Distributed Flow Algorithms for Scalable Similarity Visualization

Novi Quadrianto*, Dale Schuurmans† and Alex J. Smola‡
*SML-NICTA & RSISE-ANU, Canberra, Australia
Email: novi.quadrianto@nicta.com.au
†University of Alberta, Edmonton, Canada
Email: dale@cs.ualberta.ca
‡Yahoo! Research, Santa Clara, US
Email: alex@smola.org

***Abstract*—We describe simple yet scalable and distributed algorithms for solving the maximum flow problem and its minimum cost flow variant, motivated by problems of interest in objects similarity visualization. We formulate the fundamental problem as a convex-concave saddle point problem. We then show that this problem can be efficiently solved by a first order method or by exploiting faster quasi-Newton steps. Our proposed approach costs at most $O(|\mathcal{E}|)$ per iteration for a graph with $|\mathcal{E}|$ edges. Further, the number of required iterations can be shown to be independent of number of edges for the first order approximation method. We present experimental results in two applications: mosaic generation and color similarity based image layouting.**

***Keywords*-Visualization; Flow networks; Distributed algorithms; Linear programming;**

I. Introduction

The maximum flow problem refers to the problem of finding a feasible flow through a single-source, single-sink flow network that is maximal. This flow problem has numerous applications, ranging from image segmentation and bipartite matching to project selection and airline scheduling, among others [1]. Within data mining, machine learning and computer vision, bipartite matching itself has found applications in shape matching and object recognition [2] to document ranking [3].

Given the vast range of applications, the maximum flow problem has obviously been well studied. Many algorithms have been devised and all of the proposed algorithms have worst case time complexity that are acceptable from a practical point of view (for a comprehensive list to [4]). However, questions arise as to how to adapt those methods to exploit parallelism or the distributed nature of current computer architectures and to be able to avoid holding the entire problem in memory by instead streaming the data off a disk. Most of the existing methods (if not all) involve some sort of *global* search operations and thus it is not obvious on how to parallelize the algorithms.

We propose a simple algorithm for solving the maximum flow problem and its minimum cost variant which involves only *local* operations, thus scalable and parallelizable. We formulate the problem as a convex-concave saddle point problem. We present two algorithms to solve the saddle point problem, one based on a first order method and the other one is based on quasi-Newton method. For a graph with $|\mathcal{E}|$ edges, our first order approximation method scales as $O(|\mathcal{E}|c(\epsilon))$ where $c(\epsilon)$ is a constant that depends only on the number of iterations needed to achieve an ϵ-close approximation and is *independent* of the number of edges $|\mathcal{E}|$. Our quasi-Newton method obtains the same worst case guarantees but demonstrates superior performance in practice. Note that, the number of edges $|\mathcal{E}|$ is *always* less than or equal to $C^2_{|\mathcal{V}|}$ where $|\mathcal{V}|$ denotes the number of nodes. As the proposed approach admits parallelism in the form of matrix-vector multiplications, this could possibly further reduce the computational complexity. Parallelism of matrix-vector multiplications itself is a well-studied field started even in the period when massive parallelism architectures are not readily available (see for example [5]). Consequently, our new algorithms will facilitate large scale applications of flow problems including mosaic generation and color similarity based image layouting discussed in Section VII, among others.

The remainder of the paper is organized as follows: we will first discuss related work in Section II. Some background on the flow problem is then established in Section III. Sections IV and V are devoted to convex-concave saddle point reformulation and the two algorithms to solve it. Before concluding, we demonstrate the effectiveness of our method in the bipartite matching problems and analyze empirically the speedup gained for parallelism in a shared memory architecture in Section VII.

II. Related Work

We discuss several related work in operation research, optimization, and data mining and machine learning areas.

Anderson and Setubal [6] implemented Goldberg-Tarjan's algorithm with a *global relabeling* heuristic in a sequent symmetric multiprocessors architecture. Over a decade later, Bader and Sachdeva [7] designed a parallel algorithm with a *gap relabeling* heuristic and a cache-aware consideration on present-day symmetric multiprocessors. These parallel implementations share the common lock feature to protect every push and relabel operation. Several work have also

been done on implementing Goldberg-Tarjan's algorithm in graphics card processors [8] by relying on an atomic operation *read-update-write* for implementing the locks.

On the mathematical programming side, there are recent breakthroughs on solving non-smooth convex optimization problem. Traditionally, a first-order subgradient algorithm with complexity $O(1/\epsilon^2)$ [9] to achieve ϵ-optimal solution is considered to be the cheapest per-iteration yet not practical algorithm. This pessimistic result is based on treating the function to be optimized as a black box. By exploiting the structure of the problem, Nesterov is able to devise a first-order algorithm with convergence rate $O(1/\epsilon)$ [10]. Very recently, Gilpin, Peña and Sandholm [11] improved Nesterov's algorithm with a simple modification of lowering target accuracy at each iteration and achieved theoretical convergence rate of $O(\kappa(A)\ln(1/\epsilon))$. This actually means in theory, our first order approximation method could achieve a theoretical guarantee of $O(|\mathcal{E}|\kappa(A)\ln(1/\epsilon))$. However, in practice the constant $\kappa(A)$ (a condition measure of the associated payoff matrix A) can be quite large. In this paper, we focus on Nesterov's algorithm.

