Proc. 5th APBC-07, Vol. 5 Bioinformatics and CB, pp. 277-286, 2007

ALGORITHM ENGINEERING FOR COLOR-CODING TO FACILITATE SIGNALING PATHWAY DETECTION

FALK HÜFFNER, SEBASTIAN WERNICKE, AND THOMAS ZICHNER

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

To identify linear signaling pathways, Scott et al. [Recomb, 2005] recently proposed to extract paths with high interaction probabilities from protein interaction networks. They used an algorithmic technique known as color-coding to solve this NP-hard problem; their implementation is capable of finding biologically meaningful pathways of length up to 10 proteins within hours. In this work, we give various novel algorithmic improvements for color-coding, both from a worst-case perspective as well as under practical considerations. Experiments on the interaction networks of yeast and fruit fly as well as a testbed of structurally comparable random networks demonstrate a speedup of the algorithm by orders of magnitude. This allows more complex and larger structures to be identified in reasonable time; finding paths of length up to 13 proteins can even be done in seconds and thus allows for an interactive exploration and evaluation of pathway candidates.

1. Introduction

Motivation. Accompanying the availability of genome-scale protein–protein interaction data, various approaches have been proposed to datamine the corresponding networks[a] for biologically meaningful substructures such as dense groups of interacting proteins[3,6,9] and loop structures[2]. A special role—with respect to both biological meaning as well as algorithmic tractability—is played by the most simple structures, that is, *linear* pathways. These are easy to understand and analyze and, as demonstrated by Ideker et al.[8] for the yeast galactose metabolism, they can serve as a seed structure for experimental investigation of more complex mechanisms. Initiated by Steffen et al.[12], the automated discovery of linear pathways in protein interaction networks is hence a promising undertaking.

Unfortunately, finding linear pathways—that is, paths in a graph where each vertex occurs at most once—is an NP-hard problem[5]. However, a randomized algorithmic technique called "color-coding"[1] is known to solve this problem efficiently for small path lengths. (The details of color-coding are explained in Section 2). Consequently, Scott et al.[10] recently proposed to employ color-coding to datamine protein interaction networks for signaling pathways. They were able to demonstrate that the color-coding approach is indeed capable of identifying biologically meaningful pathways. Their implementation is limited, however, to path lengths of around 10 vertices and, moreover, it requires some hours of

[a] We use the term "network" for fields outside mathematics and computer science and "graph" for discussing algorithmic aspects.

runtime for these path lengths. In this work, we give various novel improvements for color-coding, both from a worst-case perspective as well as under practical considerations. This allows us to find pathways consisting of more than 20 vertices in some hours and the task of finding pathways of length 10 can be accomplished in a few seconds.

Structure of this work. The color-coding technique is explained in Section 2. Our algorithmic improvements and data structures are discussed in Section 3. The improved color-coding algorithm has been implemented in C++. Section 4 discusses experimental results that were obtained by using our implementation on the *S. cerevisiae* (yeast) interaction network of Scott et al.[10], the *D. melanogaster* (fruit fly) interaction network of Giot et al.[7], and random networks that are structurally similar to protein interaction networks. Our experiments demonstrate that the algorithmic improvements proposed in this work facilitate the detection of larger, more complex pathway candidates and opens the possibility for interactive exploration of smaller structures.

The source code of our color-coding implementation can be downloaded as free software from `http://theinf1.informatik.uni-jena.de/colorcoding/`.

2. The Color-Coding Technique

We model protein interaction networks as undirected graphs where each vertex is a protein and each edge is weighted by the negative logarithm of the interaction probability for the two proteins it connects. Following Scott et al.[10], we formalize the problem of pathway candidate detection to a NP-hard problem called MINIMUM-WEIGHT PATH.

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.

