An Optimization Approach for Radio Network Scheduling

Ville P.J. Koskinen, 48461N

Pauli "Pala" Alanaatu, 48425R

Jenni C. Brunila, 48376C

Tuomo M. Pyhälä, 49560K

Vesa H. Timonen, 47494R

25th April 2003

Abstract

This study presents a new approach to model a wireless radio network and methodology to find optimal routing schedules for it. Our scheduling problem, as most of them, is $\mathcal{NP}$-complete. Hence, linear programming models, which can be used to reach an exact optimum, appeared to be unsuitable for our purposes because of the high dimensionality of our problem. Therefore, we focused on developing a graph representation of the scheduling problem which can be efficiently solved by applying local search methods. In this work we applied taboo search heuristics as suggested in literature.

The computational results show that the taboo search is suitable for solving this kind of $\mathcal{NP}$-complete problems especially if a near-optimal solution suffices. This is because the search can be terminated any time and by letting the procedure to run long enough good solutions can be reached.

Preface

This work was carried out in Operations Research Project Work Course at Helsinki University of Technology at Systems Analysis Laboratory, and it was instructed by Mr. Jukka K. Nurminen at Nokia Research Center. This work has taught us diverse project work skills and provided us a great deal of insight on how to apply operations research models and methodology in practice.

It was nice to see a group of students working as a whole, as a team. Even we succeeded well in project work; the aims were reached, it was fun to work together and we made new friends within the group, a few concerns encountered during the project. Each project member had many independent projects of their own and hence compliance with the schedule appeared to be difficult from time to time. The other problem was derived from the fact that the skills needed in this project were rather diverse. Understanding of operations research models and methodology, knowledge on data networks and programming skills were required. Especially the programming skills were emphasized at the end of the project. It turned out that our team had too little experience on programming, which caused problems in keeping with the project schedule. Nevertheless, after completing this work our programming skills have developed remarkably.

We identified a couple of risks at the beginning of the project. The first one was the $\mathcal{NP}$-completeness of the problem. This caused a few problems during the project. Namely, the implemented solution algorithm needed to be optimized; otherwise the algorithm would have taken an unreasonable amount of CPU time. Another identified risk was the low level of project work expertise of the project members. Nevertheless, this did not encounter any problems in our case. No other risks were identified either during or after the project.

The whole project team wishes to sincerely thank Mr. Jukka K. Nurminen, the instructor of this work, for the possibility to work within their project. We are also grateful for course lecturer Professor Ahti Salo and course assistant Mr. Antti Punkka for giving valuable feedback during the project.

On a sunny evening in a terrace of an Irish pub, at Kaisaniemi, Helsinki, April 2003
Project team:
Ville Koskinen,
Pauli Alanaatu,
Jenni Brunila,
Tuomo Pyhälä,
Vesa Timonen.

Contents

1 Introduction

In residential areas, where the settlement is quite dense but building dedicated lines (ADSL etc.) would be too expensive, it is today possible to use radio network technique to build a local area network. Nokia Inc. provides small wireless routers, the RoofTop-routers, for this purpose.

The network is constructed as follows: each house has its own router, which transmits and receives data packets to other routers via radio waves. One of the routers operates also as a gateway, which is connected to the Internet via an optical cable. There has to be a line of sight between the routers and the maximum range of a router is not more than 1500 meters. The target is to forward data packets as efficiently as possible from sender to destination.

In radio networks only one router at a time can transmit data in its range and the other routers inside sender's range except the receiver must remain idle. This restriction constitutes a difficult scheduling problem.

Transmission of a packet from its origin to its destination takes a certain time. This time is referred to as transmission delay of the packet; evidently the smaller delay is the better the schedule is. The efficiency of the network and the schedule as a whole is measured by taking the maximum of the delays of the data packets and, hence the scheduling problem is about choosing a schedule minimizing the maximum delay.

At hardware level, the scheduling problem is solved in a decentralized manner, that is, the routers prioritize different packets, and agree on the sending periods with the other routers within their own range. The goal of this work is to find an optimal centralized solution for the scheduling problem, which is to be used as a benchmark when assessing the quality of the applied routing heuristics.

In scheduling theory literature the scheduling problem is typically described as a linear programming problem or as a graph [15], [11]. Commonly applied traditional solution methodology includes different methods related to linear programming, such as branch and bound, branch and cut, Lagrangian relaxation etc., see [7]. These methods have, however, lacked of practical usefulness due to the underlying complexities of the scheduling problem, which has been appeared to be $\mathcal{NP}$-complete. In practice, the $\mathcal{NP}$-completeness implies that no algorithm exists that can solve the problem in polynomial time as a function of the size of the problem. Hence, each solution algorithm takes an exponential time, which means that a huge amount of CPU time is required if a large enough problem is being solved.