Recently, there are interests in data mining and machine learning community to build large-scale parallel algorithms exploiting advancement in optimization techniques [12]. Specifically, Taskar et al. [13] reformulated a maximum-margin structured estimation as a convex-concave saddle-point problem solvable cheaply via Nesterov's algorithm. Shah and colleagues formulated a belief propagation (BP) algorithm in the context of the assignment problem [14] and recently of the minimum cost flow problem [15]. They prove that BP converges in pseudo-polynomial time to the optimal solution when the optimal solution is unique.

III. The Flow Network

A flow network is a directed graph with $\mathcal{V}$ set of nodes and $\mathcal{E}$ set of edges denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where each edge has a positive weight called a capacity, $c(u, v)$ for $(u, v) \in \mathcal{E}$ and each edge receives a flow. We distinguish two vertices, s-source node and t-sink node. A flow on the network is a function $f : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ that satisfies

- for any distinct vertices u and v, $f(u, v) = -f(v, u)$: *skew symmetry*;
- for any vertex v except source s or sink t, $\sum_{u\in\mathcal{V}} f(u, v) = 0$: *flow conservation*;
- for any vertices u and v, $f(u, v) \leq c(u, v)$ (where if $(u, v) \notin \mathcal{E}, c(u, v) = 0$): *capacity constraint*.

The value of the flow is the sum of the edge flows out of source s.

In this paper, we develop distributed algorithms for the maximum flow and the minimum cost flow problems associated with a given flow network.

A. The Maximum Flow Problem

The *maximum flow* problem is to input a flow network and find a flow whose value is as large as possible. This problem can be formulated as a linear program as follows

$$\underset{f}{\text{maximize}} \ R \tag{1}$$

$$\text{s.t.} \quad \sum_{u\in\mathcal{V}} f(s, u) - \sum_{v\in\mathcal{V}} f(v, s) = R \tag{2}$$

$$\sum_{u\in\mathcal{V}} f(t, u) - \sum_{v\in\mathcal{V}} f(v, t) = -R \tag{3}$$

$$\sum_{u\in\mathcal{V}} f(w, u) - \sum_{v\in\mathcal{V}} f(v, w) = 0 \quad \text{for} \quad w \in \mathcal{V} - \{s, t\} \tag{4}$$

$$0 \leq f(u, v) \leq c(u, v) \quad \text{for} \quad (u, v) \in \mathcal{E}. \tag{5}$$

In the above, the constraints (2) and (3) define the value of flow as the sum of flows out of source or equivalently as the sum of flows into sink, flow conservation (including skew symmetry) and capacity constraint are enforced in (4) and (5), respectively. The standard algorithm to solve the optimization problem in (1) is an implementation of the Fold-Fulkerson method (Edmonds-Karp and Dinic) and the more efficient Goldberg-Tarjan's push-relabel algorithm [1]. For those algorithms based on the notion of Ford-Fulkerson augmenting path, parallelism is not possible. Although Goldberg-Tarjan's approach consists mostly local operations but it needs occasional global search operations. Here we present a reformulation of the maximum flow problem as a convex-concave saddle point problem which involves only local operations thus naturally parallelizable. This reformulation will then enable us to solve large scale flow problems.

B. The Minimum Cost Flow Problem

A variation of the maximum flow problem described earlier is to find a flow which has the lowest cost. That is, there is a cost $f(u, v) \times w(u, v)$ for sending a flow of $f(u, v)$ through an edge $(u, v) \in \mathcal{E}$ with an associated cost $w(u, v)$. Thus, the corresponding linear programming formulation of the minimum cost flow problem is as follows

$$\underset{f}{\text{minimize}} \sum_{(u,v)\in\mathcal{E}} w(u, v) f(u, v), \tag{6}$$

subject to the same constraints as the maximum flow problem. Due to inherent similarity to the maximum flow problem, most of the algorithms to tackle (6) are generalizations of maximum flow algorithms [1]. Section VI describes the reduction of the widely-used assignment problem to the minimum cost problem, thus the assignment problem will be solvable in a distributed manner.

In the subsequent section, we describe a convex-concave saddle point reformulation of the maximum flow and its minimum cost flow variant.

IV. Convex-Concave Saddle Point Solution

We can rewrite the maximum flow problem in (1) in the canonical form as follows: define two vectors $x \in \mathbb{R}^{|\mathcal{E}|+1}$ and $d \in \mathbb{R}^{|\mathcal{E}|+1}$ with the following components $x = [R, f(e_1), \ldots, f(e_{|\mathcal{E}|})]^T$ and $d = [-1, 0, \ldots, 0]$ where e denotes an edge of the flow network. Further, define a matrix $A \in \mathbb{R}^{|\mathcal{V}|\times(|\mathcal{E}|+1)}$ which can be represented as $A = [A_R|A_E]$ where $A_R \in \mathbb{R}^{|\mathcal{V}|}$ denotes a column vector multiplying the variable R and $A_E \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{E}|}$ denotes a matrix multiplying the flow vector. The matrix A_E is the node-edge *incidence matrix* of the graph $\mathcal{G}$, that is $A_E[i,j] = 1$ if the edge e_j leaves vertex v_i and $A_E[i,j] = -1$ if the edge e_j enters vertex v_i. Similarly, $A_R[s,1] = -1$ and $A_R[t,1] = 1$ and 0 otherwise. Specifically, this vector A_R corresponds to an edge connecting the sink node to the source node. In the subsequent sections, the set of edges $\mathcal{E}$ includes this additional closing-the-loop edge. The maximum flow linear program is now[1]

$$\underset{x}{\text{minimize}}\ d^T x \quad \text{s.t.} \quad Ax = 0 \quad x_e \in [0, c_e] \quad \text{for} \quad e \in \mathcal{E}, \tag{7}$$