Color-Coding. Alon et al.[1] proposed a technique called *color-coding* to solve MINIMUM-WEIGHT PATH. The idea is to randomly color the vertices in the input graph with k colors[b] and then search for *colorful* paths, that is, paths where no color occurs twice. Given a fixed coloring of vertices, finding the minimum-weight colorful path is 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

$$W(v, S) = \min_{e=\{u,v\}\in E}\Big(W(u, S \setminus \{\text{color}(v)\}) + w(e)\Big). \tag{1}$$

[b] Section 3.1 shows that using only k colors is suboptimal from a runtime perspective.

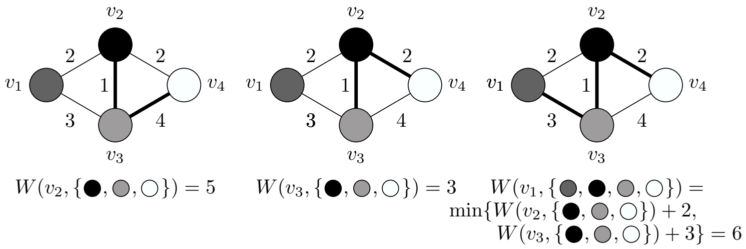
v_2
2
2
v_1
1
v_4
3
4
v_3
$W(v_2, \{●, ●, ○\}) = 5$
$W(v_3, \{●, ●, ○\}) = 3$
$W(v_1, \{●, ●, ●, ○\}) =$
$\min\{W(v_2, \{●, ●, ○\}) + 2,$
$W(v_3, \{●, ●, ○\}) + 3\} = 6$

Figure 1. Example for solving Minimum-Weight Path using the color-coding technique. Using Equation (1) a new table entry (right) is calculated using two already known entries (left and middle).

as exemplified in Figure 1.

It is easy to verify that the dynamic programming takes $O(2^k m)$ time.[c] Whenever the minimum-weight length-k path in the input graph is colored with k colors (i.e., every vertex has a different color), then it is found. The problem, 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. While the result is only optimal with a (user-specifiable) probability, setting the error probability ε to say, 0.1%, is likely to be acceptable in practice (even more so because only the logarithm of the error probability goes into the overall runtime and hence, very low error probabilities are efficient to achieve).

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[10]). Also, recently it has been demonstrated that pathway queries to a network, that is, the task of finding a pathway in a network that is as similar as possible to a query pathway, can be handled with color-coding[11].

Unless otherwise noted, we use the following variant of Minimum-Weight Path that matches the experiments by Scott et al.[10]: 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 small modifications of the global minimum-weight path).

3. Improving the Efficiency of Color-Coding

This section presents several algorithmic improvements for color-coding that lead to large savings in time and memory consumption. Whereas the improvement in Section 3.2 is of heuristic nature, the improvements in Sections 3.1 and 3.3 make color-coding also more efficient in a worst-case scenario. Note that these improvements are generally applicable to color-coding and not restricted to the protein interaction network scenario.

[c] Literature usually states the weaker bound $O(2^k km)$ that is obtained when representing the sets S explicitly instead of using a table.

3.1. *Speedup by Increasing the Number of Colors*

Clearly, we need at least k colors when trying to find a length-k path 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, yet each trial takes longer. More specifically, assume that in order to detect a path of length k we are using the color-coding technique with $k+x$ colors for some positive integer x. Then the probability P_c of a path in the input graph being colorful becomes

$$P_c = \frac{\binom{k+x}{k} \cdot k!}{(k+x)^k} = \frac{(k+x)!}{x!(k+x)^k} = \prod_{i=1}^{k} \frac{i+x}{k+x} \tag{2}$$

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

$$t_{\mathcal{A}} \leq t(\varepsilon) \cdot O(2^{k+x} \cdot m) = \left\lceil \frac{\ln \varepsilon}{\ln(1-P_c)} \right\rceil \cdot O(2^{k+x} \cdot m). \tag{3}$$

We should choose x such that the right-hand side of (3) is minimized. All works we are aware of use $x = 0$ for the 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 3.1. *The worst-case runtime of color-coding for* MINIMUM-WEIGHT PATH *with $1.3k$ colors and error probability ε is $O(|\ln \varepsilon| \cdot 4.32^k m)$.*