These computational problems have generated a number of heuristic approaches for solving $\mathcal{NP}$-complete problems. Especially in our case, there is a number of heuristic approaches for solving scheduling problems. Those approaches include, e.g., taboo search, simulated annealing, genetic algorithms and a variety of so called priority rules [2], [12]. The priority rules are easy to implement but they have proven to produce solutions that are far away from optimal [7]. Therefore, they are left beyond the scope of this work.

The evolution of the heuristic solution methods was initiated in the early 1990'es when van Laarhoven, Aarts and Lenstra [22] studied suitability of the simulated annealing. They concluded that the simulated annealing was not a suitable heuristics but they argued that taboo search methods could emerge an efficient methodology for some scheduling problems. More recently, their work has been extended by many authors, see, e.g., [1], [18], [16], [6], [2], [5], [19] and [8]. Hence, we take the taboo search as the basis of our solution methodology.

2 Radio Network Description

2.1 Assumptions

In this work we investigate a very short period of time, a snapshot, in the network. This results to a network that can be described with a *tree topology*. The *nodes* of the tree describe *routers* and the *arcs* describe *links*. The *gateway* node is on the top of the topology. As the radio links are fixed for a short period of time, they can be handled like normal cable links. The restrictions presented in the introduction remain the same.

To simplify the case the following assumptions are made:

1. There is no internal traffic in the network between the nodes, and hence all packets from and to the Internet are routed trough the gateway node.
2. Each node has three states: idle, transmitting and receiving. There is a small setup time when changing from state "transmitting" to "receiving" and vice versa.
3. The size of the data packets in the network is fixed.
4. When a node is transmitting data, all its neighbors within its range except the receiver must remain idle.
5. All arcs are identical, so the duration of transmission of a packet is equal for each router in the network. The maximum capacity of an arc is 24Mbps.
6. Each node transmits and receives data at the same constant rate, which is known in advance.

2.2 Basic Notation

Let us denote the set of *routers* by $\mathcal{M} = \{M_1, \ldots, M_N\}$ and the set of *packets* by $\mathcal{P} = \{p_1, \ldots, p_n\}$. Each packet p_i is being transmitted from the origin to the destination via certain sequence of routers $\mathcal{M}_i = \{M_{i_1}, \ldots, M_{i_k}\}$. Transmission of packet p_i from M_{i_j} to $M_{i_{j+1}}$ is called as *transmission operation* and it is denoted by O_{ij}. The durations of the operations are known in advance and they are denoted by T_{ij}.

A transmitting router interferes its neighbors and therefore we define the set of *reserved routers* $\mathcal{R}_{ij} \subseteq \mathcal{M}$ that contains the routers being reserved during the operation O_{ij}.

The sequence dependent setup times are taken into account in the model by denoting the routers that transmit and receive data during an operation O_{ij} by $M_{Tr}(O_{ij})$ and $M_{Re}(O_{ij})$ respectively. The setup time is denoted by T_v.

Thus, we can identify three types of constraints characterizing the set of *feasible schedules*:

- *The precedence constraints.* A packet must be transmitted via the routers in a fixed order $\mathcal{M}_i$.
- *The disjunctive constraints.* If two operations O_{ij} and $O_{i'j'}$ reserve same routers, i.e., $\mathcal{R}_{ij} \bigcap \mathcal{R}_{i'j'} \neq \emptyset$, they cannot be completed simultaneously. This implies that *either* O_{ij} precedes $O_{i'j'}$ *or* vice versa.
- *Setup constraints.*

Let us denote the time when packet p_i arrives to the network by T_i^0 and the time when it reaches the destination by T_i^f. Now the *total transmission time* of a packet can be defined as

$$D_i = T_i^f - T_i^0.$$

To offer high quality of service for each user of the network we minimize the greatest D_i, that is,

$$\min \max_{p_i} D_i. \tag{1}$$

For a broader view regarding the fairness in telecommunications, see, e.g., [21]. We refer the packet p_{i^*} giving the greatest delay D_i to as *critical packet.*

This formulation of the underlying problem is somewhat similar to the well known *job shop scheduling problem* (JSP) [6] provided that we interpret the packets as jobs consisting of operations and routers as machines completing the jobs. However, JSP assumes in its basic form that there are no setup times and performing an operation reserves only a single machine. Moreover, the JSP assumes that the completion time of the last job being completed is minimized.

