

# Algorithm Engineering for Color-Coding with Applications to Signaling Pathway Detection\*

Falk Hüffner<sup>†</sup>    Sebastian Wernicke<sup>‡</sup>    Thomas Zichner<sup>§</sup>

Institut für Informatik,  
Friedrich-Schiller-Universität Jena,  
Ernst-Abbe-Platz 2,  
D-07743 Jena, Germany.  
{hueffner,wernicke,tzi}@minet.uni-jena.de

## Abstract

Color-coding is a technique to design fixed-parameter algorithms for several NP-complete subgraph isomorphism problems. Somewhat surprisingly, not much work has so far been spent on the actual implementation of algorithms that are based on color-coding, despite the elegance of this technique and its wide range of applicability to practically important problems. This work gives various novel algorithmic improvements for color-coding, both from a worst-case perspective as well as under practical considerations. We apply the resulting implementation to the identification of signaling pathways in protein interaction networks, demonstrating that our improvements speed up the color-coding algorithm by orders of magnitude over previous implementations. This allows more complex and larger structures to be identified in reasonable time; many biologically relevant instances can even be solved in seconds where, previously, hours were required.

## 1 Introduction

**Motivation.** Color-coding is an elegant technique that was introduced by Alon et al. [1] in 1995 to derive (randomized) fixed-parameter algorithms for several NP-complete subgraph isomorphism problems. For example, color-coding can be used to find a  $k$ -vertex path, a  $k$ -vertex cycle, or a given subgraph of bounded treewidth<sup>1</sup> in a given graph.

---

\*An extended abstract of this paper appeared under the title “Algorithm Engineering for Color-Coding to Facilitate Signaling Pathway Detection” in the Proceedings of the 5th Asia-Pacific Bioinformatics Conference (APBC 2007), January 15–17, 2007, Hong Kong, China, volume 5 in *Advances in Bioinformatics and Computational Biology*, pages 277–286, Imperial College Press.

<sup>†</sup>Supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4.

<sup>‡</sup>Supported by the Deutsche Telekom Stiftung.

<sup>§</sup>Supported by the Deutsche Forschungsgemeinschaft, project PEAL (Parameterized Complexity and Exact Algorithms), NI 369/1 and OPAL (Optimal Solutions for Hard Problems in Computational Biology), NI 369/2.

<sup>1</sup>Treewidth is a measure of how treelike a graph is, e.g., see [22, Chapter 10].

Recently, color-coding has received considerable attention in bioinformatics, because it can be used to detect signaling pathways in protein interaction networks. In general, a signaling pathway is a cascade of successive protein interactions that the cell uses to react to various external and internal stimuli. A special role—with respect to both biological meaning as well as algorithmic tractability—is played by the most simple structures, namely *linear* pathways.<sup>2</sup> These are easy to understand and analyze and, as demonstrated by Ideker et al. [15] for the yeast galactose metabolism, they can serve as a seed structure for experimental investigation of more complex mechanisms.

Initiated by Steffen et al. [30], there have been efforts to design algorithms for the automated discovery of linear pathways in protein interaction networks. Two recent works have successfully used the color-coding technique to design algorithms that identify biologically meaningful candidates for linear signaling pathways:

- Scott et al. [27] formalize the task of pathway detection as the NP-complete MINIMUM-WEIGHT PATH problem, that is, the task of finding a minimum-weight simple path of a specified length in a weighted graph. They use color-coding to detect paths of up to 10 vertices, requiring some hours for this task.
- Shlomi et al. [29] formalize the task of pathway detection as the NP-complete PATHWAY QUERY problem, that is, the problem of finding a path in a labeled and weighted graph that best matches a pre-specified *query path* while allowing for a fixed number of insertions and deletions into it. They show that this task can be solved with color-coding, requiring some minutes for paths with up to 7 vertices.

Both works demonstrate that the color-coding approach is capable of identifying biologically meaningful pathways. Their implementations are limited, however, to path lengths of around 10 vertices. Moreover, finding paths of this length requires some hours of running time. In this work, we give various novel improvements for color-coding, both from a worst-case perspective as well as under practical considerations. These speed up the color-coding algorithms by orders of magnitude and thus allow the discovery of pathways that consist of more than 20 vertices within two hours (effectively doubling the path length that can be handled in reasonable time). Perhaps even more important, our improvements allow the frequently encountered task of finding pathways of up to 13 vertices to be accomplished in mere seconds. This allowed the implementation of an interactive application with graphical user interface for finding pathways [14].

**Previous Work.** Concerning the theoretical side of color-coding, several authors have used this technique to develop and improve fixed-parameter algorithms for the problems of SET PACKING and GRAPH PACKING [10, 18, 19, 24]. Very recently, color-coding has also inspired a new algorithmic technique that is based on randomly partitioning a graph into subgraphs and then solving subproblems on these partitions [5, 17].

---

<sup>2</sup>Biologists use the term “linear pathway” to denote a simple path in the protein interaction network, that is, a path where no vertex can occur more than once.

On the practical side, it seems somewhat surprising that not much work has been spent so far on implementing algorithms that are based on the color-coding technique, despite its elegance and wide range of applicability. Besides the implementations of Scott et al. [27] and Shlomi et al. [29], Raymann [25] discusses a color-coding implementation that determines whether an unweighted graph contains a simple  $k$ -vertex path, which, in practice, is an easier problem than the ones we consider here because the algorithm can terminate after it has found a single such path and there are generally many of them to be found. Recently, some more implementations for various applications have been made [4, 7, 8, 20], but none of them focuses on algorithm engineering aspects.

The central problem we are dealing with is MINIMUM-WEIGHT PATH: given an edge-weighted graph with  $n$  vertices and an integer  $k$ , it asks for a simple (non-crossing) path of  $k$  vertices with minimum weight. It is thus a generalization of the well-known NP-hard LONGEST PATH problem. LONGEST PATH is a classical NP-complete problem. It is NP-complete because for  $k = n$ , it is equivalent to HAMILTONIAN PATH. Moreover, even finding a constant factor approximation is NP-hard [16]. The best known polynomial-time approximations [9, 11] are capable of finding a path of length  $k^{1/(\log(n/k)+\log \log n)}$  and  $\exp(\sqrt{\log k / \log \log k})$ , respectively.

Many algorithms for LONGEST PATH, in particular those using dynamic programming, can be adapted for MINIMUM-WEIGHT PATH. The best known exact (not parameterized) algorithm is a dynamic programming based approach [13] which runs in  $O(2^n n^2)$ . Plehn and Voigt [23] gave an algorithm running in  $O(k^{O(k)} n^{\omega+1})$  time, where  $\omega$  is the treewidth of the graph.

