CSC2420 Spring 2015: Lecture 4

Allan Borodin

February 4, 2016

Announcements and todays agenda

- Assignment 1 due next Thursday, February 11. The assignment now consists of 7 questions with the addition of two questions on dynamic programming.
- We will have a number of "theoretical CS" talks this term relating to DCS faculty recruiting. I will try to announce these talks on web page.
- Not related to faculty recruiting, we have a talk tomorrow at 11 (see DCS announcement page) by Bruce Reed (Title: How to determine if a random graph with a fixed degree sequence has a giant component) and next Friday, we have a talk by Yiang Zhang (Title: Budget Feasible Mechanisms on Matchings).
- Todays agenda
  1. Continue discussion of local search.
     - ⋆ The oblivious and non oblivious local search for Exact Max-2-SAT
     - ⋆ The oblivious and non oblivious local search for the WMIS problem for $k+1$-claw free graphs.
     - ⋆ Short discussion of UFL and k-median problems
  2. Start max flow

Exact Max-k-Sat

- **Given:** An exact k-CNF formula

$$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m,$$

where $C_i = (\ell_i^1 \vee \ell_i^2 \ldots \vee \ell_i^k)$ and $\ell_i^j \in \{x_k, \bar{x}_k \mid 1 \leq k \leq n\}$.

- In the weighted version, each C_i has a weight w_i.

- **Goal:** Find a truth assignment τ so as to maximize

$$W(\tau) = w(F \mid \tau),$$

the weighted sum of satisfied clauses w.r.t the truth assignment τ.

- It is NP hard to achieve an approximation better than $\frac{7}{8}$ for exact Max-3-Sat and hence that hard for the non exact version of Max-k-Sat for $k \geq 3$.

- Max-2-Sat can be approximated to within a factor $\approx .87856$.

The natural oblivious local search

- A natural oblivious local search algorithm uses a Hamming distance d neighbourhood:
  $N_d(\tau) = \{\tau' : \tau \text{ and } \tau' \text{ differ on at most } d \text{ variables } \}$

Oblivious local search for Exact Max-k-Sat

Choose any initial truth assignment τ
WHILE there exists $\hat{\tau} \in N_d(\tau)$ such that $W(\hat{\tau}) > W(\tau)$
    $\tau := \hat{\tau}$
END WHILE

How good is this oblivious local search algorithm?

- Note: Following the standard convention for Max-Sat, I am using approximation ratios < 1.
- It can be shown that for $d = 1$, the approximation ratio for Exact-Max-2-Sat is $\frac{2}{3}$.
- In fact, for every exact 2-Sat formula, the algorithm finds an assignment τ such that $W(\tau) \geq \frac{2}{3} \sum_{i=1}^{m} w_i$, the weight of all clauses, and we say that the "totality ratio" is at least $\frac{2}{3}$.
- (More generally for Exact Max-k-Sat the ratio is $\frac{k}{k+1}$).
- This ratio is essentially a tight ratio for any $d = o(n)$.
- This is in contrast to a naive greedy algorithm derived from a randomized algorithm that achieves totality ratio $(2^k - 1)/2^k$.
- "In practice", the local search algorithm often performs better than the naive greedy and one could always start with the greedy algorithm and then apply local search.

Analysis of the oblivious local search for Exact Max-2-Sat

- Let τ be a local optimum and let
  - S_0 be those clauses that are not satisfied by τ
  - S_1 be those clauses that are satisfied by exactly one literal by τ
  - S_2 be those clauses that are satisfied by two literals by τ

  Let $W(S_i)$ be the corresponding weight.

- We will say that a clause involves a variable x_j if either $x_j$or $\bar{x}_j$ occurs in the clause. Then for each j, let
  - A_j be those clauses in S_0 involving the variable x_j.
  - B_j be those clauses C in S_1 involving the variable x_j such that it is the literal x_j or $\bar{x}_j$ that is satisfied in C by τ.
  - C_j be those clauses in S_2 involving the variable x_j.

  Let $W(A_j), W(B_j)$ be the corresponding weights.

