Power Iterated Color Refinement

Kristian Kersting
TU Dortmund University
{fn.ln}@cs.tu-dortmund.de

Martin Mladenov
TU Dortmund University
{fn.ln}@cs.tu-dortmund.de

Roman Garnett
University of Bonn
garnett@uni-bonn.de

Martin Grohe
RWTH Aachen
grohe@informatik.rwth-aachen.de

Abstract

Color refinement is a basic algorithmic routine for graph isomorphism testing and has recently been used for computing graph kernels as well as for lifting belief propagation and linear programming. So far, color refinement has been treated as a combinatorial problem. Instead, we treat it as a nonlinear continuous optimization problem and prove that it implements a conditional gradient optimizer that can be turned into graph clustering approaches using hashing and truncated power iterations. This shows that color refinement is easy to understand in terms of random walks, easy to implement (matrix-matrix/vector multiplications) and readily parallelizable. We support our theoretical results with experiments on real-world graphs with millions of edges.

Introduction

The question of efficiently determining whether two given graphs are isomorphic is a long-standing open problem in mathematics and AI. It has attracted considerable attention and effort, due both to its practical importance and its relationship to questions of computational complexity. The exact complexity status of the graph isomorphism (GI) problem remains unknown. It is known to be in the class NP, however neither an NP-completeness proof or a polynomial time solution have been found. The graph automorphism (GA) problem, in which a graph is mapped onto itself while preserving its edge-vertex connectivity, is at least as difficult, since two graphs G and H are isomorphic if and only if the disconnected graph formed by the disjoint union of G and H has an automorphism that swaps the two components.

Color refinement (**CR**, aka "naive vertex classification", "Weisfeiler-Lehman" or "color passing") is a basic algorithmic routine for graph isomorphism testing, appearing as a subroutine in almost all practical isomorphism solvers. **Color Refinement** iteratively partitions, or colors, the vertices of a graph according to an iterated degree sequence: *initially, all vertices get the same color, and then in each round of the iteration two vertices that so far have the same color get different colors if for some color c they have a different number of neighbors of color c. The iteration stops if in some step the partition remains unchanged.* The resulting partition is known as the *coarsest equitable partition* (CEP) of the graph. When applied in the context of graph isomorphism testing, the goal of CR is to partition the vertices of a graph as finely as possible, ideally one would like to compute the partition of the vertices into the orbits of the automorphism group of the graph. This is a sensible idea since (Tinhofer 1991), (Ramana, Scheinerman, and Ullman 1994) and (Godsil 1997) established a tight correspondence between CR and a relaxed formulation of GA, the fractional graph automorphisms (FGA) problem: find a doubly stochastic matrix which commutes with the adjacency matrix of the graph. When applied to AI tasks, the goal of CR is to compress models in order to speed up e.g. probabilistic inference (Ahmadi et al. 2013) and linear programming (Mladenov, Ahmadi, and Kersting 2012) and to extract features for graph kernels (Shervashidze et al. 2011).

Although Ramana *et al.* (1994) have shown that the FGA problem can be phrased as a linear program (LP), CR has been exclusively investigated through the lens of combinatorial graph theory only. On first sight this might be surprising, since the best general-purpose solvers in theory and practice for linear programming are based on continuous optimization rather than the combinatorial approach. However, solving Ramana *et al.*'s LP using off-the-shelf LP solvers does not mimic CR — they differ in the solution as well as the path to the solution — and the graph theory view already led to quasi-linear $\mathcal{O}((m+n)\log n)$ algorithms for finding FGAs of (weighted) graphs with n vertices and m edges due to asynchronous color updates, see e.g. (Berkholz, Bonsma, and Grohe 2013; Grohe et al. 2013) for more details. So, is there a fundamental gap between the combinatorial and continuous views on CR and FGAs?

Here we show that this is not the case. We present the first conditional gradient (CG) approach for computing FGAs and prove that it essentially mimics CR as presented above. Then we show that the link to continuous optimization can led to novel characterizations for computing FGAs. We prove that FGAs can be computed using iterative graph clustering approaches using hashing and truncated power iterations. This creates multiple benefits. It connects CR to main stream machine learning. It shows that CR is easy to understand in terms of random walks with restarts by extending (Boldi, Lonati, and Vigna 2006)'s result that nodes

with different PageRank values have different colors (if the restart are uniform per color class) to a complete characterization of CEPs. This also connects CR to random walks for graph matching (Gori, Maggini, and Sarti 2005) and clustering (Lin and Cohen 2010). Finally and probably most importantly, it shows that CR is easy to implement and readily scalable based on (sparse) matrix-matrix/vector multiplication, either by parallelization or just on a single PC or laptop using e.g. GraphChi (Kyrola, Blelloch, and Guestrin 2012).

We proceed as follows. After reviewing (F)GAs, we show how to use CGs for computing FGAs and develop the hashing and power iterated versions. Before concluding, we support our theoretical results with experimental evidence on real-world graphs with millions of edges. All proofs can be found in the appendix.

