An Efficient Method for Large-Scale Slack Allocation

Siddharth Joshi   Stephen Boyd

May 20, 2008

Abstract

We consider a timing or project graph, with given delays on the edges and given arrival times at the source and sink nodes. We are to find the arrival times at the other nodes; these determine the timing slacks, which must be nonnegative, on the edges. The set of possible timing slacks is a polyhedron; to choose one we maximize a separable concave utility function, such as the sum of the logarithms of the slacks. This slack allocation problem, for which we give a simple statistical interpretation, is convex, and can be solved by a variety of methods. Gradient and coordinate ascent methods are simple and scale to large problems, but can converge slowly, depending on the topology and problem data. The Newton method, in contrast, reliably computes an accurate solution, but typically cannot scale beyond problems with a few thousand nodes. In this paper we describe a custom truncated Newton method that efficiently computes an accurate solution, and scales to large graphs (say, with a million or more nodes). Our method typically requires just a few hundred iterations, with each iteration requiring a few passes over the graph; in particular, our method has approximately linear complexity in the size of the problem. The same approach can be used to solve slack allocation problems with constraints, using an interior-point method that relies on our custom truncated Newton approach.

Keywords: timing graph, slack allocation, delay budgeting, convex optimization, truncated Newton method.

1 Introduction

A timing graph is a directed acyclic graph, with an arrival time at each node, and a delay on each edge. Each edge represents a timing constraint: The difference in arrival times at the nodes connected to an edge should be greater than the edge delay. The difference between these two quantities is called the slack of the edge; nonnegative slack means the timing constraint on that edge is satisfied.

In the slack allocation problem, we are given arrival times at the source and sink nodes of the timing graph, and must choose the arrival times for all other nodes. These arrival times determine the edge slacks. The timing assignments that meet all timing constraints, *i.e.*, that correspond to nonnegative slacks, form a polyhedron. To choose a particular timing assignment, we maximize a concave utility function that is separable in the edges. This is a convex optimization problem, and can be solved by a variety of methods, depending on properties of the utility function (such as differentiability). In this paper we describe algorithms for this maximum utility slack allocation problem that scale to very large timing graphs, with millions (or more) nodes.

Timing graphs come up in several application areas. In a digital circuit, each node can represent a gate; the arrival time represents the time by which the gate output signal becomes valid. The edges represent connections from one gate to another, and the edge delay is the delay of the gate and interconnect wiring. The slack on an edge can be interpreted as an additional delay that can be tolerated, while still meeting the timing constraints. (It is also possible to associate the edges with individual transitions from an input to the output of a gate.)

The slack allocation problem, also known as delay budgeting, arises in a wide variety of circuit placement and related problems; see, *e.g.*, [SWY02, CSX^{+}05, CN07]. The zero slack allocation algorithm, introduced in [HNY87, NBHY89], has been used to solve many variations on the slack allocation problem; see, *e.g.*, [Fra92, LH05]. The log-barrier slack allocation problem, for circuit placement problems, was described in [GVL91]. For slack allocation problems with other utility objective functions, see [SKT97, YCS02].

A timing graph can also be used to represent a project network (or an activity network) [Dav66, Elm64]. The edges represent tasks, and the delay is the time required to carry out the task. A task cannot start until all the tasks associated with the its incoming edges (or predecessors) have completed. In a project network, the node arrival times have the following meaning: Tasks associated with outgoing edges cannot start until the arrival time; and tasks associated with incoming edges must complete before the arrival time. The edge slack corresponds to extra time a task has, over its delay, to complete. Zero edge slack means the task has just enough time to complete; positive slack means it has some extra time. For slack allocation and related problems in project networks see [CGT08, MMRB98, Dod84, DE85]. A project network in which the delays are random is called a stochastic activity network [Dev79, Wei86, Hel81].

The outline of the paper is as follows. In §2 we formally describe the maximum utility slack allocation problem. In §3 we give an efficient algorithm, which uses the underlying problem structure, to find a feasible slack allocation. In §4 we describe various methods to solve the slack allocation problem. We use the log-barrier formulation to demonstrate the performance of the methods on number of numerical examples in §5. In §6 we briefly describe how the methods can be applied in more general cases.

2 Slack allocation

2.1 Timing graph and slacks

We consider a directed acyclic graph with $\tilde{n}$ nodes, labeled $1, \ldots, \tilde{n}$, and m edges, labeled $1, \ldots, m$. The sets of successor and predecessor nodes of node i are defined as

$$\begin{aligned} \mathcal{S}(i) &= \{j \mid \text{there is an edge from node } i \text{ to node } j\}, \\ \mathcal{P}(i) &= \{j \mid \text{there is an edge from node } j \text{ to node } i\}. \end{aligned}$$

Node i is called a source node (or just a source) if $\mathcal{P}(i) = \emptyset$, or a sink node (or just a sink) if $\mathcal{S}(i) = \emptyset$. The set of source nodes is denoted $\mathcal{N}_{\mathrm{src}}$ and the set of sink nodes is denoted $\mathcal{N}_{\mathrm{sink}}$. We let $\mathcal{F} = \mathcal{N}_{\mathrm{src}} \cup \mathcal{N}_{\mathrm{sink}}$ denote the boundary nodes. We will order the nodes in such a way that $\mathcal{S}(i) \subseteq \{i+1, \ldots, \tilde{n}\}$, for $i = 1, \ldots, \tilde{n}$, which is possible because the graph is a directed acyclic graph.

We are given a set of nonnegative edge *delays*, denoted $d_1, \ldots, d_m$. With each node we associate an *arrival time* t_i, $i = 1, \ldots, \tilde{n}$. We refer to the vector of arrival times $t \in \mathbf{R}^{\tilde{n}}$ as a *timing assignment*. The slack of edge k, which goes from node i to node j, is given by $s_k = t_j - t_i - d_k$, $k = 1, \ldots, m$. We can express the vector of edge slacks as

$$s = \tilde{A}^T \tilde{t} - d, \tag{1}$$

where $\tilde{A} \in \mathbf{R}^{\tilde{n} \times m}$ is the incidence matrix of the graph,

$$\tilde{A}_{ik} = \begin{cases} 1 & \text{edge } k \text{ enters node } i \\ -1 & \text{edge } k \text{ leaves node } i \\ 0 & \text{otherwise.} \end{cases}$$

We say that the timing constraint is met on edge k if $s_k \geq 0$, and call the timing assignment t feasible if $s \geq 0$, and strictly feasible if $s > 0$. (These inequalities are componentwise.) We can interpret the slack s_k as the amount of additional delay that can be tolerated on edge k, while still maintaining feasibility. The set of feasible (strictly) timing assignments is denoted $\mathcal{T}$ ($\mathcal{T}_+$). Evidently these are (closed and open) polyhedra.