Analysis of the oblivious local search (continued)

- Summing over all variables x_j, we get
  - $2W(S_0) = \sum_j W(A_j)$ noting that each clause in S_0 gets counted twice.
  - $W(S_1) = \sum_j W(B_j)$
- Given that τ is a local optimum, for every j, we have

$$W(A_j) \leq W(B_j)$$

  or else flipping the truth value of x_j would improve the weight of the clauses being satisfied.

- Hence (by summing over all j),

$$2W_0 \leq W_1.$$

Finishing the analysis

- It follows then that the ratio of clause weights not satisfied to the sum of all clause weights is

$$\frac{W(S_0)}{W(S_0)+W(S_1)+W(S_2)} \leq \frac{W(S_0)}{3W(S_0)+W(S_2)} \leq \frac{W(S_0)}{3W(S_0)}$$

- It is not easy to verify but there are examples showing that this $\frac{2}{3}$ bound is essentially tight for any N_d neighbourhood for $d = o(n)$.

- It is also claimed that the bound is at best $\frac{4}{5}$ whenever $d < n/2$. For $d = n/2$, the algorithm would be optimal.

- In the weighted case, as in the max-cut problem, we have to worry about the number of iterations. And here again we can speed up the termination by insisting that any improvement has to be sufficiently better.

Using the proof to improve the algorithm

- We can learn something from this proof to improve the performance.
- Note that we are not using anything about $W(S_2)$.
- If we could guarantee that $W(S_0)$ was at most $W(S_2)$ then the ratio of clause weights not satisfied to all clause weights would be $\frac{1}{4}$.
- Claim: We can do this by enlarging the neighbourhood to include $\tau' =$ the complement of τ.

The non-oblivious local search

- We consider the idea that satisfied clauses in S_2 are more valuable than satisfied clauses in S_1 (because they are able to withstand any single variable change).

- The idea then is to weight S_2 clauses more heavily.

- Specifically, in each iteration we attempt to find a $\tau' \in N_1(\tau)$ that improves the potential function

$$\frac{3}{2}W(S_1) + 2W(S_2)$$

instead of the oblivious $W(S_1) + W(S_2)$.

- More generally, for all k, there is a setting of scaling coefficients $c_1, \ldots, c_k$, such that the non-oblivious local search using the potential function $c_1 W(S_1) + c_2 W(S_2 + \ldots + c_k W(S_k)$ results in approximation ratio $\frac{2^k-1}{2^k}$ for exact Max-k-Sat.

Sketch of $\frac{3}{4}$ totality bound for the non oblivious local search for Exact Max-2-Sat

- By renaming variables, we can assume that τ is the all true assignment.
- Let $P_{i,j}$ be the weight of all clauses in S_i containing x_j.
- Let $N_{i,j}$ be the weight of all clauses in S_i containing $\bar{x}_j$.
- Here is the key observation for a local optimum τ wrt the stated potential:

$$-\frac{1}{2}P_{2,j} - \frac{3}{2}P_{1,j} + \frac{1}{2}N_{1,j} + \frac{3}{2}N_{0,j} \leq 0$$

- Summing over variables $P_1 = N_1 = W(S_1)$, $P_2 = 2W(S_2)$ and $N_0 = 2W(S_0)$ and using the above inequality we obtain

$$3W(S_0) \leq W(S_1) + W(S_2)$$

Some experimental results concerning Max-Sat

- Of course, one wonders whether or not a worse case approximation will actually have a benefit in "practice".
- "In practice", local search becomes more of a "heuristic" where one uses various approaches to escape (in a principled way) local optima and then continuing the local search procedure. Perhaps the two most commonly used versions are Tabu Search and Simulated Annealing.
- Later, we will also discuss methods based on "random walks" and other randomized methods (and their derandomizations). .
- We view these algorithmic ideas as starting points.
- But for what it is worth, here are some 2010 experimental results both for artifically constructed instances and well as for one of the many benchmark test sets for Max-Sat.
- Recent unpublished experimental results by Matthias Poloczek show that various ways to use greedy and local search algorithms can compete (wrt. various test sets) a "state of the art" simulated annealing algorithm while using much less time.