Undirected Graphs and Automorphisms

A graph $G = (V, E)$ of size n is defined by a finite set of vertices $V = \{1, \dots, n\}$ and a set of edges $E \subset V \times V$. We consider only undirected graphs with no self-loop, i.e., such that if $(i, j) \in E$ then $(j, i) \in E$ and $(i, i) \notin E$ for any vertices $i, j \in V$. Each such graph can be equivalently represented by a symmetric adjacency matrix A of size $V \times V$, where $A_{ij} = 1$ if there is an edge between vertices i and j, and 0 otherwise. Slightly more general, a weighted graph is defined by associating nonnegative weigths $w_{ij} \in \mathbb{R}_{\geq 0}$ to all edges of a graph G. Weighted graphs can be represented by real-valued adjacency matrices A with $A_{ij} = w_{ij}$.

Given a graph G, the problem of computing automorphisms of G consists in finding a correspondence between vertices of G, which aligns G with itself. That is there is a permutation matrix P of the n vertices with $P_{ij} = 1$ if the i-th vertex of G is matched to the j-th vertex of G, and 0 otherwise. After applying the permutation defined by P to the vertices of G, we obtain a new graph, which we denote by $G' = P(G)$. The adjacency matrix A' of the permuted graph G' is simply obtained from A by $A' = PAP^T$. Each such permutation matrix P encodes a symmetry in the graph G; the identity matrix I encodes the trivial symmetry.

Conditional Gradients for Graph Matching

Computing automorphisms can be approached using continuous optimization formulations for graph isomorphisms, see e.g. (Zaslavskiy, Bach, and Vert 2009; Quadrianto et al. 2010) and references in there. In order to assess whether a permutation P defines a good 'matching' between the vertices of G, an objective function must be defined. Although other choices are possible, we start off by measuring the discrepancy between the graphs by the number of edges (in the case of weighted graphs, it will be the total weight of edges), which are present in one graph and not in the other. In terms of adjacency matrices, this can be computed using the Frobenius norm, $S(P) := ||A - A'||_F^2 = ||A - PAP^T||_F^2$, where the Frobenius matrix norm is defined by $||A||_F^2 = \mathrm{tr}(A^T A) = \sum_{i,j} |A_{ij}|^2$. Since permutation matrices are orthogonal matrices (i.e., $PP^T = I$ and $P^T P = I$), multiplying a matrix by P does not change its l_1 resp. Frobenius norm, hence, one can rewrite $S(P) = ||(A - PAP^T)P||_F^2 = ||AP - PA||_F^2$. Automorphisms are then the minima P^* of $S(P)$ over the set $\mathcal{P}$ of permutation matrices: $P^* = \arg\min_{P \in \mathcal{P}} S(P)$. Some elements of the involved sums, however, do not depend on the permuation matrix P. To see this, let $B = AP$ and $C = AP^T$ and rewrite $S(P) = \sum_{ij}(B_{ij} - C_{ji})^2 = \sum_{ij} B_{ij}^2 + \sum_{ij} C_{ji}^2 - 2\sum_{ij} B_{ij} C_{ji}$. The first two sums do not depend on P since the sums are independent of the order we sum the squared elements together. Hence, an equivalent formulation of automorphisms — meaning that they have the same optimal solutions — is $P^* = \arg\max_{P \in \mathcal{P}} F(P) := \mathrm{tr}(APAP^T)$. That is, we maximize $F(P)$ over the set $\mathcal{P}$ of permuation matrices. In other words we aim at maximizing the number of edges, which are preserved by the permutation.

As mentioned already, the complexity of solving this problem when not considering the trivial solution I is unknown and might be intractable. Hence we relax it by expanding the function $F(P)$ on the set of doubly stochastic matrices $\mathcal{D}$, i.e., $\mathcal{D} := \{D \in \mathbb{R}^{n \times n}$ where $D\mathbf{1} = \mathbf{1}$ and $\mathbf{1}^T D = \mathbf{1}^T\}$, the so-called Birkhoff polytope:

$$\textbf{(FGA)} \quad D^* = \arg\max_{D \in \mathcal{D}} F(D) \; . \tag{1}$$

This fractional graph automorphism (FGA) problem is a quadratic program with linear equality and boundary constraints. It can be solved locally[1] in polynomial time, e.g., using the Frank-Wolfe algorithm, which is a so-called conditional gradient (CG) approach, see e.g. (Jaggi 2013) for more details. CGs [2] are particularly suited to optimization over doubly stochastic matrices. The idea is to sequentially maximize linear approximations of $F(D)$. We start with an initial point $D^{(0)} \in \mathcal{D}$. At the k-th iteration, we replace F with its first order Taylor series expansion $T^{(k)}(H)$ about the point $D^{(k)}$,

$$T^{(k)}(H) := F(D^{(k)}) + \langle \nabla F(D^{(k)}), H - D^{(k)} \rangle \tag{2}$$

where $\nabla F(D^{(k)})$ is the gradient of F at the point $D^{(k)}$. This lower bound is concave and convex (it is a line) and it can be maximized effectively over the convex domain $\mathcal{D}$. In our case, the gradient is $\nabla F(D^{(k)}) = 2AD^{(k)}A$. Since $D^{(k)}$ is fixed at this stage, the maximization is equivalent to solving

$$H^{(k)} = \max_{H \in \mathcal{D}} \langle \nabla F(D^{(k)}), H \rangle \; . \tag{3}$$

This is a linear program **(LP)**, and since $\mathcal{D}$ is constrained in a unimodular fashion, the set of feasible solutions has only integral vertices, namely admissible permutation matrices. Thus, the LP is actually a linear assignment problem **(LAP)** that can be solved efficiently using freely available solvers in $\mathcal{O}(N^3)$, see e.g. (Jonker and Volgenant 1987). Moreover, since $\mathcal{D}$ is convex the line $(1 - \lambda)D^{(k)} + \lambda H^{(k)}$ between $D^{(k)}$ and $H^{(k)}$ is in $\mathcal{D}$, and since the bound is concave the maximum along the line is attained at $\lambda = 1$. One can always select $H^{(k)}$ as next point $D^{(k+1)}$ in order to maximize the lower bound. These operations are iterated until some stopping criterion is met such as $F(D^{(k)})$ is not improving anymore These steps in principle comprise the CG approach to **(FGA)**. However, it has two drawbacks:

[1] $F(D)$ is convex in D, see e.g. (Quadrianto et al. 2010).