It will sometimes be convenient to express the edge slacks in terms of the free arrival times, *i.e.*, those associated with $i \notin \mathcal{F}$. We let $t \in \mathbf{R}^n$ denote the subvector of $\tilde{t}$ containing the free arrival times, where $n = \tilde{n} - |\mathcal{F}|$. We can express the slacks as

$$s = \tilde{A}^T \tilde{t} - d = A^T t - d + f,$$

where $A \in \mathbf{R}^{m \times n}$ is the submatrix of $\tilde{A}$ formed from the rows associated with free nodes, and f is the contribution to the slacks from the fixed arrival times.

2.2 Slack allocation problem

Now we can describe the *slack allocation problem*. We are given the edge delays and the arrival times for the source and sink nodes, *i.e.*, $\tilde{t}_i$ for $i \in \mathcal{F}$. We are to choose the remaining node arrival times so as to make the slacks 'large'. We will measure 'large' using a separable utility function,

$$U(s) = U_1(s_1) + \cdots + U_m(s_m),$$

where U_i is a concave, increasing function. The (optimal) *slack allocation problem* is then

$$\begin{array}{ll} \text{maximize} & U(s) \\ \text{subject to} & s = \tilde{A}^T \tilde{t} - d \geq 0, \\ & \tilde{t}_i = T_i, \quad i \in \mathcal{F}, \end{array} \tag{2}$$

with variables $s \in \mathbf{R}^m$ and $\tilde{t} \in \mathbf{R}^{\tilde{n}}$. The problem data are $d \in \mathbf{R}^m$, T_i for $i \in \mathcal{F}$, and the utility functions $U_1, \ldots, U_m$. Since the constraints are linear and the objective, which is to be maximized, is concave, the problem (2) is a convex optimization problem.

We can express the slack allocation problem in terms of the free arrival times as

$$\begin{array}{ll} \text{maximize} & \psi(t) = U(A^T t - d + f) \\ \text{subject to} & A^T t - d \geq 0, \end{array} \tag{3}$$

with variable $t \in \mathbf{R}^n$. This problem is also a convex optimization problem.

We will focus on the special case when U is barrier function for $s \geq 0$, *i.e.*, it is twice differentiable, has domain $\mathbf{R}^m_{++}$, and satisfies $U(s) \to -\infty$ as any $s_k \to 0$. In this case, the constraint $s > 0$ is implicit, and the problem can be expressed as the unconstrained convex problem

$$\text{maximize} \quad \psi(t) = U(A^T t - d + f), \tag{4}$$

with variable t.

For $U_i(s_i) = \log s_i$, for which U is a barrier, we call the slack allocation problem the *log-barrier slack allocation problem.* This is equivalent to maximizing the product of the slacks, or finding the analytic center of $\mathcal{T}$ [BV04, §8.5.3]. Log utility and analytic centering come up in many application, *e.g.*, networking [TZB08], specifically rate control and proportional fairness [KMT97].

When U is a barrier, the optimality condition is simply $\nabla \psi(t) = 0$. We can express this condition as follows. Consider a node $i \notin \mathcal{F}$, with incoming edges $\mathcal{I}$ and outgoing edges $\mathcal{O}$. Then the optimality condition can be expressed as

$$\sum_{k \in \mathcal{I}} U_k'(s_k) = \sum_{k \in \mathcal{O}} U_k'(s_k),$$

i.e., the total marginal utilities of the incoming and outgoing edge slacks are balanced, at each free node.

2.3 Probabilistic interpretation

We can give a simple probabilistic interpretation of the slack allocation problem (2). Consider a project network in which the tasks suffer independent random additional delays $\delta_1, \ldots, \delta_m$, so the edge delays become $d_1 + \delta_1, \dots, d_m + \delta_m$. Task k meets its timing constraint if $d_k + \delta_k \le d_k + s_k$; the probability that all tasks meet their deadlines is

$$\prod_{i=1}^{m} \mathbf{Prob}(d_k + \delta_k \le d_k + s_k) = \prod_{i=1}^{m} \mathbf{Prob}(\delta_k \le s_k).$$

We can maximize this probability by maximizing its logarithm, which is $U(s)$, with slack utility functions

$$U_k(s_k) = \log \mathbf{Prob}(\delta_k \le s_k), \qquad k = 1, \dots, m. \tag{5}$$

This utility function is always nondecreasing, and it is concave for many common distributions, including uniform, exponential, normal, and log-normal. (It is concave for any log-concave distribution.) If $\delta_k \ge 0$ almost surely, and has a positive density near 0, U is a barrier, *i.e.*, $U(s) \to -\infty$ as any $s_k \to 0$.

For example, suppose that δ_k is uniform on $[0, \Delta_k]$. In this case the associated utility function is

$$U_k(s_k) = \begin{cases} -\log \Delta_k + \log s_k & s_k \le \Delta_k \\ 0 & s_k > \Delta_k. \end{cases}$$

In particular, we can think of log-barrier slack allocation as choosing the timing assignment so as to maximize the probability that all tasks satisfy the timing requirements, with uniformly distributed excess delays. (Here we ignore the saturation in U above Δ_k.)

As another example, suppose that δ_k has exponential distribution with mean λ_k. The associated utility is

$$U_k(s_k) = \log(1 - e^{\lambda_k s_k}),$$

which is concave, and a barrier.

