Experimantal evaluation of algorithmic solutions for generalised network flow models *

(Preliminary report)

Tomasz Radzik† and Shengxiang Yang†

King's College London

July 2000

Abstract

The generalised network flow problem is to maximise the net flow into a specified sink node in a network with gain-loss factors associated with edges. In practice, computation of solutions for instances of this problem is almost always done using general-purpose linear programming codes, but this may change because a number of specialized combinatorial generalised-flow algorithms have been recently proposed. To complement the known theoretical analyses of these algorithms, we develop their implementations and investigate their practical performance. We include in our study different versions of Goldberg, Plotkin, and Tardos's Fat-Path algorithm and Wayne's Push-Relabel algorithm. We compare the performance of our implementations of these algorithms with implementations of the straightforward highest-gain path-augmentation algorithms. We use various classes of networks, including a type of layered networks which may appear in the multiperiod portfolio revision problem.

Keywords: Network optimisation, Network flow algorithms, Generalised flow, Experimental evaluation.

1 Introduction

2 Definitions

The input

An instance, or an input, of the generalised flow problem is an *antisymmetric generalised flow network* $\vec{G} = (V, \vec{E}, t, e, u, \gamma)$ where:

- $(V, \vec{E})$ is a directed graph with a set of nodes V and an antisymmetric set of edges $\vec{E}$; that is, if $(v, w) \in \vec{E}$, then $(w, v) \notin \vec{E}$;
- $t \in V$ is the *sink*, the destination of flow;
- $e : V \longrightarrow R_{\geq}$ is an *(initial) excess (supply) function*; assume that $e(t) = 0$;
- $u : \vec{E} \longrightarrow R_{\geq}$ is a *capacity function*;
- $\gamma : \vec{E} \longrightarrow R_{>}$ is a *gain (loss) function.*

The assumption that the set of edges is antisymmetric is made without loss of generality. Symbols $R_{\geq}$ and $R_{>}$ stand for the nonnegative and the positive real numbers, respectively. The meaning of the gain function γ is that if x units of flow enter an edge (v, w) at node v, then $\gamma(v, w)x$ units arrive at node w.

*This work is supported by the EPSRC grant GR/L81468.

†Authors' address: Department of Computer Science, King's College London, Strand, London WC2R 2LS, UK. E-mail: {radzik,yangs}@dcs.kcl.ac.uk

A flow and the optimisation objective

A *flow* in $\vec{G}$ is a nonnegative function $f : \vec{E} \longrightarrow R_{\geq}$ which satisfies the flow conservation constraints (2): the net flow outgoing from each node v cannot be greater than the initial excess at this node; and the capacity constraints (3): the amount of flow outgoing from a node v along an edge (v, w) cannot be greater than the capacity of this edge. The objective of the generalised flow problem is to find a flow which maximises the net flow incoming into the sink t. Thus the generalised flow problem is the following linear program with decision variables $\{f(v, w) : (v, w) \in \vec{E}\}$.

$$\text{(P)} \qquad \text{maximise:} \quad \sum_{(z,t)\in\vec{E}} \gamma(z,t)f(z,t) - \sum_{(t,x)\in\vec{E}} f(t,x); \tag{1}$$

$$\text{subject to:} \quad \sum_{(v,x)\in\vec{E}} f(v,x) - \sum_{(z,v)\in\vec{E}} \gamma(z,v)f(z,v) \;\leq\; e(v), \quad \text{for each } v \in V - \{t\}; \tag{2}$$

$$f(v,w) \;\leq\; u(v,w), \qquad \text{for each } (v,w) \in \vec{E}; \tag{3}$$

$$f(v,w) \;\geq\; 0, \qquad \text{for each } (v,w) \in \vec{E}.$$

The first sum in (2) is the flow outgoing from node v and the second one is the flow incoming into v.

The dual problem

Let $\mu(v)$, for $v \in V - \{t\}$, and $\lambda(v, w)$, for $(v, w) \in \vec{E}$, be the dual variables associated with constraints (2) and (3). Introducing a constant $\mu(t) = 1$ (to simplify the formulation) we obtain the following dual problem.

$$\text{(D)} \qquad \text{minimise:} \quad \sum_{v\in V} e(v)\mu(v) \;+\; \sum_{(v,w)\in\vec{E}} u(v,w)\lambda(v,w);$$

$$\text{subject to:} \quad \mu(v) - \gamma(v,w)\mu(w) + \lambda(v,w) \geq 0, \quad \text{for each } (v,w) \in \vec{E};$$

$$\mu(v) \geq 0,\ \lambda(v,w) \geq 0, \qquad \text{for each } v \in V \text{ and each } (v,w) \in \vec{E};$$

$$\mu(t) = 1.$$

For a fixed feasible vector μ, the objective function of (D) is minimised by setting

$$\lambda_\mu(v,w) \;=\; \max\{0,\ \gamma(v,w)\mu(w) - \mu(v)\}, \quad \text{for each } (v,w) \in \vec{E}. \tag{4}$$

