forall i, j \in V \; i \neq j$*:*

$$\mathbb{P}(C_i \neq C_j | X_i, X_j) = \begin{cases} 1 & \text{if } I(X_i) = I(X_j) \\ \dfrac{C-1}{C} & \text{otherwise} \end{cases}$$

Probability that H is colorful. Recall that we want to color G such that V_H is colorful. The probability that this event occurs is an indicator of performance for a coloring strategy. In the `unif` case, $\mathbb{P}(H \text{ is colorful})$ is easily computed and does not depend on an ordering. The analysis of the `spread` strategy is detailed below.

From the property 2, we know that vertices in the same interval have distinct colors, but in general the vertices of H are likely to be spread in several intervals. Let $V_H = \{v_1, ..., v_L\}$ denote the L vertices of H and thus $\mathbb{P}(H \text{ is colorful}) = \mathbb{P}(C_{v_1} \neq ... \neq C_{v_L})$.

Let Y_k be the random variable counting the number of vertices of H in the k^{th} interval I_k. Y_k is calculated from the random variables X_i as follows: $Y_k = \left|\{i \in V_H : X_i \in I_k\}\right|$. Recall that q is the number of intervals ($n = qC$). A realization x of the random variable X can be used to compute $y \in \mathcal{Y} = \left\{(y_1, ..., y_q) \in [\![0, L]\!]^q : \sum_{k=1}^{q} y_k = L\right\}$. By the law of total probability, we have:

$$\mathbb{P}\left(C_{v_1} \neq ... \neq C_{v_L}\right) = \sum_{y \in \mathcal{Y}} \mathbb{P}\left(C_{v_1} \neq ... \neq C_{v_L}, Y = y\right) \quad (1)$$

and

$$\mathbb{P}\left(C_{v_1} \neq ... \neq C_{v_L}, Y = y\right) = \mathbb{P}\left(Y_1 = y_1, ..., Y_q = y_q\right) \times \mathbb{P}\left(C_{v_1} \neq ... \neq C_{v_L} \middle| Y_1 = y_1, ..., Y_q = y_q\right) \quad (2)$$

Let $y \in \mathcal{Y}$ be a repartition of the vertices of H in the q intervals. The first term of the product (2) is the probability that $Y = y$. With a uniform ordering strategy, it is:

$$\mathbb{P}\left(Y_1 = y_1, ..., Y_q = y_q\right) = \frac{\prod_{k=1}^{q} \binom{C}{y_k}}{\binom{n}{L}}$$

The second term of the product (2) is the probability that H is colorful knowing the repartition is fixed to y.

$$\mathbb{P}\left(C_{v_1} \neq ... \neq C_{v_L} \middle| Y_1 = y_1, ..., Y_q = y_q\right) = \frac{\binom{C}{y_1} \times \binom{C-y_1}{y_2} \times \binom{C-(y_1+y_2)}{y_3} \times ... \times \binom{C-(y_1+...+y_{q-1})}{y_q}}{\prod_{k=1}^{q} \binom{C}{y_k}} \quad (3)$$

Best and worst cases. The best case for this coloring strategy is when all the vertices of H are in the same interval, as H is then colorful with a probability of 1.

On the contrary, the worst case for this strategy is when all the vertices are ordered in different intervals. We denote this event S. Namely, $S := \{I(X_{v_i}) \neq I(X_{v_j})\ \forall v_i, v_j \in V_H\}$. We show that in this case the `spread` strategy is equivalent to the `unif` strategy.

Property 3. $\mathbb{P}\left(C_{v_1} \neq ... \neq C_{v_L} | S\right) = \frac{C!}{(C-L)!C^L}$

Proof. The event S can be defined as "exactly L intervals I_k have $y_k = 1$". Using (3), we have:

$$\mathbb{P}\left(C_{v_1} \neq ... \neq C_{v_L} | S\right) = \frac{\binom{C}{1} \times \binom{C-1}{1} \times \binom{C-2}{1} \times ... \times \binom{C-(L-1)}{1}}{\prod_{k=1}^{L} \binom{C}{1}} = \frac{C!}{(C-L)!C^L}$$

□

Thus, the new strategy always improves over the original one. Indeed, the chance for H to be colorful is always greater or equal in the `spread` case than in the `unif` case. However, as n increases, S is more likely and `spread` is more and more equivalent to `unif`. This convergence is yet slower as n grows. Figure 1 shows $\mathbb{P}(S)$ depending on n for various L.

Property 4. *With a uniform ordering,* $\mathbb{P}(S) \underset{n \to +\infty}{\longrightarrow} 1$

Proof. Let $\mathcal{U} = \left\{(y_1, ..., y_q) \in \{0, 1\}^q : \sum_{k=1}^{q} y_k = L\right\}$ be the set of realizations y such that S happens. $\left|\mathcal{U}\right| = \binom{q}{L} = \binom{\frac{n}{C}}{L}$. Every tuple (y) of $\mathcal{U}$ is equally likely to happen with probability $\frac{\prod_{k=1}^{L} \binom{C}{1}}{\binom{n}{L}} = \frac{C^L}{\binom{n}{L}}$.