JSP model has been generalized in literature to describe real manufacturing systems. For instance, Choi and Korkmaz [8] have introduced how the sequence dependent setup times can be taken into account, Mati, Nidhal and Xiaolan [16] have generalized the model to include the reservation of multiple machines and Scrich, Armentano and Laguna [19] have shown how the objective function can be varied. We combine these ideas to create models for our purposes and to develop suitable solution methods for them.

2.3 Mixed Integer Linear Programming Model (MILP)

Let us denote the *starting time* of operation O_{ij} by t_{ij} and assume that $T_{ij} = T_l$ for each O_{ij}. Now we can represent the constraints presented above in terms of inequalities as follows.

1. The disjunctive constraints

$$\begin{cases} t_{ij} + T_l \geq t_{i'j'} & \text{or} \\ t_{i'j'} + T_l \geq t_{ij} \end{cases} \qquad \forall \mathcal{R}_{ij} \bigcap \mathcal{R}_{i'j'} \neq \emptyset. \tag{2}$$

2. Setup constraints

$$\begin{cases} t_{ij} + T_l + T_v \geq t_{i'j'} & \text{or} \\ t_{i'j'} + T_l + T_v \geq t_{ij} \end{cases} \qquad \forall \begin{cases} M_{Tr}(O_{ij}) = M_{Re}(O_{i'j'}) & \text{or} \\ M_{Tr}(O_{i'j'}) = M_{Re}(O_{ij}) \end{cases} \tag{3}$$

3. The precedence constraints

$$t_{i(j+1)} \geq t_{ij} + T_v + T_l \quad \forall O_{ij}. \tag{4}$$

4. Additionally, the packets arrive to the packet's arrival to the network

$$t_{ij} > T_i^0. \tag{5}$$

By introducing a variable ε that satisfies

$$\varepsilon \geq t_{ij} - T_i^0, \tag{6}$$

we obtain an optimization problem: $\min \varepsilon$ subject to constraints (2)-(6).

The constraints (2) and (3), which are difficult *or* constraints, can be written in linear form by introducing integer variables. Hence, the problem can be rewritten as a *mixed integer linear programming* (MILP) problem. If we assume that we have even an small radio network that consists of six routers receiving and transmitting one packet, we obtain an MILP problem with 204 variables and 424 inequalities. Hence, we have a problem that is large to be solved by using traditional MILP-optimization methods, such as branch and bound method or Lagrangian relaxation [6].

2.4 Disjunctive Graph Representation

A schedule can be presented as a *graph*, see Figure 2 below representing transmission of two packets as depicted in Figure 1. The graph presents each transmission operation O_{ij} as a *node*. The nodes are aligned such that each row in the graph corresponds to a packet. You can imagine that each node contains a label that shows the set of reserved routers $\mathcal{R}_{ij}$ and the durations T_{ij} explicitly. The durations are called the *weights* of the nodes.

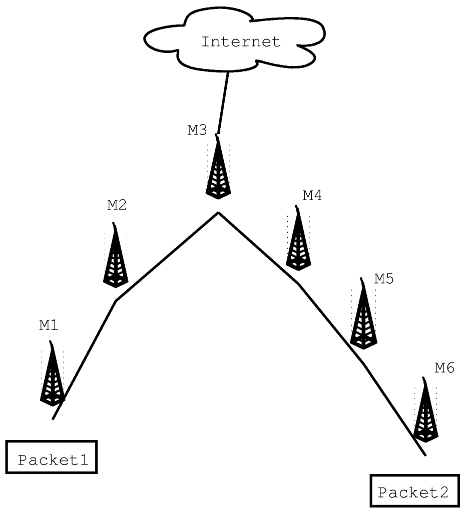

Figure 1: The radio network

The *arcs* present the order in which the operations are performed, i.e., an *directed* arc from O_{ij} to $O_{i'j'}$, denoted by $(O_{ij}, O_{i'j'})$, indicates that the operation O_{ij} is completed before $O_{i'j'}$. The solid arcs in Figure 2 correspond to the precedence constraints. The slashed arcs represent the disjunctive constraints and they are referred to as *disjunctive* arcs respectively. Initially, the disjunctive arcs are undirected and basically our problem is to choose their directions optimally.

The setup times can be included in this graph representation by setting an additional weight T_v for an arc $(O_{ij}, O_{i'j'})$ if a setup is needed between operations O_{ij} and $O_{i'j'}$.