We further replace the constraint (except the capacity constraint) by a partial Lagrangian as follows

$$\min_x \quad \max_\lambda \ d^T x - \lambda^T A x \quad \text{s.t.} \quad x_e \in [0, c_e] \quad \text{for} \quad e \in \mathcal{E}. \tag{8}$$

Here $\lambda \in \mathbb{R}^{|\mathcal{V}|}$ and λ_i are constants which act as Lagrange multipliers to ensure that suitable constraints, i.e. flow conservation property and flow value, are met.

Lemma 1: The solution of 8 and 7 are equivalent.

Proof: Denote by $L^*(\lambda)$ the value of (8) at the solution of (8). It follows that $L(\lambda)$ is concave in λ and moreover, $L(\lambda)$ is maximized for a choice of λ for which the solution of (8) satisfies the flow conservation constraints of (7). ■

Lemma 2: The following objective functions are equivalent

$$\min_x \quad \max_\lambda \ d^T x - \lambda^T A x \quad \text{s.t.} \quad x_e \in [0, c_e] \quad \text{for} \quad e \in \mathcal{E} \tag{9}$$

$$\max_\lambda \quad \min_x \ d^T x - \lambda^T A x \quad \text{s.t.} \quad x_e \in [0, c_e] \quad \text{for} \quad e \in \mathcal{E}. \tag{10}$$

Proof: First we need to show that there exist Λ such that the saddlepoint is contained in a ball of radius Λ. From the saddle point conditions, $d = A^\top \lambda$. Since the matrix A is totally unimodular and thus A is a full row rank matrix, it clearly follows that $\|A^\top \lambda\| \geq \text{eig}_0 * \|\lambda\|$ where eig_0 denotes the smallest eigenvalue. The equivalency follows from Sion's minimax theorem since the function is closed and convex in x and closed, bounded and concave in λ. ■

[1]For minimum cost flow problem, $d \in \mathbb{R}^{|\mathcal{E}|}$ is the associated cost for each edge and $A \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{E}|}$ is the node-edge incidence matrix.

We tackle the maximum flow and minimum cost flow problems in its natural saddle point form as this will allow us to exploit the structure of x and λ separately and thus will enable us to have efficient yet scalable solutions. Note that a simple strategy to alternatingly fix one of x and λ while optimizing the other will usually lead to oscillations and in general is not guaranteed to converge [16].

V. The Algorithms

We present two algorithms that sidestep the oscillations and provably converge to the saddle point solution. One is based on Quasi-Newton method [17] and the other is based on first order method called Nesterov's smoothing method [10], [13]. We will discuss both in the subsequent section.

A. First Order Method

First note that the objective function in (9) is piecewise linear in x and λ, thus there are several non-differentiable points. For a non-differentiable objective function like ours, first order subgradient algorithms are often employed. However, the required number of iterations to achieve ϵ-close solution is $O(1/\epsilon^2)$ [9]. Direct application of subgradient algorithms render the optimization problem impractical. Here we focus on first order method which only required $O(1/\epsilon)$ iterations [10]. We now turn our attention to the following convex-concave problem

$$\max_\lambda \quad \min_x \ \underbrace{d^T x - \lambda^T A x + \frac{1}{2\mu}\|x - x_c\|_{\ell_2}^2 - \frac{1}{2\mu}\|\lambda - \lambda_c\|_{\ell_2}^2}_{\mathcal{L}(x,\lambda)} \tag{11}$$

$$\text{s.t.} \quad x_e \in [0, c_e] \quad \text{for} \quad e \in \mathcal{E}.$$

for some setting of constant $\mu \in \mathbb{R}$ and x_c and λ_c denote (arbitrary) proximal centers for x and λ, respectively. This additional quadratic term on both primal variables x and dual variables λ is commonly known as a proximal regularization term [18], [19]. Note that, at x_c, the gradient of the primal-proximal regularization term is zero and similarly, at λ_c, the gradient of the dual-proximal regularization term is zero. Crucially, our problem is now smooth and strongly convex in x and smooth and strongly concave in λ. Denote a joint feasible space, $\mathcal{Z} = \mathcal{X} \times \Lambda$ where $x \in \mathcal{X}$ and $\lambda \in \Lambda$, our gradient operator on this joint space is now

$$\begin{bmatrix} \nabla_x \mathcal{L}(x,\lambda) \\ -\nabla_\lambda \mathcal{L}(x,\lambda) \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & -A^T \\ -A & 0 \end{bmatrix}}_{:=F} \underbrace{\begin{bmatrix} x \\ \lambda \end{bmatrix}}_{:=z} - \underbrace{\begin{bmatrix} d \\ 0 \end{bmatrix}}_{:=a}. \tag{12}$$

A summary of the primal-proximal dual-proximal approach based on Nesterov's smoothing [10], [13] is described in Algorithm 1.