Experiment for unweighted Max-3-Sat

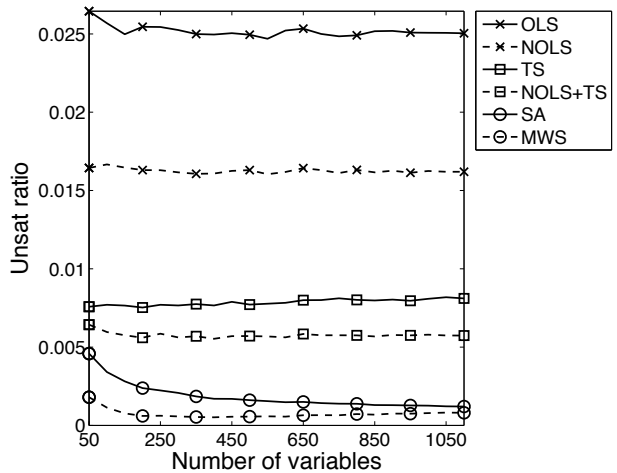

Fig. 1. Average performance when executing on random instances of exact MAX-3-SAT.

[From Pankratov and Borodin]

Experiment for Benchmark Max-Sat

	OLS	NOLS	TS	NOLS+TS	SA	MWS
OLS	0	457	741	744	730	567
NOLS	160	0	720	750	705	504
TS	0	21	0	246	316	205
NOLS+TS	8	0	152	0	259	179
SA	30	50	189	219	0	185
MWS	205	261	453	478	455	0

Table 2. MAX-SAT 2007 benchmark results. Total number of instances is 815. The tallies in the table show for how many instances a technique from the column improves over the corresponding technique from the row.

[From Pankratov and Borodin]

Oblivious and non-oblivious local search for $k+1$ claw free graphs

- We again consider the weighted max (independent) vertex set in a $k+1$ claw free graph. (Recall the argument generalizing the approximation ratio for the k set packing problem.)
- The standard greedy algorithm and the 1-swap oblivious local search both achieve a $\frac{1}{k}$ approximation for the WMIS in $k+1$ claw free graphs. Here we define an "ℓ-swap" oblivous local search by using neighbrourhoods defined by bringing in a set S of up to ℓ vertices and removing all vertices adjacent to S.
- For the unweighted MIS, Halldórsson shows that a a 2-swap oblivious local search will yield a $\frac{2}{k+1}$ approximation.

Berman's [2000] non-oblivious local search

- For the weighted MIS, the "ℓ-swap" oblivous local search results (essentially) in an $\frac{1}{k}$ locality gap for any constant ℓ.
- Chandra and Halldóssron [1999] show that by first using a standard greedy algorithm to initialize a solution and then using a "greedy" k-swap oblivious local search, the approximation ratio improves to $\frac{3}{2k}$.
- Can we use non-oblivious local search to improve the locality gap? Once again given two solutions V_1 and V_2 having the same weight, when is one better than the other?

Berman's [2000] non-oblivious local search

- For the weighted MIS, the "ℓ-swap" oblivous local search results (essentially) in an $\frac{1}{k}$ locality gap for any constant ℓ.
- Chandra and Halldóssron [1999] show that by first using a standard greedy algorithm to initialize a solution and then using a "greedy" k-swap oblivious local search, the approximation ratio improves to $\frac{3}{2k}$.
- Can we use non-oblivious local search to improve the locality gap? Once again given two solutions V_1 and V_2 having the same weight, when is one better than the other?
- Intuitively, if one vertex set V_1 is small but vertices in V_1 have large weights that is better than a solution with many small weight vertices.