Suppose that we have one schedule, i.e., we have all the disjunctive arcs directed, as shown in Figure 2. Consider a *path* $P = \{O_{i_1 j_1}, \ldots, O_{i_n j_n}\}$ in the graph and define the *length of a path* as the sum of the weights of the nodes and arcs on the path. Hence, the length of a path can be interpreted as a lower bound of the time lag between the operations $O_{i_1 j_1}$ and $O_{i_n j_n}$. This is because the directed arcs give the completion order of the operations.

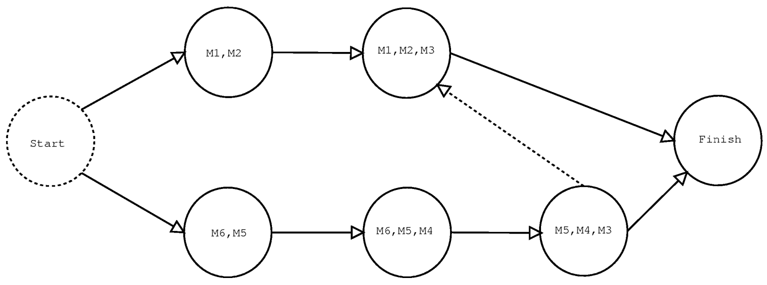

Figure 2: An example of a schedule

This representation allows us to determine the total transmission time for each packet p_i. Choose the operations O_{i1} and $O_{ij_{\max}}$ latter of which refers to the last operation of the packet p_i. Now we can calculate a *critical path* for p_i, that is the longest path from O_{i1} to $O_{ij_{\max}}$. Its length equals the total transmission time of the packet and hence we are able to compute the transport delays of the packets for the schedule as shown by Corment, Leiserson and Rivest [10].

The critical path can be found by transforming the longest path subproblem as the shortest path problem by switching the signs of the weights of the arcs and nodes. Hence, it is possible to apply Bellmann-Ford algorithm [5] or Depth first search (DFS) to find out a complete completion order of the operations that can be employed to compute the corresponding maximum delay as presented by [10].

Note that the schedule described by a directed graph is feasible provided that the graph is *acyclic*, i.e., there are no cycles in the graph. This result can be shown easily by assuming the controversy. Suppose that there is a cycle $\{O_{ij}, O_{i'j'}, \ldots, O_{ij}\}$, which implies that the operation O_{ij} needs to be completed before O_{ij}. On the other hand, if there is no cycle in the graph, a path P gives a completion order of a certain subset of operations.

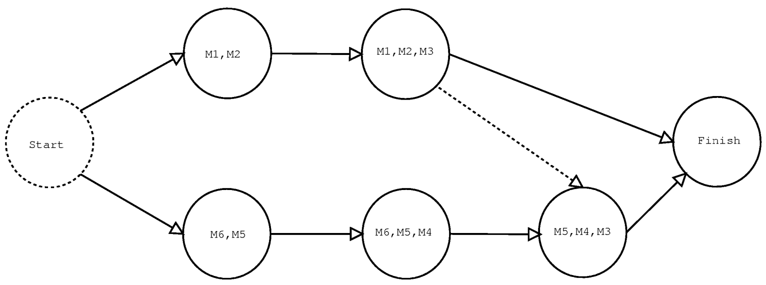

Figure 3: An example of an improved schedule

We denote the set of feasible schedules by $\mathcal{S}$. We also interchangeably refer the schedules to as *solutions* and respectively the set of feasible schedules to as the set of *feasible solutions*.

3 A Taboo Search Algorithm

The basic idea in *local search* methods is to define a *neighborhood* for each solution, iteratively search for the best solution by exploring this neighborhood and thus gradually reach the optimum (c.f. steepest ascent methods [4] in nonlinear programming). The Figures 2 and 3, where a disjunctive arc is redirected, give a simple illustration on how a solution can be locally improved by reversing one or more disjunctive arcs.

This iterative exploration of the neighborhood does not, however, guarantee a global convergence of the method but convergence to a local optimum only. Hence, a *taboo list* is additionally defined in *taboo search* methods. A taboo list declares a certain subset of the neighborhood as taboo to prevent the algorithm to terminate in a local optimum.

To favor good solutions it is also often defined an *aspiration criteria*. It allows the algorithm to choose a taboo neighbor as the next solution if a taboo neighbor solution is the best solution reached so far. This kind of aspiration criteria is referred to as *elitism*.

