Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
Electronic Notes in Theoretical Computer Science 141 (2005) 97–114
www.elsevier.com/locate/entcs

Interactions in Transport Networks

Nigel Walker[1] Marc Wennink[2]

BT Research,
B54, 141, Adastral Park,
Ipswich, IP5 3RE, UK

Abstract

We present a model that captures basic interactions occurring in transport networks, including routing and flow control. Many network processes can be seen as solving an optimisation problem, or seeking a balance between competing interests. The problem structure is illustrated by means of a 'component graph', which dictates the communication and interaction patterns between different parts of the system. We show how the same formalism also captures interactions in electrical circuits.

Keywords: Optimisation, routing, Lagrangian duality, interaction, graphical models.

1 Introduction

We are interested in developing techniques to model and specify processes and interactions occurring in communications networks, primarily transport networks, but ultimately other components of shared infrastructure, much of which also incorporates elements of network functionality. In this setting interactions can take place across many different 'axes' (between different users, between users and operators, between nodes or between layers of the network, between flows and costs), and over quite different timescales. Each process or interaction takes place within a larger spatial and temporal environment.

Many network processes can be seen as solving some kind of optimisation problem (e.g. minimum cost routing) or, more generally, as seeking a bal-

[1] Email: nigel.g.walker@bt.com
[2] Email: marc.wennink@bt.com

doi:10.1016/j.entcs.2005.05.018

ance between competing interests (e.g. sharing available capacity). From a computational point of view, the problem is to find the values of parameters (variables) in the system that achieve such an optimum, or balance point. This must be done dynamically, and often as a distributed calculation, in response to changes in the environment. For example, a routing protocol must continually respond to changes in availability of links, adjusting flows accordingly.

Although we do not report on it directly in this paper, we believe there is much value to be gained in developing high level (programming) languages to analyse and organise network functionality and structure. Advantages we anticipate from such an approach include better statement of management issues, exposure of options for refinement into different protocols, comparison of different design options within a common language framework, and precision about aspects such as the amount of network state, numbers of variables, and naming structure. Here we restrict our attention only to a model of interaction that should underpin such a language. We draw on the mathematics of optimisation, which puts our work in parallel with other recent work using optimisation theory in the design and analysis of networks and protocols [9,11].

An outline of our general approach is as follows. A network task for which a distributed solution is sought is first formulated as a mathematical optimisation problem, involving an objective function and some constraints. The problem is then relaxed by incorporating the constraints into the objective function. The modified objective function is called the *Lagrangian*, and is standard in optimisation theory [4]. It depends on the variables of the original objective function, and an additional set of *dual* variables which quantify the cost of violating the constraints. The problem then becomes to find a saddle point of this new function, rather than a minimum or maximum. The Lagrangian can usually be written as the sum of different components, and its structure can be described as a graph highlighting the connections between these components. This graph illuminates the structure of the original problem in several respects: it allows for *decomposition*, where different parts of the graph are considered as subproblems; these subproblems can then be *distributed* by allocating them to different nodes or processors in the network; it specifies the *communication* channels that must be supported between the subproblems; it suggests *distributed algorithms*; and its structure exposes different axes of *interaction* and *feedback* in the system. Different decompositions of the component graph lead to different algorithms, and in this way the design space for solving the original problem is laid out.

We illustrate these points through an example based on shortest path routing in a simple network. We present the Lagrangian for this problem,

and show how the decomposition procedure leads to a distributed version of the Bellman-Ford algorithm. We also present a second algorithm, based on a dynamic system equation, that converges to a saddle point for any differentiable Lagrangian satisfying a *convex-concave* property. We then extend our shortest path example to show how interactions between routing flow control can be studied in this framework.

Many networking problems of practical importance, including the routing and flow control problems which we discuss in this paper, can be captured by a convex-concave Lagrangian. However, some combinatorial problems do not have this property, and we discuss the implications of this limitation in section 7.

2 Saddle points and duality

As just mentioned, the general mathematical setting for our model is actually more general than optimisation. We study the problem of finding a *saddle point* of a convex-concave function of typically many real-valued variables. We usually call this (real-valued) function, L, the *Lagrangian*, on the basis that it can often (but not always) be derived from a Lagrange relaxation of an optimisation problem. In this section we first present general saddle point conditions, then specialise these to smooth approximations of Lagrangian functions derived from linear programs.

The arguments of L are separated into *primal decision variables*, $x = (x_1, \ldots, x_n)$ and *dual decision variables*, $y = (y_1, \ldots, y_m)$. A Lagrangian, L, defined over a domain $X \times Y$ is *convex-concave* if and only if

- for any $x_1 \in X$, $x_2 \in X$, $y \in Y$ and $0 \leq \alpha \leq 1$ we have $\alpha x_2 + (1-\alpha)x_1 \in X$, and $L(\alpha x_2 + (1-\alpha)x_1, y) \leq \alpha L(x_2, y) + (1-\alpha)L(x_1, y)$
- for any $y_1 \in Y$, $y_2 \in Y$, $x \in X$ and $0 \leq \beta \leq 1$ we have $\beta y_2 + (1-\beta)y_1 \in Y$, and $L(x, \beta y_2 + (1-\beta)y_1) \geq \beta L(x, y_2) + (1-\beta)L(x, y_1)$

A point (x^*, y^*) is a saddle point of $L(x, y)$ if

$$L(x^*, y) \leq L(x^*, y^*) \leq L(x, y^*) \qquad \forall x \in X,\ \forall y \in Y. \tag{1}$$

A saddle point can be interpreted as an equilibrium configuration for a game in which players are associated with variables. The primal variables try to minimise the value of the Lagrangian, and the dual variables try to maximise it. Mutually conflicting interests reach an accommodation in a saddle point.