A technical issue

A *symmetric* generalised flow network $H = (V, F, t, e, u, \gamma)$ has a symmetric set of edges F (that is, if $(v, w) \in F$, then also $(w, v) \in F$), and $\gamma(v, w) = 1/\gamma(w, v)$, for each $(v, w) \in F$. For such a symmetric network, the generalised flow problem is formulated in the following way.

$$\text{(P}_s\text{)} \qquad \text{maximise:} \quad \sum_{(z,t)\in F} \gamma(z,t)f(z,t);$$

$$\text{subject to:} \quad f(w,v) \;=\; -\gamma(v,w)f(v,w), \qquad \text{for each } (w,v) \in F; \tag{5}$$

$$\sum_{(v,x)\in F} f(v,x) \;\leq\; e(v), \qquad \text{for each } v \in V - \{t\}; \tag{6}$$

$$f(v,w) \;\leq\; u(v,w), \qquad \text{for each } (v,w) \in F.$$

Only positive edge flows are "real" flows. Symmetric edges and condition (5) are used to simplify the notions of residual paths and residual networks. In the case of an antisymmetric network, we have to refer to "sending flow along an edge (v, w)," as well as to "reversing some of the flow sent before along (v, w)." In the case of a symmetric network, we can refer only to "sending flow along an edge," either (v, w) or its symmetric (w, v).

For our antisymmetric input network $\vec{G} = (V, \vec{E}, t, e, u, \gamma)$, let G denote the symmetric network (V, E, t, e, u, γ), where E is obtained from $\vec{E}$ by adding to $\vec{E}$ the symmetric edge (w, v) for each edge $(v, w) \in \vec{E}$, and setting $u(w, v) = 0$ and $\gamma(w, v) = 1/\gamma(v, w)$. For a flow f in $\vec{G}$, the corresponding flow in G is obtained by extending f according to (5). Problem (P) for network $\vec{G}$ is the same as problem (P_s) for network G, and from now on, for convenience, we refer only to symmetric networks. (We began with antisymmetric networks to formulate the generalised flow problem in a more natural way and to have an immediate derivation of the dual problem.)

Residual network

Let f be a flow in network G. The *residual capacities* of edges in E and the *residual excesses* of nodes in V are defined in the following way.

$$\begin{aligned} u_f(v, w) &= u(v, w) - f(v, w), \quad \text{for each } (v, w) \in E; \\ e_f(v) &= e(v) - \sum_{(v,w)\in E} f(v, w), \quad \text{for each } v \in V. \end{aligned}$$

Note that the sum above is the *net* flow outgoing from node v, since it covers the outgoing edges as well as the incoming ones (through their symmetric edges). A *residual edge* is an edge in E with positive residual capacity and a *residual path* is a path which consists only of residual edges. Let V_f denote the set of nodes which can reach the sink t along residual paths, and let $E_f = E \cap V_f \times V_f$ be the set of the edges between the nodes in V_f. The *residual network* is the generalised flow network $G_f = (V_f, E_f, t, e_f, u_f, \gamma)$, with functions e_f, u_f, and γ restricted to sets V_f and E_f. If $e_f(v) > 0$ for a node $v \in V_f$, then we say that v is a node with (residual) excess.

If h is a flow in G_f, then $f + h$ is a flow in G creating the excess $e_f(t) + e_h(t)$ at the sink. (Assume that $h(v, w) = 0$ for each $(v, w) \in E - E_f$.) If h is a maximum flow in G_f, then $f + h$ is a maximum flow in G.

Let $e_{\text{opt}}(t)$ denote the maximum possible net flow into the sink over all flows in G. We say that a flow f in G is ξ-*optimal*, if $e_f(t)(1 + \xi) \geq e_{\text{opt}}(t)$.

Flow generating cycles

The gain $\gamma(P)$ of a path P is the product of the gains of the edges on this path. Sending x units of flow from a node v along a residual path P to the sink increases the residual excess at the sink by $\gamma(P)x$.

A *flow generating cycle* is a residual cycle with gain greater than 1. If Γ is a flow generating cycle, then we can increase flows on the edges of Γ to created (or increase) the excess at at least one node on this cycle. This additional created excess can be send subsequently to the sink.