Note in Algorithm 1, $\Pi_{\mathcal{Z}}(.)$ refers to a projection to the joint feasible space. In the primal space, $\mathcal{X}$ this only involves a simple projection into a box constraint associated with the

Algorithm 1 Primal-Proximal Dual-Proximal Method

Initialize $\hat{z} \in \mathcal{Z}$
Initialize $s_{-1} = 0$
for $t = 0$ **to** τ **do**
  $y = \Pi_{\mathcal{Z}}(\hat{z} + \mu s_{t-1})$
  $z_t = \Pi_{\mathcal{Z}}(y - \mu(Fy - a))$
  $s_t = s_{t-1} - (Fz_t - a)$
end for
output $z_\tau = \frac{1}{1+\tau}\sum_{t=0}^{\tau} z_t$.

maximum allowed flow to be sent across a particular edge, i.e. x_e to $[0, c_e]$ via $x_e \leftarrow \max(0, \min(c_e, x_e))$. There is no projection into feasible space involved in the dual space, Λ. The stepsize μ is set to be the inverse of the Lipschitz constant of the gradient operator with respect to some norm $\|.\|$ [10]. This is a direct consequence of relationship between strong convexity and Lipschitz continuous gradient [20]. For our joint gradient operator, the Lipschitz constant, for example, with respect to ℓ_2-norm is upper bounded by

$$L \equiv \max_{z,z' \in \mathcal{Z}} \frac{\|F(z - z')\|_{\ell_2}}{\|z - z'\|_{\ell_2}} \leq \|F\|_{\ell_2} \leq \max \text{ degree}(\mathcal{G}). \tag{13}$$

In the above, $\|F\|_{\ell_2}$ denotes the largest singular value of F and the last inequality is due to the Gershgorin circle theorem. It is clear that our method will scale well for sparse graph or graphs with small largest eigenvalue [21]. An example of this kind of graph is social network graphs.

Convergence: The convergence of the first order method is usually described as a function of objective values. Define the following gap function

$$\mathcal{G}(x, \lambda) = \left[\max_{\hat{\lambda}} \{d^T x - \hat{\lambda}^T A x\}\right] - \left[\min_{\hat{x}} \{d^T \hat{x} - \lambda^T A \hat{x}\}\right]. \tag{14}$$

For an optimal point (x^*, λ^*), the gap $\mathcal{G}(x^*, \lambda^*)$ is zero and a point (x, λ) is an ϵ-solution if and only if $\mathcal{G}(x, \lambda) \leq \epsilon$.

Lemma 3: [10, Theorem 2] Denote z_t as the sequence generated by Algorithm 1 and $z_\tau = \frac{1}{1+\tau}\sum_{t=0}^{\tau} z_t$. Let $p(x, x') := \frac{1}{2\mu}\|x - x'\|_{\ell_2}^2$ be the proximal function and $\mathcal{G}_r(x, \lambda) := \left[\max_{\hat{\lambda}} \{\mathcal{L}(x, \hat{\lambda}) : p(\lambda, \lambda_c) \leq D_\lambda\}\right] - \left[\min_{\hat{x}} \{\mathcal{L}(\hat{x}, \lambda) : p(x, x_c) \leq D_x\}\right]$ be the gap function with restricted proximal regularizers. Then, for L-Lipschitz continuous gradient operator, after τ iterations the gap of $(x_\tau, \lambda_\tau) = z_\tau$ is

$$\mathcal{G}_r(x_\tau, \lambda_\tau) \leq \frac{L(D_x + D_\lambda)}{\tau + 1}. \tag{15}$$

The above lemma implies $O(1/\epsilon)$ iterations are required to achieve $\epsilon-$close solution. Note that, although the lemma is upper-bounding $\mathcal{G}_r(.,.)$, for $D_x \geq p(x_c, x^*)$ and $D_\lambda \geq p(\lambda_c, \lambda^*)$, the restricted gap function $\mathcal{G}_r(.,.)$ coincides with the gap function $\mathcal{G}(.,.)$ [10].

Lemma 4: The cost per iteration of primal-proximal dual-proximal method is at most $O(|\mathcal{E}|)$.

Proof: The most costly operation in our primal-proximal dual-proximal method involves a vector-matrix multiplication, λA in the primal updates or a matrix-vector multiplication, Ax in the dual updates. Exploiting the structure of the node-edge incidence matrix A, we know that A is sparse with number of non-zero elements at most $2 \times |\mathcal{E}|$. Thus, the matrix-vector multiplication costs at most $O(|\mathcal{E}|)$. ■

Lemma 5: The time complexity of Algorithm 1 to achieve $\epsilon-$close solution is $O((\frac{L(D_x+D_\lambda)}{\epsilon})|\mathcal{E}|)$.