The Lagrangian corresponding to a linear program is convex-concave, and many network flow problems can be stated as a linear programs [1]. Let A be an $m \times n$-matrix, b and y be m-vectors, and c and x be n-vectors. A linear

program of the form

$$\min cx \quad \text{s.t. } Ax = b,\ x \geq 0 \tag{2}$$

has a corresponding Lagrangian function

$$L(x, y) = cx - yAx + yb, \quad x \geq 0. \tag{3}$$

If (x^*, y^*) is a saddle point of $L(x, y)$ in (3) then x^* is an optimal solution to the corresponding linear program (2).

More generally, a convex-concave Lagrangian can be associated with *any* convex optimisation problem, not just those formulated as linear programs. Saddle points for a Lagrangian derived in this way are characterised by the well-known Karush-Kuhn-Tucker conditions [4].

In section 4 we will present dynamic system equation and an algorithm that requires the Lagrangian to be differentiable in the region of its saddle point. Even if the Lagrangian function appears to be differentiable, this condition can still fail if the saddle point lies on the border of its domain. For example, this would be the case if one of the non non-negativity conditions $x \geq 0$ in (3) was tight (i.e. for some component x_i , $x_i = 0$). To circumvent this problem we can introduce *barrier functions*. The idea is to add a (differentiable) component to the Lagrangian which has high value as x approaches the boundary of its domain. Then the saddle point of the modified Lagrangian will lie at some finite distance from the domain boundary. This expedient parallels that used in *interior point methods* [4], which similarly add barriers directly to the objective function of an optimisation problem (rather than its Lagrangian) in order to ensure that it is differentiable in the region of its minimum (or maximum). A favourite choice is a logarithmic barrier function. Thus a constraint $z > 0$ would be respected by adding a component $-\epsilon \ln(z)$. By choosing ϵ sufficiently small for each such barrier function, the solution of the modified problem (the values of the decision variables and the value of the objective function) approximates that of the original problem arbitrarily accurately. Interior point methods proceed by finding the minimum objective for a sequence of values of ϵ until the desired accuracy is obtained. We work with fixed barrier functions, and with this modification (3) becomes

$$L(x, y) = cx - yAx + yb - \sum_{j=1}^{n} \epsilon_j \ln(x_j), \quad x > 0. \tag{4}$$

where $\epsilon_j > 0$, $j = 1 \ldots n$. Now it is sufficient to require that

$$\frac{\partial L}{\partial x_j} = 0, \quad j = 1, \ldots, n \qquad \frac{\partial L}{\partial y_i} = 0, \quad i = 1, \ldots, m \tag{5}$$

at (x^*, y^*) for this to be a saddle point. As the values of ϵ_j are reduced, the saddle point of (4) more closely approximates that of (3). If we had started

with a linear program of the form

$$\min cx \quad \text{s.t. } Ax \geq b,\ x \geq 0 \tag{6}$$

instead of (2), then the Lagrangian (3) would also be restricted to $y \geq 0$ and barrier functions $+\delta_i \ln(y_j)$, $i = 1, \ldots, m$ would be added to (4).

3 Decomposition and distribution

To explain how a saddle point problem can be distributed we introduce a running example. We seek the shortest paths to a given destination in a communication network, which can be formulated in the standard way as a minimum cost flow linear program, with Lagrangian of the form (3). The flow over link j is determined by a primal decision variable x_j, while the dual variable y_i becomes the distance (or cost) from node i to the destination node. A cost of c_j per unit flow is imposed at link j, and flow b_i is injected into node i. We can choose $b_i = 1$ if i is one of the $m-1$ ingress nodes, and $b_i = 1-m$ to sink all the flow at destination node i. The matrix A is the incidence matrix of the network; $A_{ij} = 1$ if node i is a source of link j, $A_{ij} = -1$ if node i is a target of link j, and $A_{ij} = 0$ otherwise. The network is assumed connected. The solution is degenerate in the dual variables, which is normally handled by requiring the destination node to set $y_i = 0$.

The Lagrangian function can be written as a sum of separate components, the structure of which can be presented graphically. We show this for a small 5-node (=m), 7-link (=n) network in Fig. 1. Primal variables are enclosed in circles and dual variables in squares. A component of the Lagrangian function is represented by a blob, which is connected to the variables on which it depends. Each link in the communication network corresponds to a column of the incidence matrix. For example, in Fig. 1, x_a participates in two components $-y_1 x_a$ and $+y_2 x_a$ deriving from the incidence matrix, together with cost component $c_a x_a$, and an explicit log-barrier component, $-\epsilon_a \ln(x_a)$, discussed above, which we represent by an open circle. This graphical presentation is related to factor graphs in belief propagation networks, Tanner graphs in decoding theory, and constraint graphs in constraint programming [2,3]. We use the name 'component graph' to emphasise that it derives from a straightforward translation of the Lagrangian function. A nice feature is that, for this problem formulation, it naturally reflects the underlying network topology. In section 5 we will see that the same holds for component graphs derived from electrical circuits.