Recall that we aim to minimize the objective function $f(x) = \max D_i$ over feasible solutions $x \in \mathcal{S}$. Let us now denote the neighborhood of a solution $x \in \mathcal{S}$ by $\mathcal{N}(x) \subset \mathcal{S}$. Assume that taboo search has generated a sequence $(x_1, x_2, \ldots, x_k)$ of solutions. Moreover, denote the taboo neighbors by $\mathcal{T}(x_k, x_{k-1}, \ldots, x_{k-K})$, the neighbors satisfying the aspiration criteria by $\mathcal{A}$ and the best solution found so far by x^* and its the related maximum delay by $f^* = f(x^*)$.The parameter K is referred to as *length of the taboo list.*

Now, the generic rough formulation of the taboo search is given as follows: [13]

1. Set $k = 0$ and choose an initial solution $x_0 \in \mathcal{S}$
2. Choose the next solution $x_{k+1} = \text{argmin}\{f(x) | x \in \mathcal{N}(x_k) \backslash \mathcal{T}(x_k, x_{k-1}, \ldots, x_{k-K}) \bigcup \mathcal{A}\}$
3. If $f(x_{k+1}) < f^*$, set $x^* = x_{k+1}$ and $f^* = f(x_{k+1})$.

4. Set $k = k + 1$. Go to step 2.

The choice of the initial solution x_0 and the neighborhood $\mathcal{N}$ and the definition of the taboo list $\mathcal{T}$ are strongly problem dependent. In the following subsections we describe how the neighborhood and the taboo list are chosen in our scheduling problem.

3.1 Generating the initial solution

We fix a arbitrary total order between the packets. Using this order as basis we generate a feasible initial solution for the taboo search. For any two transmission operations O and O' reserving a common router we define a order by using the order between packets which these belong. Namely if p is the packet that O belongs and p' is the packet that O' belongs, then we create a arc (O, O') if $p < p'$ and (O', O) otherwise.

3.2 Neighborhood Structure

The neighborhood structure applied in our case is based on *reversals* of the disjunctive arcs on a critical path of a critical packet as alluded in Chapter 2.4. This approach is reasonable because it pays attention on reordering the operations that are limiting the performance of the current solution.

In our case, a neighbor of a solution x is obtained by reversing an arc on a critical path of a critical packet, and the whole neighborhood $\mathcal{N}(x)$ can be generated by reversing each of those arcs separately. This choice of the neighborhood follows the ideas presented in JSP literature.

It is also possible to define more sophisticated neighborhood structures. Typically those structures require reversing of more than one arc at a time, see, e.g., [18], [3] and [14].

However, this makes the neighborhood larger and, hence, its exploration requires more CPU time.

3.3 Taboo List

The taboo list is commonly defined through arc reversals, too. Simply, once an arc is reversed it is forbidden to re-reverse it for K next iterations. Most of the existing implementations of the taboo search presented in JSP literature use this approach, see, e.g., [18].

Besides the selection of the neighborhood structure, the choice of the length of the taboo list plays a crucial role when fitting the algorithm to the problem being solved.

If K is too small the algorithm terminates to a local optimum only, because the set of taboo neighbors in too small and hence the algorithm is greedy in searching the best solutions locally. Likewise, too large values of K may prevent the algorithm to converge even to a local optimum.

4 Implementation of the Solution Method

We have implemented the taboo search method for our scheduling problem by applying C++ programming language [20] and the standard template library (STL) [17].

Here the implementation is presented roughly; for technical details we suggest the reader to familiarize oneself with the source code.

4.1 Inputs

The program takes as an input a data matrix presenting the network topology, that is, the routers and the arcs describing which routes the data flows through the network. Because we assume that each router transmits packets of constant size at constant rate, the traffic in the data network is presented by entering the time it takes to transmit a packet and the time interval between individual packets to be transmitted or received. The input is given as a text file. The times are given as integers.

4.2 Outputs

The implementation offers a variety of options to represent the solution to support varying user needs. It is possible to view the optimal schedule using one of three different output options, or using an arbitrary permutation of those. The output is printed in a text file too.

First output option produces a list of transmit-receive events in chronological order. At each row time of the event, transmitting router, receiving router and related the job is printed. In case of multiple coincident events each of them is printed out on separate row.

Second output option produces a table of events in a router oriented form. Each row of the table describes events in the data network at a single moment. The first column contains event and other columns contain the possible event in the corresponding router. Events are described with following notation: 's' refers to sending (transmitting) a packet, 'r' receiving a packet. Packet number is presented in brackets after event type.

Third output option produces a table of events in a packet oriented form. As in the second option, each row of the table describes event(s) in the net at a single moment. The first column contains the event times in chronological order of and other columns contain the possible event related to the corresponding packet. Sending-receiving event of packet at issue is described with notation 's>r', where 's' refers to the sending router and 'r' to the receiving router.