Fixed-parameter algorithms [22] are an approach to exactly solving NP-hard problems by confining the combinatorial explosion to a parameter  $k$ . More precisely, a problem is fixed-parameter tractable with respect to a parameter  $k$  if an instance of size  $n$  can be solved in  $f(k) \cdot n^{O(1)}$  time for an arbitrary function  $f$ . For MINIMUM-WEIGHT PATH, the length of the path is a natural parameter. Monien [21] gave the first fixed-parameter algorithm with a running time of  $O(k! \cdot nm)$ . Bodlaender [2] gave an algorithm running in  $O(2^k k! \cdot n)$  time using dynamic programming. Introducing the color-coding method, Alon et al. [1] presented an algorithm solving MINIMUM-WEIGHT PATH in  $O(5.44^k \cdot m)$  time with high probability. In Section 3.1, we show how to speed up this algorithm to achieve an  $O(4.32^k \cdot m)$  time bound.

**Contribution and structure of this work.** As Niedermeier [22, p. 180] notes, there is very little “substantial practical experience with [color-coding]” so far. Motivated by its application in signaling pathway detection, we advance the practical experience with color-coding here and propose a number of algorithmic improvements. Practical experiments on biological and random networks show that this increases the speed of pathway detection by some orders of magnitude.

The color-coding technique is explained in Section 2. Our algorithmic improvements and data structures are discussed in Section 3. The improved color-coding algorithms have been implemented in C++; the source is available at <http://theinf1.informatik.uni-jena.de/colorcoding/> as free software. Section 4 discusses experimental results that were obtained by using our implementation on the *S. cerevisiae* (yeast) interaction network of Scott et al. [27], the *D. melanogaster* (fruit fly) interaction networks of Giot et al. [12], the data

of Shlomi et al. [29], and random networks that are structurally similar to protein interaction networks. These experiments demonstrate that the algorithmic improvements proposed in this work improve the efficiency of color-coding by some orders of magnitude. Concerning the detection of signaling pathways, this means that the detection of larger pathway candidates is possible, and moreover it opens the possibility of interactive exploration of smaller structures.

## 2 The Color-Coding Technique

The central idea that underlies color-coding is to randomly color each vertex of the input graph using a small set of colors and to “hope” that the path that is sought after becomes *colorful* in the process, i.e., that each of its vertices acquires a different color. One then searches only for such colorful paths, which can be done much more efficiently than a search for all paths of a certain length.

As we later discuss in more detail, finding a colorful path can be accomplished in  $O(2^k m)$  time for an  $m$ -edge graph by dynamic programming. Whenever the path that is sought after in the input graph (that is, the path that constitutes an optimal solution) is colorful, it is found by this dynamic programming. The catch, of course, is that the coloring of the input graph is random and hence many coloring *trials* have to be performed to ensure that the minimum-weight path is found with a high probability. More precisely, the probability of any  $k$ -vertex path (including the optimal one) being colorful in a single trial is

$$P_c = \frac{k!}{k^k} > \sqrt{2\pi k} e^{-k} \quad (1)$$

because there are  $k^k$  ways to arbitrarily color  $k$  vertices with  $k$  colors and  $k!$  ways to color them such that no color is used more than once. Using  $t$  trials, a path of length  $k$  is found with probability  $1 - (1 - P_c)^t$ . To ensure that a colorful path is found with a probability greater than  $1 - \varepsilon$  (for some  $0 < \varepsilon \leq 1$ ), at least

$$t(\varepsilon) = \left\lceil \frac{\ln \varepsilon}{\ln(1 - P_c)} \right\rceil = |\ln \varepsilon| \cdot O(e^k) \quad (2)$$

trials are therefore needed. While the result of this approach is only optimal with a (user-specifiable) probability, setting the error probability  $\varepsilon$  to a very low value of, say, 0.1% is likely to be acceptable in practice: Note that only the logarithm of the error probability goes into the overall running time and hence, very low error probabilities are efficient to achieve.

As Alon et al. [1] note, it is also possible to derandomize color-coding using a result of Schmidt and Siegel [26], that is, we can achieve a deterministic color-coding algorithm that maintains the  $2^{O(k)}$  part of the running time bound. Chen et al. [5] give an improved derandomization. However, the randomized algorithm remains preferable in practice: the combinatorial explosion of the derandomized algorithms is much worse, while the error probability of the randomized algorithm is only a logarithmic factor in its running time and can hence be easily made sufficiently small for practical purposes.

It remains to discuss the problem-specific dynamic programming algorithms that are used by Scott et al. [27] and Shlomi et al. [29] in each color-coding trial in order to find the “best” colorful paths in a graph for a given coloring. For this purpose, a protein interaction network is modeled as an undirected

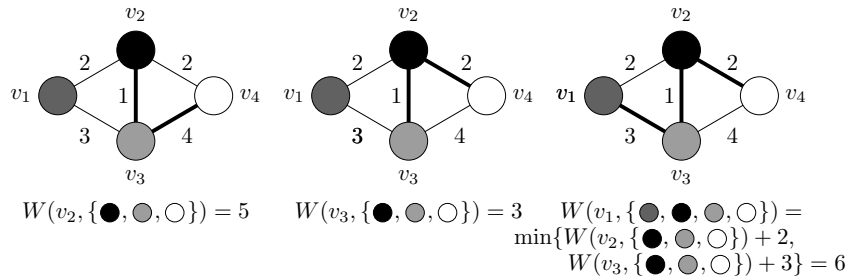


Figure 1: Example for solving MINIMUM-WEIGHT PATH using the color-coding technique. Using Equation (3) a new table entry (right) is calculated using two already known entries (left and middle).

graph  $G = (V, E)$  with  $n := |V|$  and  $m := |E|$ . Each vertex in  $V$  represents a protein; an edge signifies that the proteins represented by its endpoints are assumed to interact. Each edge  $e \in E$  is weighted with a probability  $0 < p(e) \leq 1$  that expresses the likelihood of the interaction (we set  $p(e) = 0$  for  $e \notin E$ ).

## 2.1 Finding Minimum-Weight Paths

Scott et al. [27] demonstrated that *high-scoring simple paths* in a protein interaction network constitute plausible candidates for linear signal transduction pathways, *simple* meaning that no vertex occurs more than once and *high-scoring* meaning that the product of edge weights is maximized. For easier handling, we work with the *weight*  $w(e) := -\log p(e)$  of the edges, such that the goal is to minimize the sum of weights for the edges of a path. To make the finding of signaling pathway candidates algorithmically feasible and biologically meaningful, the number of vertices that these paths contain is restricted by some reasonably small integer  $k$ . Formally stated, the NP-hard problem that needs to be solved in order to find minimum-weight simple paths thus is the following:

### MINIMUM-WEIGHT PATH

**Input:** An undirected edge-weighted graph  $G = (V, E)$  with  $n := |V|$  and  $m := |E|$  and an integer  $k$ .

**Task:** Find a length- $k$  path in  $G$  that minimizes the sum over its edge weights.

Given a fixed coloring of vertices, finding the minimum-weight colorful path can be accomplished by dynamic programming: Assume that for some  $i < k$  we have computed a value  $W(v, S)$  for every vertex  $v \in V$  and cardinality- $i$  subset  $S$  of vertex colors; this value denotes the minimum weight of a path that uses every color in  $S$  exactly once and ends in  $v$ . Clearly, this path is simple because no color is used more than once. We can now use this to compute the values  $W(v, S)$  for all cardinality- $(i + 1)$  subsets  $S$  and vertices  $v \in V$  because a colorful length- $(i + 1)$  path that ends in a vertex  $v \in V$  can be composed of a colorful length- $i$  path that does not use the color of  $v$  and ends in a neighbor of  $v$ . More precisely, we let

$$W(v, S) = \min_{e=\{u,v\} \in E} \left( W(u, S \setminus \{\text{color}(v)\}) + w(e) \right). \quad (3)$$

as exemplified in Figure 1.