As a last example, we consider the case when δ_k is log-normal, *i.e.*, $\log \delta_k \sim \mathcal{N}(\mu_k, \sigma_k^2)$. Then we have

$$U_k(s_k) = \log \mathbf{Prob}(\delta_k \le s_k) = \log \Phi((\log s_k - \mu_k)/\sigma_k),$$

where Φ is the cumulative distribution function of the standard Gaussian random variable. This utility is concave, although it is not immediately obvious. It can be shown several ways, for example, by observing that $\log \Phi$ is increasing and concave, and $(\log s_k - \mu_k)/\sigma_k$ is concave in s_k.

In a stochastic activity or project network, another quantity of interest is the completion time, which is the maximum of the arrival times at the sink nodes [RT76, Dev79, Dod85]. Optimization problems involving the distribution of the completion time can be found in [HJ90, Wei86, KBY$^+$07]. The slack allocation problem is different, and a bit easier than, completion time minimization problems.

3 Slack allocation feasibility

Feasibility (and strict feasibility) of the slack allocation problem is easily determined. One simple method is to recursively compute the smallest possible arrival time at each node, as follows. For the source nodes we set $t_i^{\min} = T_i$. Then we traverse the nodes in increasing order, setting

$$t_i^{\min} = \max_{j \in \mathcal{P}(i)} (t_j^{\min} + d_k),$$

where edge k goes from node j to node i. If for each $i \in \mathcal{N}_{\text{sink}}$ we have $t_i^{\min} \leq T_i$, we have constructed a feasible timing assignment; otherwise the slack allocation problem is infeasible. The slack allocation problem is strictly feasible if and only if for each $i \in \mathcal{N}_{\text{sink}}$ we have $t_i^{\min} < T_i$. We will show below how to construct a strictly feasible timing assignment, when the problem is strictly feasible.

Once it is determined that the problem is strictly feasible, we reset $t_i^{\min} = T_i$, $i \in \mathcal{N}_{\text{sink}}$. We compute the largest possible value of the arrival times by carrying out a backward recursion, starting from the sinks. We first assign $t_i^{\max} = T_i$ for $i \in \mathcal{N}_{\text{sink}}$. We then traverse the graph, in reverse order, setting

$$t_i^{\max} = \min_{j \in \mathcal{S}(i)} (t_j^{\max} - d_k),$$

where edge k goes from node i to node j. (The slack allocation problem is feasible (strictly feasible) if and only if for each $i \in \mathcal{N}_{\text{src}}$ we have $t_i^{\max} \geq T_i$ ($t_i^{\max} > T_i$).) We reset $t_i^{\max} = T_i$, $i \in \mathcal{N}_{\text{src}}$. Thus we obtain $t^{\min}$ and $t^{\max}$ feasible, but not strictly feasible, timing assignments.

To obtain a strictly feasible timing assignment t^{init} we compute the maximum slack that can be allocated to an edge. The maximum slack of edge k (which goes from node i to node j), denoted $s_k^{\max}$, is given by

$$s_k^{\max} = t_i^{\max} - t_j^{\min} - d_k, \qquad k = 1, \ldots, m.$$

(The slack allocation problem is strictly feasible if and only if $s_k^{\max} > 0$, $k = 1, \ldots, m$.) Let l_k be the length of a longest path through edge k. (The length of a path is the number of edges in the path.) Like the maximum slack $s_k^{\max}$, l_k for all edges can computed by a forward and a backward recursion. We set the slack on edge k to be $s_k^{\max}/l_k$, and compute the timing assignment by a forward recursion: For the source nodes we set $t_i^{\text{init}} = T_i$, $i \in \mathcal{N}_{\text{src}}$, then traversing the nodes in increasing order we set

$$t_i^{\text{init}} = \max_{j \in \mathcal{P}(i)} (t_j^{\text{init}} + d_k + s_k^{\max}/l_k),$$

where edge k goes from node j to node i. Finally, we reset $t_i^{\text{init}} = T_i$, $i \in \mathcal{N}_{\text{sink}}$. The timing assignment t^{init} is strictly feasible, since the resulting slack of edge k, s_k, satisfies $s_k \geq s_k^{\max}/l_k > 0$.

4 Solving the slack allocation problem

In this section we focus on the case when U is a barrier, in which case the unconstrained slack allocation problem (4) can be solved by any of the many methods for smooth unconstrained minimization. We mention a few of these methods below, before coming to our proposed method, which is a truncated Newton method. All of these methods start from a strictly feasible point t, compute a search direction Δt, and update the timing assignment to $t+\theta\Delta t$, where θ is the step size computed by a standard line search method, *e.g.*, backtracking. The trade-off among the different algorithms is between the effort required to compute each search direction, and the quality of the search direction, as judged by the convergence progress made per iteration.

For future use we give the gradient and Hessian of the objective $\psi(t) = U(A^Tt - d + f)$. We have

$$\nabla\psi(t) = AU'(A^Tt - d + f), \qquad \nabla^2\psi(t) = A\,\mathbf{diag}(U''(A^Tt - d + f))A^T,$$

where $U' = (U_1', \dots, U_m')$ and $U'' = (U_1'', \dots, U_m'')$. We define H_{diag} as the diagonal matrix with entries equal to the diagonal of the Hessian $\nabla^2\psi(t)$. The matrix H_{diag} is given by

$$\begin{aligned} H_{\text{diag}} &= \mathbf{diag}\,(\nabla^2\psi(t)_{11}, \dots, \nabla^2\psi(t)_{nn}) \\ &= \mathbf{diag}\,\Big((A \circ A)U''(A^Tt - d + f)\Big), \end{aligned}$$

where $\circ$ is the Hadamard (elementwise) product.

4.1 Coordinate ascent method

One simple approach to solve the slack allocation problem (4) is to cycle over the components of t, and optimize each component individually, keeping all others fixed. This method is called the *cyclic coordinate ascent method* [Lue84, §7.9]. Using Newton's method, each one-variable subproblem can be solved very quickly, requiring only a small number of floating point operations, given the arrival times of the node's predecessors and successors.

Cyclic coordinate ascent can be partially parallelized. We partition the free arrival times into groups, with no edges between any two nodes in each group. We can then optimize all the timing assignments in each group in parallel, and cycle through the groups.