Let f be a flow in G and h be a flow in the residual network G_f such that the residual network G_{f+h} of the combined flow $f + h$ does not have flow generating cycles. Such a flow h *cancels (all) flow generating cycles* in the residual network G_f, and computation of h is called *canceling (all) flow generating cycles.*

Relabeled network

Let μ be positive node labels in G_f. the *relabeled (residual) capacities*, the *relabeled gain factors*, and the *relabeled (residual) excesses* are defined as

$$\begin{aligned} u_{f,\mu}(v, w) &= u_f(v, w)\mu(v), \\ \gamma_\mu(v, w) &= \gamma(v, w)\mu(w)/\mu(v), \\ e_{f,\mu}(v) &= e_f(v)\mu(v). \end{aligned}$$

The *relabeled network* $G_{f,\mu} = (V_f, E_f, s, u_{f,\mu}, e_{f,\mu}, \gamma_\mu)$ is equivalent to network G_f (they are equivalent instances of the generalised flow problem). For a flow h in G_f, define the flow h_μ in $G_{f,\mu}$:

$$h_\mu(v, w) = h(v, w)\mu(v), \quad \text{for each } (v, w) \in E_f.$$

```
{ INPUT: a generalised flow network G }
f ← a flow which cancels all flow generating cycles in G;
while  there is a node v ∈ V_f with positive residual excess  do
   1: compute a highest gain path P from v to t in G_f;
   2: update flow f by sending flow from v to t along P
      (transfering as much of the excess from v to t as the edge capacities on P allow);
   { invariant: there are no flow generating cycles in G_f }
end_while
{ OUTPUT: optimal flow f in G }
```

Figure 1: Onaga's algorithm (code HIGHESTPATH).

Flows h and h_μ are actually the same flow, but h expresses that flow in terms of the network G_f while h_μ expresses it is terms of the equivalent network $G_{f,\mu}$. For a path P from v to w, $\gamma_\mu(P) = \gamma(P)\mu(v)/\mu(w)$, and for a cycle Γ, $\gamma_\mu(\Gamma) = \gamma(\Gamma)$.

There are no flow generating cycles in G_f if and only if there exist node labels in G_f, called *proper node labels*, such that the relabeled gain of each residual edge in G_f is at most 1. If G_f does not have flow generating cycles, then the node labels μ in G_f such that $\mu(v)$ is equal to the highest gain of a path from v to t in G_f are well defined, are proper, and are called the *canonical node labels*. The canonical node labels, and the highest-gain paths to the sink, can be computed in $O(mn)$ time using the Bellman-Ford-Moore shortest paths algorithm. If we have proper node labels, then this computation can be done in $O(m + n \log n)$ time using Dijkstra's shortest paths algorithm with Fibonacci heaps [**?**]. (Adapt shortest-path algorithms by replacing summing the edge weights with multiplying the edge gains.)

Optimality conditions

Lemma 1. *If f is a flow in G, then the following four conditions are equivalent.*

1. *Flow f is optimal.*
2. *(a) There is no path from a node with excess to the sink t in the residual network G_f, and*
   (b) there is no flow generating cycle in the residual network G_f.
3. *(a) There is no path from a node with excess to the sink t in the residual network G_f, and*
   (b) there exist node labels μ in G_f such that the relabeled gain of each residual edge is at most 1.
4. *There exist node labels μ in G such that the feasible primal vector f and the feasible dual vector (μ, λ_μ) for the primal and the dual problems (P) and (D), satisfy the complementary slackness property (λ_μ is defined by (4)).*

Proof.
(1) $\Rightarrow$ (**??**). If not (**??**) or not (**??**), then we would be able to send more flow into the sink.

(**??**) $\Rightarrow$ (**??**). Discussed above.

(**??**) $\Rightarrow$ (**??**). Extend the canonical node labels μ in G_f to V by setting $\mu(v) = 0$ for each $v \in V - V_f$, and check that vectors f and (μ, λ_μ) satisfy the complementary slackness property for the primal-dual pair (P) and (D).