It is easy to verify that the dynamic programming takes  $O(2^k m)$  time.<sup>3</sup> Using the number of trials that was established in (2) to achieve an error probability of  $\varepsilon$ , this bounds the overall running time for solving MINIMUM-WEIGHT PATH by  $O(|\ln \varepsilon| \cdot 2^{O(k)} \cdot m)$ .

A particularly appealing aspect of the color-coding method is that it can be easily adapted to many practically relevant variations of the problem formulation: For example, the set of vertices where a path can start and end can be restricted (such as to force it to start in a membrane protein and end in a transcription factor [27]). Unless otherwise noted, for experiments we use the following variant of MINIMUM-WEIGHT PATH that matches the experiments by Scott et al. [27]: With an error probability of  $\varepsilon = 0.1\%$ , we seek 100 minimum-weight paths which must differ from each other in at least 30% of the vertices (to ensure that they are not only minor modifications of the global minimum-weight path).

## 2.2 Querying Paths

Pathway queries are an important problem in the analysis of protein interaction networks [28]: Once a pathway has been identified in a protein interaction network, one is interested in whether similar pathways also exist in protein interaction networks of other species, for example, in order to enable a knowledge transfer from well-studied networks. Of course, one will usually not look for the *exact* same pathway since—in the course of evolution—proteins might have been added, deleted, or replaced. Taking these changes into account led Shlomi et al. [29] to the following problem formalization:

### PATHWAY QUERY

**Input:** An undirected graph  $G = (V, E)$  with an edge weight function  $w : E \rightarrow \mathbb{R}^+$ , a length- $\ell$  query sequence  $Q = q_1, \dots, q_\ell$ , a match weight function  $h : \{q_1, \dots, q_\ell\} \times V \rightarrow \mathbb{R}^+$ , and two nonnegative integers  $N_{\text{ins}}$  and  $N_{\text{del}}$ .

**Task:** Find an *alignment*, that is, a path  $P = p_1, \dots, p_k$  in  $G$  together with a mapping  $M$  from  $\{q_1, \dots, q_\ell\}$  to  $\{p_1, \dots, p_k\} \cup \{\perp\}$  such that no vertex in  $P$  has more than one preimage. The alignment must have at most  $N_{\text{ins}}$  *insertions* (that is, vertices in  $P$  that have no preimage in  $Q$ ) and at most  $N_{\text{del}}$  *deletions* (that is, vertices in  $Q$  that are mapped to  $\perp$ ). Further, the weight of the alignment must be minimal, that is, one must minimize

$$\sum_{i=1}^{\ell-1} w(p_i, p_{i+1}) + \sum_{\substack{1 \leq i \leq \ell \\ M(q_i) \neq \perp}} h(q_i, M(q_i)).$$

Note that PATHWAY QUERY is a generalization of MINIMUM-WEIGHT PATH and becomes equivalent to this problem in the special case where the match weight function  $h$  is unit and  $N_{\text{ins}} = N_{\text{del}} = 0$ .

<sup>3</sup>Literature usually states the weaker bound  $O(2^k km)$  that is obtained when representing the sets  $S$  explicitly instead of using a table.

Shlomi et al. [29] show how to solve PATHWAY QUERY by color-coding. The basic process is the same as for MINIMUM-WEIGHT PATH: the input graph is randomly colored with  $k := \ell + N_{\text{ins}}$  colors and it is hoped that the optimal path becomes colorful in the process. However, the dynamic programming step (3) from solving MINIMUM-WEIGHT PATH needs to be adapted in several ways in order to account for the more general problem formulation of PATHWAY QUERY:

- New dimensions are added to the dynamic programming table to track the number of deletions  $\theta$  and the number of matched vertices  $i$ .
- The vertex match weights are taken into account.
- New recurrences for the process of insertion and deletion are added.

Thus, a table entry  $W(v, i, \theta, S)$  contains the minimum weight of a partial alignment that matches  $q_1, \dots, q_i$ , ends at  $v$ , contains  $\theta$  deletions, and uses the colors in  $S$  for  $P$ . The precise recurrences are:

$$W(v, i, \theta, S) = \min \begin{cases} W(u, i-1, \theta, S \setminus \{\text{color}(v)\}) + w(u, v) + h(q_i, u) & \{u, v\} \in E \\ W(u, i, \theta, S \setminus \{\text{color}(v)\}) + w(u, v) & \{u, v\} \in E, |S| - i < N_{\text{ins}} \\ W(v, i-1, \theta-1, S) & \theta < N_{\text{del}}. \end{cases} \quad (4)$$

Letting  $k := \ell + N_{\text{ins}}$  and assuming that both  $N_{\text{ins}}$  and  $N_{\text{del}}$  are constants, calculating these recurrences requires  $O(2^k \cdot k \cdot m)$  time for a given instance of PATHWAY QUERY, that is, an additional factor of  $k$  compared to the recurrence for MINIMUM-WEIGHT PATH. Details of the implementation of calculating (4) are given in Section 3.3.

### 3 Speeding Up Color-Coding

This section presents several algorithmic improvements for color-coding that lead to large savings in time and memory consumption. Whereas most of the improvements in Sections 3.2 and 3.3 are of a heuristic nature, the improvement in Section 3.1 makes color-coding also more efficient in a worst-case scenario. Most of our improvements are applicable to color-coding in general and not restricted to the bioinformatics scenario that we apply them to.

#### 3.1 Speedup by Increasing the Number of Colors

Assume that we want to solve MINIMUM-WEIGHT PATH for a  $k$ -vertex path. We clearly need at least  $k$  colors to find a minimum-weight  $k$ -vertex path when using the color-coding technique. Increasing the number of used colors beyond this leads to a tradeoff: Fewer trials have to be performed to ensure the same error bound (because the path that we seek after is more likely to become colorful in a trial), yet the dynamic programming step of each single trial takes longer.