Berman's [2000] non-oblivious local search

- For the weighted MIS, the "ℓ-swap" oblivous local search results (essentially) in an $\frac{1}{k}$ locality gap for any constant ℓ.
- Chandra and Halldóssron [1999] show that by first using a standard greedy algorithm to initialize a solution and then using a "greedy" k-swap oblivious local search, the approximation ratio improves to $\frac{3}{2k}$.
- Can we use non-oblivious local search to improve the locality gap? Once again given two solutions V_1 and V_2 having the same weight, when is one better than the other?
- Intuitively, if one vertex set V_1 is small but vertices in V_1 have large weights that is better than a solution with many small weight vertices.
- Berman chooses the potential function $g(S) = \sum_{v \in S} w(v)^2$. Ignoring some small ϵ's, his k-swap non-oblivious local search achieves a locality gap of $\frac{2}{k+1}$ for WMIS on $k+1$ claw-free graphs.

The (metric) facility location and k-median problems

- Two extensively studied problems in operations research and CS algorithm design are the related uncapacitated facility location problem (UFL) and the k-median problem. In what follows we restrict attention to the (usual) metric case of these problems defined as follows:

Definition of UFL

Input: (F, C, d, f) where F is a set of faciltites, C is a set of clients or cities, d is a metric distance function over $F \cup C$, and f is an opening cost function for facilities.
Output: A subset of facilities F' minimizing $\sum_{i \in F'} f_i + \sum_{j \in C} d(j, F')$ where f_i is the opening cost of facility i and $d(j, F') = min_{i \in F'} d(j, i)$.

- In the capacitated version, facilities have capacities and cities can have demands (rather than unit demand). The constraint is that a facility can not have more assigned demand than its capacity so it is not possible to always assign a city to its closest facility.

UFL and k-median problems continued

Deifnition of k-median problem

Input: (F, C, d, k) where F, C, d are as in UFL and k is the number of facilities that can be opened.
Output: A subset of facilities F' with $|F'| = k$ minimizing $\sum_{j \in C} d(j, F')$

- These problems are clearly well motivated. Moreover, they have been the impetus for the development of many new algorithmic ideas which we will hopefully at least touch upon throughout the course.
- There are many variants of these problems and in many papers the problems are defined so that $F = C$; that is, any city can be a facility. If a solution can be found when F and C are disjoint then there is a solution for the case of $F = C$.

UFL and k-median problems continued

- It is known (Guha and Khuller) that UFL is hard to approximate to within a factor better than 1.463 assuming NP is not a subset of $DTIME(n^{\log \log n})$ and the k-median problem is hard to approximate to within a factor better than $1 + 1/e \approx 1.736$ (Jain, Mahdian, Saberi).
- The UFL problem is better understood than k-median. After a long sequence of improved approximation results the current best polynomial time approximation is 1.488 (Li, 2011).
- For k-median, until recently, the best approximation was by a local search algorithm. Using a p-flip (of facilities) neighbourhood, Arya et al (2001) obtain a $3 + 2/p$ approximation which yields a $3 + \epsilon$ approximation running in time $O(n^{2/\epsilon})$.
- Li and Svennsson (2013) have obtained a $(1 + \sqrt{3} + \epsilon)$ approximation running in time $O(n^{1/\epsilon^2})$. Surprisingly, they show that an α approximate "pseudo solution" using $k + c$ facilities can be converted to an $\alpha + \epsilon$ approximate solution running in $n^{O(c/\epsilon)}$ times the complexity of the pseudo solution.

Some (almost) concluding comments (for now) on local search

- We will hopefully return later to local search and in particular non-oblivious local search.
- An interesting (but probably difficult) open problem is to use a non oblivious local search for either the UFL or k-median problems.
- But suffice it to say now that local search is the basis for many practical algorithms, especially when the idea is extended by allowing some well motivated ways to escape local optima (e.g. simulated annealing, tabu search) and combined with other paradigms.
- Although local search with all its variants is viewed as a great "practical" approach for many problems, local search is not often theoretically analyzed. It is not surprising then that there hasn't been much interest in formalizing the method and establishing limits.
- LP is itself often solved by some variant of the simplex method, which can also be thought of as a local search algorithm, moving fron one vertex of the LP polytope to an adjacent vertex.

Ford Fulkerson max flow based algorithms

A number of problems can be reduced to the max flow problem. As already suggested, max flow itself can be viewed as a local search algorithm.