4.3 Some Details of the Implementation

A text file containing the matrix representation of the network topology and the network parameters is passed for the main program. Class Initializer creates all the sending and receiving tasks in the radio network. After that, class Schedule builds up a feasible initial schedule, which is being improved upon by the taboo search to gradually reach better and better solutions. UML presentation of the main program and the relationship between class Initializer and class Schedule are introduced in Figure 4. Figures 5 and 6 give details of the Initializer and Schedule classes.

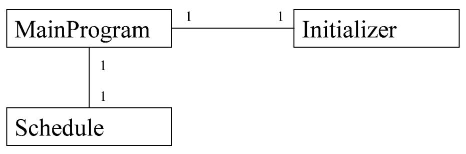

Figure 4: Rough UML representation of the implementation

Most important part in the algorithmic sense is the Schedule::MaxDelay routine which

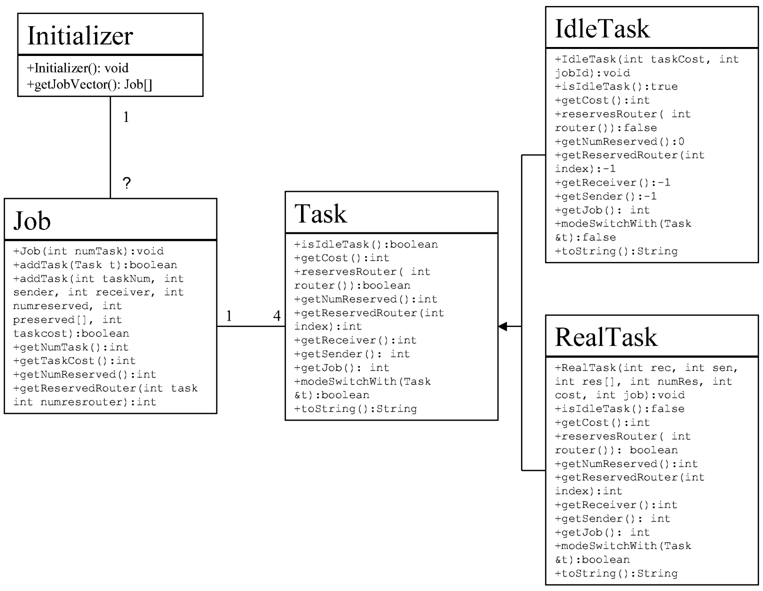
Initializer
+Initializer(): void
+getJobVector(): Job[]
1
?
Job
+Job(int numTask):void
+addTask(Task t):boolean
+addTask(int taskNum, int sender, int receiver, int numreserved, int preserved[], int taskcost):boolean
+getNumTask():int
+getTaskCost():int
+getNumReserved():int
+getReservedRouter(int task int numresrouter):int
1
4
Task
+isIdleTask():boolean
+getCost():int
+reservesRouter(int router()):boolean
+getNumReserved():int
+getReservedRouter(int index):int
+getReceiver():int
+getSender(): int
+getJob(): int
+modeSwitchWith(Task &t):boolean
+toString():String
IdleTask
+IdleTask(int taskCost, int jobId):void
+isIdleTask():true
+getCost():int
+reservesRouter(int router()):false
+getNumReserved():0
+getReservedRouter(int index):-1
+getReceiver():-1
+getSender():-1
+getJob(): int
+modeSwitchWith(Task &t):false
+toString():String
RealTask
+RealTask(int rec, int sen, int res[], int numRes, int cost, int job):void
+isIdleTask():false
+getCost():int
+reservesRouter(int router()): boolean
+getNumReserved():int
+getReservedRouter(int index):int
+getReceiver():int
+getSender(): int
+getJob(): int
+modeSwitchWith(Task &t):boolean
+toString():String