We can distribute the problem of finding a saddle point by partitioning the component graph. Figure 1 shows such a partitioning for the shortest path problem. Each node 'owns' the variables in its shaded region: one primal

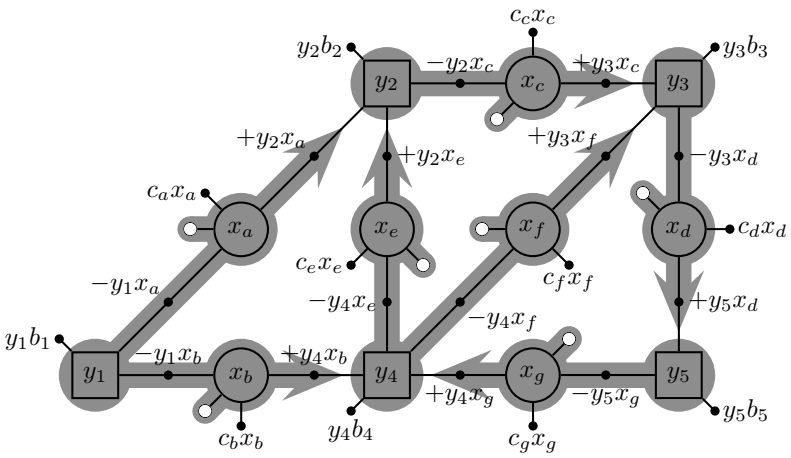

Fig. 1. Component graph for shortest path routing, showing underlying network structure, and decomposition for Bellman-Ford algorithm.

variable for each out-link and one dual variable for the distance label. For each node, we can obtain a locally perceived Lagrangian by collecting all the components that involve any of the variables owned by that node. For node 4, for example, we find

$$L_4(x_e, x_f, y_4) = (c_e + \underline{y_2})x_e + (c_f + \underline{y_3})x_f - y_4 x_e - y_4 x_f + y_4(b_4 + \underline{x_b} + \underline{x_g}) - \epsilon_e \ln(x_e) - \epsilon_f \ln(x_f) \tag{7}$$

where we use the notation $\underline{y_i}$ and $\underline{x_j}$ to indicate that a variable is owned by another node. Thus, from the point of view of node 4, y_2 is perceived as environmental. A locally perceived Lagrangian changes whenever neighbouring nodes change the values of their decision variables. If a node has found a saddle point of its local Lagrangian, the derivatives (5) with respect to the variables it owns are zero. Because every decision variable in the global Lagrangian is owned by exactly one node, it follows that the network as a whole is in a saddle point if and only if all nodes are simultaneously in a saddle point of their local Lagrangian. Each local saddle point problem is formally similar to the global saddle point problem, and the decomposition into sub-problems can, in general, be continued recursively.

The above discussion suggests a straightforward distributed approach to finding the network saddle point: after a node receives new information about the values of its neighbours' variables, it solves its local problem and in turn communicates the newly selected values of its own variables to the relevant neighbouring nodes. In the case of the shortest path problem, this procedure converges to a solution. The operations performed by node 4 in our example network can be summarised as follows

- maintain a table of the distance labels y_2 and y_3 received from downstream neighbours, nodes 2 and 3.
- maintain a table of flow values x_b and x_g received from upstream neighbours, nodes b and g.
- Periodically perform the following
  - update its distance label to be the least of the costs of sending the flow over each out-link. Thus $y_4 = \min(y_2 + c_e, y_3 + c_f)$.
  - send all the inflow over the out-link which offered the least cost. If this was link f then set $x_f = x_b + x_g$ and $x_e = 0$, if it was link e then set $x_e = x_b + x_g$ and $x_f = 0$.
  - Send the new distance label y_4 to upstream nodes 1 and 5.
  - Send the flow values x_e and x_f to the downstream neighbours, nodes 2 and 3 respectively.

The other non-destination nodes perform similar operations. If we ignore the the values of the flow variables, and record only which out-link is used, then these operations recreate a distributed asynchronous version of the Bellman-Ford algorithm, which is at the heart of distance vector protocols, such as RIP, widely used in the Internet [8].

The procedure described above, in which it is implied that the variables owned by an individual node (or 'sub-process') are adjusted instantaneously to the locally perceived saddle point, does not necessarily converge to a global saddle point for an arbitrary Lagrangian. An obvious expedient to fix this is instead to adjust the values of the decision variables incrementally. This line of reasoning leads to a dynamic system equation, described in the next section, in which the decision variables converge to a global saddle point for any differentiable strictly convex-concave Lagrangian.

4 Evaluation as dynamic system

Here we assume that the values of the variables change continuously, and at a rate such that the propagation delay incurred when exchanging messages can be ignored. We discuss these assumptions later. It is also assumed that the Lagrangian is at least once differentiable. The following dynamic equation can then be motivated as an intuitive method of finding the saddle point, i.e. the minimum of the Lagrangian with respect to x and the maximum with respect to y,

$$\frac{dx}{dt} = -\lambda \,.\, \nabla_x L, \quad \frac{dy}{dt} = +\mu \,.\, \nabla_y L, \tag{8}$$

where $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ are vectors containing the values of the decision variables, $\lambda \in \mathbb{R}^{n \times n}$ and $\mu \in \mathbb{R}^{m \times m}$ are positive definite matrices, usually assumed diagonal and $\nabla_x L$ and $\nabla_y L$ are the vectors of partial derivatives of L with respect to the primal and dual variables respectively. In the rest of this section present a Lyapunov function that proves convergence of this dynamic equation, and then discuss a message passing algorithm motivated by it.