$$\begin{aligned}
\mathbb{P}(S) &= \sum_{y \in \mathcal{U}} \mathbb{P}\left(Y_1 = y_1, ..., Y_q = y_q\right) = \sum_{y \in \mathcal{U}} \frac{C^L}{\binom{n}{L}} = \binom{\frac{n}{C}}{L} \frac{C^L}{\binom{n}{L}} \\
&= \frac{\frac{n}{C} \times (\frac{n}{C} - 1) \times ... \times (\frac{n}{C} - L + 1)}{L!} \times \frac{C^L \times L!}{n \times (n-1) \times ... \times (n - L + 1)} \\
&= \frac{n \times (n \times (n - C) \times ... \times (n - C(L-1))}{L! \times C^L} \times \frac{C^L \times L!}{n \times (n-1) \times ... \times (n - L + 1)} \\
&= \left(\frac{n - C}{n - 1}\right) \times \left(\frac{n - 2C}{n - 2}\right) \times ... \times \left(\frac{n - C(L-1)}{n - (L-1)}\right)
\end{aligned}$$

Each term of this product converges to 1 as n tends to $+\infty$, so the product converges to 1. □

Property 5. *The derivative* $\frac{\partial \mathbb{P}(S)}{\partial n}$ *is in* $O\left(\frac{1}{n^2}\right)$.

Proof. $\mathbb{P}(S) = \prod_{i=1}^{L-1} \frac{(n-iC)}{(n-i)} = \frac{\prod_{i=1}^{L-1} (n-iC)}{\prod_{i=1}^{L-1} (n-i)}$

$\mathbb{P}(S)$ is the quotient of two polynomials of the same degree $L-1$, so the result is a quotient with one degree less in the numerator. Thus $\mathbb{P}(S) = \frac{u}{v}$ with u a polynomial of degree at most $L-2$ and v is still a polynomial of degree $L-1$. The derivative $\frac{\partial \mathbb{P}(S)}{\partial n}$ is a polynomial quotient with a polynomial of degree at most $2L-4$ as numerator and a polynomial of degree $2L-2$ as denominator. Therefore $\frac{\partial \mathbb{P}(S)}{\partial n}$ is in $O\left(\frac{1}{n^2}\right)$. □

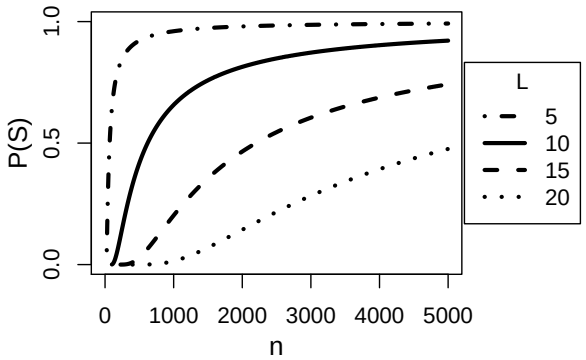

Figure 1: $\mathbb{P}(S)$ with $C = L$

The `spread` strategy is at least as good as the `unif` strategy. For small n, the improvement is significant but the gap between the two strategies follows a decreasing function in $O(\frac{1}{n^2})$. As an example, let $L = C = 10$: when $n = 30$, the probability that H is colorful increases more than fivefold, and when $n = 100$ the factor is

about 1.6. A major property of our coloring strategy is that some subsets of vertices (the intervals I_k) are colorful. Thus, if the vertices of H could be gathered in a single interval, then the `spread` strategy would always make H colorful. In the next section, we propose a new ordering that takes advantage of the graph structure to address this goal.

5 Ordering strategy for connected subgraph isomorphism problems

The `spread` coloring strategy is very efficient when the coloring sequence defined by the ordering strategy puts the vertices of H in the smallest number of intervals. Even more, an ordering where the vertices of H belong to a single interval ensures a probability of 1 that H is colorful. This ideal ordering cannot be easily found but we can define a strategy that orders close to each other two vertices that may belong to H.

We propose a new strategy dedicated to this purpose when H is connected. We compute with the Floyd-Warshall algorithm the distance function $d : V \times V \Rightarrow \mathbb{N} \cup \{+\infty\}$ where $d(i, j)$ is the number of arcs of a shortest path (with respect to the **number** of arcs) between i and j. We define the extended neighborhood $\gamma(i)$ of vertex i as the set of vertices that may appear in a subgraph H including i.

Our first idea is to control the maximum difference between the position of two extended neighbors, defined as $\Delta = \max_{\substack{i \in V \\ j \in \gamma(i)}} |x_i - x_j|$. V_H is contained in a coloring subsequence of size at most Δ. To create an ordering that minimizes Δ, we solve the following optimization problem:

$$\begin{aligned} \Delta^* = \min\ \Delta = & \max_{\substack{i \in V \\ j \in \gamma(i)}} |x_i - x_j| \\ & x_i \neq x_j \qquad \forall i \neq j \in V \\ & x_i \in \{0, ..., n-1\} \qquad \forall i \in V \end{aligned} \tag{4}$$

This problem is equivalent to the graph bandwidth problem in an auxiliary graph $G' = (V, E')$ with the same set of vertices V but which contains an edge (ij) if i and j are extended neighbors. For the L-path problem, we look for a path having L vertices, so $L - 1$ arcs. In this case the extended neighborhood is defined $\forall i \in V$ as $\gamma(i) = \{j \in V : d(i, j) < L \text{ or } d(j, i) < L\}$ and G' is the $L - 1^{th}$ power graph of G.