Flow Networks

A flow network $\mathcal{F} = (G, s, t, c)$ consists of a "bi-directional" graph $G = (V, E)$, a source s and termnal node t, and c is a non-negative real valued (*capacity*) function on the edges.

What is a flow

A flow f is a real valued function on the edges satisfying the following properties:

1. $f(e) \leq c(e)$ for all edges e (capacity constraint)
2. $f(u, v) = -f(v, u)$ (skew symmetry)
3. For all nodes u (except for s and t), the sum of flows into (or out of) u is zero. (Flow conservation).
   Note: this is the "flow in = flow out" constraint for the convention of only having non negative flows.

The max flow problem

- The goal of the max flow problem is to find a valid flow that maximizes the flow out of the source node s. As we will see this is also equivalent to maximizing the flow in to the terminal node t. (This should not be surprising as flow conservation dictates that no flow is being stored in the other nodes.) We let $val(f) = |f|$ denote the flow out of the source s for a given flow f.
- We will study the Ford Fulkerson augmenting path scheme for computing an optimal flow. I am calling it a scheme as there are many ways to instantiate this scheme although I dont view it as a general paradigm in the way I view (say) greedy and DP algorithms.
- I am assuming that many people in the class have seen the Ford Fulkerson algorithm so I will discuss this quickly. I am following the development of the model and algorithm as in Cormen et al (CLRS). That is, we have negative flows which simplifies the analysis but may be less intuitive.

A flow f and its residual graph

- Given any flow f for a flow network $\mathcal{F} = (G, s, t, c)$, we can define the residual graph $G_f = (V, E(f))$ where $E(f)$ is the set if all edges e having *positive* residual capacity ; i.e. the residual capacity of e wrt to f is $c_f(e) = c(e) - f(e) > 0$.
- Note that $c(e) - f(e) \geq 0$ for all edges by the capacity constraint. Also note that with our convention of negative flows, even a zero capacity edge (in G) can have residual capacity.
- The basic concept underlying Ford Fulkerson is that of an augmenting path which is an $s - t$ path in G_f. Such a path can be used to augment the current flow f to derive a better flow f'.
- Given an augmenting path π in G_f, we define its residual capacity wrt f as $c_f(\pi) = \min\{c_f(e) | e \text{ in the path } \pi\}$.

The Ford Fulkerson scheme

Ford Fulkerson

$f := 0$;
$G_f := G$ %initialize
While there is an augmenting path in G_f
    Choose an augmenting path π
    $f := f + f_{pi}; f := f$ % Note this also changes G_f
End While

I call this a scheme rather than a well specified algorithm since we have not said how one chooses an augmenting path (as there can be many such paths)

The max flow-min cut theorem

Ford Fulkerson Max Flow-Min Cut Theorem

The following are equivalent:

1. f is a max flow
2. There are no augmenting paths wrt flow f; that is, no $s-t$ path in G_f
3. $val(f) = c(S, T)$ for some cut (S, T) ; hence this cut (S, T) must be a min (capacity) cut since $val(f) \leq c(S, T)$ for all cuts.

Hence the name max flow $(=)$ min cut

Comments on max flow - min cut theorem

- As previously mentioned, the Ford Fulkerson algorithms can be viewed as local search algorithms.
- This is a rather unusual local search algorithm in that any local optimum is a global optimum.
- Suppose we have a flow network in which all capacities are integral. Then :
  1. Any Ford Fulkerson implementation must terminate.
  2. If the sum of the capacities for edges leaving the source s is C, then the algorithm terminates in at most C iterations and hence with complexity at most $O(mC)$.
  3. Ford Fulkerson implies that there is an optimal integral flow. (There can be other non integral optimal flows.)