Figure 5: UML representation of the initializer class

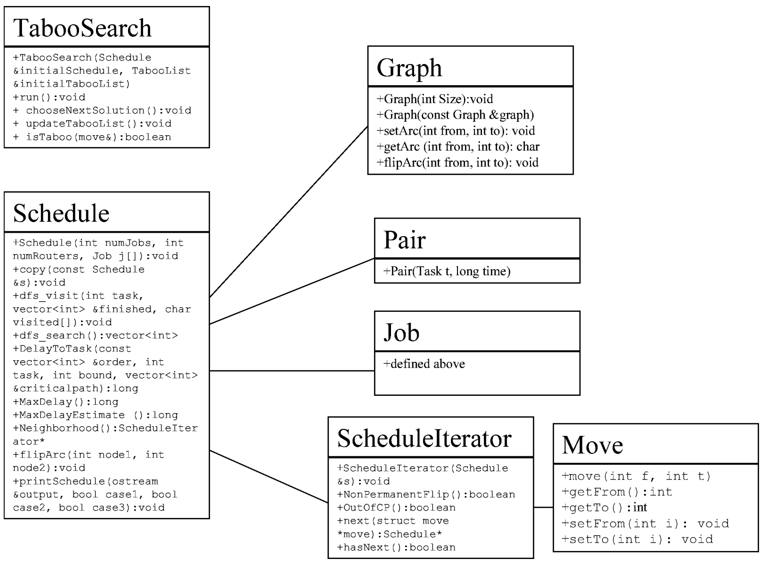
TabooSearch
+TabooSearch(Schedule &initialSchedule, TabooList &initialTabooList)
+run():void
+ chooseNextSolution():void
+ updateTabooList():void
+ isTaboo(move&):boolean
Schedule
+Schedule(int numJobs, int numRouters, Job j[]):void
+copy(const Schedule &s):void
+dfs_visit(int task, vector<int> &finished, char visited[]):void
+dfs_search():vector<int>
+DelayToTask(const vector<int> &order, int task, int bound, vector<int> &criticalpath):long
+MaxDelay():long
+MaxDelayEstimate ():long
+Neighborhood():ScheduleIterator*
+flipArc(int node1, int node2):void
+printSchedule(ostream &output, bool case1, bool case2, bool case3):void
Graph
+Graph(int Size):void
+Graph(const Graph &graph)
+setArc(int from, int to): void
+getArc (int from, int to): char
+flipArc(int from, int to): void
Pair
+Pair(Task t, long time)
Job
+defined above
ScheduleIterator
+ScheduleIterator(Schedule &s):void
+NonPermanentFlip():boolean
+OutOfCP():boolean
+next(struct move *move):Schedule*
+hasNext():boolean
Move
+move(int f, int t)
+getFrom():int
+getTo():int
+setFrom(int i): void
+setTo(int i): void

Figure 6: UML representation of the schedule class

calculates the maximum delay experienced during the snapshot. We take advantage of the acyclicness of the schedule. We also noticed that one does not have to recalculate all the delays after reversing one arc.

We calculate the earliest starting times of the operations by first using DFS to do a topological sort and afterwards calculating the earliest starting time for each task. If we have previously calculated the earliest starting times for some schedule and reverse an arc (a,b), then we do not have to recalculate all the earliest starting times for new schedule. To achieve this goal, we need to perform a DFS starting from node b and producing new ordering only for the nodes affected by the reversion of the arc. Thus, we have to recalculate the delays only for these nodes.

For interested reader more details can be found in the source code from the class Schedule. The methods are MaxDelay for calculating maximum delays, dfs_search for DFS and InitializeDelays for calculating the earliest starting times. Routine flipArc recalculates the maximum delay after flipping an arc. It uses partial_dfs_search to complete DFS search only starting from a certain node.

Schedule graph is stored as a matrix which is suitable for dense graphs. We believe schedules on the small networks to be rather dense, and on the other hand the limiting factor in computational resources seems to be the computation speed, not the memory taken by the program.

5 Numerical Examples

The implementation was tested with two different cases. The first case represented a simplified network and the other case was closer to a real network.

According to a real network the bandwidth was specified to be 24 Mbps and the transmission rate was assumed to be asymmetric: each router received 256 kbps from the Internet and sent 128 kbps to the Internet.

As the transmission time of a packet was set to 1 ms, the following parameters were fixed:

- packet size 24kb (3kB)
- time interval between packets coming from Internet 100ms
- time interval between packets sent to Internet 200ms

The setup time was set to 3 ms.

When running the taboo search algorithm the following parameters were varied

- number of packets sent to and from the Internet
- maximum number of iterations
- the length of the taboo list

5.1 Small Scale Radio Network

We simulated the taboo search algorithm with a network described in Figure 7.

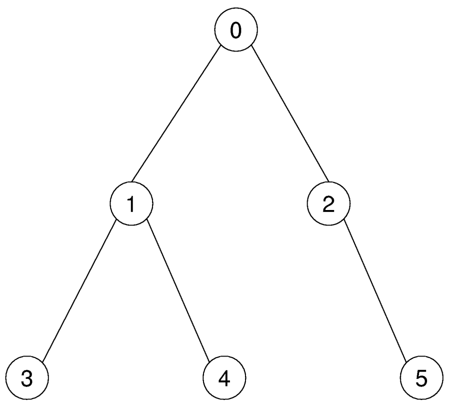

Figure 7: The small scale network

The network consist of six routers (zero refers to the gateway router) and radio links between them.