The graph bandwidth problem on G' aims at labeling the vertices of G' with distinct integers such that the maximum difference between the label of two adjacent vertices is minimized. In our case the label of a vertex i is its position X_i and solving the bandwidth problem provides a coloring sequence x minimizing $\Delta = \max_{(ij) \in E'} |x_i - x_j| = \max_{\substack{i \in V \\ j \in \gamma(i)}} |x_i - x_j|$.

The optimal solution Δ^* of (4) is called the bandwidth of G', denoted by $\varphi(G')$. Finding the bandwidth of a graph is an NP-hard problem [23] for which various algorithms exist, as the widely used Cuthill-McKee heuristic and its reversed version [9, 16]. Many of them are heuristics but some exact approaches are able to solve instances of reasonable size (250 nodes). We refer the reader to [29] for a comprehensive review.

The graph bandwidth problem focuses only on the maximum difference between the positions of two extended neighbors and does not control the average difference between them. However, it could be efficient to move the positions of two extended neighbors away from each other, at the risk of increasing Δ, if that means many other extended neighbors are contained in a smaller coloring subsequence. We thus propose to solve a slightly different problem that minimizes the sum of all differences, that is $\delta = \sum_{(ij) \in E'} |x_i - x_j| = \sum_{\substack{i \in V \\ j \in \gamma(i)}} |x_i - x_j|$. This problem is known as the (minimum) linear arrangement problem [1] that was proven to be NP-complete [15].

$$\begin{aligned} \delta^* = \min\ \delta = & \sum_{(ij) \in E'} |x_i - x_j| \\ & x_i \neq x_j \qquad \forall i \neq j \in V \\ & x_i \in \{0, ..., n-1\} \qquad \forall i \in V \end{aligned}$$

In this case, Δ can be big, but in average the difference between the positions of two extended neighbors might be small. We therefore define a new ordering strategy based on this linear assignment problem, called `la` ordering, which constructs an ordering aimed at minimizing δ. We use a simple local search approach swapping the vertices in the sequence that we do not detail in the present paper. This strategy gathers extended neighbors in the coloring sequence and its quality depends on its running time as the local search is an anytime algorithm that provides an ordering whenever it is asked. The longer it runs, the better the ordering, *i.e.*, the smaller δ. Note that the solution to the linear arrangement problem does not have to be optimal, even if its quality has an impact on the efficiency of the coloring. The running time is determined by the user depending on the algorithm application and the time allocated to the dynamic program of the second phase.

Even when the positions of V_H are close enough, `la` ordering cannot guarantee that they belong to the same interval I_k. Indeed, the size of the position sequence of V_H may be smaller than C while being split into two intervals. In this case and with a standard color coding, the `spread` coloring strategy would not find H with a probability of 1. This motivates the new color coding technique introduced in the next section.

6 `Order-and-shift` technique

Given a coloring sequence x and a subgraph H, we can compute the size of the coloring subsequence containing all the vertices of H: $\Delta_H := \max_{\substack{i \in V_H \\ j \in \gamma(i)}} |x_i - x_j|$. Recall that the `spread` coloring strategy colors the vertices of G by intervals of size C. However, even if Δ_H is smaller than C, the coloring subsequence of H can straddle two coloring intervals, and H can be non colorful.

We introduce a new coloring strategy, called `shifted-spread`, making the color coding more expensive (C dynamic programs calls by iteration) but stronger. We apply C times the `spread` coloring while shifting the coloring sequence between each iteration. Thus, if $\Delta_H \leq C$, at least one coloring sequence will put the vertices of H in a single coloring interval I_k and H will be colorful. As we do not know H, we cannot compute Δ_H, but the algorithm guarantees that when $\Delta \leq C$ only one iteration of the color coding algorithm, with C calls to the dynamic program, is required to find H.

Property 6. *With a `shifted-spread` coloring strategy, if $\Delta \leq C$ then $\mathbb{P}(H \text{ is colorful}) = 1$*

We saw in Section 4 that the `spread` strategy gives a significant improvement for small n which does not scale with n. When com-

bined with a `shifted-spread` ordering strategy, this improvement now depends on Δ rather than n. Indeed, the probability that H is colorful if G is colored with a `shifted-spread` strategy is at least as good as the probability that H is colorful in a graph with Δ vertices colored with a `spread` strategy. Algorithm 1 details the complete `la-and-shifted-spread` color coding using our ordering and coloring strategies.

Algorithm 1 `la-and-shifted-spread` color coding

Input: `#iterations`, `tmax`
  apply `la` ordering within time limit `tmax` $\rightarrow x = (x_1, ..., x_n)$
  if $\Delta \leq C$ **then** `#iterations` $\leftarrow 1$
  for 1...`#iterations` **do**
    apply a `shifted-spread` coloring:
    for $k : 1...C$ **do**
      • color: draw C_i $\forall i \in V$ using a `spread` coloring
      • solve: apply an algorithm that finds H if colorful
      • shift the ordering: $x = (x_{1+k}, ..., x_n, x_1, ..., x_k)$

7 Experimental results

We present experimental results for the color coding algorithm applied to a minimum-weighted L-path problem. This problem emerged from the pricing step of the column generation framework solving the kidney exchange problem [22].