More specifically, assume that to detect a minimum-weight  $k$ -vertex path we are using the color-coding technique with  $k + x$  colors for some positive

integer  $x$ . Then the probability  $P_c$  of a  $k$ -vertex path in the input graph being colorful becomes

$$P_c = \frac{\binom{k+x}{k} \cdot k!}{(k+x)^k} = \frac{(k+x)!}{x!(k+x)^k} = \prod_{i=1}^k \frac{i+x}{k+x} \quad (5)$$

because there are  $(k+x)^k$  ways to color  $k$  vertices with  $k+x$  colors and of these ways exactly  $\binom{k+x}{k} \cdot k!$  use mutually different colors. The overall running time  $t_{\mathcal{A}}$  of the color-coding algorithm to ensure an error probability of at most  $\varepsilon$  is a product of two factors, namely the running time of a single trial and the number of trials  $t(\varepsilon)$  to perform. As discussed in Section 2.1, the worst-case running time for each trial when solving MINIMUM-WEIGHT PATH is  $O(2^{k+x} \cdot m)$ , and we obtain

$$t_{\mathcal{A}} \leq t(\varepsilon) \cdot O(2^{k+x} \cdot m) = \left\lceil \frac{\ln \varepsilon}{\ln(1 - P_c)} \right\rceil \cdot O(2^{k+x} \cdot m). \quad (6)$$

Obviously, the value of  $x$  should be chosen such that the right-hand side of (6) is minimized. Nearly all works we are aware of use  $x = 0$  for their running time analysis, which yields

$$t_{\mathcal{A}} = O(|\ln \varepsilon| \cdot e^k \cdot 2^k m) = O(|\ln \varepsilon| \cdot 5.44^k m).$$

While this choice can be argued for with respect to memory requirements for a trial (after all, these are a major bottleneck for dynamic programming algorithms), it is not optimal concerning  $t_{\mathcal{A}}$ :

**Theorem 1.** *The worst-case running time of color-coding for MINIMUM-WEIGHT PATH with  $1.3k$  colors and error probability  $\varepsilon$  is  $O(|\ln \varepsilon| \cdot 4.32^k m)$ .*

*Proof.* To estimate the factorials in Equation (5), we use the double inequality

$$\sqrt{2\pi n}^{n+1/2} \cdot \exp(-n + 1/(12n + 1)) < n! < \sqrt{2\pi n}^{n+1/2} \cdot \exp(-n + 1/(12n))$$

derived from Stirling's approximation. This yields

$$\begin{aligned} P_c &\geq \frac{\sqrt{2\pi}(k+x)^{k+x+1/2} \cdot \exp\left(-k-x + \frac{1}{12k+12x+1}\right)}{\sqrt{2\pi}x^{x+1/2} \cdot \exp\left(-x + \frac{1}{12x}\right)} \cdot (k+x)^{-k} \\ &= \left(\frac{k}{x} + 1\right)^{x+1/2} \cdot \exp\left(-k - \frac{1}{12x} + \frac{1}{12k+12x+1}\right). \end{aligned}$$

Setting  $x := 0.3k$  and using the inequality  $\ln(1 - P_c) < -P_c$  (which is valid because the probability  $P_c$  satisfies  $0 < P_c < 1$ ) we obtain

$$t_{\mathcal{A}} \leq \left\lceil \frac{\ln \varepsilon}{\ln(1 - P_c)} \right\rceil \cdot O(2^{k+x} \cdot m) < \left(\frac{\ln \varepsilon}{-P_c} + 1\right) \cdot O(2^{k+x} \cdot m)$$

where

$$\frac{1}{P_c} < 4.33^{-0.3k-1/2} \cdot \exp\left(k + \frac{1}{12x}\right) = O\left(\frac{e^k}{1.552^k}\right) = O(1.752^k)$$



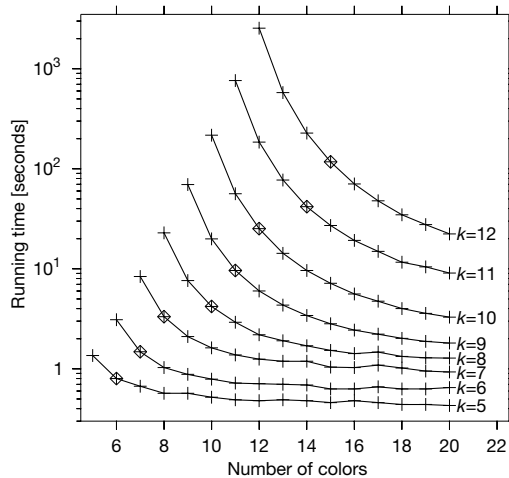


Figure 2: Running times for finding the 20 minimum-weight paths of different lengths  $k$  in the yeast protein interaction network of Scott et al. [27]. Increasing the number of colors yields a speedup of up to two orders of magnitude. No lower bound function (Section 3.2.1) was used; the highlighted point of each curve marks the optimal choice when assuming worst-case trial running time.

which finally yields

$$t_{\mathcal{A}} \leq |\ln \varepsilon| \cdot (O(1.752^k) + 1) \cdot O(2^{1.3k} m) = O(|\ln \varepsilon| \cdot 4.32^k m)$$

as claimed by the theorem.  $\square$

Increasing the number of colors has been independently examined by Deshpande et al. [7]. They also suggest using  $1.3k$  colors; however, their analysis only derives an  $O(4.5^k)$  bound for the exponential part of the running time.

Analogously to this theorem for MINIMUM-WEIGHT PATH, the worst-case running time that is required to solve PATHWAY QUERY for a  $(k - N_{\text{ins}})$ -vertex query path can also be significantly improved by setting the number of colors close to  $1.3k$ .

Unfortunately, it seems difficult to algebraically solve for the value of  $x$  that minimizes the right-hand side of (6). Numerical evaluation, however, suggests that setting  $x$  close to  $0.3k$  is an optimal choice to minimize the worst-case running time when solving MINIMUM-WEIGHT PATH or PATHWAY QUERY; concrete evaluations of (6) can be used to determine whether to round the number  $1.3k$  up or down.

For a practical implementation, while we could fix the number of colors at the worst-case optimum  $1.3k$ , it is most likely beneficial to use even more colors because various algorithmic tweaks and the underlying graph structure can keep the running time of a single trial significantly below the worst-case estimate. This in turn causes the increase in running time per trial by choosing more colors to be even more overcompensated by a decrease in the total number of trials needed, as is demonstrated in Figure 2 for the case of MINIMUM-WEIGHT PATH. In fact, for a small path size of 8–10 we can choose the number of colors to be the maximum our implementation allows (that is, 31), and get by

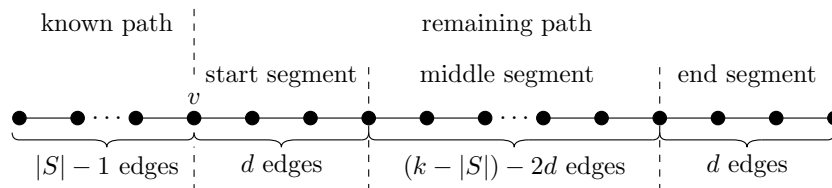


Figure 3: Calculation of a lower weight bound for a length- $k$  path when already  $|S|$  of the vertices are given.

with a very small number of trials ( $\approx 15-30$ ). Based on such observations, our implementation uses an adaptive approach to the number of colors, starting with the maximum of 31 and decreasing this in case a trial runs out of memory.