[2] It coincides with DC programming and CCCP in our case.

Algorithm 1: CGCR(A): CG for Color Refinement

1 Set $D^{(0)} = \frac{1}{n}\mathbf{1} \in \mathcal{D}$, i.e., the flat partition matrix;
2 Set $k := 1$;
3 **repeat**
4 $\quad B := \textsc{CharactMat}\big(\nabla F(D^{(k)})\big)$;
5 $\quad S := \mathrm{diag}(B^T\mathbf{1})$ /* diagonal mat. of class sizes */;
6 $\quad$ Update $D^{(k+1)} := BS^{-1}B^T$;
7 $\quad$ Set $k := k + 1$;
until $F(D^{(k)}) \leq F(D^{(k-1)})$;
8 **return** $D^{(k)}$

- It does not scale to large graphs due to the cubic running time of LAP solvers.
- It finds integral solutions, typically the trivial one, and typically misses the coarsest equitable partitions (CEP).

The reason for the latter drawback is that CG computes iteratively permutation and not doubly stochastic matrices. In turn, it cannot employ the "more general than" relation that actually exists among all (even doubly stochastic) solutions: a solution D is more general than — a condensation of — another solution D' if $D'_{ij} = 0$ implies $D_{ij} = 0$ for all nodes i and j. Consequently, CG will miss the CEP — the most general solution — found by CR.

In the following, we will show how to adapt CG so that it is guaranteed to converge to the CEP. The approach is based on a novel symmetry-regularized solver for the induced linear subproblems of the CG approach. Actually, we show that the resulting CG approach coincides with CR.

Conditional Gradients for Color Refinement

Reconsider the linear approximation of $F(D)$ and the corresponding linear program for selecting the hindsight direction (3). The main idea underlying our approach is to exploit symmetries already here and, in turn, to favor flat doubly stochastic matrices as solutions.

Intuitively, if two vertices of the graph G have identical subgradients, i.e., the ith and jth rows of $\nabla F(D^{(k)})$ are identical $\nabla_i F(D^{(k)}) = \nabla_j F(D^{(k)})$ both vertices are interchangeable. So, whenever we can 'match' vertex i with vertex j, we could also have 'matched' vertex j with vertex i. Consequently, instead of solving the original LAP, we can also cluster together the vertices with identical subgradients and solve a LAP of reduced dimension. Let $B \in \{0,1\}^{n\times c}$ encode the partition induced by $\nabla F(D^{(k)})$ over the vertices

$$B_{ij} = \begin{cases} 1 & i\text{th vertex is in } j\text{th cluster of } \nabla F(D^{(k)}), \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

and $S \in \mathbb{N}^{c\times c}$ the diagonal matrix with S_{ii} being the number of vertices in color class i, where in analogy to CR we call the clusters of vertices with identical subgradients color classes. Now one solves the LP of reduced dimensionality, namely $h^{(k)} = \arg\max_{h\in\lceil}\langle B^T\nabla F(D^{(k)})S^{-1}B, h\rangle$ where $\lceil$ is the Birkhoff polytope of reduced dimension. The solution $h^{(k)}$ of reduced dimensionality can be expanded to a full solution by considering $H^{(k)} = S^{-1}Bh^{(k)}B^T$.

Algorithm 2: $\textsc{Colors}(M)$

1 Let $U \in \mathbb{R}^{c\times n}$ be the system of representatives of the rows of M;
2 **return** *the index vector v s.t. $M_{i\bullet} = U_{v(i)\bullet}$ for the rows*

Algorithm 3: $\textsc{CharactMat}(M)$

1 $v := \textsc{Colors}(M)$;
2 Initialize the characteristic matrix $B = \mathbf{0} \in \mathbb{R}^{n\times\max(c)}$;
3 **for** $i := 1, 2, \ldots, n$ **do**
4 $\quad B_{iv(i)} := 1$;
5 **return** B

Indeed, this is akin to lifted linear programming (Mladenov, Ahmadi, and Kersting 2012), however, it turns out that we do not have to evoke an LP solver at all for solving the reduced LAP if we are willing to follow ascent instead of steepest direction. CGCR in Alg. 1 is doing this. We start with the flat partition matrix, line 1, use the doubly stochastic matrix H induced by the row clustering of the gradient as update, line 4, and iterate. This computes the CEP in a linear number of iterations as the next two Theorems show.

Theorem 1. CGCR *converges to a local maximum of F.*

Intuitively, H restricts the length of a line search between $D^{(k)}$ and I. We can go at most to H and not all the way 'down' to I. Thus, clustering the gradient matrix according to its row-symmetries acts as a regularizer. This symmetry-regularized CG converges to the CEP.