7.1 Protocol

Recall that our problem is to find an optimal path of length L in a weighted directed graph. We want to measure how the coloring of the first phase impacts the probability to find any optimal path in the second phase. This happens each time an optimal path is colorful at the end of the first phase and does not depend on the second phase. As the performances of the dynamic program are not affected by our strategies, our experiments focus only on the first phase. Thus, we do not run the dynamic program and assume that the optimal paths are known[2], constituting the set of paths $\mathcal{P}$. To assert the effectiveness of the various coloring strategies, we only check if one path among $\mathcal{P}$ is colorful. We execute 10 000 iterations of each strategy and compute the number of times one of the optimal paths is colorful out of the 10 000 trials. This frequency estimates the probability of the first phase to be successful.

Choice of the instances. We test our algorithms on two benchmarks of graphs and with L in $\{10, 15, 20\}$.

The first benchmark, denoted by KBEA, contains graphs that come from the pricing step of our column generation framework solving the KEP. Initial KEP instances are created using a Saidman-based generator [25] available at `http://www.dcs.gla.ac.uk/~jamest/kidney-webapp/#/generator` using realistic parameters, leading to sparse graphs. The size of the graphs varies between 68 and 334 vertices. Depending on the two parameters $|\mathcal{P}|$ and L, 18 or 21 instances are solved.

We use another benchmark to analyze our results on different and structured instances that can be found online for reproducibility purposes. It is composed by graphs designed for a tree-width problem of the 2016 PACE challenge [10], available at `http://bit.ly/pace16-tw-instances-20160307`. We select graphs having a size between 30 and 3300 vertices and containing enough paths of the given length L. We decompose this benchmark into two parts: the PACE-exact (up to 600 vertices) and the PACE-heuristic instances which contains larger instances. Depending on the two parameters $|\mathcal{P}|$ and L, 93 to 97 instances are solved.

[2] actually $\mathcal{P}$ is composed by random paths playing the role of optimal paths, the optimality is not important here

Choice of the path set $\mathcal{P}$. As none of the used strategies depends on the weight function, any path can belong to the set $\mathcal{P}$, so we draw them randomly in the graph. Their number $|\mathcal{P}|$ varies in $\{3, 10, 50\}$.

Choice of the algorithms. We compare four strategies:
1. color coding with `unif` coloring (standard version)
2. color coding with uniform ordering and `spread` coloring
3. color coding with `la` ordering and `spread` coloring
4. `la-and-shifted-spread` color coding

The point of these experiments is mainly to establish the impact of the `spread` coloring strategy and how a *good* ordering can affect it. Thus, even if it is known that using more colors than L leads to a higher chance that H is colorful, we stand in a simpler case where $C = L$. For the same reason, the time limit given to the local search of the `la` ordering is sufficiently long (5 minutes) to hope for a good ordering. This time limit can be adapted to fit the need of the various applications of the color coding. Note that both strategies 3 and 4 use the same `la` ordering.

7.2 Analysis

Coloring `spread` versus `unif`. The `spread` coloring (strategy 2) reveals moderately better results than the `unif` coloring. The average frequency with which a path of $\mathcal{P}$ is found increases from 2.26% to 3.54%. As the ordering does not take profit from any structure and because of the size of our instances, the benefit of this coloring strategy alone is quite small. This is consistent with the analysis made in Section 4 for large values of n.

`la` ordering versus uniform ordering. The frequencies with which at least one path of $\mathcal{P}$ is colorful are denoted by f_u, f_3 and f_4 for strategies 1, 3 and 4 respectively. They estimate the probability to find an optimal path within one iteration of the color coding. The average frequencies on a set of instances are denoted by $\overline{f_u}$, $\overline{f_3}$ and $\overline{f_4}$. We also computed the minimum (resp. average) number of ordering intervals I needed to cover a path of $\mathcal{P}$, denoted by $\#I_u^m$ (resp. $\overline{\#I_u}$) for the uniform ordering and $\#I_{la}^m$ (resp. $\overline{\#I_{la}}$) for `la` ordering.

The gain on the frequencies estimates the gain on the probability that an optimal path is colorful, which is, equivalently, a gain on the number of color coding iterations needed to find an optimal path. We are however interested in the gain on the number of calls to the costly dynamic program. For the non shifted strategy, each iteration calls only one dynamic program, so $\overline{g_3} = \overline{f_3}/\overline{f_u}$. On the contrary, an iteration of `la-and-shifted-spread` calls C dynamic programs, thus, we compute the gain of the normalized frequency $normf_4 = f_4/C$ and $\overline{g_4} = \overline{normf_4}/\overline{f_u}$.

Tables 1, 2 and 3 show the aggregated results for the benchmarks KBEA, PACE-exact and PACE-heuristic respectively: the average frequency of `unif` coloring ($\overline{f_u}$), the average frequency and gain for `la` ordering with `spread` ($\overline{f_3}$ and $\overline{g_3}$) and the average frequency with its net gain for `la-and-shifted-spread` ($\overline{f_4}$ and $\overline{g_4}$). They also display the average (over instances) minimum number of intervals covering a path of $\mathcal{P}$ for each ordering ($\overline{\#I_u^m}$ and

$\overline{\#I_{la}^m}$). In these tables, # denotes the number of instances of the benchmark that were tested. We also detail individual results in Table 4 for KBEA and PACE-exact benchmarks, $L = C = 15$ and $|\mathcal{P}| = 3$ (chosen as examples).