Proof. To estimate the factorials in Equation (2), 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

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

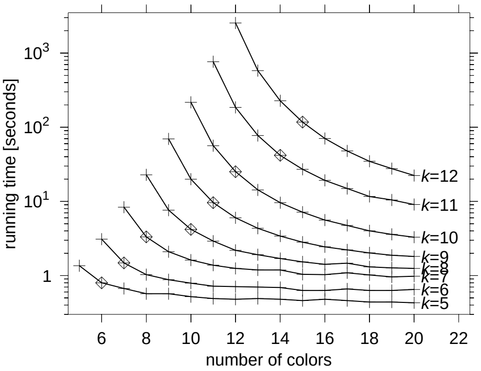

Figure 2. Runtimes for finding the 20 minimum-weight paths of different lengths in the yeast protein interaction network of Scott et al.[10]. No lower bound function (Section 3.2) was used. The highlighted point of each curve marks the optimal choice when assuming worst-case trial runtime.

where

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

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. □

Numerical evaluation suggests that setting x close to $0.3k$ (whether to round up or down should be determined numerically for a concrete k) as done in the theorem is actually an optimal choice from a runtime perspective.

For a practical implementation, while we could fix the number of colors at the worst-case optimum $k + x$, it is most likely beneficial to choose x even larger, because various algorithmic tweaks and the underlying graph structure can keep the runtime of a trial significantly below the worst-case estimate. This in turn causes the increase in runtime per trial by choosing a larger x to be even more overcompensated by a decrease in the total number of trials needed, as is demonstrated in Figure 2. In fact, for a small path length of 8–10 we can choose the number of colors to be the maximum our implementation allows (that is, 31), and get by 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. *Speedup by Lower Bounds and Cache Preheating*

In a color-coding trial, every vertex carries entries for up to 2^{k+x} color sets, each of them 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

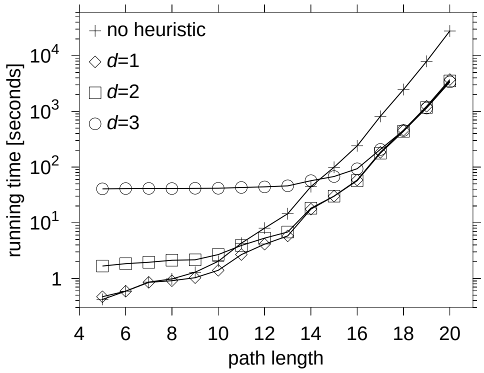

Figure 3. Runtime comparison with heuristic evaluation functions for different values of d (seeking the 20 lowest-weight paths in the yeast network of Scott et al.[10] that differ in at least 30% of participating vertices).

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 $W(u, S)$ corresponding to some partial path. To obtain a length-k path, we need to append another $k - |S|$ edges, and so a lower bound for the total weight of a length-k path expanded from this entry is $W(u, S) + (k - |S|)w_{\text{min}}$, where w_{min} is the minimum weight of any edge in the graph. We improve upon this simple bound by dividing the path length left not into single edges, but short segments of d edges, calculating a lower bound separately for each segment, and summing up these bounds.

We prepare the lower bound calculation in a preprocessing phase by dynamic programming on the uncolored graph. 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.[10], 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 3).

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. (Especially in database applications, similar techniques are known as “preheating the cache.”) 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 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.

Table 1. Basic properties of the network instances YEAST (Scott el al.[10]) and DROSOPHILA (Giot et al.[7]). 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	average degree	maximum degree
YEAST	4 389	14 319	0.067	6.5	237
DROSOPHILA	7 009	20 440	0.030	5.8	175

3.3. *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 (1) 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[4]) 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.

As to the data structure for the color sets at each vertex, they are managed as a Patricia tree, that is, a compact representation of a radix tree[4] where any node which is an only child is merged with its parent. A color set is represented as a bit string of fixed length. The Patricia tree 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.

4. Experimental Results

Method and 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 1200 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".