Cyclic coordinate ascent can perform well for small problems, but for larger problems, it typically makes some initial rapid progress, and then stalls, *i.e.*, converges very slowly. The convergence is quite dependent on the topology of the graph.

4.2 Diagonally scaled gradient method

The diagonally scaled gradient method uses the search direction $\Delta t^{\text{dsg}} \in \mathbf{R}^n$ given by

$$\Delta t^{\text{dsg}} = -H_{\text{diag}}^{-1}\nabla\psi(t),$$

which can be computed very efficiently, once the slacks of the incoming and outgoing edges have been evaluated. We use this search direction, with a backtracking line search, to update t. We stop when, for example, $\nabla\psi$ is small enough. (For more on this see [BV04, NW99].)

The scaled gradient method performs like the cyclic coordinate ascent method: It typically makes some rapid progress, and then convergence slows down. As with cyclic coordinate ascent, the convergence is very much affected by the topology of the graph.

4.3 Newton method

The Newton method uses the search direction

$$\Delta t^{\mathrm{nt}} = -\nabla^2 \psi(t)^{-1} \nabla \psi(t),$$

with a backtracking line search to ensure convergence. The Newton method performs extremely well, typically converging to a highly accurate solution within a few tens of iterations (and often far fewer). This performance is not particularly dependent on the graph topology, or other problem data. Of course the cost of computing the Newton direction $\Delta \tilde{t}^{\mathrm{nt}}$ is far higher than the cost of computing the diagonally scaled gradient direction $\Delta \tilde{t}^{\mathrm{dsg}}$, or carrying out one cycle of the cyclic coordinate ascent method.

To compute the Newton direction, we must solve the Newton equation

$$\left(A\,\mathbf{diag}(-U'')A^T\right)\Delta t^{\mathrm{nt}} = AU', \tag{6}$$

where, to simplify notation, we omit the arguments of U' and U''. The coefficient matrix has no more than $n + 2m$ nonzero elements, which means for m small enough, it will be worthwhile to use a sparse Cholesky factorization to compute the search direction. The complexity of this method depends on the fill-in that occurs in the factorization, which in turn depends primarily on the graph topology and the method used to order the variables (see, *e.g.*, [Dem97]). Depending on the fill-in, the complexity of this method can range from $O(m)$ (for very simple sparse graphs, or chordal graphs) to $O(mn^2)$ (for graphs with much, or complete, fill-in). If the topology (and ordering method) is such that the fill-in is small, the Newton method is very efficient, and can scale to very large timing graphs. But fill-in typically limits the Newton method to problems with a few thousand nodes and edges.

4.4 Truncated Newton method

In a truncated Newton method [NW99, SF92], we compute the search direction as an approximate solution of the Newton equation (6), obtained by an iterative method, such as the preconditioned conjugate gradient (PCG) method with diagonal preconditioning [Kel95, NW99, Saa03]. The matrix $A\,\mathbf{diag}(-U'')A^T$ is diagonally dominant, which suggests that PCG with diagonal preconditioning should work well.

In each iteration of the PCG algorithm, we must multiply a vector z by $A\,\mathbf{diag}(-U'')A^T$. This can carried out as $A(\mathbf{diag}(-U'')(A^T z))$, which is very efficient; in particular, we never form or store the matrix $A\,\mathbf{diag}(-U'')A^T$. We also need to multiply a vector by the inverse of the diagonal preconditioner $-H_{\mathrm{diag}}^{-1}$, which also is very efficient because H_{diag}^{-1}, the diagonal of the Hessian, can be obtained without forming the Hessian.

If we terminate the PCG algorithm after one iteration, the resulting search direction is the scaled gradient direction (scaled by a positive amount). Since the scaled gradient method decreases the norm of the gradient rapidly for the first few iterations, we terminate the PCG algorithm after one iteration until the relative decrease in the gradient norm is less than some threshold ζ. After that, we run the PCG algorithm until the search direction Δt satisfies

$$\frac{\|(A\,\mathbf{diag}(-U'')A^T)\Delta t - AU'\|}{\|AU'\|} \leq \eta, \tag{7}$$

i.e., the relative Newton equation residual is less than or equal to η, or until the number of iterations reaches some limit $N^{\max}$. (In this case we use the search direction with the least relative residual.) The thresholds ζ, η, and $N^{\max}$ are set by experimentation.

The truncated Newton method performs very well, combining the good qualities of the scaled gradient and Newton methods. It makes rapid initial progress, but does not stall. The total number of PCG steps required to compute a very accurate solution is typically less than 100, even for very large timing graphs.

4.5 Comparison of methods

To compare the performance of the coordinate ascent and diagonally scaled gradient methods to the truncated Newton method, we need a rough estimate of the effort required per iteration, compared to the effort required in one PCG iteration.

In one PCG iteration we need to multiply a vector by $A\,\mathbf{diag}(-U'')A^T$, multiply a vector by $-H_{\mathrm{diag}}^{-1}$ (the preconditioning step), and compute a couple of inner-products of size n vectors. (For each search direction computation in the truncated Newton method we need to compute U', U'', and $\nabla\psi(t)$. The computation cost for computing these quantities, when amortized over the number of PCG iterations required to compute a search direction, is negligible.)

In one iteration of the diagonally scaled gradient method we need to compute U', U'', $\nabla\psi(t)$, H_{diag}^{-1}, and, finally, Δt^{dsg}. This requires approximately the same effort as one PCG iteration.

One cycle of arrival time updates in the coordinate ascent method is considered as one iteration. This requires an effort approximately equal to the average number of Newton iterations to solve the one-variable optimization problems for the n nodes, times the effort required in one iteration of the diagonally scaled gradient method. A typical number of Newton steps required to optimize one coordinate is around 5, so each cycle of coordinate ascent corresponds to 5 iterations of diagonally scaled gradient method, which is approximately 5 PCG iterations.

To summarize, we take the effort required in one iteration of cyclic coordinate ascent to be equal to 5 PCG iterations, and one iteration of the diagonally scaled gradient method to be equal to 1 PCG iteration.

5 Examples

We will show the performance of the methods on some numerical instances of the log-barrier slack allocation problem.