The bold values in Tables 1, 2 and 3, enlighten the significant gain of our methods in almost every cases, so we can expect a substantial reduction of the dynamic program executions to find an optimal path. This gain sometimes reaches several orders of magnitude, in particular for the PACE-exact benchmark (Table 2). Note that `la-and-shifted-spread` gives very high frequencies, but the third strategy has a slightly better net gain and the shift seems unnecessary. In one configuration (out of 9), even these high frequencies do not compensate this shifting technique and $\overline{g_4}$ is smaller than 1. However, only `la-and-shifted-spread` provides the guarantee that a path is found in one iteration if $\Delta \leq C$. Actually, as discussed in Section 5, Δ might big (see Figure 2), `la-and-shifted-spread` still outperforms `unif` and we can observe a number of instances with $f_4 = 1$ but $\Delta > C$ in Table 4. This probability of one occurs when $\#I_{la}^m = 1$, *i.e.*, when our ordering sufficiently gathered the vertices of one path of $\mathcal{P}$ so that one of the shifts puts them in a single coloring interval. This happens many times for the PACE-exact benchmark, which explains the good performance of `la-and-shifted-spread`. The side effect of the normalization on values bounded by 1 explains the important gap between $\overline{g_3}$ and $\overline{g_4}$ for this benchmark. More generally, when $\overline{\#I_{la}^m}$ is much smaller than $\overline{\#I_u^m}$, we observe a significant improvement of the colorful frequency, for both strategies 3 and 4, since our ordering clusters the vertices of the paths of $\mathcal{P}$ in a small number of intervals. On the contrary, when $\overline{\#I_{la}^m}$ and $\overline{\#I_u^m}$ are close, our methods behave similarly to `spread` alone since the ordering strategy gathers the vertices of the paths in almost as many intervals as a uniform ordering strategy. When this happens, the gain is attributable only to `spread` and remains, therefore, rather small.

$\lvert\mathcal{P}\rvert$	L	#	uniform ordering		la ordering				
			unif	$\overline{\#I_u^m}$	spread		shifted-spread		$\overline{\#I_{la}^m}$
			$\overline{f_u}$		$\overline{f_3}$	$\overline{g_3}$	$\overline{f_4}$	$\overline{g_4}$	
3	10	21	0.0108	6.1	0.0278	**2.6**	0.1945	**1.8**	5.2
	15	18	0.0001	7.2	0.0003	**2.7**	0.0051	**2.6**	6.3
	20	18	3.7E^{-7}	7.2	1E^{-5}	**27**	0.0002	**26.1**	6.1
10	10	21	0.034	5.7	0.1088	**3.2**	0.4537	**1.3**	4.6
	15	18	0.0004	6.5	0.0011	**2.4**	0.0159	**2.4**	5.8
	20	18	6.7E^{-6}	6.7	2.9E^{-5}	**4.3**	0.0006	**4.7**	5.9
50	10	18	0.1571	4.8	0.2015	**1.3**	0.8918	0.57	4.6
	15	18	0.0022	5.8	0.0053	**2.4**	0.0749	**2.2**	5.3
	20	18	2.6E^{-5}	6.1	0.0002	**5.9**	0.003	**5.7**	5.4

Table 1: Results for KBEA

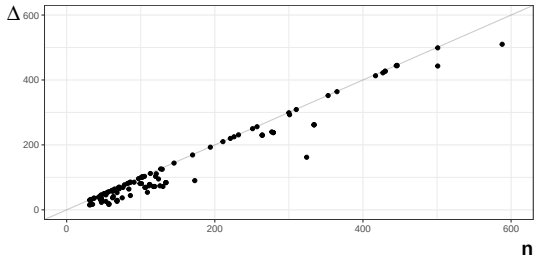

Figure 2: Δ for each instance of size < 600 with $L = 10$

$\lvert\mathcal{P}\rvert$	L	#	uniform ordering		la ordering				
			unif	$\overline{\#I_u^m}$	spread		shifted-spread		$\overline{\#I_{la}^m}$
			$\overline{f_u}$		$\overline{f_3}$	$\overline{g_3}$	$\overline{f_4}$	$\overline{g_4}$	
3	10	40	0.0106	4.8	0.2419	**22.8**	0.6109	**5.8**	2
	15	39	0.0001	4.8	0.0978	**765.2**	0.3803	**198.3**	2
	20	39	1E^{-6}	4.4	0.0859	**83734**	0.2261	**11021**	2.6
10	10	40	0.0336	4.4	0.4449	**13.2**	0.8904	**2.6**	1.7
	15	39	0.0004	4.4	0.2435	**568.7**	0.5133	**79.9**	1.8
	20	39	4.4E^{-6}	4.2	0.1653	**37184**	0.3021	**3399**	2.3
50	10	39	0.1308	3.9	0.7907	**6**	0.9953	0.76	1.4
	15	38	0.0019	4.2	0.4667	**248.9**	0.7251	**25.8**	1.7
	20	36	1.9E^{-5}	4.2	0.1954	**10243**	0.425	**1114**	2.1