## 3.2 Improvements for Minimum-Weight Path

As is standard practice in dynamic programming, we do not allocate memory for the complete table and evaluate (3) recursively, but rather work layer-wise starting from an initial set of entries. More precisely, we seed the table with all entries corresponding to a one-vertex path ending at a vertex  $v$ , and then repeatedly generate the next layer by extending all entries by one vertex, after which the old layer can be discarded. This saves memory compared to holding a complete table and allows the improvements of Section 3.2.1. However, it requires extra memory to carry along enough information in each entry to reconstruct a solution; we show in Section 3.2.2 how to do this efficiently.

### 3.2.1 Lower Bounds and Cache Preheating

In a color-coding trial for solving MINIMUM-WEIGHT PATH, every vertex carries entries for up to  $2^{k+x}$  color sets, each representing a partial colorful path with a certain weight. Because each entry may get expanded to an exponentially large collection of new entries, pruning even a small fraction of them can lead to a significant speedup. The pruning strategy that we employ makes use of the fact that we are only looking for a fixed number of minimum-weight paths. As soon as we have found this number of candidates, we can always remove entries where the weight of the corresponding partial path is certain to exceed the weight of the worst known path in the current collection of paths when completed.

Consider an entry  $W(v, S)$  corresponding to some partial path. To obtain a length- $k$  path, we need to append another  $k - |S|$  edges. Thus, a trivial lower bound for the total weight of a length- $k$  path expanded from this entry would be  $W(v, S) + (k - |S|)w_{\min}$ , where  $w_{\min}$  is the minimum weight of any edge in the graph. We improve upon this simple bound by dividing the remaining path length not into single edges, but rather—as illustrated in Figure 3—into three segments, calculating a lower bound separately for each of them and summing up these bounds.

The lower bound calculation is prepared in a preprocessing phase on the uncolored graph. There, we determine by dynamic programming for every vertex  $v$  and a range of lengths  $1 \leq i \leq d$  the minimum weight  $w_{\min}(v, i)$  of a path of  $i$  edges that starts at the vertex  $v$ . If the paths are restricted to end only in

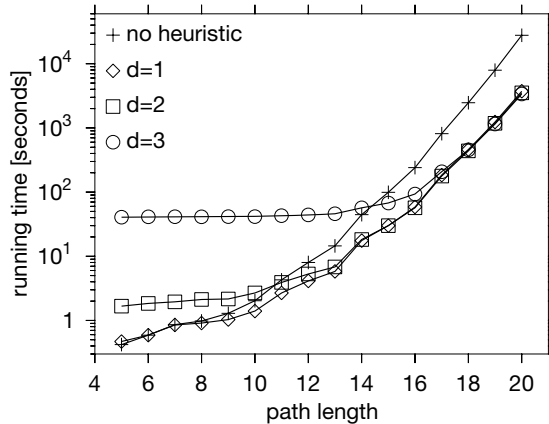


Figure 4: Running time comparison with heuristic evaluation functions for different values of  $d$  (seeking the 20 lowest-weight paths in the yeast network of Scott et al. [27] that differ in at least 30% of participating vertices).

a certain set of “goal vertices” (for example, when the signaling pathway candidates are restricted to end in a transcription factor), we additionally determine the minimum weight  $g_{\min}(v, i)$  of a path of  $i$  edges starting in  $v$  and ending in a goal vertex. After this preprocessing, to get a lower bound for the minimum weight of a path with  $\ell < k$  edges starting in  $v$  and ending in a goal vertex, we can directly look up  $g_{\min}(v, \ell)$  whenever  $\ell \leq d$ . Since calculating  $w_{\min}$  and  $g_{\min}$  takes  $O(n^d)$  time and space in the worst case, we generally have to choose  $d < k$ . We can still get a lower bound if  $\ell > d$ . For example, if  $\ell = c \cdot d$  for some  $c \geq 2$ , we calculate

$$w_{\min}(v, d) + \frac{\ell - 2d}{d} \cdot \min_{u \in V} w_{\min}(u, d) + \min_{u \in V} g_{\min}(u, d). \quad (7)$$

If  $d$  does not evenly divide  $\ell$ , we add a suitable correction term for the middle segment. If  $\ell < 2d$ , we additionally try all ways of dividing the bound between  $w_{\min}(v, \ell_1)$  and  $\min_{u \in V} g_{\min}(u, \ell_2)$  for  $\ell_1 + \ell_2 = \ell$ .

Clearly, there is a trade-off between the time invested in the preprocessing (depending on  $d$ ) and the time saved in the main algorithm. For the yeast network of Scott et al. [27], setting  $d = 2$  seems to be a good choice with an additional second of preprocessing time. For  $d = 3$ , the preprocessing time increases to 38 seconds, an amount of time that is only recovered when searching for paths of length at least 19 (see Figure 4).

Using lower bounds is only effective once we have already found as many paths as we are looking for. Therefore, it is important to quickly find some low-weight paths early in the process. We achieve this acquisition of lower bounds by prepending a number of trials with a thinned-out graph, that is, for some  $0 < t < 1$ , we consider a graph that contains only the  $t|E|$  lightest edges of the input graph.<sup>4</sup> Trials for a certain value of  $t$  are repeated with different random colorings until the lower bound does not improve any more. By default,  $t$  is increased in steps of  $1/10$ ; should we run out of memory, this step size is

<sup>4</sup>In particular in database applications, similar techniques are known as “preheating the cache.”

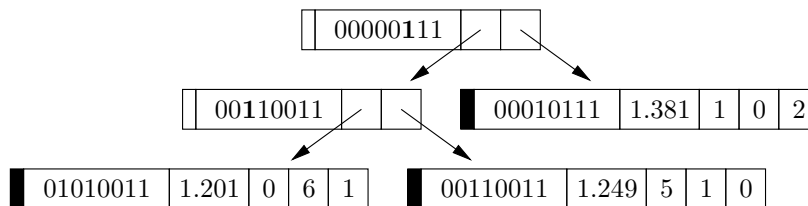


Figure 5: Representation of color sets at some node  $u$  with color 4. Inner nodes contain the marker bit (leftmost box), the common suffix (branch bit in bold), and two pointers to children. Leaf nodes contain the marker bit, the color set, the weight, and the vertices of the path.

halved. This allows to successfully complete trials in the thinned out graphs, making trials feasible on the original graphs by providing them with powerful bounds for pruning.

### 3.2.2 Efficient Storage of Color Sets

Since one is not only interested in the weight of a solution, but in the vertices (that is, proteins) that it consists of, it is common to not only store the weight of a partial colorful path in Equation (3) but also a concrete sequence of vertices that realizes this weight. This accounts for the bulk of the memory requirement of a color-coding implementation, because  $k \lceil \log |V| \rceil$  bits per stored path are required. We propose to save memory here by noting that it suffices to store only the order in which the colors appear on a path: after completing a color set at some vertex  $u$ , the path can be recovered by running a shortest path algorithm (e.g., Dijkstra’s algorithm [6]) for the source vertex  $u$  while allowing it to only travel edges that match the color order. This reduces the memory cost per entry to  $k \lceil \log k \rceil$  bits, which, for our application, amounts to a saving factor of about 2–4. Because of the resulting increase in computer cache effectiveness, this usually also leads to a speedup except when either short path lengths are used (where memory is not an issue anyway) or when many solution paths are found and have to be reconstructed.