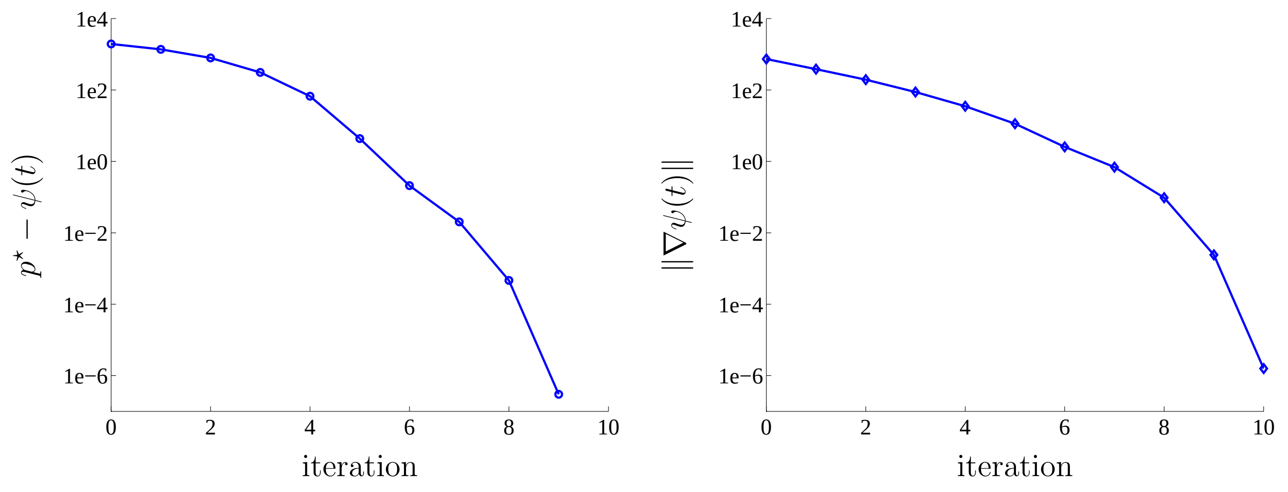

Figure 1: Error and norm of the gradient versus iteration for Newton method.

5.1 Medium size example

In the first example we consider a graph with $\tilde{n} = 1000$ nodes. The edges are chosen randomly, as follows. For $i = 1, \dots, \tilde{n}$ an edge from node i to node j, $j \in \{i+1, \dots, \min\{\tilde{n}, i+100\}\}$ is chosen with a probability $5/100$. Thus on average there are around 5 successors of each node. The resulting graph has $m = 4662$ edges, 34 source nodes, and 27 sink nodes. The delays are chosen independently from a uniform distribution on $[0, 1]$. The arrival times at the source nodes are chosen randomly from a uniform distribution on $[0, 1]$. The minimum arrival times at the sink nodes, for the problem to be feasible, are computed by a forward pass. We look at the differences between the arrival times at the sink and source nodes; the maximum of these differences is denoted T_{span}. The arrival times of the sink nodes are set to the (minimum) arrival time plus $0.05T_{\text{span}}$. The idea is to give 5% slack to the critical path.

We first show the performance of the Newton method. The Newton method solves the problem to high accuracy in 9 iterations. Figure 1 shows the error and the norm of the gradient, versus iteration.

The performance of the other methods is plotted in Figure 2, with effort shown in terms of equivalent PCG iterations, as described in §4.5. The error and the norm of the gradient for the diagonally scaled gradient and the cyclic coordinate descent methods decrease rapidly in the first few iterations, but thereafter converge very slowly to the optimal value. The truncated Newton method, however, converges to a highly accurate solution in around 200 PCG iterations. The performance shown in these plots is quite typical.

For the truncated Newton method we set $\zeta = 1\%$, *i.e.*, the ascent direction is the same as that obtained by the diagonally scaled gradient method, until the relative decrease in gradient norm falls below 1%. After that, the truncation rule is switched to the criterion (7) with $\eta = 10\%$, and $N^{\max} = 100$. These thresholds were chosen after some experimentation, but do not need to be fine tuned; very similar performance is obtained for a wide range of these parameters. For example, we can choose ζ large, which means that we switch immediately

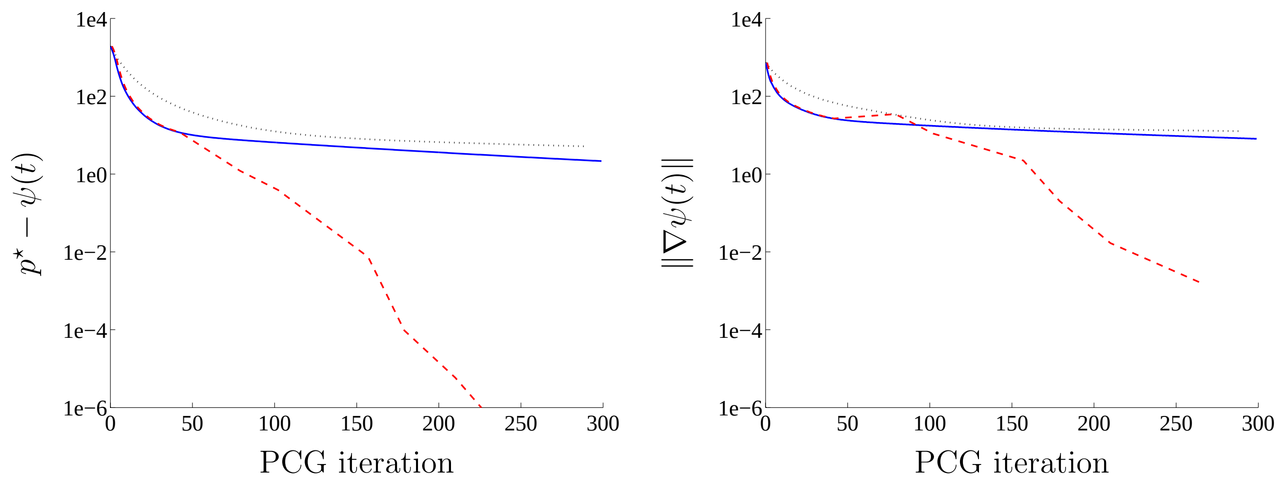

Figure 2: Error and norm of the gradient versus effort, expressed in PCG iterations, for cyclic coordinate ascent method (dotted curve), diagonally scaled gradient method (solid curve), and truncated Newton method (dashed curve).

to the PCG termination criterion (7). In this case the number of PCG iterations to get a highly accurate solution is still approximately 200.