Table 2: Results for PACE-exact

$\lvert\mathcal{P}\rvert$	L	#	uniform ordering		la ordering				
			unif	$\overline{\#I_u^m}$	spread		shifted-spread		$\overline{\#I_{la}^m}$
			$\overline{f_u}$		$\overline{f_3}$	$\overline{g_3}$	$\overline{f_4}$	$\overline{g_4}$	
3	10	55	0.0109	5.8	0.0187	**1.7**	0.1651	**1.5**	5
	15	52	0.0001	6.3	0.0006	**4.5**	0.0077	**4.1**	5.6
	20	52	1.2E^{-6}	6.3	3.8E^{-5}	**33**	0.0007	**29**	5.7
10	10	55	0.0355	5.3	0.0597	**1.7**	0.4327	**1.7**	4.5
	15	52	0.0004	5.9	0.0019	**4.3**	0.0262	**3.9**	5.1
	20	52	4.7E^{-6}	6.1	0.0001	**22.8**	0.0021	**22.3**	5.3
50	10	55	0.1635	4.7	0.2568	**1.6**	0.921	0.66	4.1
	15	52	0.0022	5.4	0.0089	**4**	0.1169	**3.5**	4.6
	20	52	2.8E^{-5}	5.5	0.0006	**20.1**	0.0111	**20**	4.9

Table 3: Results for PACE-heuristic

8 Conclusion

This paper introduced a new framework for the color coding approach including a new coloring strategy and a new ordering strategy. The coloring strategy significantly improves the probability to find a subgraph for small graphs by spreading colors uniformly. When combined with the new ordering strategy based on the graph structure, the proposed algorithm dominates the original color coding, as it preserves the improvement of the coloring strategy in graph with small Δ, a parameter related to the bandwidth.

The complete framework also includes a shifting technique that guarantees to find an optimal solution with only C calls to the dynamic program when the ordering step makes Δ bounded by C. With or without the shifting, a randomized color coding approach using our algorithm needs far fewer calls to the dynamic programming step to expect the same chance to obtain an optimal solution than in the standard version using a uniform coloring. These algorithms were tested on realistic instances of the kidney exchange problem, an application domain widely studied in recent years [8], but also on graphs coming from structural problems [10]. Similar results can be expected for graphs from other domains, particularly if the graphs are sparse. We now intend to investigate the consequences of our strategies for derandomization purposes.

ACKNOWLEDGEMENTS

We would like to thank our colleague Louis Esperet for his precious advices and his hint about the graph bandwidth, as well as the three anonymous reviewers who helped us improve this paper.