Good and bad ways to implement Ford Fulkerson

- There are bad ways to implement the networks such that
  1. There are networks with non rational capacities where the algorithm does not terminate.
  2. There are networks with integer capacities where the algorithm uses exponential (in representation of the capacities) time to terminate.
- There are various ways to implement Ford-Fulkerson so as to achieve polynomial time. Edmonds and Karp provided the first polynomial time algorithm by showing that a shortest length augmenting path yields the time bound $O(|V| \cdot |E|^2)$. For me, the conceptually simplest polynomial time analysis is the Dinitz algorithm which has time complexity $O(|V|^2|E|)$ and also has the advantage of leading to the best known time bound for unweighted bipartite matching. I think the best known worst case time for max flow is the preflow-push-relabel algorithm of Goldberg and Tarjan with time $O(|V| \cdot |E|\ \mathit{polylog}(|E|)$ or maybe $O(|V| \cdot |E|)$.

The Dinitz (sometimes written Dinic) algorithm

- Given a flow f, define the leveled graph $L_f = (V', E')$ where $V' = \{v | v \text{ reachable from } s \text{ in } G_f\}$ and $(u, v) \in E'$ iff $level(v) = level(u) + 1$. Here $level(u) =$ length of shortest path from s to u.
- A blocking flow $\tilde{f}$ is a flow such that every s to t path in L_f has a saturated edge.

The Dinitz Algorithm

Initialize $f(e) = 0$ for all edges e
While t is reachable from s in G_f (else no augmenting path)
    Construct L_f corresponding to G_f
    Find a blocking flow $\hat{f}$ wrt L_f and set $f := f + \hat{f}$
End While

The run time of Dinitz' algorithm

Let $m = |E|$ and $n = |V|$

- The algorithm halts in at most $n - 1$ iterations (i.e. blocking steps).
- The residual graph and the levelled graph can be computed in time $O(m)$ with breadth first search and using depth first search we can compute a blocking path in time $O(mn)$. Hence the total time for the Dinitz blocking flow algorithm is $O(mn^2)$
- A unit network is one in which all capaities are in $\{0,1\}$ a nd for each node $v \neq s, t$, either v has at most one incoming edge (i.e. of capacity 1) or at most one outgoing edge. In a unit network, the Dinitz algorithm terminates within $2\sqrt{n}$ iterations and hence on such a network, a max flow can be computed in time $O(m\sqrt{n})$ (Hopcroft and Karp [1973].

Application to unweighted bipartite matching

- We can transform the maximum bipartite matching problem to a max flow problem.
  Namely, given a bipartite graph $G = (V, E)$, with $V = X \cup Y$, we create the flow network $\mathcal{F}_G = (G', s, t, c)$ where
  - $G' = (V', E')$ with $V' = V \cup \{s, t\}$ for nodes $s, t \notin V$
  - $E' = E \cup \{(s, x) | x \in X\} \cup \{(y, t) | y \in Y\}$
  - $c(e) = 1$ for all $e \in E'$.

  .

Claim: Every matching M in G gives rise to an integral flow f_M in $\mathcal{F}_G$ with $val(f_M) = |M|$; conversely every integral flow f in $\mathcal{F}_G$ gives rise to a matching M_f in G with $|M| = val(f)$.

- Hence a maximum size bipartite matching can be computed in time $O(m\sqrt{n})$ using the Hopcroft and Karp adatpion of the blocking path algorithm.
- Similar ideas allow us to compute the maximum number of edge (or node) disjoint paths in directed and undirected graphs.

Additional comments on maximum bipartite matching

- There is a nice terminology for augmenting paths in the context of matching. Let M be a matching in a graph $G = (V, E)$. A vertex v is *matched* if it is the end point of some edge in M and otherwise if is *free*. A path π is an alternating path if the edges in π alternate between M and $E - M$.
- Abusing terminology briefly, an augmenting path (relative to a matching M) is an alternating path that starts and ends in a free vertex. An augmenting path in a graph shows that the matching is not a maximum and can be immediately improved.
- Clearly the existence of an augmenting path in a bipartite graph G corresponds to an augmenting path in the flow graph $\mathcal{F}_G$ used to show that bipartite matching reduce to flows.