Assume that the Lagrangian is strictly convex-concave. Then it has a unique saddle point (x^*, y^*). We would like to know that a solution $(x(t), y(t))$ of (8) converges to (x^*, y^*) as $t \to \infty$. We can do this by constructing a Lyapunov function. Assume the saddle point (x^*, y^*) of the Lagrangian is at the origin $(0, 0)$ and define

$$\phi = \frac{1}{2}\left(x^t \,.\, \lambda^{-1} \,.\, x + y^t \,.\, \mu^{-1} \,.\, y\right) \tag{9}$$

where x, y and hence ϕ are time varying quantities, and x^t denotes the transpose of x. Note that λ and μ are positive definite matrices, so can be inverted. Then

$$\frac{d\phi}{dt} = -x^t \,.\, \nabla_x L + y^t \,.\, \nabla_y L \tag{10}$$

If $L(x, y)$ is once differentiable then, on account of its strict convex-concavity

$$\begin{aligned} L(x_2, y) - L(x_1, y) &> (x_2 - x_1)^t \,.\, [\nabla_x L](x_1, y) \\ L(x, y_2) - L(x, y_1) &< (y_2 - y_1)^t \,.\, [\nabla_y L](x, y_1) \end{aligned} \tag{11}$$

for all x, y and $x_1 \neq x_2$, $y_1 \neq y_2$ [4]. Setting $x_1 = x$, $y_1 = y$, $x_2 = x^* = 0$ and $y_2 = y^* = 0$ in (11) and substituting in (10) gives

$$\frac{d\phi}{dt} < L(x^*, y) - L(x, y^*) \leq 0 \tag{12}$$

where the second inequality follows from the definition of a saddle point (1). The function $\phi(x, y)$ decreases along trajectories, yet is non-negative, and therefore constitutes a suitable Lyapunov function. If we are given a Lagrangian that is convex-concave, but not strictly so, then it may be possible to modify it so that the above result is applicable. For example, the Lagrangian of (4) can be made strictly convex-concave by subtracting a term yMy, where M is a small positive definite matrix.

To design a protocol that finds the saddle point in a distributed system, we could try to arrange the messages and calculations so that the decision variables follow the dynamic system equations (8), at least approximately. This can be done through a simple protocol in which messages are (visualised as) passed in both directions 'over the component graph', as illustrated in Fig. 2. The protocol does not distinguish between primal variables or dual variables, so the symbol z is used for either. Here $V(k)$ is the set of suffixes of

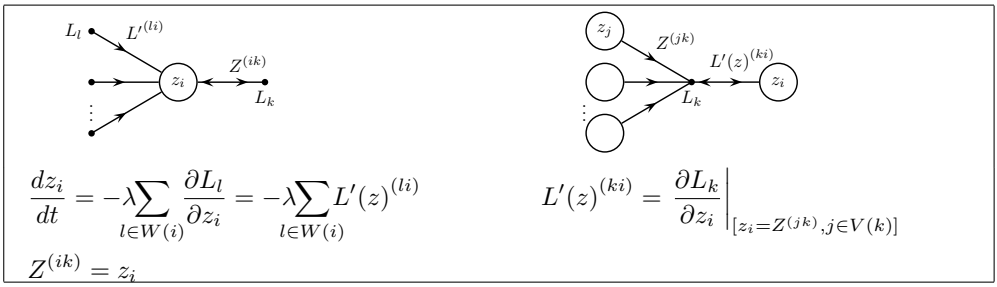

Fig. 2. Message passing rules to implement dynamic system evaluation strategy of equation (8).

the variables participating in component L_k, and $W(i)$ is the set of suffixes of the components that depend on z_i. A message $Z^{(ik)}$ sent from variable node i to Lagrangian component L_k carries the (most recent) value of the decision variable z_i, and a message $L'^{(ki)}$ sent from component k to variable i is the partial derivative $\partial L_k / \partial z_i$ evaluated at the most recently received $z_j = Z^{(jk)}$, $j \in V(k)$. The variable node i maintains a value for z_i which is adjusted in accordance with the local dynamic equation shown in Fig. 2. The matrices λ and μ are assumed diagonal. The component node k must calculate the gradient of function L_i with respect to all the z_i, $i \in V(k)$.

As always when designing a discrete system to emulate, or approximate, a smooth dynamics, the question arises as to what conditions ensure the approximation is accurate. For equations (8) two groups of parameters need to be considered:

- The values of λ and μ, which determine the rates at which the decision variables are adjusted. These have to be chosen sufficiently small so that propagation delay of messages can be neglected.
- The frequency of sending messages. This must be sufficiently large compared with the rate of adjustment of the decision variables.

The question of determining upper limits for λ and μ, or lower limits on the frequency of sending messages is complicated in general, and is the province of control theory, dynamic systems analysis, and sampling theory.