Theorem 2. CGCR *converges in a linear number of iterations to the coarsest equitable partition (CEP). It is sufficient to cluster $AB^{(k)}$ instead of $AD^{(k)}A$ yielding Alg. 4.*

Theorem 2 establishes a fundamental link between combinatorial and nonlinear optimization views on CR. Its proof shows that CR actually maximizes a lower bound on F using a conditional gradient (CG) approach. However, whereas a standard CG fails to find the CEP, a symmetry regularization of its induced linear subproblems forces it to find the CEP. And, CG directly applies to weighted graphs where other approaches such as Saucy, see e.g. (Katebi, Sakallah, and Markov 2012) and references in there, do not apply.

Although conceptually simple and akin to power iteration methods well known from web mining, naively carrying CGCR out will not scale (already for unweighted graphs). Although sparse matrix operations allow one to compute $AB^{(k)}$ in $\mathcal{O}(m)$ flops (where m is the number of edges), a naive computation of the characteristic matrix B is quadratic in the number of nodes n. We scan e.g. each column to compute the index vectors of identical elements in $\mathcal{O}(m^{(k)}n)$. Since now the entries are integers, we can apply radix sort to compute the index vectors of identical rows in $\mathcal{O}(m^{(k)}n)$. In the worst case this is $\mathcal{O}(n^2)$ since $m^{(k)}$ might be equal to n. Overall, this yields a running time that is cubic: we have n refinement rounds in the worst-case, and each round

Algorithm 4: CGCR(A): CG for Color Refinement

$B^{(0)} := \mathbf{1}$, i.e., the all 1 column vector;
$m^{(0)} := 1$ (the current maximal color) and $k := 1$;
repeat
  $B^{(k+1)} :=$ CHARACTMAT$(AB^{(k)})$;
  Set $m^{(k+1)}$ to the number of columns of $B^{(k+1)}$;
  $k := k + 1$;
until $m^{(k)} = m^{(k-1)}$;
return B^k

Algorithm 5: HCGCR(A): Hashed CGCR

Let π an array where $\pi(i)$ equals to the ith prime;
$c^{(0)} := \mathbf{1}$, i.e., the all 1 column vector;
$m^{(0)} := 1$ (the maximal color) and $k := 1$;
repeat
  $c^{(k+1)} :=$ COLORS$\big(c^{(k)} + A\log(\pi(c^{(k)}))\big)$;
  $m^{(k+1)} = \max(c^{(k+1)})$;
  $k := k + 1$;
until $m^{(k)} = m^{(k-1)}$;
return $c^{(k)}$

takes time $\mathcal{O}(m + n^2)$. This is far too expensive for scaling to unweighted graphs with millions of nodes and edges. In contrast, the naive implementation of CR is known to be $\mathcal{O}((n + m)n)$, i.e., quadratic. So, can we close the gap for unweighted graphs? The answer is yes when using hashing, which can also realize the best known running time for CR, $\mathcal{O}((n + m)\log n)$, using asynchronous color updates.

Hashed and Power Iterated Color Refinement

Hashing is known to help to solve continuous optimization problems more efficiently, see e.g. (Shi et al. 2009). It turns out by using a perfect hash we can directly work with the current color vector $c^{(k)}$ instead of $B^{(k)}$ using $h := c^{(k)} + A\log\big(\pi(c^{(k)})\big)$ instead of $AB^{(k)}$. Here $c^{(k)}$ is the vector with $B_{ic_i^{(k)}} = 1$, and $\pi(j)$ denotes the j-th prime.

Theorem 3. *For two nodes i and j in a graph G, $h_i = h_j$ if and only if $c_i^{(k)} = c_j^{(k)}$ and $\delta\big([AB^{(k)}]_{i\bullet}, [AB^{(k)}]_{j\bullet}\big) = 1$ where δ is the Kronecker delta function and $[\cdot]_{i\bullet}$ the ith row.*

Thus, for unweighted graphs CGCR reduces to iteratively computing the perfect hash values of the colors of all nodes. This is realized in HCGCR in Alg. 5. Next to the $\mathcal{O}(n+m)$ flops when using sparse matrices per iteration, the main additional costs is to store a precomputed table of size $\mathcal{O}(n)$ of the logs of the first n primes. Since there are more than a billion primes known, see e.g. `http://www.bigprimes.net`, this scales well to graphs with billions of nodes. We also note that one could realizes asynchronous color updates. This would lead to stochastic CG approaches akin to stochastic gradients and the quasi-linear CR developed by Berkholz *et al.* (2013). We are not going into details.

We have just closed the gap between the combinatorial and continuous optimization views on CR for unweighted graphs. Now, we will illustrate the benefit of this by developing power iterated CR connecting fractional automorphisms to web mining. Actually, HCGCR is already akin to the well-known power iteration (PI) method for computing eigenvalues. The following Theorem shows that the connection of CR to eigenvalue problems is deeper.