(**??**) $\Rightarrow$ (1). From the complementary slackness optimality conditions. ■

```
{ INPUT: a generalised flow network G }
f ← a flow which cancels all flow generating cycles in G;
while there is a node v ∈ V_f with positive residual excess do
    1: compute the canonical node labels μ in G_f
    2: compute a maximum flow h in G_{f,μ} from the nodes with excesses to t
       using only edges with (relabeled) gains equal to 1;
    3: f ← f + h;
    { invariant: there are no flow generating cycles in G_f }
end_while
{ OUTPUT: optimal flow f in G }
```

Figure 2: Truemper's algorithm (code MAXFLOW).

```
{ INPUT: a generalised flow network G and a number ξ > 0 }
Δ ← initialise;            { Δ ≤ e_opt(t) ≤ mΔ }
f ← zero flow in G;
while mΔ > ξ · e_f(t) do
    { e_opt(t) − e_f(t) ≤ mΔ }
    h ← a flow which cancels all flow generating cycles in G_f;
    f ← f + h;
    while there is a Δ-fat path from an excess node v to t in G_f do
        update flow f by sending flow from v to t along a highest gain Δ-fat path;
    end_while
    {e_opt(t) − e_f(t) ≤ mΔ/2}
    Δ ← Δ/2;
end_while
{ OUTPUT: ξ-optimal flow f in G }
```

Figure 3: The Fat-Path algorithm.

{ INPUT: a generalised flow network G and a number $\xi > 0$ }
$f \leftarrow$ a flow which cancels all flow generating cycles in G;
$\mu \leftarrow$ the canonical labeling in G_f;
let TOTRESEX$_{f,\mu}$ stand for the current sum of the residual relabeled excesses in $V_f - \{t\}$;
{ TOTRESEX$_{f,\mu}$ is an upper bound on $e_{\text{opt}}(t) - e_f(t)$ }
while TOTRESEX$_{f,\mu} > \xi \cdot e_f(t)$ **do**
  $\Delta \leftarrow$ TOTRESEX$_{f,\mu}/(2(m+n))$;
  PHASE(Δ);
  { no flow generating cycles in G_f, μ – the canonical labeling of G_f, TOTRESEX$_{f,\mu} < (m+n)\Delta$ }
end_while
{ OUTPUT: ξ-optimal flow f in G }.

PHASE(Δ):
  { no flow generating cycles in G_f, μ – the canonical labeling of G_f }
  for each $v \in V_f - \{t\}$ and $(v,w) \in E_f$ **do** $\tilde{e}_{f,\mu}(v) \leftarrow e_{f,\mu}(v)$; $\tilde{e}_{f,\mu}(v,w) \leftarrow 0$; **end_for**
  while there exists $v \in V_f - \{t\}$ such that $\tilde{e}_{f,\mu}(v) \geq \Delta$ **do**
    PUSHFLOWINPATH(v);
    $\mu \leftarrow$ the canonical labeling in G_f;
  end_while.

PUSHFLOWINPATH(v):
  while $v \neq t$ and $\tilde{e}_{f,\mu}(v) \geq \Delta$ **do**
    $w \leftarrow$ the next node from v on the highest gain path to t;
    PUSHFLOWINEDGE(v,w);
    $v \leftarrow w$;
  end_while.

PUSHFLOWINEDGE(v,w):
  do the following updates in $G_{f,\mu}$ (note that $\gamma_\mu(v,w) = 1$):
    move Δ units (of flow) from $\tilde{e}_{f,\mu}(v)$ to $\tilde{e}_{f,\mu}(v,w)$;
    send $\alpha = \min\{\tilde{e}_{f,\mu}(v,w), u_{f,\mu}(v,u)\}$ units of flow along (v,w), increasing $\tilde{e}_{f,\mu}(w,v)$ by α;
    move $\min\{\tilde{e}_{f,\mu}(w,v), \Delta\}$ units from $\tilde{e}_{f,\mu}(w,v)$ to $\tilde{e}_{f,\mu}(w)$.

Figure 4: Goldfarb, Jin and Orlin's excess scaling algorithm.

3 Onaga's and Truemper's algorithms

4 Excess scaling algorithms

4.1 The Fat-Path algorithm

4.2 Goldfarb, Jin and Orlin's excess scaling algorithms

EXSC0: implementation of the algorithm described in Figure **??**

EXSC1: Each PUSHFLOWINPATH(v) computation sends $k\Delta$ units of flow from node v to the sink t along the highest gain residual path, for the largest possible integer k.

EXSC2: Each PUSHFLOWINEDGE(v,w) computation sends $k\Delta$ units of flow from node v along edge (v,w) to node w, for the largest possible integer k.