5.2 Large examples

We now show the performance of the truncated Newton method for larger examples. We consider graphs with $\tilde{n} = 10^3$, 10^4, 10^5, and 10^6 nodes. For each value of $\tilde{n}$ we generate 3 random topologies, by choosing an edge from node i to node j, $j \in \{i+1, \dots, \min(i+\nu, \tilde{n})\}$ independently with probability $5/n$, taking $\nu = 0.1n, 0.2n, 0.3n$. Sometimes a node remains unconnected and is removed; thus the number of nodes is slightly less than the desired number of nodes. The delays are chosen independently from a uniform distribution on $[0,1]$. The arrival times at the source nodes are chosen randomly from a uniform distribution on $[0,1]$. As described earlier for the first example, T_{span} is calculated and the (minimum) arrival times of the sink nodes are moved forward by μT_{span}, where for μ we take three values $\mu = 0.05, 0.10, 0.15$. Thus we have a total of 36 numerical examples: for each of the 4 problem sizes we have 3 topologies and 3 sets of fixed arrival times.

For each of the 36 examples we run the truncated Newton method until the root mean square value of the gradient is below 10^{-3}, *i.e.*, $\|\nabla\psi(t)\|/\sqrt{n} \le 10^{-3}$. The truncation rule for the PCG algorithm is set to (7) from the very beginning. The number of PCG iterations required versus problem size is plotted in Figure 3. There is a very slight increase in the number of PCG iterations with problem size. We can see that around 700 PCG iterations are enough to solve almost all problems, even problems with 10^6 nodes and around 5×10^6 edges. We show, in Table 1, the number of times a search direction was computed and the number of PCG iterations required to approximately solve the Newton system, for each problem size, averaged over various examples. The important point to observe is that the number of PCG

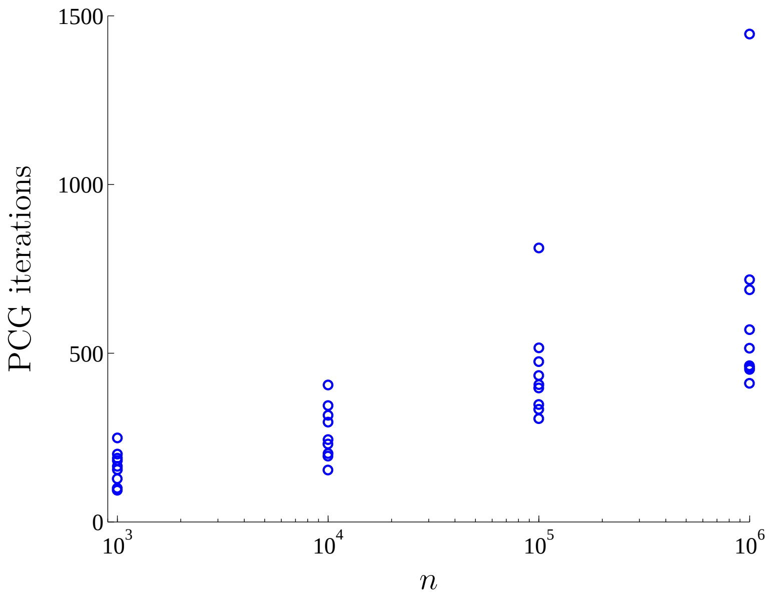

Figure 3: Cumulative PCG iterations required versus number of nodes in the problem.

Nodes	10^3	10^4	10^5	10^6
Newton system approximately solved	10	12	15	16
PCG iterations required/solve	17	23	31	40

Table 1: Number of PCG iterations per search direction computation and number of times the Newton system is approximately solved, averaged over various topologies and various set of fixed arrival times, for each problem size. (Values are approximate.)

iterations required to approximately solve the Newton system is very much less the size of the Newton system; only 40 or so PCG iterations are required to approximately solve the Newton system for the problems with 10^6 nodes. We see that the truncated Newton method scales to very large problems.

6 Extensions

In this section we show how the methods described in §4 apply to more general cases.

6.1 Worst-case slack allocation

A variation on the slack allocation problem is to allocate the slacks to maximize the minimum slack. This is called the *worst-case slack allocation* problem:

$$\begin{array}{ll} \text{maximize} & \min_{k\in\{1,\ldots,m\}} s_k \\ \text{subject to} & s = A^T t - d + f, \end{array} \tag{8}$$

with variables s and t. The optimization problem (8) is a convex optimization problem, and is equivalent to the problem

$$\begin{array}{ll} \text{maximize} & r \\ \text{subject to} & r \le s_k, \quad k = 1,\ldots,m \\ & s = A^T t - d + f, \end{array} \tag{9}$$

with variables r, s and t.

The problem (9) can be solved efficiently using the bisection method. We note that feasibility, for a particular value of r, can be efficiently determined using the method for determining slack allocation feasibility described in §3, with the delay of the gates equal to $d_k + r$ instead of d_k. As a result we obtain a timing assignment for which all the resulting slacks are greater than or equal to r, or we determine that the value of r is infeasible. The bisection method works as follows. We start with an interval $[0, r^{\max}]$ in which the optimal value of r, $r^\star$, lies. We check if $r = r^{\max}/2$ is feasible, and if so then the new interval in which $r^\star$ lies is $[r^{\max}/2, r^{\max}]$; if not, the new interval is $[0, r^{\max}/2]$. If the desired accuracy is ϵ the bisection method requires $\lceil \log_2(r^{\max}/\epsilon)\rceil$ steps, each requiring one pass over the graph. The length of an initial interval, *i.e.*, $r^{\max}$, can be chosen is many ways. One way is to compute the maximum slack $s^{\max}$, after checking the feasibility of $r = 0$, and choose $r^{\max} = \min_{i=1,\ldots,m} s_i^{\max}$.