The weighted bipartite matching problem

- Can the flow algorithm for unweighted bipartite matching be modified for weighted bipartite matching?
- The obvious modification would set the capacity of $< x, y > \in E$ to be its weight $w(x, y)$ and the capacity of any edge $< s, x >$ could be set to $\max_y\{w(x, y)\}$ and similarly for the weight of edges $< y, t >$.
- Why doesnt this work?
- It is true that if G has a matching of total weight W then the resulting flow network has a flow of value W.
- But the converse fails! Why?
- We will return to the weighted biparitite matching problem (also known as the *assignment problem*) discussing both the i optimal "Hungarian method formalize by Kuhn [1955] and attributed by Kuhn to Konig and Egevary [1931].
- We will see that this method is intimately tied to linear programming duality.
- We will also discuss the online matching problem and variants.

The metric labelling problem

We consider a problem well motivated by applications in, for example, information retrieval. (See Kleinberg and Tardos text)

The metric labelling problem

Given: graph $G = (V, E)$, a set of labels $L = \{a_1, \ldots, a_r\}$ in a metric space M with distance δ, and a cost function $\kappa : V \times L \to \Re^{\geq 0}$. The goal is to construct an assignment α of labels to the nodes V so as to minimize $\sum_{i \in V} \kappa(i, \alpha(i)) + \sum_{(i,j) \in E} p_{i,j} \cdot \delta(\alpha(i), \alpha(j))$

The idea is that κ represents a cost for labelling the node (e.g. a penalty for a bad classification of a web page), p represents the importance of that edge (e.g. where in a web page a particular link occurs) and δ represents the (basic or unweighted) cost of giving different labels to nodes that are related (e.g. the penalty for different labellings of web pages that are linking to each other or otherwise seem to be discussing similar topics.

The binary label case

- A case of special interest and the easiest to deal with is when the metric is the binary $\{0, 1\}$ metric; that is, $\delta(a, b) = 1$ if $a \neq b$ and 0 otherwise. (When there are only two labels, the binary $\{0, 1\}$ metric is the only metric.)
- The case of two labels suggests that the problem might be formulated as a min cut problem. Indeed this can be done to achieve an optimal algorithm when there are only two labels.
- For more than two labels, the binary metric case becomes NP hard but a there is a 2-approximation via a local search algorithm that uses min cuts to search a local neighbourhood.

The case of two labels

- The problem for two labels can be restated as follows: find a partition $V = A \cup B$ of the nodes so as to minimize $\sum_{i \in A} b_i + \sum_{j \in B} a_j + \sum_{(i,j) \in A \times B} p_{i,j}$
- We transform this problem to a min cut problem as follows: construct the flow network $\mathcal{F} = (G', s, t, c)$ such that
  - $G' = (V', E')$
  - $V' = V \cup \{s, t\}$
  - $E' = \{(u, v) | u \neq v \in V\} \cup \{(s, u) | u \in V\} \cup \{(u, t) | u \in V\}$
  - $c(i, j) = c(j, i) = p_{i,j}$; $c(s, i) = a_i$; $c(i, t) = b_i$

Claim:

For any partition $V = A \cup B$, the capacity of the cut $c(A, B) = \sum_{i \in A} b_i + \sum_{j \in B} a_j + \sum_{(i,j) \in A \times B} p_{i,j}$.

-

Flow networks with costs

We now augment the definition of a flow network $\mathcal{F} = (G, s, t, c, \kappa)$ where $\kappa(e)$ is the non negative cost of edge e. Given a flow f, the cost of a path or cycle π is $\sum_{e \in \pi} \kappa(e) f(e)$.

MIn cost flow problem

Given a network $\mathcal{F}$ with costs, and given flow f in $\mathcal{F}$, the goal is to find a flow f of minimum cost. Often we are only interested in a max flow min cost.

- Given a flow f, we can extend the definition of an augmenting path in $\mathcal{F}$ to an augmenting cycle which is just a simple cycle (not necessarily including the source) in the residual graph G_f.
- If there is a negative cost augmenting cycle, then the flow can be increased on each edge of this cycle which will not change the flow (by flow conservation) but will reduce the cost of the flow.
- A negative cost cycle in a directed graph can be detected by the Bellman Ford DP for the single source shortest path problem.