Proof: Follows directly from Lemma 3 and Lemma 4. ■

Memory Efficiency and Parallelism: Note that while Algorithm 1 is presented in the form of joint values (x, λ), primal flow variables x and dual Lagrange variables λ can indeed be solved separately. Note that y, z and s are vectors with elements consist of concatenation of flow vector x and dual vector λ. Denote $y|x$ as the variables y restricted to x components and define similarly for $z_t|x, s_t|x$ and for λ restricted components. The updates in Algorithm 1 can now be explicitly written as:

$$y|x = \Pi_{\mathcal{X}}(\hat{z}|x + \mu \cdot s_{t-1}|x) \tag{16}$$
$$y|\lambda = \hat{z}|\lambda + \mu \cdot s_{t-1}|\lambda \tag{17}$$
$$z_t|x = \Pi_{\mathcal{X}}(y|x - \mu \cdot [d - A^T \times y|\lambda]) \tag{18}$$
$$z_t|\lambda = y|\lambda - \mu \cdot [A \times y|x] \tag{19}$$
$$s_t|x = s_{t-1}|\lambda - [d - A^T \times z_t|\lambda] \tag{20}$$
$$s_t|\lambda = s_{t-1}|\lambda - [A \times z_t|x]. \tag{21}$$

Noticeably, primal gradients computations as well as dual gradients computations involve only matrix-vector multiplication operations and thus are amenable to parallelism.

B. Quasi-Newton Method

It is known that for a convex but non-smooth objective function, many off-the-shelf optimization algorithms including the widely used BFGS [17], are not applicable. However, recent progress has been made to systematically modify BFGS for non-smooth objective functions [22]. We first note that the primal flow variable in (8) will have three possible solutions:

$$x_e = \begin{cases} 0 & \text{if } \left[d - A^T\lambda\right]_e > 0 \\ c_e & \text{if } \left[d - A^T\lambda\right]_e < 0 \\ [0, c_e] & \text{if } \left[d - A^T\lambda\right]_e = 0 \end{cases}$$

Define $g(\lambda) := \min_x \{d^T x - \lambda^T A x\}$, the subdifferential[2]

[2]For a convex function $f : \mathbb{R}^d \to \mathbb{R}$, the subdifferential $\partial f(x)$ at x is the set of all vectors g such that $f(y) \geq f(x) + g^T(y - x)$ for all $y \in \mathbb{R}^d$ and elements belonging to the subdifferential are known as subgradients.

Algorithm 2 Quasi-Newton Method

Initialize λ randomly
repeat
Set the value of primal flow variables according to $\text{sign}(d - A^T\lambda)$
For each flow variables which have zero value of $(d - A^T\lambda)$, set their values by solving $\underset{x|\{x_e=0 \text{ for } [d-A^T\lambda]_e>0;\ x_e=c_e \text{ for } [d-A^T\lambda]_e<0\}}{\text{minimize}} \|Ax\|_{\ell_2}^2$
Solve outer maximization in (8) using non-smooth BFGS [22], obtain new λ
until λ has converged $(\lambda = \lambda^*)$

$\partial g(\lambda)$ is then

$$\{Ax : x \in \arg\min\{d^T x - \lambda^T Ax\}\}. \tag{22}$$

At the optimum, we know that $0 \in \partial g(\lambda)$. Our strategy is now clear, for those undefined values of primal variables (whenever $\left[d - A^T\lambda\right]_e = 0$), we can then optimally set them by solving $\underset{x}{\text{minimize}}\ \|Ax\|_{\ell_2}^2$. For a summary of Quasi-Newton method, refer to Algorithm 2.

Convergence: BFGS is widely known to have a convergence rate at worst in the same rate as first order methods [17], [22]. Although the analysis on required number of iterations is missing, there exist empirical studies showing the superiority of quasi-Newton method which practically exploits second order information to other optimizers including first order methods (see for example [23]).

Below lemma states that quasi-Newton incurs the same cost per iteration as first order method.

Lemma 6: The cost per iteration of Quasi-Newton method is at most $O(|\mathcal{E}|)$.

Proof: Same as 4. The additional optimal selection of undefined flow variables does not add a signification computational overhead. ■

Memory Efficiency and Parallelism: Limited-memory BFGS [24] is available as a variant of BFGS with a matrix-free approach designed for reducing the cost of storing and updating approximate Hessian from $O(|\mathcal{V}|^2)$ to $O(|\mathcal{V}|m)$ where m is freely chosen with typical values $6 - 10$ regardless of problem size. In term of parallelism, the updates of the primal flow variables can be done independently to each other, i.e. each element of x can be set based on the value of $d_e - A_{:,e}^T\lambda$. Subsequently the updated flow variables are re-distributed to BFGS to compute the next iterate of the dual variables. BFGS requires computation of subdifferential, $\partial g(\lambda) = \{Ax\}$ and further parallelization can be done for this costly operation.

There are several libraries that provide optimized sparse matrix-vector operations: Intel MKL and OSKI[3].

[3]http://bebop.cs.berkeley.edu/oski/

VI. Assignment Problem as a Flow Problem

In this section, we describe the reduction of the widely-used assignment problem to a flow problem, this will enable us to solve the assignment problem in a distributed manner.