The real-world network instances used for speed measurements were the *Saccharomyces cerevisiae* interaction network used by Scott el al.[10] and the *Drosophila melanogaster* interaction network described by Giot et al.[7]. 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 runtime to various graph parameters (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

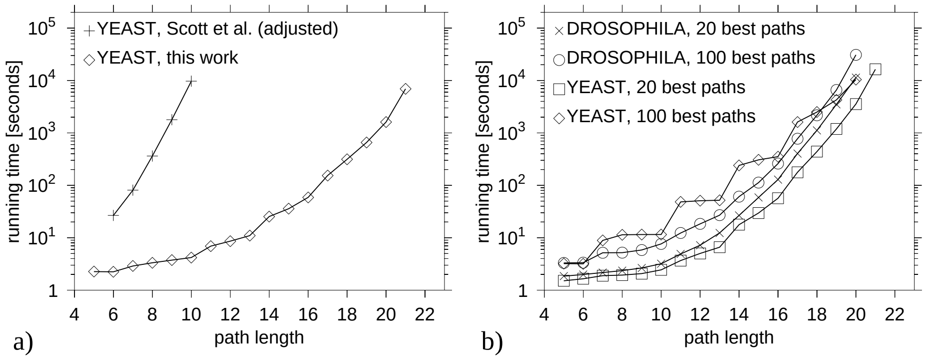

Figure 4. **(a)** Runtimes for YEAST as reported by Scott et al. [10] (adjusted for speed difference of the testing machines) and measured with our implementation. In both cases, paths must start at a membrane protein, and end at a transcription factor. Memory requirements were, e. g., 3 MB for $k = 10$ and 242 MB for $k = 21$.
(b) Comparison of the runtimes 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.

that were generated with the algorithm described by Volz[13]. The results of all experiments and details as to the experimental setting are given in Figures 4 and 5.

Note that Scott et al.[10] obtained their runtimes on a dual 3.0 GHz Intel Xeon processor with 4 GB main memory. To make their runtimes comparable with ours, Figure 4 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.[10] that most likely overestimates the speed of our machine).

Discussion. Compared to the (machine-speed adjusted) runtimes from Scott et al.[10], our implementation is faster by a factor of 10 to 2 000 on YEAST (see Figure 4a). 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 4b shows that the runtimes 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 4b also demonstrates that a major factor in the runtime 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 5a, 5b, and 5d show that the runtime of the color-coding algorithm appears to be somewhat 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 runtimes for graphs with less than 500 vertices in Figure 5a are explained by the fact that the number of length-10 and length-15 paths in these networks