EXSC3: During the computation PHASE(Δ), we apply the PUSHFLOWINPATH(v) computation to each node v with excess $\tilde{e}_{f,\mu}(v) \geq \Delta$ before computing the canonical labeling, and the new highest-gain paths tree. See Figure **??**. Node next(v) is the next node on the path from v to t in the last computed highest-gain paths tree. Procedure PUSHFLOWINPATH(v) may now terminate not only when the sink has been reached or the current node has excess $\tilde{e}_{f,\mu}(v)$ less than Δ, but also when an edge $(x, \text{next(x)})$ which no

```
PHASE(Δ):
  { no flow generating cycles in G_f, μ – the canonical labeling of G_f }
  for each v ∈ V_f − {t} and (v,w) ∈ E_f do ẽ_{f,μ}(v) ← e_{f,μ}(v); ẽ_{f,μ}(v,w) ← 0 end_for
  while there exists v ∈ V_f − {t} such that ẽ_{f,μ}(v) ≥ Δ do
      L ← {v ∈ V_f − {t} : ẽ_{f,μ}(v) ≥ Δ};
      for each v ∈ L do PUSHFLOWINPATH(v) end_for
      μ ← the canonical labeling in G_f;
  end_while.

PUSHFLOWINPATH(v):
  while v ≠ t, ẽ_{f,μ}(v) ≥ Δ, and u_f(v, next(v)) > 0 do
      PUSHFLOWINEDGE(v, next(v));
      v ← next(v);
  end_while.
```

Figure 5: Modification of Goldfarb, Jin and Orlin's algorithm (code EXSC3).

longer is residual has been reached. Procedure PUSHFLOWINEDGE(v,w) is as in EXSC2.

5 Push-relabel algorithms

5.1 Single-phase push-relabel algorithm

5.2 Multi-phase push-relabel algorithm

6 Generators of generalised flow networks

To create test inputs for our implementations, we use two input generators of generalised flow networks, They are loosely based on possible applications of generalised flow problem in financial analysis. To simplify our initial tests, the generated networks do not have flow generating cycles.

Layered networks: multiperiod network portfolio models

Generator LAYERS creates a network with the set of nodes $V = \{(b,t) : a = 1,2,\ldots,K; \tau = 1,2,\ldots,T\} \cup \{t\}$, A node (b,t) represents asset b at the time period τ. An edge $e = ((b',\tau),(b'',\tau+1))$ represents possibility of exchanging an amount x of asset b' available at the time period τ into the $\gamma(e)\cdot x$ amount of asset b'', which will be available at the next time period $\tau+1$. For each node (b,τ), $\tau < T$, generator LAYERS randomly chooses D such edges outgoing from this node to nodes at the next time period. Each node at the last time period is connected by an edge to the sink t.

Generator LAYERSX is similar to generator LAYERS, but the edges can go from one time period to any other time period. A "forward" egde, from a time period τ' to a time period $\tau'' > \tau'$ represents exchanging assets (with possible delayed availability of the bought assets). A "backward" egde, from a time period τ' to a time period $\tau'' < \tau'$ represents borrowing.

The gains and the capacities of the edges are randomly selected from intervals $[\gamma_{\min},\gamma_{\max}]$ and $[u_{\min},u_{\max}]$, respectively. It seems that in this model one should expect the edge gains to be close to 1 (assuming that the data comes in a "normalised" form), so our intention is to set the parameters $\gamma_{\min}$ and $\gamma_{\max}$ close to 1.

Grid of cliques: exchanging and trasfering currencies

The generator GRIDOFCLIQUES creates a network with the set of nodes $V = \{(c,q) : c = 1,2,\ldots,K; q = 1,2,\ldots,Q\} \cup \{t\}$, where a node (c,q) represents currency c at market q. For each $q = 1,2,\ldots,Q$, the nodes $\{(c,q) : c = 1,2,\ldots,K\}$ form a (directed) clique; and for each $c = 1,2,\ldots,K$, the nodes

```
{ INPUT: a generalised flow network G and a number ξ > 0 }
f ← zero flow in G;
loop
    cancel all flow generating cycles in G_f and compute the canonical labeling μ;
    if TOTRESEX ≥ ξ e_f(t) then terminate;
    h ← PHASE(G_{f,μ}, 1/2);
    f ← f + h;
end_loop
{ OUTPUT: ξ-optimal flow f }.

PHASE(G, ξ):
    { G is a generalised flow network with edge gains of residual edges at most 1 }
    b ← (1 + ξ)^{1/n};
    round edge gains in G to powers of b and let H be the resulting network;
    h(v, w) ← 0, for each (v, w) in H;
    μ(v) ← 0, for each v in H;
    while exists v ∈ H_h with positive excess do
        if exists an admissible edge (v, w) { a residual edge in H_h with γ_μ(v, w) > 1 } then
            { push: } update f by sending min{e_h(v), u_h(v, w)} units of flow along (v, w);
        else
            { relabel: } μ(v) ← μ(v)/b^{1/n};
    end_while
    { h is an optimal flow in H }
    g ← interpretation of flow h in G;
    { g is a ξ-optimal flow in G }
    return g.
```