						uniform ordering			la ordering				
						unif			spread	shift			
instance			n	m	d	f_u	$\overline{\#I_u}$	$\#I_u^m$	f_3	f_4	Δ	$\overline{\#I_{la}}$	$\#I_{la}^m$
KBEA-realistic	final	100-10-4-l12	112	731	0.059	0.0001	5.7	5	0.0004	0.0074	78	5	5
		100-25-4-l12	134	834	0.047	0.0001	6	5	0.0004	0.0063	84	6	5
		100-5-4-l12	106	679	0.061	0.0001	5	5	0.0007	0.0112	69	4.7	3
		250-10-4-l12	279	4652	0.06	0.0001	9.7	9	0.0002	0.0029	238	10.7	8
		250-25-4-l12	334	4670	0.042	0.0002	11	9	0.0001	0.0028	262	10.3	8
		250-5-4-l12	264	4524	0.065	0.0001	9.7	9	0.0002	0.0032	230	9.3	8
	first	100-10-4-l12	112	731	0.059	0.0002	5.7	5	0.0005	0.0068	78	5	4
		100-25-4-l12	134	834	0.047	0.0001	7	7	0.0005	0.0073	84	5.3	4
		100-5-4-l12	106	679	0.061	0.0001	5	5	0.0007	0.0078	69	4.3	4
		250-10-4-l12	279	4652	0.06	0.0001	9.7	9	0.0002	0.0026	238	10.7	9
		250-25-4-l12	334	4670	0.042	0.0002	9.7	9	0.0002	0.0029	262	9.7	8
		250-5-4-l12	264	4524	0.065	0.0001	10.3	9	0.0002	0.003	230	9.3	9
	middle	100-10-4-l12	112	731	0.059	0.0002	5.7	5	0.0004	0.0059	78	5.3	5
		100-25-4-l12	134	834	0.047	0.0002	6.3	6	0.0003	0.0068	84	5.7	5
		100-5-4-l12	106	679	0.061	0.0001	5	5	0.0005	0.0063	69	5	4
		250-10-4-l12	279	4652	0.06	0.0001	8.7	8	0.0002	0.0026	238	11	8
		250-25-4-l12	334	4670	0.042	0.0001	11.3	10	0.0002	0.0025	265	11	8
		250-5-4-l12	264	4524	0.065	0.0001	9.3	9	0.0002	0.0031	230	8.7	8
PACE-exact-easy	contiki	cxmac-input-packet	91	284	0.035	0.0002	6	6	0.0022	0.0183	90	4	3
		dhcpc-dhcpc-init	35	102	0.086	0.0001	3	3	0.1364	**1**	**20**	**2**	**1**
		httpd-cfs-send-file	45	140	0.071	0.0001	3	3	0.0016	0.0219	44	3	3
		ifft-ifft	173	532	0.018	0.0002	9	7	0.0009	0.0138	143	4	3
		ircc-list-channel	71	222	0.045	0.0001	4.7	4	0.0265	0.0776	70	3.7	3
		lpp-send-packet	117	356	0.026	0.0001	6.3	6	0.14	**1**	**116**	**2.7**	**1**
		polite-announcement-send-timer	32	93	0.094	0.0001	2.3	2	0.0578	**1**	**23**	**2**	**1**
		powertrace-add-stats	47	140	0.065	0.0001	3.3	3	0.0097	0.0627	46	2.7	2
		powertrace-powertrace-print	324	969	0.009	0.0001	10.3	10	0.0064	**1**	**162**	**2.3**	**1**
		profile-profile-episode-start	32	95	0.096	0.0001	2.7	2	**1**	**1**	**29**	**2**	**1**
		psock-psock-generator-send	62	197	0.052	0.0001	2	3	0.0012	0.0155	61	3.7	3
		rudolph1-rudolph1-send	31	88	0.095	0.0002	3	3	**1**	**1**	**30**	**4**	**1**
		shell-shell-register-command	43	132	0.073	0.0001	3	3	0.0043	0.0338	42	2.7	2
		shell-collect-view-process-thread -collect-view-data-process	62	185	0.049	0.0001	4.7	4	0.0046	**1**	**54**	**2.7**	**1**
		shell-rime-recv-collect	63	190	0.049	0.0001	4.3	4	0.0034	0.357	50	2	2
		shell-rime-ping-recv-mesh	48	141	0.062	0.0001	3.7	3	0.1345	0.3154	47	3	2
		tcpip-eventhandler	99	322	0.033	0.0001	6	5	0.0018	**1**	**95**	**3.3**	**1**
		uip-neighbor-uip-neighbor-add	68	209	0.046	0.0001	4.7	4	0.002	0.0247	57	3	2
		uip-over-mesh-recv-data	86	261	0.036	0.0002	5.7	5	0.0195	**1**	**43**	**2.3**	**1**
		webclient-senddata	109	326	0.028	0.0001	7.3	6	0.0117	0.0521	72	2.3	2
	fuzix	devf-fd-transfer	120	377	0.026	0.0001	6.7	6	0.0038	0.0697	119	2.3	2
		devio-kprintf	70	225	0.047	0.0002	5	5	0.0005	0.009	69	4.7	4
		difftime-difftime	75	220	0.04	0.0001	4.7	4	0.0647	**1**	**37**	**2**	**1**
		fgets-fgets	54	169	0.059	0.0002	4	4	0.0007	0.01	53	4	3
		filesys-filename	46	141	0.068	0.0001	3	3	0.1343	0.3535	45	3	2
		filesys-getinode	53	166	0.06	0.0001	4	4	0.0029	0.0345	52	3.3	2
		filesys-i-open	130	415	0.025	0.0001	8.7	8	0.004	0.0539	129	3	2
		getpass–gets	32	101	0.102	0.0001	3	3	0.0041	0.0701	31	2.7	2
		malloc—insert-chunk	105	336	0.031	0.0001	6	6	0.0008	0.0182	104	4.7	4
		process-getproc	33	102	0.097	0.0002	2.3	2	0.0022	0.0602	32	3	2
		ran-rand	47	142	0.066	0.0001	4	4	0.0214	**1**	**23**	**2**	**1**
		regexp-regcomp	119	376	0.027	0.0001	7.3	7	0.0007	0.0126	116	4	3
		se-ycomp	84	275	0.039	0.0001	5.3	5	0.0009	0.0112	83	3.7	3
		stat-statfix	53	154	0.056	0.0001	4	4	**1**	**1**	**26**	**1.7**	**1**
		tty-tty-read	124	397	0.026	0.0001	6.7	6	0.0031	**1**	**123**	**3**	**1**
	stdlib-sincoshf		111	344	0.028	0.0002	7	6	0.0016	0.0342	98	3.3	2
PACE-exact-hard-DoubleStarSnark			31	120	0.129	0.0001	12	11	0.0004	0.0051	30	6.3	4
PACE-exact-hard-contiki-dhcpc-handle-dhcp			277	902	0.012	0.0001	2	2	0.0046	0.0701	270	2	2
PACE-exact-hard-fuzix-vfscanf-vfscanf			588	1923	0.006	0.0001	12	12	0.0007	0.0268	584	4.7	2

Table 4: Results for $L = C = 15$ and $|\mathcal{P}| = 3$
$n = |V|$, $m = |E|$ and $d = \frac{m}{n(n-1)}$ is the density of the graph $G = (V, E)$