Algorithm 6: PICGCR(A): Power Iterated CGCR

$B^{(0)} = \mathbf{1}$;
$m^{(0)} := 1$ (the current maximal color) and $k := 1$;
repeat
  $B^{(k+1)} :=$ CHARACTMAT$\big(\Pi(A, \alpha, B^{(k)})\big)$;
  Set $m^{(k+1)}$ to the number of columns of $B^{(k+1)}$;
  $k := k + 1$;
until $m^{(k)} = m^{(k-1)}$;
return $B^{(k)}$

Theorem 4. *The CEP can be computed iteratively by clustering PI vectors personalized by the current condensation, see Alg. 6. This converges in a linear number of iterations.*

Thus, CEPs of unweighted graphs can be found by repeatedly clustering globally the steady-state distributions of "color preferred' random walks. This extends Boldi *et al.*'s (2006) results from computing condensations of B^* to computing B^* itself and connects fractional automorphisms to recursive and local spectral clustering approaches, see e.g. (Kannan, Vempala, and Vetta 2004; Dasgupta et al. 2006; Mahoney, Orecchia, and Vishnoi 2012). Since Theorem 4 holds for any restart value α, we can trade off the length l of the random walks with the number of CG iterations k. It is known that PI's rate of convergence is the rate at which α^l goes to zero. A rough estimate of the number of PI iterations l needed to reach a tolerance level ϵ is $\tau = \log(\epsilon)/\log(\alpha)$. However, color classes also change a lot in early CG iterations. To balance both, we propose to set l to $\min(2k, \tau)$ in the k-th CG iteration since $2k$ is our current best estimate of the diameter of the graph (length of longest shortest path).

Empirical Illustration

To investigate whether CR based on matrix operations is practical, we implemented naive versions of CGCR, HCGCR (denoted as **Hashing** and available at `https://github.com/rmgarnett/fast_wl/`) and PICGCR in Matlab on a single Linux machine(4×3.4 GHz cores, 32 GB main memory). The convergence threshold ϵ for PI was set to 10^{-8} and the damping factor α to 0.95. The maximum number of PI iterations was set to 500 (denoted as **PIfix**) resp. to $\min(2k, \tau)$ (**PIflex**). To reduce running time, we randomly selected half of the color class and the currently largest color class, and computed their union as preference vector[3]; the clustering was done with the resulting PI vector and the color vector resulting from one HCGCR iteration to ensures convergence. We measured the running time

[3]That is we solve the subproblem exactly only with probability p resulting in $1/p$ more CG iterations (Jaggi 2013). To avoid this, we coupled this approximate solution with the hashing solution.

Name / Description	# nodes	# edges	Avg. (5 reruns) time in sec. / median # CG iteration Hashing		PIfix		PIflex		S	C
chain100001: Chain graph	100001	100,000	699.62	50002	420.18	•1252	•136.59	2892	< 0.01	49%
grid1000: Grid graph	1,000,000	1,998,000	96.26	501	77.44	•21	•23.87	41	0.43	13%
email-EuAll: Email comm. netw., EU res.	265,214	365,030	•0.32	8	6.09	•5	0.51	•5	0.05	81%
soc-Epinions1: Who-trusts-whom netw.	75,888	405,740	•0.08	5	1.60	5	0.14	•4	0.02	30%
web-Google: Web graph from Google	875,713	4,322,051	•5.61	17	163.49	•11	14.67	•11	1.03	40%
flickr: 2005 crawl of flickr.com by D. Gleich	820,878	9,837,214	•1.32	•5	59.70	6	3.47	•5	0.61	40%
lung2: Transp. in lung, Uni. Aukland	109,460	492,564	4.84	227	3.39	•10	•1.48	26	0.06	59%
xenon2: Complex zeolite, sodalite crystals	157,464	1,933,344	0.96	35	3.81	•5	•0.78	10	0.16	59%
		Total •	4	1	0	6	4	4		

Table 1: Scaling results on graphs generated from `http://www.cise.ufl.edu/research/sparse/matrices/index.html` (matrices were turned into graphs using $A := A + A^T$ and thresholding $|A| > 0$). C is the compression ratio (ratio of # of color classes and n). • denotes best value in a row (among proposed approaches) and < 0.01 smaller than 0.01. CGCR ran out of memory.

Name / # graphs / avg. # nodes	Hashing	WL	CGCR	S
MUTAG / 188 / 17.93	•0.23	0.53	0.6	–
ENZYMES / 600 / 29.87	•0.64	3.46	2.08	–
NCI1 / 111 / 29.87	•5.25	16.07	93.81	–
Weighted MUTAG	–	–	•0.40	–
Weighted ENZYMES	–	–	•1.82	–
Weighted NCI1	–	–	•111.53	–

Table 2: Running times (sec.) for computing CEPs of graph sets taking from (Shervashidze et al. 2011) for Weisfeiler-Lehman graph kernels. – denotes not applicable. Weighted X denotes row normalized $(X + 0.0001)$.

and the number of iterations. The results in Tab. 1 show that CEPs (they were always found) are readily computable using sparse matrix operations.The CG approaches can scale to graphs with millions of nodes. Hashing seems to be best on real world graphs with small diameters such as social networks; it is fastest per CG iteration and there is not much we can gain from PI. For graphs with large diameters such a chains and grids (as they often appear in AI tasks) the PI methods are faster since hashing may take many CG iterations. **PIfix** takes the fewest CG iterations but each iteration may take quite some time. **PIflex** balances both the best. To get references for the running times, we compared to Saucy **S**[4], see also Tab. 1, and to the combinatorial CR implementation **WL** in Matlab of (Shervashidze and Borgwardt 2009; Shervashidze et al. 2011) used for fast Weisfeiler-Lehman graph kernels, see Tab. 2. The results show that **S** is faster for unweighted graphs since it can rely on integer arithmetic whereas Matlab only supports double sparse matrices. However, **S** cannot be used for Shervashidze *et al.*'s graph kernels due to its asynchronous color updates and cannot deal with weighted graphs. Compared to **WL**, (sparse) matrix approaches can be faster and deal with weighted graphs. In summary, this shows that recent AI techniques (Ahmadi et al. 2013; Mladenov, Ahmadi, and Kersting 2012; Shervashidze et al. 2011) based on WL can efficiently be realized using (sparse) matrix-matrix/vector multiplications.