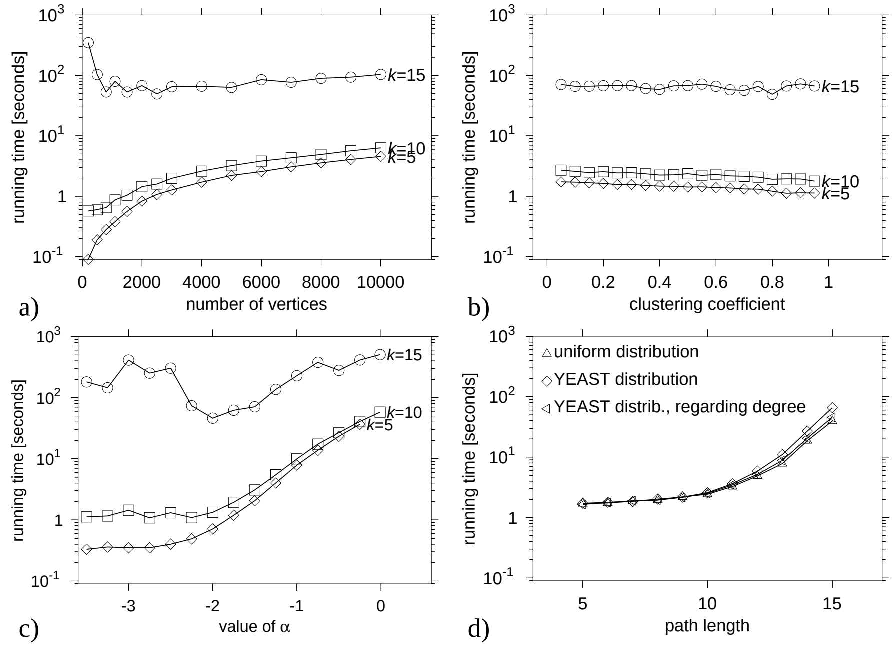

Figure 5. Runtime for our color-coding implementation on random networks, seeking after 20 minimum-weight paths. Unless a parameter is the variable of a measurement, the following default values are used (we have empirically found them to result in networks that are quite similar to YEAST): 4 000 vertices; 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 α is -1.6; edge weights are distributed as in YEAST; the clustering coefficient is 0.1. The data shown reports the average runtime over five runs each. **(a)** Dependency on the number of vertices. **(b)** Dependency on the clustering coefficient. **(c)** Dependency on the parameter α 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.

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 5c 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 runtime for $-3 \leq \alpha \leq -2$ can most is most likely due to a decrease in the total number of length-15 paths as compared to larger values of α.

5. Conclusion

We have given various algorithmic improvements that enable the color-coding technique as a tool both for fast exploration of small pathway candidates as well as for finding larger structures than previously possible. The protein interaction networks of yeast and fruit fly are the so-far most extensively analyzed and understood; as more high-quality data becomes available in the future, it would be interesting to further investigate the practical application of color-coding beyond the detection of linear pathways. Most of the improvements we have given are also useful for detecting other structures that can also be handled with color-coding such as cycles and trees. Finally, we plan further research into the applicability of color-coding to querying pathways in a network (as recently done by Shlomi et al.[11]) and are working on extending our color-coding implementation to a user-friendly tool for pathway candidate detection.

Acknowledgments

Falk Hüffner was supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4, Sebastian Wernicke was supported by the Deutsche Telekom Stiftung, and Thomas Zichner was supported by the Deutsche Forschungsgemeinschaft project PEAL (Parameterized Complexity and Exact Algorithms), NI 369/1. The authors are grateful to Jacob Scott (Cambridge, MA) for providing them with the yeast interaction network discussed in Ref. 10 and to Hannes Moser and Rolf Niedermeier (Jena) for discussions and comments.

References

1. N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.
2. J. S. Bader, A. Chaudhuri, J. M. Rothberg, and J. Chant. Gaining confidence in high-throughput protein interaction networks. *Nature Biotech.*, 22(1):78–85, 2004.
3. T. Can, O. Çamoğlu, and A. K. Singh. Analysis of protein–protein interaction networks using random walks. In *Proc. 5th ACM SIGKDD Workshop on Data Mining in Bioinformatics (BIOKDD '05)*, 2005.
4. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.
5. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
6. G. K. Gerber, Z.-B. Joseph, T. I. Lee, et al. Computational discovery of gene modules and regulatory networks. *Nature Biotech.*, 21(11):1337–1342, 2003.
7. L. Giot, J. S. Bader, C. Brouwer, et al. A protein interaction map of Drosophila melanogaster. *Science*, 302(5651):1727–1736, 2003.
8. 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.
9. T. Ito, T. Chiba, R. Ozawa, et al. A comprehensive two-hybrid analysis to explore the yeast protein interactome. *PNAS*, 98(8):4569–4574, 2001.
10. J. Scott, T. Ideker, R. M. Karp, and R. Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. *J. Comp. Biol.*, 13(2):133–144, 2006. Preliminary version appeared in *Proc. RECOMB'05*.
11. 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.
12. M. Steffen, A. Petti, J. Aach, P. D'haeseleer, and G. Church. Automated modelling of signal transduction networks. *BMC Bioinformatics*, 3:34, 2002.
13. E. Volz. Random networks with tunable degree distribution and clustering. *Phys. Rev. E*, 70:056115, 2004.