An application of a min cost max flow

- We return to the weighted interval scheduling problem (WISP) .
- We can optimally solve the m machine WISP using DP but the time for this algorithm is $O(n^m)$.
- Using the local ratio (i.e. priority stack) algorithm we can approximate the optimal within a factor of $2 - \frac{1}{m}$.
- Arkin and Silverberg [1987] show that the m machine WISP can be reduced efficiently to a min cost problem resulting in an algorithm having time complexity $O(n^2 \log n)$.
- The Arkin and Silverberg reduction is (for me) a subtle reduction which I will sketch.

The Arkin-Silverberg reduction of WISP to a min cost flow problem

- The reduction relies on the role of maximal cliques in the interval graph.
- An alternative characterization of an interval graph is that each job (i.e. interval) is contained in consecutive maximal cliques.
- The goal will be the crreate a flow graph so that a min cost flow (with value = maximum size clique - m) will correspond to a minimum weight set of intervals whose removal will leave a feasible set of intervals (i.e. can be scheduled on the m machines).
- If $q_1, \ldots, q_r$ are the maximal cliques then the (directed) flow graph will have nodes $v_0, \ldots, v_r$ and three types of edges:
  1. (q_i, q_{i-1}) representing the clique q_i with cost 0 and infinite capacity
  2. If an interval J_i occurs in cliques $q_j, \ldots, q_{j+\ell}$, there is an edge $(v_{j-1}, v_{j+\ell})$ with cost w_i and capacity 1.
  3. For each clique q_i not having maximum size, we have an edge (v_{i-1}, v_i) of cost 0 and capacity maximum clique size - size of q_i. (These can be thought of as "dummy intervals".)

Example of interval graph transformation

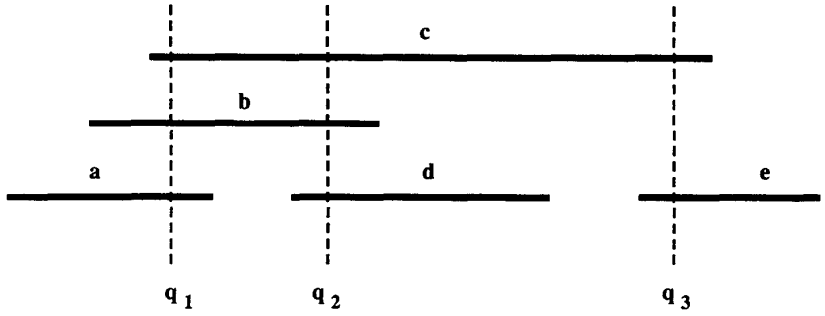

Fig. 1a. The jobs.

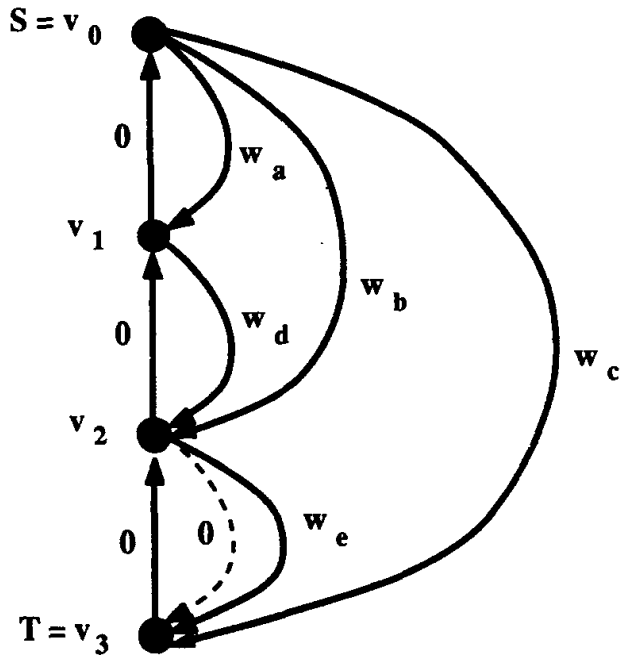

Fig. 1b. The directed graph.

Note: The dotted arc represents a dummy job.