Significantly, it is possible to 'derive' the Bellman-Ford algorithm from this generic protocol. Consider the messages exchanged between nodes 1 and 2 in the shortest path routing problem of Fig. 1. Let $L_k = y_2 x_a$. Node 2 sends $Z^{(2k)} = y_2$, i.e. its distance label, to its upstream neighbour and gets in return $L'^{(k2)} = \partial L_k / \partial y_2 = x_a$, the value of the flow variable. Similarly for the messages exchanged between other nodes. Now reduce the rate at which messages are sent *between* nodes, while maintaining a high rate of message passing, and rate of adjusting variables, *within* a node. Each node then ap-

pears to adjust the values of its variables as a *transition* when observed at the timescale commensurate with messages passed between nodes. This gives the version of the Bellman-Ford algorithm discussed in section 3. Evidently, exchange of messages according to rules such as those in Fig. 2 can yield useful algorithms, even if the assumptions of the previous paragraph are broken.

Performing the Bellman-Ford calculations within each node explicitly using the dynamic system message passing equations is computationally expensive: the calculations can be performed more efficiently using the operations described in section 3. However, we have been experimenting with an implementation of a prototype language, and have coded this particular scenario as it illuminates the middle ground between a transition semantics and a dynamic system semantics of the computation over the component graph. At one extreme the propagation delay between nodes is completely hidden (communication is assumed to be instantaneous), and the dynamics are manifest as a continuous evolution of the system variables. In the other case the evolution of the variables is completely hidden (computation is assumed instantaneous), and the dynamics are manifest as a sequence of propagation delays followed by state transitions. In our coding experiments both the propagation delay between nodes and the computation time within a node were exposed.

The general protocol described above, and illustrated in Fig. 2, has strong similarities with so-called 'message passing algorithms' from decoding and information theory [2], and constraint propagation algorithms from constraint programming [3]. We arrived at the rules in Fig. 2 through a metamorphosis of the 'min-plus' algorithm described in [2], though there the messages exchanged between nodes are functions, rather than values. In constraint programming the messages correspond to reading or writing constraints to a store.

5 Analogy with electrical circuits

It is worth emphasising a formal correspondence between electrical networks and communication networks, as this provides considerable scope for the transfer of concepts such as impedance, passivity, small signal analysis, frequency domain techniques, etc. as well as the notion of interaction, from one setting to the other. It is routine for electrical engineers to characterise the behaviour of circuits differently at different frequencies, or over different timescales, whereas this style of thinking appears to be less thoroughly exploited in reasoning about communications networks and protocols.

Figure 3 shows a simple LCR circuit using conventional electrical symbols, and its translation into a component graph and associated Lagrangian function. The flow (current) and potential (voltage) variables may now be

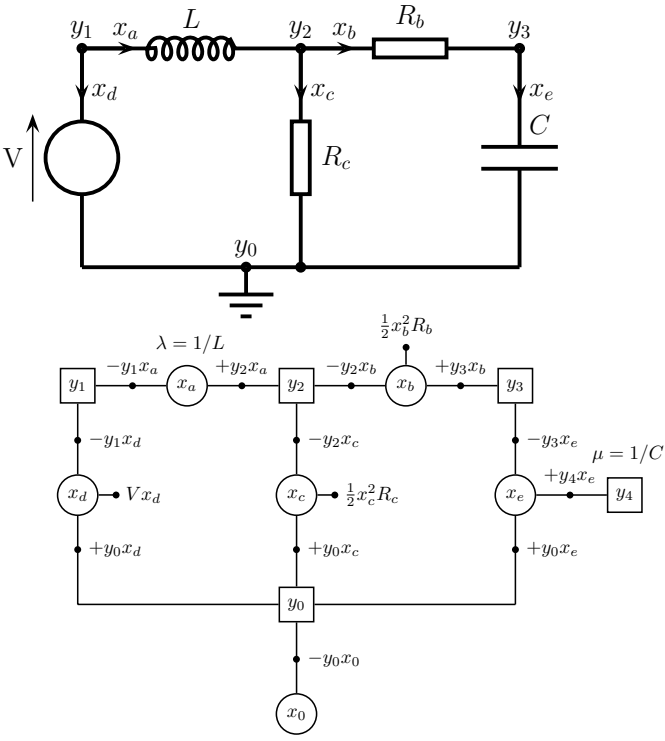

Fig. 3. Simple LCR circuit and its component graph

either positive or negative, so no barrier functions are required. The main structure of the component graph is still determined by the incidence matrix. A resistor gives rise to a quadratic 'cost' component on the link flow variable. A link inductance L associates $\lambda = 1/L$ with a flow variable, and a capacitor C gives $\mu = 1/C$ for a potential variable. The corresponding λ's and μ's for the remaining variables are assumed to be high. In other words, they are parasitic capacitances or inductances. The interpretation of λ and μ as reciprocal inductance and capacitance gives an energy interpretation to the Lyapunov function (9). The implication of the circuit diagram is that the parasitic modes of oscillation can be ignored—they are assumed to be 'out of band', and to decay quickly—so that all the interesting dynamics are determined by the values of the explicitly indicated inductance and capacitance. Although we do not present the details, the procedure by which these parasitic modes are eliminated, thereby recovering the circuit equations that would conventionally be associated with the circuit diagram, is standard in dynamic systems theory [10].