### 3.2.3 Data Structure

We represent color sets as a bit string of fixed length. This allows to use a Patricia tree, that is, a compact representation of a radix tree [6] where any node which is an only child is merged with its parent (see Fig. 5 for an example).

Inner nodes of the tree contain a color set. The highest 1-bit of this color set is the *branch bit*. For all leaf nodes below an inner node, the bits below the branch bits are equal to the corresponding bits in this inner node. The left subtree contains color sets where the branch bit is 0, and the right subtree those where it is 1. We additionally need a marker bit to distinguish inner nodes and leaves. A leaf stores the complete color set, the weight of the corresponding partial path, and the colors in the order of occurrence on the path (except for the last one, which is redundant).

The height of the tree is naturally limited by the number of colors, so no balancing is needed. This data structure allows for very quick insertions and

iterations with a moderate memory overhead of, e.g., 12 bytes per color set on a 32-bit system. Memory allocation time and space overhead is minimized by using a memory pool.

The data structure has the additional advantage that it is possible to quickly skip over color sets containing a certain color by noting that the corresponding bit is set in the suffix at some inner node.

### 3.3 Specific Improvements for Pathway Query

If we wish to exploit the heuristic cutoffs and the resulting sparseness of the dynamic programming table when solving an instance of PATHWAY QUERY, we cannot use recurrence (4) from Section 2.2 directly; rather, the entries must be built up inductively. For this purpose, the dimension of  $i$  is represented implicitly by working layerwise from  $i = 1$  to  $l$  and accessing only the previous layer. The dimensions of  $v$  and  $\theta$  are represented explicitly as an array, while the values of  $S$  are covered by one Patricia tree per combination of  $v$  and  $\theta$ . The calculation of layer  $i + 1$  from layer  $i$  starts by expanding each entry  $W(v, i, \theta, S)$  with weight  $w_i$  in layer  $i$  by possible matchings or deletions of a single vertex:

$$\begin{aligned} &\text{if } \{u, v\} \in E \wedge \text{color}(u) \notin S : \\ &\quad W(u, i + 1, \theta, S \cup \{\text{color}(u)\}) \leftarrow w_i + w(u, v) + h(q_{i+1}, u) \\ &\text{if } \theta < N_{\text{del}} : \\ &\quad W(v, i + 1, \theta + 1, S) \leftarrow w_i. \end{aligned}$$

The update is skipped if an entry with lower weight is already present. Since insertions do not increment  $i$ , we then have to update the table for layer  $i + 1$  by entries with an arbitrary additional number of insertions. Each entry  $W(v, i + 1, \theta, S)$  with weight  $w_{i+1}$  (including those generated in this process) is expanded:

$$\begin{aligned} &\text{if } \{u, v\} \in E \wedge \text{color}(u) \notin S \wedge |S| - (i + 1) < N_{\text{ins}} : \\ &\quad W(u, i + 1, \theta, S \cup \{\text{color}(u)\}) \leftarrow w_{i+1} + w(u, v). \end{aligned}$$

Fortunately, the Patricia tree structure allows to do the insertion updates in a straightforward way: a single in-order walk of the tree will do, since any newly inserted entry contains one more color and will therefore be encountered later in the walk.

For initialization, all possibilities to delete  $0, \dots, N_{\text{del}}$  query vertices and then match the next with  $v \in V$  have to be entered into the table. Alignments starting with insertions are not considered, since they cannot be optimal.

A deletion is not allowed after an insertion, since alignments that only differ in the order of deletions and insertions between two actual matches will have the same score, and are not reasonable to differentiate for our application.

In order to use the heuristic lower bounds that we use for MINIMUM-WEIGHT PATH (Section 3.2.1) also for PATHWAY QUERY, these have to be slightly adapted: First, possible deletions have to be taken into account when considering the minimum additional edge that must be incurred. Second, we improve the heuristic by adding the minimum match weight of all query vertices that are yet to be matched (also considering possible deletions).

Table 1: Basic properties of the network instances YEAST (Scott et al. [27]) and DROSOPHILA (Giot et al. [12]). The *clustering coefficient* is the probability that  $\{u, v\} \in E$  for  $u, v, x \in V$  with  $\{u, x\} \in E$  and  $\{x, v\} \in E$ .

	vertices	edges	clustering coefficient	avg. degree	max. degree
YEAST	4 389	14 319	0.067	6.5	237
DROSOPHILA	7 009	20 440	0.030	5.8	175

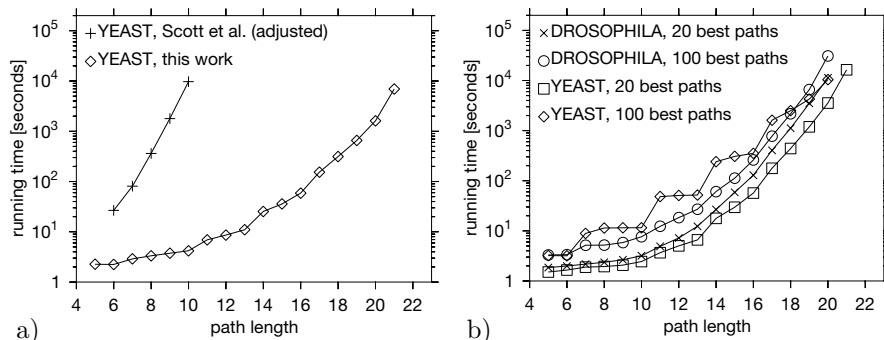


Figure 6: **(a)** Running times for YEAST as reported by Scott et al. [27] (adjusted for speed difference of the testing machines) and measured with our implementation. In both cases, paths must start at a membrane protein, and end at a transcription factor. Memory requirements were, e. g., 3 MB for  $k = 10$  and 242 MB for  $k = 21$ .  
**(b)** Comparison of the running times of our implementation when applied to YEAST and DROSOPHILA for various path lengths, seeking after either 20 or 100 minimum-weight paths that mutually differ in at least 30% of their vertices. There were no restrictions as to the sets of start and end vertices.

## 4 Experimental Results

We have implemented the color-coding technique with the improvements described in the last section. The source code of the program is available from <http://theinf1.informatik.uni-jena.de/colorcoding/>; it is written in the C++ programming language and consists of approximately 1700 lines of code. The testing machine is an AMD Athlon 64 3400+ with 2.4 GHz, 512 KB cache, and 1 GB main memory running under the Debian GNU/Linux 3.1 operating system. The program was compiled with the GNU g++ 4.2 compiler using the options “-O3 -march=athlon”.

### 4.1 Minimum-Weight Path

The real-world network instances used for speed measurements were the *Saccharomyces cerevisiae* interaction network used by Scott et al. [27] and the *Drosophila melanogaster* interaction network described by Giot et al. [12]. Some properties of these networks, which we will refer to as YEAST and DROSOPHILA, are summarized in Table 1.