Let $\mathcal{G}_b = (\mathcal{V}_1 \cup \mathcal{V}_2, \mathcal{E})$ be a bipartite graph and let $w : \mathcal{E} \to \mathbb{R}_+$ be a weight function on the edges. A *perfect* matching of $\mathcal{G}_b$ is a subset $M \subseteq \mathcal{E}$ of the edges such that for every node $v \in \mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$ there is exactly one incidence edge $e \in M$. The weight of a matching M is given by the sum of the weights of its edges, that is $w(M) = \sum_{e \in M} w(e)$. Now the minimum (maximum) weight bipartite matching problem is to find for a given bipartite graph $\mathcal{G}_b$ and a given weight function w a perfect matching of minimum (maximum) weight. This problem is also known as assignment problem in operations research area. We assume without loss of generality that $|\mathcal{V}_1| = |\mathcal{V}_2|$ as dummy nodes with the corresponding super-high or super-low weights can be introduced to achieve this matched size.

It is known that the minimum weight matching problem for bipartite graphs can be reduced to the minimum cost flow problem [25]. The source and sink nodes, s and t respectively, are introduced to the bipartite graph. The source node is connected by an edge (s, v) with capacity $c(s, v) = 1$ and weight $-|\mathcal{V}| \times \max_{e \in \mathcal{E}}(w(e))$ to every node $v \in \mathcal{V}_1$ and every node $v \in \mathcal{V}_2$ is connected to the sink node by an edge (v, t) with capacity $c(v, t) = 1$ and weight $w(v, t) = 0$. The large negative weights on the outgoing edges of source node assure that there will be maximum flow on these edges and therefore every node will be matched. For every edge in the original bipartite graph from $\mathcal{V}_1$ to $\mathcal{V}_2$ admits capacity of 1 and the original weight. The edge (t, s) has weight 0 and capacity ∞. It is now clear that the integral solution of minimum cost flow problem corresponds to a minimum weight matching in the original bipartite graph.

VII. Experiments

We perform weighted bipartite matching experiments on randomly generated data as well as on image data for generating a mosaic and layouting images.

A. Toy Data

In this experiment, we compare the running time of our quasi-Newton approach with the Hungarian method which is known to have worst case time complexity of $O(|\mathcal{V}|^3)$ [26]. We implement our method in Python and use the following implementation of the Hungarian algorithm[4]. For our method, instead of performing optimal selection of primal flow variables as described in Algorithm 2, we randomly select subgradient whenever we are at the non-differentiable points. We generate our toy bipartite graph data with edge weight which is sampled from a uniform distribution, $w(e) \sim \text{Unif}[0, 10]$. The result for 10 repeat

[4]http://www.clapper.org/software/python/munkres/

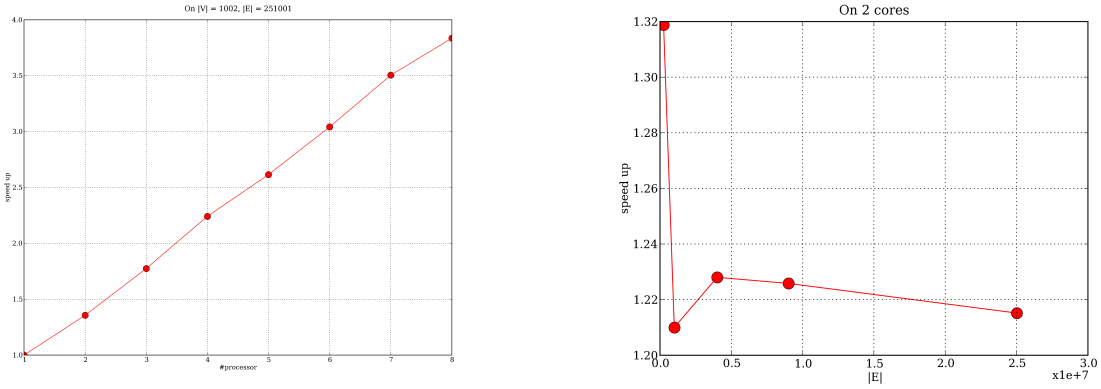


Figure 1. Left: Speedup (in wallclock time) as a function of number of processors; Right: Speedup (in wallclock time) as a function of problem size.

Figure 2. Left: Mosaic of President Obama generated with Quasi-Newton minimum cost flow algorithm. The equivalent flow problem has $|\mathcal{E}| = 29170801$ and $|\mathcal{V}| = 10802$. Right: Color-based image layouting with quasi-Newton algorithm. Images with similar color are found in proximal locations, for example, black colored images are located at the top right while white colored images are put at the bottom left.