This example also illuminates the assumption concerning propagation delay outlined in the previous section. It is a standard 'lumped circuit' treatment

of an electrical circuit. The diagram of Fig. 3 implies that the delay can be ignored at the frequencies that are of interest. Outside of this regime, a propagation delay must be made explicit by including a waveguide in the circuit, and performing a transmission line analysis.

6 Congestion routing

We now extend our running network example, 'zooming out' to include more of the environment. We no longer focus on flow to a single destination, but consider three different flows, $\alpha = i, \ldots, iii$, each with a different destination. The injected flow levels are no longer constant, but determined by variable demands d^α. We use utility functions $u^\alpha(d^\alpha)$ to capture the users' appetite for sending flow. Also, the cost of each link has a variable component, associated with the level of congestion on that link. This variable, p_j, depends on the capacity k_j of the link and the total flow it carries. (We could have zoomed out even further and included provisioning within the model, promoting the capacities themselves to become dynamic variables.)

The Lagrangian for this scenario is shown as a component graph in Fig. 4. It can be related to a variant of the multi-commodity flow *optimisation* problem, but here we want to use it to expose various types of *interactions* within transport networks.

As drawn, Fig. 4 emphasises the interaction between demands (right), chosen routes (middle), and congestion levels (left). When demands increase, the congestion levels on the shortest paths will increase, forcing the routing process to find alternative paths. In most transport networks, routing and congestion control are not combined as directly as this because delay can lead to instabilities, so-called 'route-flapping'. The usual practice is to decouple flow control from routing, as is the case in TCP/IP for example [8]. However, this decoupling can be illusory. Observed over a long enough timescale these interactions *do* occur.

An alternative lay-out of the component graph can be created to emphasise the interaction between the different nodes in the network. Each variable is associated with one of the nodes. For example, we can associate the demand variable d^α with the source node of flow α and the congestion variable p_j with the source node of link j. As in the case of the Bellman-Ford decomposition in Fig. 1, we can then identify the local problems that have to be solved by each node and the patterns of communication that have to be established between the nodes.

A third decomposition is obtained by grouping the variables involved in the demand and routing processes by flow type. This view emphasises the interac-