Figure 6: Tardos and Wayne's Push-Relabel algorithm.

```
{ INPUT: a generalised flow network G }
f ← zero flow in G;
k ← 0;
loop
    k ← k + 1;
    cancel all flow generating cycles in G_f and compute the canonical labeling μ;
    if TOTRESEX ≥ ξ e_f(t) then terminate;
    h ← PHASE(ξ_k);
    f ← f + h;
end_loop
{ OUTPUT: ξ-optimal flow f }.

PHASE(ξ):
    { G_f does not have flow generating cycles, μ is the canonical labeling in G_f }
    b ← (1 + ξ)^{1/n};
    let γ̃_μ(v, w) be γ_μ(v, w) rounded down to integer power of b, for each residual edge (v, w) in G_f;
    while exists v ∈ G_f with positive excess do
        if exists an admissible edge (v, w) { a residual edge in G_f with γ̃_μ(v, w) > 1 } then
            { push: } update f by sending min{e_f(v), u_f(v, w)} units of flow along (v, w);
        else
            { relabel: } μ(v) ← μ(v)/b^{1/n};
    end_while
    { [e_opt(t) − e_{f''}(t)] ≤ [ξ/(1 + ξ)] · [e_opt(t) − e_{f'}(t)],
      where f' and f'' are the flows at the beginning and at the end of the computation }
```

Figure 7: Adaptation of Tardos Wayne's Push-Relabel algorithm.

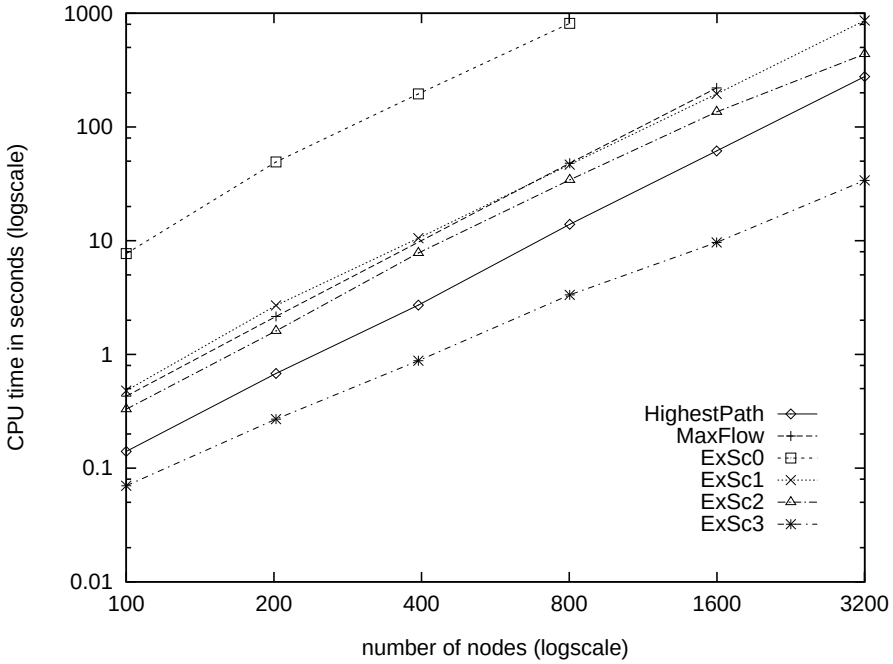
$\{(c, q) : q = 1, 2, \ldots, Q\}$ form a clique. Thus a currency c at a market q can be exchanged into any other currency within the same market and can be transfer into the same currency at any other market. Similarly as above, generator GRIDOFCLIQUES has also parameters $\gamma_{\min}$, $\gamma_{\max}$, $u_{\min}$ and $u_{\max}$ defining ranges of the edge gains and capacities, and $\gamma_{\min}$ and $\gamma_{\max}$ should preferably be set close to 1.

7 Experiments

In our experiments, we consider the various values of nodes as shown in the following Table 1.