Conclusions

We described novel and simple methods for color refinement (CR), a basic algorithmic routine for graph isomorphism, fast graph kernels as well as lifted belief propagation reps. linear programming. The methods are easy to understand and implement using (sparse) matrix-matrix/vector multiplications, readily parallelizable, and efficient and scalable in terms of time and space. Moreover, they open up many interesting doors. Using disk-based systems such as GraphChi (Kyrola, Blelloch, and Guestrin 2012) the fractional automorphisms of massive graphs with billion of edges are readily computable on just a single PC or laptop. The results also suggest novel graph clustering and kernel approaches based on iterative PI vectors where additionally the lower number of CG iterations is promising. They may also lead to novel notions of fractional automorphisms imposing e.g. norm instead of rank constraints.

Acknowledgments: The authors would like to thank the anonymous reviewers for their feedback. This work grew out of the DFG Collaborative Research Center SFB 876 and discussions with Petra Mutzel and Christian Sohler, and was partly funded by the DFG, KE 1686/2-1 and GA 1615/1-1.

Appendix

Proof of Theorem 1: Reconsider the Taylor series approximation (2) used in the kth iteration of CG at the point $H = BS^{-1}B^T$ where B denotes the partition induced by $\nabla F(D^{(k)})$ as computed in (4). By construction, H is a doubly stochastic matrix. To see this note that $B^T\mathbf{1}$ is the columns vector of the sizes of color classes. So, $S^{-1}B^T\mathbf{1} = \mathbf{1}$ and hence $BS^{-1}B^T\mathbf{1} = \mathbf{1}$. Now, $\mathbf{1}^TBS^{-1}B^T = \mathbf{1}^T$ follows from the symmetry of H, i.e., $H = H^T$. We are now going to prove that $\langle \nabla F(D^{(k)}), H\rangle = \langle \nabla F(D^{(k)}), I\rangle$. Due to the Birkhoff theorem , the matrix H is a convex combination $H = \sum_i w_i P_i$ of permutation matrices P_i. Thus $\langle \nabla F(D^{(k)}), H\rangle = \sum_i w_i \langle \nabla F(D^{(k)}), P_i\rangle$. For each P_i its inverse P_i^T is among the P_is, since if we can exchange row i by row j, then we could also have exchanged row j by row i within the LAP. Thus, $\sum_i w_i \langle \nabla F(D^{(k)}), P_i\rangle = \sum_i w_i \langle \nabla F(D^{(k)}), P_i^T\rangle$. It holds $\langle \nabla F(D^{(k)}), P_i^T\rangle = \mathrm{tr}(\nabla^T F(D^{(k)})P_i^T) = \mathrm{tr}((P_i \nabla F(D^{(k)}))^T)$. By construction of B we know $P_i \nabla F(D^{(k)}) = \nabla F(D^{(k)})$. Hence the last equation simplifies to $\mathrm{tr}((P_i \nabla F(D^{(k)}))^T) = \mathrm{tr}(\nabla^T F(D^{(k)})) = \langle \nabla F(D^{(k)}), I\rangle$. Here, we have employed the invariance of the trace operator un-

[4] `http://vlsicad.eecs.umich.edu/BK/SAUCY/`. We modified the C/C++ code s.t. it stops after computing the CEP.

der cyclic permutations and transposition. Consequently, $\langle \nabla F(D^{(k)}), H\rangle = \sum_i w_i \langle \nabla F(D^{(k)}), I\rangle = \langle \nabla F(D^{(k)}), I\rangle \sum_i w_i = \langle \nabla F(D^{(k)}), I\rangle$. In other words, plagging this result into (2) at point H ,we get

$$T_k(H) = T_k(I) \ . \tag{5}$$

Intuitively, although H is not identical to I, it behaves like I w.r.t. the lower bound. Using (5), we now show that $H - D^{(k)}$ is an ascent direction. The convexity of F implies that a necessary condition for a local maximum is the inequality $t_k(D) := \langle \nabla F(D^{(k)}), D - D^{(k)}\rangle \geq 0$ for all $D \in \mathcal{D}$. Since $\mathcal{D}$ is convex, the line joining H and $D^{(k)}$ is contained in $\mathcal{D}$ and so the vector $H - D^{(k)}$ is a feasible direction. Moreover, due to (5), $t_k(H) = t_k(I) \geq \langle \nabla F(D^{(k)}), D^{(k)} - D^{(k)}\rangle = 0$. Now, either $t_k(H) > 0$ or $t_k(H) = 0$. In the former case, $H - D^{(k)}$ is an ascent direction, and we can improve the objective value in this direction. So assume $t_k(H) = 0$. Then $t_k(I) = 0$ since $t_k(H) = t_k(I)$, and we get the same directions at point $D^{(k)}$ as at the trivial solution I. Thus $t_k(H) \geq t_k(D) = \langle \nabla F(D^{(k)}), D - D^{(k)}\rangle \ \ \forall D \in \mathcal{D}$, and so $D^{(k)}$ is a local maximum and CGCR terminates.