Though we can solve the worst-case slack allocation problem (8) efficiently, even for large-scale problems, it is not appealing because of the following property its solution set. At an optimal slack allocation the timing graph has as least one path with all edge slacks equal to r^*. Such a path is called a *critical path.* For each non-critical path, there is a lot of flexibility in allocating the slacks along the edges. This leads to a large solution set of the worst-case slack allocation problem (8); in contrast the solution of the log-barrier slack allocation problem, the analytic center of the timing polyhedron $\mathcal{T}$, is unique.

6.2 Non-barrier utility functions

If the utility U is not a barrier, to solve the slack allocation problem (4) we add barrier terms to the objective of the problem to maintain the positivity of the slack variables. Specifically if U_k is not a barrier, for the inequality $s_k \geq 0$, which no longer is implicit, we add the barrier term $\kappa \log s_i$ to the objective, where κ is a positive constant. The slack allocation problem (4) is solved by solving a sequence of optimization problems each of the form

$$\text{maximize} \quad \phi(t) = U^{\text{bar}}(A^T t - d + f), \tag{10}$$

where $U^{\text{bar}}(s) = \sum_{i=1}^m U_i^{\text{bar}}(s_i)$,

$$U_i^{\text{bar}}(s_i) = U_i(s_i) + \kappa \log s_i, \qquad i = 1, \ldots, m,$$

for a decreasing sequence of values of κ. (Here we have added a log-barrier term for each of the slack variables.) The solution of the problem for a value of κ serves as initial point of the next problem with smaller value of κ. Usually the value of κ is decreased according to a preselected schedule, but it can also be decreased adaptively. Let $t^\star$ be a timing assignment that maximizes ψ (the original objective), and t^* be the timing assignment that maximizes ϕ. The suboptimality of the timing assignment t^*, defined as $\psi(t^\star) - \psi(t^*)$, is at most $m\kappa$. If the desired level of accuracy is ϵ, then the sequence of optimization problems is solved until $\kappa \leq \epsilon/m$. If the desired accuracy ϵ is large enough, which is the case for many practical applications, we can choose to solve just one optimization problem (10) with the required value of κ. This choice of either solving one optimization problem or a sequence of optimization problems depends on the particular application, the quality of the initial point, the desired accuracy, etc., and is usually made after some experimentation. Some applications in which the optimization problem is solved for only one value of κ are sensor selection [JB07] and model predictive control [WB08].

The utility U^{bar} is a barrier and to solve the problem (10) the truncated Newton method can be applied. The Hessian $\nabla^2 \phi(t)$ is

$$\nabla^2 \phi(t) = -AGA^T,$$

where the diagonal matrix $G \in \mathbf{R}^{m \times m}$ is given by

$$G_{kk} = U_k''(t_j - t_i - d_k) + \kappa/(t_j - t_i - d_k)^2, \qquad k = 1, \ldots, m,$$

where edge k goes from node i to node j. The matrix AGA^T is diagonally dominant.

The method we have outlined in this section is the interior-point algorithm for solving the optimization problem (4) when U is not a barrier. The step of solving the optimization problem (10) is called *centering*. See [Nes03, NN94, Ye97, Wri97] for details on interior-point methods. The important point is that the problem (10) can be solved efficiently using truncated Newton type methods.

6.3 Constraints on arrival times

To impose an *equality constraint* on the arrival time of a free node, we split the node in two nodes: a source and a sink, since the only fixed nodes in our formulation are the sources and sinks. The successors of the node are the successors of the newly introduced source node, and, similarly, the predecessors of the node are the predecessors of the newly introduced sink node.

The *inequality constraints* on the arrival time of a free node:

$$T_i^{\mathrm{l}} \leq t_i \leq T_i^{\mathrm{u}},$$

can also be incorporated in our formulation. To impose the above constraints we introduce a source and a sink node with fixed arrival times T_i^{l} and T_i^{u}, respectively. We add directed edges: one from the source node to node i, and one from node i to the sink node, each having a delay 0. The slacks on these edges are $t_i - T_i^{\mathrm{l}}$ and $T_i^{\mathrm{u}} - t_i$, and we introduce the utility $\kappa \log(.)$ for these slacks, where κ is a positive constant. The slack allocation problem (4) with the inequality constraints is approximated by the optimization problem

$$\text{maximize} \quad \varphi(t) = U(A^T t - d + f) + \kappa \sum_{i=1}^{n} \left(\log(t_i - T_i^{\mathrm{l}}) + \log(T_i^{\mathrm{u}} - t_i) \right). \tag{11}$$

To solve the slack allocation problem (4) with the inequality constraints, we solve a sequence of optimization problems of the form (4) with decreasing value of κ. The solution of the problem (4) for a value of κ serves as the initial point for the next problem with smaller value of κ. If the desired level of accuracy is ϵ, the sequence of optimization problems is solved until $\kappa \leq \epsilon/2n$.

The objective function of the problem (11) is a barrier (assuming U is a barrier): If any of the slacks, including the slacks introduced by the inequality constraints, goes to 0, the objective function value goes to $-\infty$. The truncated Newton method can be used to solve the problem (11). The Hessian $\nabla^2 \varphi(t)$ is

$$\nabla^2 \varphi(t) = A\, \mathbf{diag}(U'') A^T - \kappa B,$$

where the matrix $B \in \mathbf{R}^{n \times n}$ is a diagonal matrix with the entries

$$B_{ii} = 1/(t_i - T_i^{\mathrm{l}})^2 + 1/(T_i^{\mathrm{u}} - t_i)^2, \qquad i = 1, \ldots, n.$$

The matrix $-\nabla^2 \varphi(t)$ is diagonally dominant; in fact the matrix $-\nabla^2 \varphi(t)$ is more diagonally dominant than the matrix $A\, \mathbf{diag}(-U'') A^T$ because of the diagonal matrix κB, and one can expect better convergence of the preconditioned conjugate gradient method with diagonal preconditioning.