To explore the sensitivity of the running time to various graph parameters

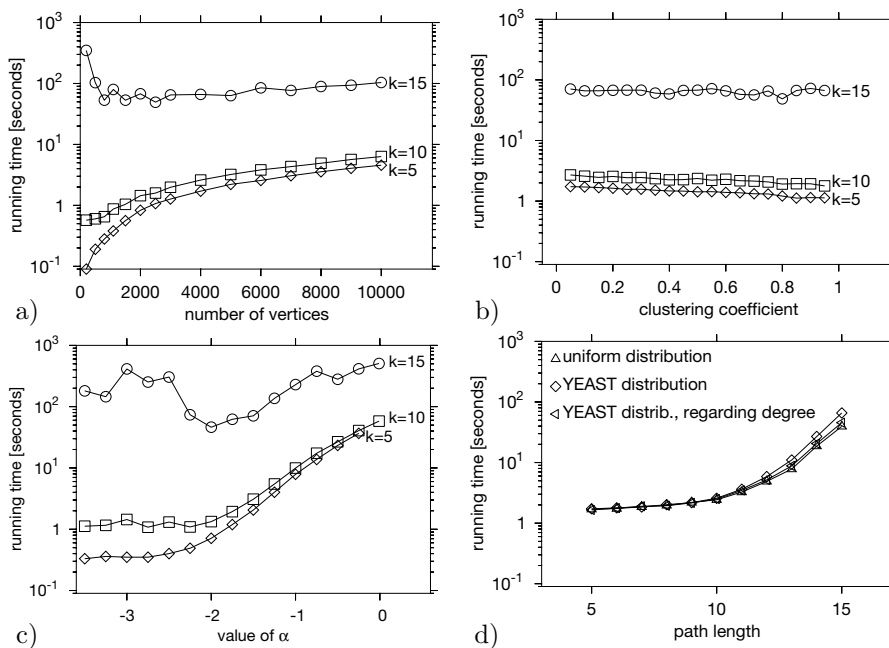


Figure 7: Running time for our color-coding implementation on random networks, seeking after 20 minimum-weight paths. Unless a parameter is the variable of a measurement, the following default values are used (we have empirically found them to result in networks that are quite similar to YEAST): 4000 vertices; degree distribution is a power law with exponential cutoff, that is, the fraction  $p_k$  of vertices with degree  $k$  satisfies  $p_k \sim k^\alpha \cdot e^{-k/1.3} \cdot e^{-45/k}$ ; the default value for  $\alpha$  is  $-1.6$ ; edge weights are distributed as in YEAST; the clustering coefficient is 0.1. The data shown reports the average running time over five runs each. **(a)** Dependency on the number of vertices. **(b)** Dependency on the clustering coefficient. **(c)** Dependency on the parameter  $\alpha$  of the power law distribution. **(d)** Dependency on the distribution of edge weights for three different distributions: A uniform  $[0, 1]$ -distribution, the distribution of YEAST, and the distribution of YEAST under consideration of vertex degree.

(namely, the number of vertices, the clustering coefficient, the degree distribution, and the distribution of edge weights), the implementation was also run on a testbed of random graph instances that were generated with the algorithm described by Volz [31]. The results of all experiments and details as to the experimental setting are given in Figures 6 and 7.

Note that Scott et al. [27] obtained their running times on a dual 3.0 GHz Intel Xeon processor with 4 GB main memory. To make their running times comparable with ours, Figure 6 does not report their original times here, but divides them by 1.2 (which is a very conservative estimate in favor of Scott et al. [27] that most likely overestimates the speed of our machine).

Only a few of the most difficult instances hit the predefined 768 MB memory limit and required additional preheating cycles.

**Discussion.** Compared to the (machine-speed adjusted) running times from Scott et al. [27], our implementation is faster by a factor of 10 to 2000 on YEAST (see Figure 6a). Scott et al. discuss findings for paths up to a length of 10 which they were able to find in about three hours. These can be found within seconds by our implementation, allowing for interactive queries and displays. The range of feasible path lengths is more than doubled.

Figure 6b shows that the running times for both YEAST and DROSOPHILA are roughly equal. The only exception is the search for the best 100 paths within YEAST which not only takes unexpectedly long but also displays step-like structures. Most likely, these two phenomena can be attributed to the fact that certain path lengths allow for much fewer well-scoring paths than others in YEAST, causing the lower-bound heuristic to be less effective. Figure 6b also demonstrates that a major factor in the running time is actually the number of paths that is sought after. This is because a larger number of paths worsens the lower bound of the heuristic which cannot cut off as many partial solutions and maintaining the list of paths and checking the “at least 30% of vertices must differ” criterion becomes more involved.

Figures 7a, 7b, and 7d show that the running time of the color-coding algorithm appears to be insensitive to the size of the graph (increasing linearly with increasing graph size) as well as the clustering coefficient and the distribution of edge weights. The somewhat unexpectedly high running times for graphs with less than 500 vertices in Figure 7a are explained by the fact that the number of length-10 and length-15 paths in these networks is very low, causing the heuristic lower bounds to be rather ineffective (this also explains why the effect is worse for  $k = 15$  than it is for  $k = 10$ ).

Figure 7c shows that the algorithm is generally faster when the vertex degrees are unevenly distributed. This comes as no surprise because for low-degree vertices, fewer color sets have to be maintained in general and the heuristic lower bounds are often better. For  $k = 15$ , two points in the curve require further explanation: First, the drop-off in running time for  $\alpha < -3$  is explained by the random graph “disintegrating” into small components. Second, the increased running time for  $-3 \leq \alpha \leq -2$  is most likely due to a decrease in the total number of length-15 paths as compared to larger values of  $\alpha$ .

## 4.2 Pathway Queries

**Method and Results.** To evaluate the performance of our improved color-coding for PATHWAY QUERY, we conducted experiments similar to those of Shlomi et al. [29]: The data basis are their protein interaction networks of *S. cerevisiae* and *D. melanogaster* as well as a matrix of protein similarity scores described in [29]. To obtain query paths of various lengths  $\ell = 4, \dots, 9$ , we determined the set of the 100 minimum-weight paths of each length in the *S. cerevisiae* network, using the constraint that no two paths of the same length are allowed to overlap by more than 20% of their vertices. We then determined the best match for each of these query paths in the *D. melanogaster* network, allowing up to 3 insertions and up to 3 deletions. Table 2 shows the obtained running times for these queries.

**Discussion.** Shlomi et al. [29] were able to answer path queries for paths of length 7 on average within 8 minutes on a Pentium 4 with 1.7GHz. On



Path length	Avg. Time [s]	Max. Time [s]	Successful Queries
4	2.24	2.57	98%
5	2.33	3.61	93%
6	3.00	23.02	81%
7	4.52	93.32	52%
8	7.49	225.61	31%
9	11.38	245.78	13%

Table 2: Running times for path queries in the *D. melanogaster* network. The rightmost column gives the percentage of query paths for which a matching path in the *D. melanogaster* network could be found.

average, our implementation solves these within a few seconds and is able to answer queries even for length  $\ell = 9$  within reasonable time.<sup>5</sup> In general, we found the algorithm to be significantly slowed down if the network contains no path that matches the query, which explains the large deviations between the average and maximum required time as the query path length increases.