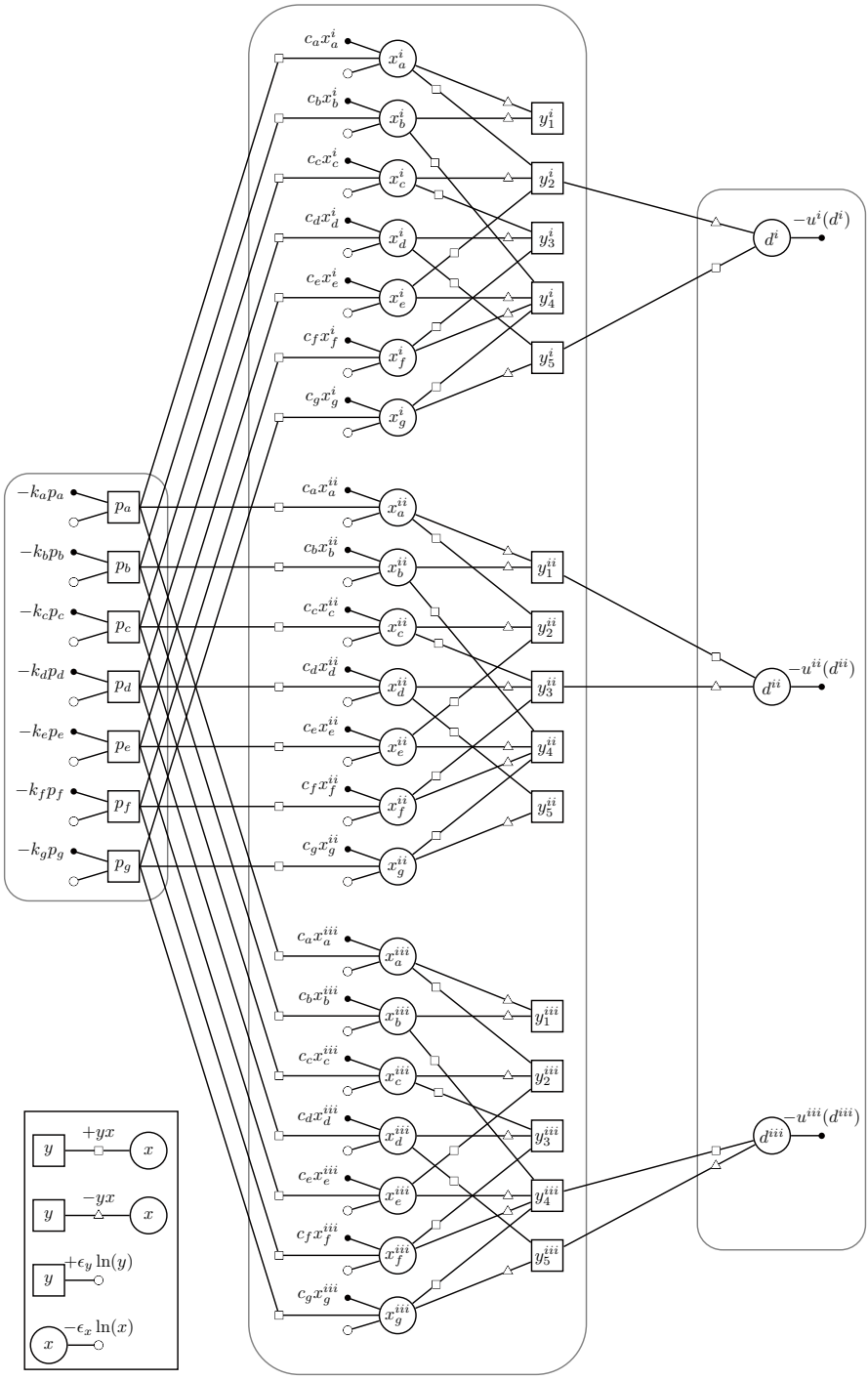

Fig. 4. Component graph for congestion routing

tion between the three flows, competing for the limited available capacity. We can then interpret the left of Fig. 4 as a market process mediating between the flows, and the p_j as *congestion prices*. High congestion prices force flows to re-route, or demands to reduce. This economic perspective has permeated recent work on congestion control mechanisms in communication networks [9,12,6], and market management mechanisms for distributed systems generally [14,5]. On the other hand, we could develop an electrical reading in which notions such as resistance (increase in congestion price with demand), dissipation, passivity, inductance and capacitance provide the intuitive framework.

7 Discussion

We have investigated a variety of interactive computations in networks through the techniques described in this paper, including layering, flow control, facility placement, and building of multi-cast trees, in addition to routing and congestion control along the lines of the examples presented above. There is much to be said for starting with an optimisation problem, as it forces the choices made about objectives and constraints to be made explicit, particularly those concerning fairness issues (an interesting analysis of TCP flow control, retrospectively extracting users' utility functions, has been performed along these lines in [11].)

The component graphs themselves also provide much insight into the structure of a problem. As mentioned in the introduction, they expose different options for decomposition (i.e. choice of subproblems), which lead to different system designs and different distributed algorithms. In this way alternative designs can be laid out and compared. The overall structure of interaction patterns and feedback loops are made apparent from the topology of the graph.

In using the component graphs, we have found that a certain amount of modelling intuition and idiom quickly builds up, to the extent that it becomes easy, sometimes natural, to by-pass the separate statement of the optimisation problem: the graphs become a useful design tool in their own right. We hope that a hint of this is conveyed in the main text. Furthermore, we have experimented with various language ideas, using an extension of the Scheme programming language, for specifying component graphs and different types of message exchange. A particular top-level problem is solved by writing a program that expresses a suitable decomposition into subproblems, just as is the case for solving any problem in any programming language. In our view the expressiveness and precision of a language framework offers many benefits for reasoning about network processes, as well as providing opportunities for transfer of techniques and know-how between the two subject areas.

The resulting programming paradigm is related to constraint programming and, hence, also logic programming. However, there are some differences of emphasis. In our framework duality plays a key role, and the 'denotational semantics' of a program is a saddle point of the Lagrangian, which is a (stable) equilibrium configuration of the system, or part of the system under consideration. The underlying conceptual picture of programming becomes one of specifying a collection of interacting feedback loops, rather than satisfying a collection of constraints. In this way, control theory and optimisation become equally relevant in building programming intuition.

The question arises as to whether a system converges to its saddle point. From a programming perspective, this is a question of whether a program will evaluate successfully. Here the convex-concave property of the Lagrangian plays a key role. In the regime in which the values of the decision variables change sufficiently slowly so that propagation delay can be ignored, the dynamic system equations (8) provide a naïve 'evaluation strategy'. Our proof of convergence is based on convex-concavity, and the assumption that the environment is stable, so the Lagrangian remains constant. In the case of the multi-commodity flow example of section 6, link failures, or users joining and leaving, would break this assumption. However, proofs of convergence of distributed algorithms (or of the behaviour of passive electrical circuits) typically make a similar assumption. It is also important to explore the opposite extreme in which decision variables owned by subproblems can change their values over timescales much shorter than the propagation delay of the communication channels. Convergence criteria are also required for this regime, where we should think in terms of sequences of (state) transitions executed by subproblems, as would be the case in a practical implementation of the Bellman-Ford algorithm.

The convex-concave property of the Lagrangian is a strong assumption. It ensures (i) that solutions of the problem are well-defined as saddle points, and (ii) the existence of effective (gradient descent) distributed algorithms for finding a saddle point. From the algorithmic point of view, compared to general constraint satisfaction algorithms, or logic programming, it avoids the need for any backtracking or search, which would be even more difficult and expensive to perform in a distributed setting than it already is in a local context. The uniqueness of a saddle point (in the case of a *strictly* convex-concave Lagrangian) guards against unwanted equilibrium states, or system-wide transitions between such equilibrium states. A failure to maintain this property can lead to bi-stability phenomena in networks, where routing or congestion control can stabilise in either a high-throughput or an unwanted low-throughput state. Similarly, configuration problems occurring in BGP routing suggest a

failure to ensure convexity, and can lead to unwanted equilibrium states and instabilities in the inter-domain routing infrastructure [7].