8 Conclusion

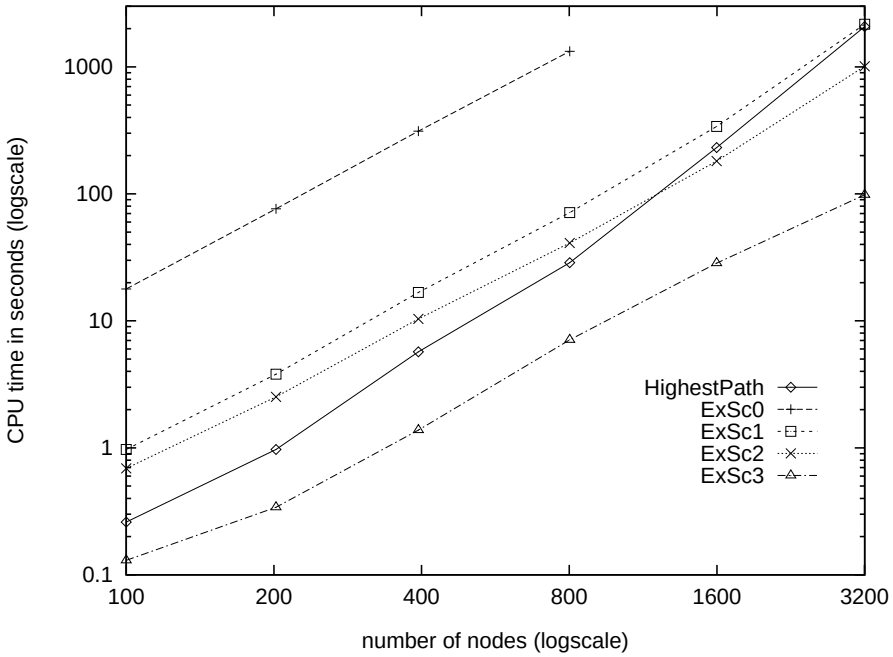
- Excess scaling: full SP computation from scratch seems to be the bottleneck, we have to come up with a suitable practically fast update of the SP tree.
- Push-Relabel:
    1. periodical computation of "good" labels may significantly improve the running times;
    2. avoiding of "canceling flow generating cycles" could be done by saturating edges with relabeled gains $> 1 + \epsilon$ (but "deficit" nodes would be created, so the overall benefit of this approach is not clear).
- Truemper's algorithm with cost scaling seems to be an interesting option, which is worth experimental evaluation.
- A simple method which combines together the excess scaling with the gain scaling is needed.



nodes/edges	HighestPath	MaxFlow	ExSc0	ExSc1	ExSc2	ExSc3
100/400	0.14	0.43	7.69	0.48	0.33	0.07
200/800	0.68	2.16	49.22	2.70	1.61	0.27
400/1600	2.71	-	195.10	10.54	7.80	0.88
800/3200	13.95	48.02	815.46	46.68	34.23	3.34
1600/6400	61.65	220.30	-	195.66	135.76	9.67
3200/12800	277.32	-	-	862.73	437.79	33.94

Figure 8: Running times on inputs created by the generator Layers.

- The new approximation "packing" algorithms for generalised maximum flow problem ([?] and [?], based on [?]) may also be worth experimental evaluation.



nodes/edges	HighestPath	ExSc0	ExSc1	ExSc2	ExSc3
100/500	0.26	17.83	0.97	0.69	0.13
200/1000	0.97	76.29	3.80	2.52	0.34
400/2000	5.70	312.33	16.72	10.36	1.38
800/4000	28.69	1326.39	71.29	41.02	7.08
1600/8000	231.88	-	339.45	181.22	28.46
3200/16000	2074.96	-	2169.16	1009.39	98.24

Figure 9: Running times on inputs created by the generator LayersX.

References

[1] R. K. Ahuja, M. Kodialam, A.K. Mishra, and J. B. Orlin. Computational investigation of maximum flow algorithms. *Europ. J. of Operational Research*, 97:509–542, 1997.

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

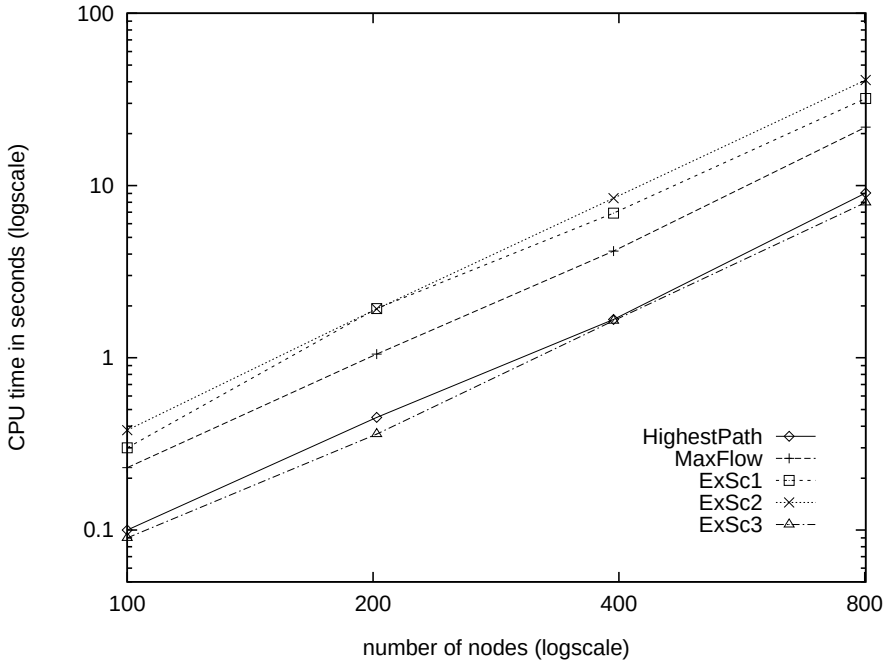
[3] E. Cohen and N. Megiddo. New algorithms for generalized network flows. *Math. Programming*, 64(3):325–336, 1994.