## 5 Conclusion

We have given various algorithmic improvements for the color-coding technique. In the applications scenario of detecting signaling pathway candidates in protein interaction networks, these enable the fast exploration of small pathway candidates as well as finding much larger structures than previously possible. To some extent, our work also closes the gap in practical experience with color-coding.

There remain two interesting open questions for future research: First, the recently devised algorithms that are based on random partitions of a graph into its subgraphs [3, 17] have a better worst-case bound than color-coding. It is not clear whether this better bound carries over in practice because the random partition approach has the exponential part of the running time “hard-coded” into recursive function calls whereas color-coding is more dependent on the input graph structure, which usually is favorable in this respect. As a second interesting field for future research, one should further look into ways to derandomize color-coding algorithms so efficiently that the resulting deterministic algorithms are of practical use (recently, Chen et al. [5] made substantial progress in this direction).

## Acknowledgments

The authors are grateful to Jacob Scott (Cambridge, MA) for providing them with the yeast interaction network discussed in [27] and to Tomer Shlomi (Tel Aviv) for providing them with the data from [29]; Roded Sharan (Tel Aviv) helped us in establishing both contacts. Hannes Moser and Rolf Niedermeier (Jena) made some very helpful comments in various discussions of this work.

<sup>5</sup>Longer queries are of doubtful relevance on this data set, since matches are rarely found.

## References

- [1] N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.
- [2] H. L. Bodlaender. On linear time minor tests with depth-first search. *Journal of Algorithms*, 14(1), 1993.
- [3] L. Cai, S. M. Chan, and S. O. Chan. Random separation: a new method for solving fixed-cardinality optimization problems. In *Proc. 2nd Int. Workshop on Parameterized and Exact Computation (IWPEC'06)*, volume 4169 of *LNCS*, pages 239–250. Springer, 2006.
- [4] P. Cappanera and M. G. Scutellà. Balanced paths in telecommunication networks: some computational results. In *Proc. 3rd Int. Network Optimization Conference (INOC'07)*, 2007.
- [5] J. Chen, S. Lu, S.-H. Sze, and F. Zhang. Improved algorithms for path, matching, and packing problems. In *Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA'07)*, pages 298–307. ACM-SIAM, 2007.
- [6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.
- [7] P. Deshpande, R. Barzilay, and D. R. Karger. Randomized decoding for selection-and-ordering problems. In *Proc. Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technologies (NAACL HLT'07)*, pages 444–451. Association for Computational Linguistics, 2007.
- [8] B. Dost, T. Shlomi, N. Gupta, E. Ruppín, V. Bafna, and R. Sharan. QNet: A tool for querying protein interaction networks. In *Proc. 11th Annual Int. Conference on Research in Computational Molecular Biology (RECOMB'07)*, volume 4453 of *Lecture Notes in Bioinformatics*, pages 1–15. Springer, 2007.
- [9] T. Feder and R. Motwani. Finding large cycles in hamiltonian graphs. In *Proc. 16th ACM-SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 166–175. SIAM, 2005.
- [10] M. R. Fellows, C. Knauer, N. Nishimura, P. Ragde, F. A. Rosamond, U. Stege, D. M. Thilikos, and S. Whitesides. Faster fixed-parameter tractable algorithms for matching and packing problems. In *Proc. 12th European Symposium on Algorithms (ESA'04)*, volume 3221 of *LNCS*, pages 311–322. Springer, 2004.
- [11] H. N. Gabow. Finding paths and cycles of superpolylogarithmic length. In *Proc. 36th ACM Symposium on Theory of Computing (STOC'04)*, pages 407–416. ACM, 2004.
- [12] L. Giot, J. S. Bader, C. Brouwer, et al. A protein interaction map of *Drosophila melanogaster*. *Science*, 302(5651):1727–1736, 2003.

- [13] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.
- [14] F. Hüffner, S. Wernicke, and T. Zichner. FASPAD: fast signaling pathway detection. *Bioinformatics*, 2007. To appear.
- [15] T. Ideker, V. Thorsson, J. A. Ranish, et al. Integrated genomic and proteomic analyses of a systematically perturbed metabolic network. *Science*, 292(5518):929–934, 2001.
- [16] D. R. Karger, R. Motwani, and G. D. S. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997.
- [17] J. Kneis, D. Mölle, S. Richter, and P. Rossmanith. Divide-and-color. In *Proc. 32nd Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG'06)*, volume 4271 of *LNCS*, pages 58–67. Springer, 2006.
- [18] I. Koutis. A faster parameterized algorithm for Set Packing. *Information Processing Letters*, 94(1):7–9, 2005.
- [19] L. Mathieson, E. Prieto, and P. Shaw. Packing edge disjoint triangles: A parameterized view. In *Proc. 1st Int. Workshop on Parameterized and Exact Computation (IWPEC'04)*, volume 3162 of *LNCS*, pages 127–137. Springer, 2004.
- [20] I. Mayrose, T. Shlomi, N. D. Rubinstein, J. M. Gershoni, E. Ruppin, R. Sharan, and T. Pupko. Epitope mapping using combinatorial phage-display libraries: a graph-based algorithm. *Nucleic Acids Research*, 35(1):69–78, 2007.
- [21] B. Monien. How to find long paths efficiently. *Annals of Discrete Mathematics*, 25:239–254, 1985.
- [22] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
- [23] J. Plehn and B. Voigt. Finding minimally weighted subgraphs. In *Proc. 16th Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG'90)*, volume 484 of *LNCS*, pages 18–29. Springer, 1990.
- [24] E. Prieto and C. Sloper. Looking at the stars. *Theoretical Computer Science*, 351(3):437–445, 2006.
- [25] D. Raymann. Implementation of Alon-Yuster-Zwick's color-coding algorithm. Diplomarbeit, Institute of Theoretical Computer Science, ETH Zürich, Switzerland, 2004.
- [26] J. P. Schmidt and A. Siegel. The spatial complexity of oblivious  $k$ -probe hash functions. *SIAM Journal on Computing*, 19(5):775–786, 1990.
- [27] J. Scott, T. Ideker, R. M. Karp, and R. Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. *Journal of Computational Biology*, 13(2):133–144, 2006.

- [28] R. Sharan and T. Ideker. Modeling cellular machinery through biological network comparison. *Nature Biotechnology*, 24(4):427–433, 2006.
- [29] T. Shlomi, D. Segal, E. Ruppin, and R. Sharan. QPath: a method for querying pathways in a protein–protein interaction network. *BMC Bioinformatics*, 7:199, 2006.
- [30] M. Steffen, A. Petti, J. Aach, P. D’haeseleer, and G. Church. Automated modelling of signal transduction networks. *BMC Bioinformatics*, 3:34, 2002.
- [31] E. Volz. Random networks with tunable degree distribution and clustering. *Physical Review E*, 70:056115, 2004.