RESOURCE ALLOCATION AND ECONOMICAL ANALYSIS

ABHAY SHANKAR K

1. Theoretical optimality

We are given a static network in the form of a graph $\mathcal{G} = (V, \mathcal{E})$, and a demand vector $\mathcal{D}$. The demand vector is a mapping from the set of all pairs of nodes in the graph to a non-negative integer.

Each edge is a 3-tuple (c, τ, Γ) with

- c being the cost of the link
- τ being the cooldown time
- Γ being the capacity of the link - the number of bell pairs one can send across.

Each element of the demand vector is a 3-tuple (u, v, d), with

- α: The demand number, to uniquely identify each demand.
- u: source node
- v: destination node
- d: the number of bell pairs to be sent

Objective: We seek to find a routing protocol that minimizes the cost of the network, while satisfying the demand.

Secondary Objective : We also wish to find the theoretically optimal routing scheme, i.e. an Allocation function, to serve as a benchmark for subsequent practical algorithms.

1.1. **Allocation function.** .

We may interpret the problem of optimal network utilisation as assigning each edge to a given set of demands for a time slot, i.e., finding a function $f : T \times \mathcal{E} \to \mathcal{P}(\mathcal{D} \times \mathbb{N})$ that minimises the cost

$$\mathcal{C} = \sum_{t \in T} \left(\sum_{e \in E'(t)} c\,(e) \right)$$

where

- $E'(t) = \{e \in \mathcal{E} \mid f(t, e) \neq \{\}\}$ is the set of all edges that are active at time t
- $T \subset \mathbb{N}$ is the set of all time slots during which the network operates.
- The output of the allocation function is a set of (α, n) pairs, which specify the number of qubits (n) of which demand (α) the edge should transmit. This corresponds to $((\alpha_1, \gamma_1)..(\alpha_k, \gamma_k))$.
  - Here $\forall k\, \gamma_k < \alpha_k$`.demand`, i.e. the edge cannot transmit more qubits than the demand,
  - $\Sigma_k \gamma_k \leq \Gamma$ for a given edge,
  - Flow conservation: $\forall t \in T\ \forall u \in V\ \forall g \in \mathcal{E}_u\ \Sigma_g s(g) f(t, g) = 0$ where $\mathcal{E}_u$ is the set of edges incident on u and $s(g)$ is 1 for outgoing edges and -1 for incoming edges.
  - α_k is a valid demand.

To elaborate, in each timeslot, a link/edge $e(c, \tau, \Gamma)$ transmits $\gamma \leq \Gamma$ qubits, each of which accrues a cost c. Each qubit belongs to a certain demand, as qubits are only transmitted if there is a demand. We specify which demands are catered to bby any given link in each timeslot, and how many qubits of that demand are transmitted.

While the allocation function method is indeed optimal in the number of demands serviced, it may result in starvation - a single demand may be delayed indefinitely in favour of other, smaller demands. Thus, we may choose to optimise other quantities (see below).

This is a linear programming problem, and can be solved using standard techniques. However, in the interest of efficiency, it may be fruitful to pursue alternative regimes.

1.2. **Routing.** .

We may also approach this problem from the viewpoint of classical networks. We shall consider peering models now and discuss hierarchial models later.

Each vertex v in $\mathcal{G}$ maintains a vector of routing information, viz. the routing vector $\mathcal{R}_v$, which it uses to create quantum links. When a request for routing arises (i.e. an entangled qubit and a destination address), the node consults it's routing vector and 'forwards' the qubit onto the next node in its path, by a bell-state swap.

The construction of the routing vector may be handled in several ways of increasing complexity and efficacy. The simplest is to use a shortest path algorithm, (Dijkstra), and compute te shortest paths betwen every pair of nodes in the network. While the routing vector may change in each timeslot, it need not be recomputed.

This is ineffctive, as several topologies may attempt to funnel all data through a single link, causing queueing delays due to the severe bottleneck. While this is no flaw if we seek only to minimise the cost, it becomes pertinent for other quantites we may wish to optimise, as mentioned later.