As already mentioned in the introduction, many network flow problems can be formulated as linear programs, and these have convex-concave Lagrangian functions. In problems formulated this way, flow variables are typically real-valued: a flow of one unit can be split into two half-unit flows. Imposing an additional constraint, as is sometimes required, that the flows must be integer-valued, breaks the convex-concavity property. In the special case of the minimum cost flow problem, integer solutions can be found among the solutions of the original (convex) linear program, so no extra cost is incurred by imposing the integer constraints (this property is shared by all linear programs having a totally unimodular A matrix and integer-valued c and b vectors [13].) The version of the problem with real-valued flow variables is therefore a useful *relaxation* of the problem with integer-valued flow variables, insofar as choosing any solution of the integer-valued problem that is close to (or co-incident with) a solution of the real-valued problem, incurs little extra cost in the objective function. Therefore, one approach to solving a distributed integer-valued problem could be to solve first the distributed relaxed problem, then seek a close integer-valued solution. On the other hand, if the problem is purely combinatorial, having no large-scale convex-concave structure in the cost function, then it seems unlikely that algorithms based on local gradient information will be effective.

The preceding argument suggests that, especially in a distributed setting, any convex-concave component to the problem (or a relaxation of the problem) should be identified and exploited as much as possible. Moreover, it may be necessary to impose the convex-concave property, or design it in to a system as a meta-constraint, if certain desirable behaviour is to be achieved. For example, such considerations might guide the design of routing policies, or policy languages if it is required to ensure that routing converges, or that the same route is selected after failure and restoration of any one link.

There is a question as to whether the types of processes we describe here, i.e. ongoing distributed optimisations, are of interest as an instance of *interactive computation*. The emphasis seems different to other models of computation, such as process algebra. Part of the apparent difference seems to be the treatment given to the notion of *state*. Usually computation is viewed as a sequence of state transitions. In our model, the overall state space is a real-valued space of dimension equal to the number of decision variables, but we normally only think of one global equilibrium state—the saddle point—into which the whole system converges: the idea of a sequence of transitions between equilibrium states is suppressed. In our model, computational com-

plexity comes directly from the high-dimensionality of the space, rather than the combinatorial character of state transitions.

Another answer to the previous question is to point out that network problems, such as the examples presented in the main text, provide a non-trivial class of interactive computations of considerable importance to society. Reasoning about their structure and behaviour relies on the same techniques of abstraction, composition, decomposition and reuse of patterns that are the central concerns of any high level computing *language*. Our coding experiments, viewed as experimental *evaluators*, require an extension of the structure used in, say, functional programming, as variables must be dynamic, evaluation must take place over cyclic (component) graphs, and different timescales must be taken into consideration. Seen through the eyes of these experimental evaluators, network processes do indeed constitute an interesting class of computation.

8 Conclusion

The model as presented captures a wide variety of distributed processes. As exemplified by the congestion routing example of section 6, it exposes interactions between parts of infrastructure that are often studied or designed assuming they are completely independent of each other. It emphasises the importance of treating primal and dual variables with equal status in the computation. Backwards compatibility with electrical circuit theory offers some intriguing avenues for development, as well as for transfer of concepts between these subjects, such as richer frequency domain analysis. It is also compatible with much economic theory. The fact that the same model underpins different interpretations applicable at different levels is encouraging, and suggests it might have a role to play in understanding aspects of global computation. Finally, communication networks expose an important part of the 'parameter space' of interactive computing.

References

[1] Ahuja, R. K., T. L. Magnanti and J. B. Orlin, "Network Flows," Prentice-Hall, 1993.

[2] Aji, S. M. and R. J. McEliece, *The generalized distributive law*, IEEE Transactions on Information Theory **46** (2000), pp. 325–343.

[3] Apt, K. R., "Principles of Constraint Programming," Cambridge University Press, 2003.

[4] Boyd, S. and L. Vandenberghe, "Convex Optimization," Cambridge University Press, 2004.

[5] Dash, R. K., N. R. Jennings and D. C. Parkes, *Computational-mechanism design: A call to arms*, IEEE Intelligent systems (2003), pp. 40–47.

[6] Gibbens, R. J. and F. P. Kelly, *Resource pricing and the evolution of congestion control.*, Automatica **35** (1999), pp. 1969–1985.

[7] Griffin, T. G., F. B. Shepherd and G. Wilfong, *The stable paths problem and interdomain routing*, IEEE Transactions on Networking **10** (2002), pp. 232–243.

[8] Huitema, C., "Routing in the Internet," Prentice Hall, 2000.

[9] Kelly, F., A. Maulloo and D. Tan, *Rate control in communication networks: shadow prices, proportional fairness and stability*, Journal of the Operational Research Society **49** (1998), pp. 237–252.

[10] Khalil, H. K., "Nonlinear Systems," Pearson Education, 2000, 3 edition.

[11] Low, S., F. Paganini and J. Doyle, *Internet congestion control: An analytical perspective*, IEEE Control Systems Magazine (2002).

[12] *M3i, market managed multiservice internet*, http://www.m3i.org/.

[13] Schrijver, A., "Theory of Linear and Integer Programming," Wiley, 1987.

[14] Wellman, M. P., *A market-oriented programming environment and its application to distributed multicommodity flow problems*, Journal of Artificial intelligence Research **1** (1993), pp. 1–23.