Proof of Theorem 2: Reconsider line 4 of CGCR in Alg. 1 where we cluster vertices together that have identical rows in the gradient matrix. Two nodes i and j are in the same color class iff $\delta_{ij}^T A D^{(k)} A = \mathbf{0}^{1\times n}$ where $\delta_{ij} = (\mathbf{e}_i - \mathbf{e}_j)$, $\mathbf{e}_i$ denotes the ith unit column vector, and $\mathbf{0}^{1\times n}$ is a row vector of n zeros. Since $AD^{(k)}A$ is symmetric, right multiplication with δ_{ji} yields $\delta_{ij}^T A D^{(k)} A \delta_{ji} = 0$. Now, recall that $D^{(k)} = BS^{-1}B^T = XX^T$ with $X = BS^{-1/2}$. Plugging this into the last equation yields $\delta_{ij}^T AX(XA)^T \delta_{ji} = 0$. Noting that $\delta_{ji} = -\delta_{ij}$ this simplifies to $-YY^T = 0$ with $Y = \delta_{ij}^T AX$. Hence $Y = \mathbf{0}^{1\times c}$ since AX is non-negative. Unfolding Y this means $\delta_{ij}^T AB S^{-1/2} = \mathbf{0}^{1\times c}$. Right multiplication by $S^{1/2}$ yields $\delta_{ij}^T AB = \mathbf{0}^{1\times c}$. This leads us to modify our original algorithm as summarized in Alg. 4: instead of $AD^{(k)}A$ we use $AB^{(k)}$ for clustering in each iteration, saving us from having to construct and store the doubly stochastic matrix $D^{(k)}$ in each iteration, while providing us with the same exact results[5]. Moreover, instead of using $F(D^{(k)})$ to check for optimality, we can simply check whether new colors have been created. If not, we have converged. That is, we do not have to compute $D^{(k)}$. This converges in at most a linear number of iterations since in each step we either create a new color or we stop; overall, however, there can only be at most n colors. Finally, CGCR fulfills conditions (1.1) and (1.2) of (Grohe et al. 2013) and hence converges to the coarsest equitable partition.

Proof of Theorem 3: We have $A \log\big(\pi(c^{(k)})\big)_i = \sum_{l=1}^n A_{il} \log\big(\pi(c_l^{(k)})\big) = \sum_{c=1}^{m^{(k)}} \log\big(\pi(c)\big) \sum_{l=1}^n A_{il} \cdot \delta(c_l, c)$ where $m^{(k)}$ is the currently maximal color. The inner sum simplifies to $[AB^{(k)}]_{ic}$. Now assume $h_i = h_j$. Exponentiating both sides (which turns the sums into products) and setting $N_i := \prod_{c=1}^{m^{(k)}} \pi(c)^{(AB^{(k)})_{ic}}$ yields

$$e^{c_i^{(k)}} N_i = e^{c_j^{(k)}} N_j \tag{6}$$

where both N_i and N_j are integers. Rearranging (6) gives $e^{c_i^{(k)} - c_i^{(k)}} = N_j / N_i$. Note that the right-hand side is rational although all nonzero integral powers of e are irrational. Therefore we conclude that $c_i^{(k)} = c_j^{(k)}$ and (6) simplifies to $N_i = N_j$. From the fundamental theorem of arithmetic, we have that their prime factorizations coincide. Therefore $AB^{(k)})_{ic} = AB^{(k)})_{jc}$ for all $1 \leq c \leq m^{(k)}$, which is equivalent to $\delta\big([AB^{(k)}]_{i\bullet}, [AB^{(k)}]_{j\bullet}\big) = 1$. Since the argument goes in both directions, we have proven the theorem.

Proof of Theorem 4: Reconsider CGCR in Alg. 4. Starting from the all flat partition[6] $B^{(0)} = \mathbf{1}$, it computes $B^{(1)}$ by clustering $AB^{(0)}$. That is, we group together nodes i and j if $\delta_{ij}^T AB^{(0)} = 0$ with $\delta_{ij} = (\mathbf{e}_i - \mathbf{e}_j)$, $\mathbf{e}_i$ begin the ith unit column vector. Then it computes $B^{(2)}$ by clustering $AB^{(1)}$ and so on. The value $[AB^{(k)}]_{ij}$ is the number of 1-step walks that start in a node of class j and end in node i. Thus, it groups together nodes that have the same number of 1-step walks coming from each current color classes $j = 1, 2, \ldots$. Hence once nodes i and j get assigned to different color classes, they will stay in different color classes. That is, $B^{(k)}$ is a condensation[7] of B^*. So, CGCR iteratively computes condensations $B^{(k)}$ of and converges to the CEP B^*.