7 Conclusion

We have developed a custom truncated Newton type method to solve the slack allocation problem, which scales efficiently to large-scale problems, *i.e.*, problems with the timing graph consisting of a million or more nodes. The method exploits the underlying problem structure via the preconditioned conjugate gradient method, which efficiently computes a good search direction, as judged by the overall performance.

References

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[CGT08] J. Castro, D. Gómez, and J. Tejada. A rule for slack allocation proportional to the duration in a PERT network. *European Journal of Operational Research*, 187(2):556–570, 2008.

[CN07] J. Cong and G. Nam. *Modern Circuit Placement: Best Practices and Results.* Springer, 2007.

[CSX$^+$05] J. Cong, J. Shinnerl, M. Xie, T. Kong, and X. Yuan. Large-scale circuit placement. *ACM Transactions on Design Automation of Electronic Systems*, 10(2):389–430, 2005.

[Dav66] E. Davis. Resource allocation in project network models – A survey. *The Journal of Industrial Engineering*, 17(4):177–187, 1966.

[DE85] B. Dodin and S. Elmaghraby. Approximating the criticality indices of the activities in PERT networks. *Management Science*, 31(2):207–223, February 1985.

[Dem97] J. Demmel. *Applied Numerical Linear Algebra.* Society for Industrial and Applied Mathematics, 1997.

[Dev79] L. Devroye. Inequalities for the completion times of stochastic PERT networks. *Mathematics of Operations Research*, 4(4):441–447, 1979.

[Dod84] B. Dodin. Determining the k most critical paths in PERT networks. *Operations Research*, 32:859–877, 1984.

[Dod85] B. Dodin. Bounding the project completion time distribution in PERT networks. *Operations Research*, 33:862–881, 1985.

[Elm64] S. Elmaghraby. An algebra for the analysis of generalized activity networks. *Management Science*, 10:494–514, 1964.

[Fra92] J. Frankle. Iterative and adaptive slack allocation for performance-driven layout and FPGA routing. *Proceedings of the 29th ACM/IEEE conference on Design automation conference*, pages 536–542, 1992.

[GVL91] T. Gao, P. Vaidya, and C. Liu. A new performance driven placement algorithm. *IEEE International Conference on Computer-Aided Design*, pages 44–47, 1991.

[Hel81] U. Heller. On the shortest overall duration in stochastic acyclic network. *Methods of Operations Research*, 42:85–104, 1981.

[HJ90] J. Hagstrom and N. Jane. Computing the probability distribution of project duration in a PERT network. *Networks*, 20:231–244, 1990.

[HNY87] P. Hauge, R. Nair, and E. Yoffa. Circuit placement for predictable performance. *Proceeding of International Conference on Computer Aided Design*, pages 88–91, 1987.

[JB07] S. Joshi and S. Boyd. Sensor selection via convex optimization. *Submitted to IEEE Transactions on Signal Processing*, 2007.

[JB08] S. Joshi and S. Boyd. An efficient method for large-scale gate sizing. *To appear in IEEE Transactions on Circuits and Systems I*, 2008.

[KBY$^+$07] S. Kim, S. Boyd, S. Yun, D. Patil, and M. Horowitz. A heuristic for optimizing stochastic activity networks with applications to statistical digital circuit sizing. *Optimization and Engineering*, 8(4):397–430, 2007.

[Kel95] C. Kelley. *Iterative Methods for Linear and Nonlinear Equations*, volume 16 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, 1995.

[KM02] A. Kahng and I. Markov. Min-max placement for large-scale timing optimization. *Proceedings of the 2002 international symposium on Physical design*, pages 143–148, 2002.

[KMT97] F. Kelly, A. Maulloo, and D. Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49:237–252, 1997.

[LH05] Y. Lin and L. He. Leakage efficient chip-level dual-Vdd assignment with time slack allocation for FPGA power reduction. *Proceedings of the 42nd annual conference on Design automation*, pages 720–725, 2005.

[Lue84] D. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, second edition, 1984.

[MMRB98] A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco. An exact algorithm for the resource constrained project scheduling problem based on a new mathematical formulation. *Management Science*, 44:714–729, 1998.

[NBHY89] R. Nair, C. Berman, P. Hauge, and E. Yoffa. Generation of performance constraints for layout. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(8):860–874, 1989.

[Nes03] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course.* Kluwer Academic Publishers, Boston, 2003.

[NN94] Y. Nesterov and A. Nemirovsky. *Interior-point Polynomial Methods in Convex Programming*, volume 13 of *Studies in Applied Mathematics*. SIAM, 1994.

[NW99] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, New York, 1999.

[RT76] P. Robillard and M. Trahan. The completion times of PERT networks. *Operations Research*, 25:15–29, 1976.

[Saa03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2nd edition, 2003.

[SF92] T. Schlick and A. Fogelson. TNPACK–A truncated newton minimization package for large-scale problems: I. Algorithm and usage. *ACM Transactions on Mathematical Software (TOMS)*, 18(1):46–70, 1992.

[SKT97] M. Sarrafzadeh, D. Knol, and G. Tellez. A delay budgeting algorithm ensuring maximum flexibility in placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(11):1332–1341, 1997.

[SWY02] M. Sarrafzadeh, M. Wang, and X. Yang. *Modern Placement Techniques*. Kluwer Academic Publishers, 2002.

[TZB08] N. Trichakis, A. Zymnis, and S. Boyd. Dynamic network utility maximization with delivery contracts. *To appear in IFAC World Congress*, July 2008.

[WB08] Y. Wang and S. Boyd. Fast model predictive control using online optimization. *To appear in IFAC World Congress*, July 2008.

[Wei86] G. Weiss. Stochastic bounds on distributions of optimal value functions with applications to PERT, network flows and reliability. *Operations Research*, 34(4):595–605, 1986.

[Wri97] S. Wright. *Primal-Dual Interior-Point Methods*. Society for Industrial and Applied Mathematics, 1997.

[YCS02] X. Yang, B. Choi, and M. Sarrafzadeh. Timing-driven placement using design hierarchy guided constraint generation. *ICCAD 2002*, pages 177–180, November 2002.

[Ye97] Y. Ye. *Interior Point Algorithms: Theory and Analysis*. Discrete Mathematics and Optimization. Wiley, New York, 1997.