trials is summarized in Figure 3(a). Our approach seems to have a comparable scaling properties in comparison to the Hungarian algorithm. We also assess the parallelism property of our method on a simple shared memory architecture. The program with primal-parallel and gradient-parallel is run on a Intel Xeon Core 2 Quad 2GHz machine. Our machine is a Quad core with 2 hyperthreads per core and this might not give us a full speedup as compare to an Octa core machine. The speedup as a function of varying number of processors from one to eight is illustrated in Figure 1–Left. As expected, the more the number of cores, the higher the speedup. Note that we are distributing $0.251 \cdot 10^6$ flow variables on $\{1, \ldots, 8\}$ processors. The better way to utilize current computer architectures is to move to graphics card processors or to computer clusters where massive parallelism is achievable albeit with the trade-off between communication cost and speedup gained. This factor will be investigated in our future research. Figure 1–Right shows the scaling properties when the problem size varies, $\{0.251, 1, 4, 9, 25.010\} \cdot 10^6$, while fixing the number of processors at two. Although at first we might expect the matrix-vector multiplication is memory-bounded which can potentially introduce serious degradation

in the speedup gained, experimental analysis shows more gracious decaying. Lastly, we also assess the performance of our first order approximation method in Algorithm 1. As a baseline, we use a direct application of subgradient algorithm which exhibit $O(1/\epsilon^2)$ convergence. The results are summarized in Figure 3(b). It is evident that as the number of iterations increased, the dual Lagrange variables which basically enforce flow conservation property (enforce a one-to-one assignment in the weighted bipartite matching problem) gradually approach the optimum point.

B. Mosaic Generation

The generation of mosaics is a popular application in the processing of composite images. A template image and a collection of reference images are needed to generate a mosaic where the individual 'pixels' of the mosaic are taken from the set of reference images such that the mosaic best resembles the given template. This is exactly a minimum weight bipartite matching problem. For our experiment, we use a President Obama image as the template image. The set of reference images are taken from LabelMe dataset [27]. We simply extract RGB color features from images and use ℓ_2 norm distance between features as the weight of the bipartite graph. The generated mosaic are shown in Figure 2–Left.

C. Image Similarity Visualization

A recently proposed machine learning technique called kernelized sorting [28] allows, among others, layouting of images into arbitrary structures such as grids or spheres so that images that are visually or semantically similar are placed in proximal locations. This layouting is shown to be advantageous for visualization, web image browsing and photo album summarization [29], [30]. Kernelized sorting is a general technique to perform matching between pairs of objects from different domains which only requires similarity measure within each of the two domains. In this visualization experiment, images form one set of objects while the other set of objects contains coordinate positions of the structure. At the heart of kernelized sorting, a succession of linear assignment problems is solved. We layout images from LabelMe dataset with respect to a simple 2D grid structure. Lab color features, which approximate human visual perception better than RGB space, are extracted from images and are used as the basis of layouting such that images with similar color are to be found nearby, as shown in Figure 2–Right.

VIII. Discussion and Conclusion

We present scalable and distributed algorithms for flow problem by casting it as a convex-concave saddle point problem. Our first order approximation method scales as $O(|\mathcal{E}|/\epsilon)$[5] to achieve an ϵ-close approximation. Thus, this method is particularly appealing when the desired accuracy ϵ is not too small. Our second method exploits quasi-Newton steps and while the number of required iterations can not be characterized, we find that this method works amazingly well in practice. Further, as the two proposed methods involve only local operations, they admit a straightforward parallelism which is attractive considering the evolvement of today's computer architectures. Exploring the usage of our proposed algorithms for other large scale similarity visualizations is an ongoing research.

[5] vide Related Work on the convergence rate that the algorithm could have achieved.

Acknowledgment

The authors would like to thank Yaoliang Yu and Tiberio Caetano for discussion and comments. The authors would also like to thank James Petterson, Mirwaes Wahabzada, Rui Yang, Joseph Anthony and Kristian Kersting for helping in the data and providing computing facilities. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[2] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 509–522, 2002.

[3] Q. V. Le and A. J. Smola, "Direct optimization of ranking measures," *CoRR*, vol. abs/0704.3359, 2007.

[4] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *J. ACM*, vol. 35, no. 4, pp. 921–940, 1988.

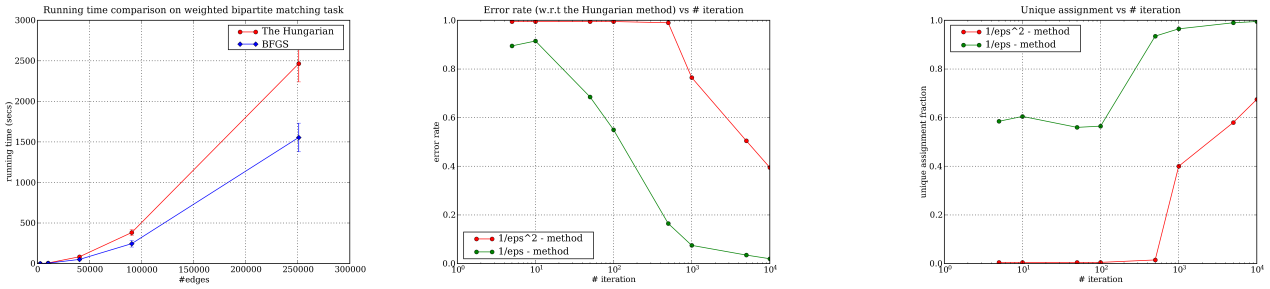
[5] W. F. McColl, "Scalable computing," *Computer Science Today*, pp. 46–61, 1995.