We now combine this sandwiching behavior with a seminal result due to Boldi *et al.* (2006). They have proven that the steady state distribution of any Markov chain with restart induced by G also induces a condensation of D^*. More precisely, let $P = C^{-1}A$ be the transition matrix induced by G where C is the degree matrix of G. Now, consider the steady states distribution $\pi(\alpha, v)$ of the Markov chain with restart induced by $\alpha P + (1-\alpha)\mathbf{1}^T v$ where v is the preference vector for the restart. Then $\delta_{ij}\pi(\alpha) = 0$ for $0 \leq \alpha < 1$ if nodes i and j are in the same color class in B^*. In general, the partition induced by $\pi(\alpha, v)$, however, will not be B^* but this can be fixed using CGCR. Specifically, let's compute $\pi(\alpha, v)$ using power iteration, that is $x^{(i+1)} := \alpha P x^{(i)} + (1-\alpha)\mathbf{1}^T v$ with $x^{(0)} = v$. Induction over the power iterations shows the equality holds for any iteration i, i.e., $\delta_{ij} x^{(i)} = 0$ for any $i = 1, 2, 3, \ldots$. provided that the preference vector v is color class-wise uniform. This, however is the case for the columns of $B^{(k)}$ modulo normalization. So, let $\Pi(A, \alpha, B^{(k)})$ be the matrix whose uth column is the power iteration vector. We have just proven that $\delta_{ij}^T \Pi(A, \alpha, B^{(k)}) = 0$ for all iterations k. In other words, we can cluster in each iteration the matrix $\Pi(A, \alpha, B^{(k)})$. Doing so can result in fewer CG iterations as we might skip condensations but still converges to B^* since we are always sandwiched between one condensation B^l ($l \geq k$) computed by HCGCR and B^*.

[5] In particular in early iterations $D^{(k)}$ is very densly populated with millions of entries when n is large.

[6] We use $B^{(k)}$, $D^{(k)}$ and the corresponding partition of the graph in an interchangeable way.

[7] $B^{(k)}$ is a condensation of B^* if $D^{(k)}$ is a condensation of D^* . That is each color class in $B^{(k)}$ is a union of classes in B^*.

References

Ahmadi, B.; Kersting, K.; Mladenov, M.; and Natarajan, S. 2013. Exploiting symmetries for scaling loopy belief propagation and relational training. *Machine Learning Journal* 92(1):91–132.

Berkholz, C.; Bonsma, P.; and Grohe, M. 2013. Tight lower and upper bounds for the complexity of canonical colour refinement. In *21st Annual European Symposium on Algorithms (ESA)*, 145–156.

Boldi, P.; Lonati, V.; and Vigna, M. S. S. 2006. Graph fibrations, graph isomorphism, and pagerank. *Theoretical Informatics and Applications* 40(2):227–253.

Dasgupta, A.; Hopcroft, J.; Kannan, R.; and Mitra, P. 2006. Spectral clustering by recursive partitioning. In *14th Annual European Symposium on Algorithms (ESA-2006)*, 256–267.

Godsil, C. 1997. Compact graphs and equitable partitions. *Linear Algebra and its Applications* 255:259–266.

Gori, M.; Maggini, M.; and Sarti, L. 2005. Exact and approximate graph matching using random walks. *IEEE Trans. Pattern Anal. Mach. Intell.* 27(7):1100–1111.

Grohe, M.; Kersting, K.; Mladenov, M.; and Selman, E. 2013. Dimension reduction via colour refinement. In *arXiv:1307.5697*.

Jaggi, M. 2013. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on Machine Learning (ICML); JMLR Proceedings vol. 28*, 427–435.

Jonker, R., and Volgenant, A. 1987. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing* 38:325–340.

Kannan, R.; Vempala, S.; and Vetta, A. 2004. On clusterings: Good, bad and spectral. *J. ACM* 51(3):497–515.

Katebi, H.; Sakallah, K.; and Markov, I. 2012. Graph symmetry detection and canonical labeling: Differences and synergies. In Voronkov, A., ed., *Turing-100*, volume 10 of *EPiC Series*, 181–195.

Kyrola, A.; Blelloch, G.; and Guestrin, C. 2012. Graphchi: Large-scale graph computation on just a pc. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

Lin, F., and Cohen, W. 2010. Power iteration clustering. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 655–662.

Mahoney, M.; Orecchia, L.; and Vishnoi, N. 2012. Local spectral method for graphs: with applications to improving graph partitions and exploring data graphs locally. *J. Machine Learning Research* 13:2339–2365.

Mladenov, M.; Ahmadi, B.; and Kersting, K. 2012. Lifted linear programming. In *15th Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2012)*, 788–797. Volume 22 of JMLR: W&CP 22.

Quadrianto, N.; Smola, A.; Song, L.; and Tuytelaars, T. 2010. Kernelized sorting. *IEEE Trans. Pattern Anal. Mach. Intell.* 32(10):1809–1821.

Ramana, M.; Scheinerman, E.; and Ullman, D. 1994. Fractional isomorphism of graphs. *Discrete Mathematics* 132:247–265.

Shervashidze, N., and Borgwardt, K. 2009. Fast subtree kernels on graphs. In *Proceedings of the 23rd Annual Conference on Neural Information Processing Systems (NIPS)*, 1660–1668.

Shervashidze, N.; Schweitzer, P.; van Leeuwen, E.; Mehlhorn, K.; and Borgwardt, K. 2011. Weisfeiler–Lehman Graph Kernels. *Journal of Machine Learning Research* 12:2539–2561.

Shi, Q.; Petterson, J.; Dror, G.; Langford, J.; Smola, A.; and Vishwanathan, S. V. N. 2009. Hash kernels for structured data. *Journal of Machine Learning Research* 10:2615–2637.

Tinhofer, G. 1991. A note on compact graphs. *Discrete Applied Mathematics* 30:253–264.

Zaslavskiy, M.; Bach, F.; and Vert, J.-P. 2009. A path following algorithm for the graph matching problem. *IEEE Trans. Pattern Anal. Mach. Intell.* 31(12):2227–2242.