References

[1] D Adolphson and T Ch Hu, 'Optimal linear ordering', *SIAM Journal on Applied Mathematics*, **25**(3), 403–423, (1973).
[2] Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and S. Cenk Sahinalp, 'Biomolecular network motif counting and discovery by color coding', *Bioinformatics*, **24**, i241–i249, (2008).
[3] Noga Alon, Raphael Yuster, and Uri Zwick, 'Color-coding', *Journal of the ACM (JACM)*, **42**(4), 844–856, (1995).
[4] Max Bannach, Christoph Stockhusen, and Till Tantau, 'Fast parallel fixed-parameter algorithms via color coding'. arXiv preprint arXiv:1509.06984, 2015.
[5] Ralf Borndörfer, Martin Grötschel, and Marc E Pfetsch, 'A column-generation approach to line planning in public transport', *Transportation Science*, **41**(1), 123–132, (2007).
[6] Leizhen Cai, Siu Man Chan, and Siu On Chan, 'Random separation: A new method for solving fixed-cardinality optimization problems', in *International Workshop on Parameterized and Exact Computation*, pp. 239–250. Springer, (2006).
[7] Jianer Chen, Joachim Kneis, Songjian Lu, Daniel Mölle, Stefan Richter, Peter Rossmanith, Sing-Hoi Sze, and Fenghui Zhang, 'Randomized divide-and-conquer: Improved path, matching, and packing algorithms', *SIAM Journal on Computing*, **38**(6), 2526–2547, (2009).
[8] COST. European network for collaboration on kidney exchange programmes. `http://www.cost.eu/COST_Actions/ca/CA15210`, 2016.
[9] Elizabeth Cuthill and James McKee, 'Reducing the bandwidth of sparse symmetric matrices', in *Proceedings of the 1969 24th national conference*, pp. 157–172. ACM, (1969).
[10] Holger Dell, Thore Husfeldt, Bart MP Jansen, Petteri Kaski, Christian Komusiewicz, and Frances A Rosamond, 'The first parameterized algorithms and computational experiments challenge', in *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, (2017).
[11] Banu Dost, Tomer Shlomi, Nitin Gupta, Eytan Ruppin, Vineet Bafna, and Roded Sharan, 'Qnet: a tool for querying protein interaction networks', *Journal of Computational Biology*, **15**(7), 913–925, (2008).
[12] Mosh Dror, 'Note on the complexity of the shortest path models for column generation in vrptw', *Operations Research*, **42**(5), 977–978, (1994).
[13] Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen, 'An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle and routing problems', *Networks: An International Journal*, **44**(3), 216–229, (2004).
[14] Michael R Fellows, Christian Knauer, Naomi Nishimura, Prabhakar Ragde, F Rosamond, Ulrike Stege, Dimitrios M Thilikos, and Sue Whitesides, 'Faster fixed-parameter tractable algorithms for matching and packing problems', in *European Symposium on Algorithms*, pp. 311–322. Springer, (2004).
[15] Michael R Garey, David S Johnson, and Larry Stockmeyer, 'Some simplified np-complete problems', in *Proceedings of the sixth annual ACM symposium on Theory of computing*, pp. 47–63. ACM, (1974).
[16] J Alan George, 'Computer implementation of the finite element method', Technical report, Stanford University, Computer Science Department, (1971).
[17] Falk Hüffner, Sebastian Wernicke, and Thomas Zichner, 'Algorithm engineering for color-coding with applications to signaling pathway detection', *Algorithmica*, **52**(2), 114–132, (2008).
[18] Joachim Kneis, Alexander Langer, and Peter Rossmanith, 'Derandomizing non-uniform color-coding I', Technical report, RWTH Aachen - Department of Computer Science, (08 2011).
[19] Ioannis Koutis, 'A faster parameterized algorithm for set packing', *Information processing letters*, **94**(1), 7–9, (2005).
[20] Itay Mayrose, Tomer Shlomi, Nimrod D Rubinstein, Jonathan M Gershoni, Eytan Ruppin, Roded Sharan, and Tal Pupko, 'Epitope mapping using combinatorial phage-display libraries: a graph-based algorithm', *Nucleic acids research*, **35**(1), 69–78, (2006).
[21] Moni Naor, Leonard J Schulman, and Aravind Srinivasan, 'Splitters and near-optimal derandomization', in *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pp. 182–191. IEEE, (1995).
[22] Lucie Pansart, Hadrien Cambazard, and Nicolas Catusse Gautier Stauffer, 'Column generation for the kidney exchange problem', in *12th International Conference on MOdeling, Optimization and SIMulation - MOSIM18 - June 27-29 2018 Toulouse - France "The rise of connected systems in industry and services"*, (2018).
[23] Ch H Papadimitriou, 'The np-completeness of the bandwidth minimization problem', *Computing*, **16**(3), 263–270, (1976).
[24] Benjamin Plaut, John P Dickerson, and Tuomas Sandholm, 'Hardness of the pricing problem for chains in barter exchanges'. arXiv preprint arXiv:1606.00117, 2016.
[25] Susan L Saidman, Alvin E Roth, Tayfun Sönmez, M Utku Ünver, and Francis L Delmonico, 'Increasing the opportunity of live kidney donation by matching for two-and three-way exchanges', *Transplantation*, **81**(5), 773–782, (2006).
[26] Jacob Scott, Trey Ideker, Richard M Karp, and Roded Sharan, 'Efficient algorithms for detecting signaling pathways in protein interaction networks', *Journal of Computational Biology*, **13**(2), 133–144, (2006).
[27] Tomer Shlomi, Daniel Segal, Eytan Ruppin, and Roded Sharan, 'Qpath: a method for querying pathways in a protein-protein interaction network', *BMC bioinformatics*, **7**(1), 199, (2006).
[28] Dekel Tsur, 'Faster deterministic parameterized algorithm for k-path'. arXiv preprint arXiv:1808.04185, 2018.
[29] Chen Wang, Chuan Xu, and Abdel Lisser, 'Bandwidth minimization problem', in *10th International Conference on MOdeling, Optimization and SIMulation - MOSIM14 - November 5-7 2014 Nancy - France "From linear economy to circular economy."*, (2014).
[30] Meirav Zehavi, 'Mixing color coding-related techniques', in *Algorithms-ESA 2015*, 1037–1049, Springer, (2015).