[6] R. J. Anderson and a. C. Setubal, Jo "On the parallel implementation of goldberg's maximum flow algorithm," in *SPAA '92: Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*. ACM, 1992, pp. 168–177.

[7] D. A. Bader and V. Sachdeva, "A cache-aware parallel implementation of the push-relabel network flow algorithm and experimental evaluation of the gap relabeling heuristic," in *ISCA PDCS*, 2005, pp. 41–48.

[8] V. Vineet and P. J. Narayanan, "Cuda cuts: Fast graph cuts on the gpu," *Computer Vision and Pattern Recognition Workshop*, pp. 1–8, 2008.

[9] J. L. Goffin, "On the convergence rate of subgradient optimization methods," *Mathematical Programming*, 1977.



(a) (b)

Figure 3. (a) Running time comparison of the Hungarian algorithm and our quasi-Newton approach with 10 repeat trials; (b) Iteration numbers comparison of first order approximation methods with $O(1/\epsilon^2)$ the subgradient and $O(1/\epsilon)$ Nesterov's convergence rate. Left: Error rate with respect to the Hungarian method; Right: Fraction of unique one-to-one mapping.

[10] Y. Nesterov, "Dual extrapolation and its applications to solving variational inequalities and related problems," *Technical Report, CORE, Catholic University of Louvain.*, 2003.

[11] A. Gilpin, J. Peña, and T. Sandholm, "First-order algorithm with o(ln(1/ϵ)) convergence for ϵ -equilibrium in two-person zero-sum games," in *AAAI'08: Proceedings of the 23rd national conference on Artificial intelligence*. AAAI Press, 2008, pp. 75–82.

[12] K. P. Bennett and E. Parrado-Hernández, "The interplay of optimization and machine learning research," *J. Mach. Learn. Res.*, vol. 7, pp. 1265–1281, 2006.

[13] B. Taskar, S. Lacoste-Julien, and M. I. Jordan, "Structured prediction, dual extragradient and bregman projections," *J. Mach. Learn. Res.*, vol. 7, pp. 1627–1653, 2006.

[14] M. Bayati, D. Shah, and M. Sharma, "Max-product for maximum weight matching: Convergence, correctness, and lp duality," *IEEE Transactions on Information Theory*, vol. 54, no. 3, pp. 1241–1251, 2008.

[15] D. Gamarnik, D. Shah, and Y. Wei, "Belief propagation for min-cost network flow: Convergence & correctness," in *SODA*, 2010, pp. 279–292.

[16] G. M. Korpelevich, "The extragradient method for finding saddle points and other problems," *Ekonomika i Matematicheskie Metody (in Russian; English translation in Matekon)*, vol. 12, pp. 747–756, 1976.

[17] J. Nocedal and S. J. Wright, *Numerical Optimization*, ser. Springer Series in Operations Research. Springer, 1999.

[18] K. C. Kiwiel, "Proximity control in bundle methods for convex nondifferentiable minimization," *Mathematical Programming*, vol. 46, pp. 105–122, 1990.

[19] P. Ravikumar, A. Agarwal, and M. J. Wainwright, "Message-passing for graph-structured linear programs: proximal projections, convergence and rounding schemes," in *ICML '08: Proceedings of the 25th international conference on Machine learning*. New York, NY, USA: ACM, 2008, pp. 800–807.

[20] J. Hiriart-Urruty and C. Lemaréchal, *Convex Analysis and Minimization Algorithms, I and II*. Springer-Verlag, 1993, vol. 305 and 306.

[21] G. Rodgers, K. Austin, B. Kahng, and D. Kim, "Eigenvalue spectra of complex networks," *Journal of Physics A: Mathematical and General*, vol. 38, no. 43, pp. 9431–9437, 2005.

[22] J. Yu, S. V. N. Vishwanathan, S. Günter, and N. N. Schraudolph, "A quasi-Newton approach to nonsmooth convex optimization," *J. Mach. Learn. Res.*, vol. 11, pp. 1145–1200, 2010.

[23] T. P. Minka, "A comparison of numerical optimizers for logistic regression," Microsoft Research, Technical Report, 2007.

[24] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 3, pp. 503–528, 1989.

[25] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, no. 2, pp. 248–264, 1972.

[26] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. New Jersey: Prentice-Hall, 1982.

[27] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, "Labelme: A database and web-based tool for image annotation," *Int. J. Comput. Vision*, vol. 77, no. 1-3, pp. 157–173, 2008.

[28] N. Quadrianto, A. J. Smola, L. Song, and T. Tuytelaars, "Kernelized sorting," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 32, pp. 1809–1821, 2010.

[29] A. Torralba, B. Russell, and J. Yuen, "Labelme: online image annotation and applications," MIT CSAIL, Tech. Rep., 2009, http://people.csail.mit.edu/torralba/publications/labelmeApplications.pdf.

[30] N. Quadrianto, K. Kersting, T. Tuytelaars, and W. L. Buntine, "Beyond 2d-grids: a dependence maximization view on image browsing," in *MIR '10: Proceedings of the international conference on Multimedia information retrieval*. New York, NY, USA: ACM, 2010, pp. 339–348.