[4] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. Assoc. Comput. Mach.*, 34:596–615, 1987.

[5] N. Garg and J. Könemann. Faster and Simpler Algorithms for Multicommodity Flow and other Fractional Packing Problems. In *Proc. 39th IEEE Annual Symposium on Foundations of Computer Science*, 1998.

[6] A. V. Goldberg, S. A. Plotkin, and É. Tardos. Combinatorial Algorithms for the Generalized Circulation Problem. *Math. Oper. Res.*, 16(2):351–381, 1991.

[7] D. Goldfarb and Z. Jin. A faster combinatorial algorithm for the generalized circulation problem. *Math. Oper. Res.*, 21:529–539, 1996.



nodes/edges	HighestPath	MaxFlow	ExSc0	ExSc1	ExSc2	ExSc3
100/1000	0.10	0.23	38.16	0.30	0.38	0.09
200/3000	0.45	1.05	341.18	1.93	1.92	0.36
400/8000	1.67	4.16	-	6.91	8.44	1.64
800/23000	9.05	21.84	-	32.00	40.93	7.98

Figure 10: Running times on inputs created by the generator GridOfCliques.

[8] D. Goldfarb, Z. Jin, and J. Orlin. Polynomial-time highest-gain augmenting path algorithms for the generalized circulation problem. *Math. Oper. Res.*, 22:793–802, 1997.

[9] J.D. Oldham. Combinatorial approximation algorithms for generalized flow problems. In *Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999.

[10] K. Onaga. Dynamic Programming of Optimum Flows in Lossy Communication Nets. *IEEE Trans. Circuit Theory*, 13:308–327, 1966.

[11] K. Onaga. Optimal Flows in General Communication Networks. *J. Franklin Inst.*, 283, 1967.

[12] T. Radzik. Faster algorithms for the generalized network flow problem. *Math. Oper. Res.*, 23(1):69–100, 1998.

[13] É. Tardos and K.D. Wayne. Simple generalized maximum flow algorithms. In *Proc. 6th International Conference on Integer Programming and Combinatorial Optimization*, pages 310–324, 1998.

[14] K. Truemper. On Max Flows with Gains and Pure Min-Cost Flows. *SIAM J. Appl. Math.*, 32:450–456, 1977.

[15] K.D. Wayne. *Generalized maximum flow algorithms.* PhD thesis, Cornell University, January 1999.

[16] K.D. Wayne. A polynomial combinatorial algorithm for generalized minimum cost flow. In *Proc. 31st Annual ACM Symposium on Theory of Computing*, 1999.

[17] K.D. Wayne and L. Fleischer. Faster approximation algorithms for generalized flow. In *Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999.

(a)

nodes/edges	ExSc3	PushRelabel	CPLEX-p	CPLEX-d
100/500	0.11	0.06	0.03	0.04
200/1000	0.37	0.19	0.08	0.14
400/2000	1.13	0.78	0.22	0.39
800/4000	5.78	2.68	0.72	1.54
1600/8000	27.64	16.80	2.35	5.00
3200/16000	122.82	102.37	31.32	16.00

(b)

nodes/edges	ExSc3	PushRelabel	CPLEX-p	CPLEX-d
100/1000	0.08	0.09	0.03	0.06
200/3000	0.21	0.27	0.11	0.27
400/8000	1.45	2.06	0.26	1.37
800/23000	7.16	11.95	1.89	9.73

Figure 11: Running times on inputs created by the generators (a) LayersX, and (b) GridOfCliques.

nodes/edges	ExSc3	PushRelabel	CPLEX-p	CPLEX-d
6000/20000	105	32	35	29
9000/30000	193	51	57	68
12000/40000	287	91	111	120
15000/50000	389	145	145	167
18000/60000	523	311	210	279

Figure 12: Average running times on inputs created by generator LayersX.