This may be solved using a slight modification of Dijksta's shortest path algorithm. The algorithm calculates all the shortest paths from a given node to all other nodes in the network - while it is doing so, we may also calculate the capacity of each of these paths, a divide the demand between them. This redueces the load on any one path, and reduces the queueing delays, but does not guarantee it's elimination.

Further modifications may introduce a new mertic combining both the cost and the capacity of an edge, and use this instead of the costs when calculating shortest paths. Suggestions for such metrics include their ratio, $\frac{c}{\Gamma}$, or a saturation test: 0 if the edge is saturated, c otherwise. The latter approach assumes a knowledge of local and global network state, which may be obtained by a centralised controller, or by a distributed algorithm.

1.3. **Note about omniscience.** Given all the tasks a machine must perform in the future, it may compute a priori the optimal resource-allocation function, and use it to perform those tasks without any additional computational overhead.

An example of this is the SJF (shortest job first) scheduling algorithm, which is optimal for minimizing the average waiting time of a set of tasks but requires foreknowledge of their duration.

In real networks, such knowledge is unavailable, and this solution is inapplicable. However, it is useful to consider this case as a benchmark for the performance of other algorithms.

The allocation function ~~appears to be~~ is of this form, and cannot be utilised in case of a dynamic demand vector.

1.4. **Optimisation Alternatives.** Alternatively, we could also choose to optimise any of the following quantities, or establish relationships between their optimal values:

- Minimise the total time taken, i.e. $|T|$
- Maximise the utilisation of the network, i.e. $\dfrac{\sum_{t\in T} |E'(t)|}{|\mathcal{E}|\,|T|}$
- Minimise the maximum utilisation of the network, i.e. $\max\limits_{t\in T} \dfrac{|E'(t)|}{|\mathcal{E}|}$, e.g. congestion control.
- Maximise the minimum utilisation of the network, i.e., $\min\limits_{t\in T} \dfrac{|E'(t)|}{|\mathcal{E}|}$, e.g. load balancing.

or even some other metric built from these.

1.5. **Assumptions.** Our assumptions are as follows

- Constant classical topology, i.e. quantum links are consumed and revived but classical links are fixed.
- Constant edge properties - i.e. cost, capacity, cooldown time do not change with time.
- Static demand vector, without duplicate elements between the same pair of nodes.
- Discretized times slots, with instantaneous operations occuring within a timeslot in a well-defined oorder.
- Measurement Infallibility: The bell-pair measurement never fails.
- Link Infallibility: The quantum links between two nodes do not fail.
- Synchronicity: All measurements are instantaneous, and occur simultaneously relative to a synchronised global clock.

In the following sections, we shall relax several of these assumptions, progressing towards a more practical model.

2. Dynamic demands

2.1. **Allocation function.** .

We now consider the case where the demand vector is not static, but changes over time. We will assume that the demand vector is a function

$$\mathcal{D} : T \times V \times V \to \mathbb{N}$$

and that the demand at time t from node u to node v i.e. $\mathcal{D}(t, u, v)$ is known at t but not at t - 1 $\forall t \in T, u, v \in V$.

The cost function remains the same, but the allocation function (our potential objective) needs to be recomputed at each time instance or each time the demand vector changes.

2.2. **Routing.** Alternatively, if we use a routing scheme, we approach a paradigm more resembling classical networks. Our discussion regarding routing in the previous section is still applicable, except for the recomputation costs due to changing topologies.

We can thus consider an algorithm that does not recompute the entire routing vector, but only those impacted by a given topology change. We may also consider distance-vector routing algorithms, where fewer routers must recompute their routing vectors as fewer routers possess information regarding the altered topology.

2.3. **Oracular dynamic demands.** As an intermediate between truly dynamic demand vectors and a static demand, we can consider a demand matrix $\mathcal{D}$ where the i'th row represents the demand vector at time i. We can then use the same model as before - no changes are necessary. This means that $\forall tu, v\mathcal{D}(u, v, t)$ is known at $t = 0$.

3. Fallible links

This is a Link-layer concern, and independent of the routing protocol. Thus, we will discuss this elsewhere.