Four different small scale test cases are reported here to illustrate the features of our solution methodology by completing 10000 iterations and setting the taboo list length to 10, 50 and 1000:

1. Each router receives one packet and transmits two packets. Maximum delay of the initial solution was 246.
2. Each router receives two packets and transmits four packets. Maximum delay of the initial solution was 646.
3. Each router receives four packets and transmits eight packets. Maximum delay of the initial solution was 1446.
4. Each router receives eight packets and transmits 16 packets. Maximum delay of the initial solution was 3064.

The results of these instances are presented in Tables 1–4 and Figures 8–11.

The results show that the quality of the solution is strongly dependent on the length of the

Table 1: Reached maximum delay when each router transmits one packet and receives two.

Length of the taboo list	MaxDelay	Relative improvement
10	224	8.94%
50	16	93.50%
100	15	93.90%

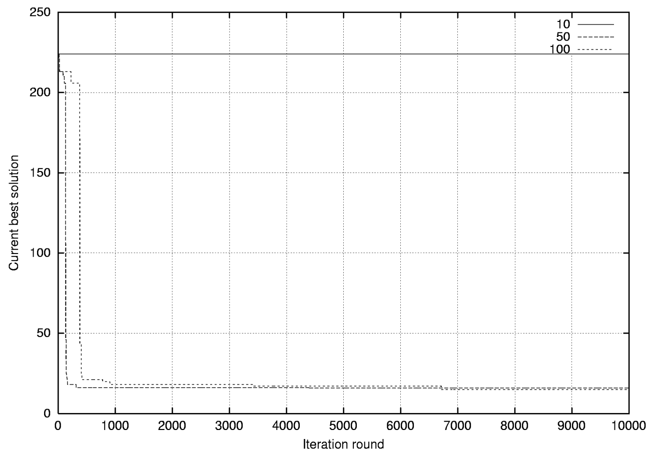

Figure 8: Reached maximum delay as a function of iterations when each router transmits a packet and receives two packets

Table 2: Reached maximum delay when each router transmits two packets and receives four.

Length of the taboo list	MaxDelay	Relative improvement
10	624	3.41%
50	315	51.24%
100	120	81.42%

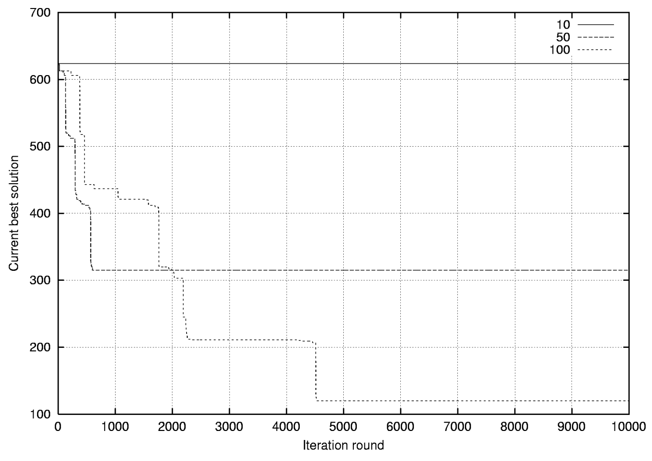

Figure 9: Reached maximum delay as a function of iterations when each router transmits two packets and receives four packets

Table 3: Reached maximum delay when each router transmits four packets and receives eight.

Length of the taboo list	MaxDelay	Relative improvement
10	1424	1.52%
50	1221	15.56%
100	819	43.36%

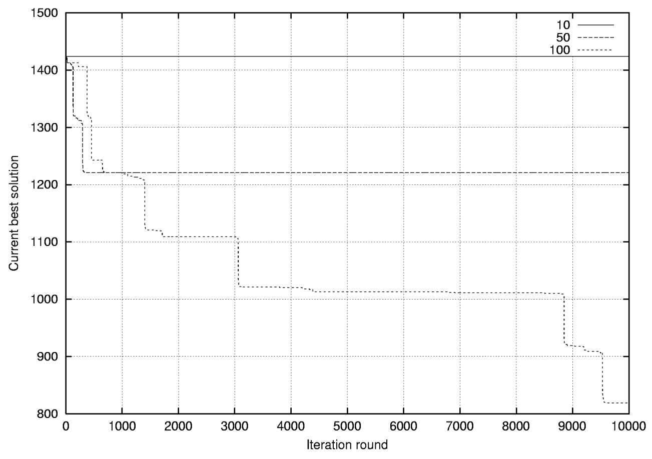

Figure 10: The reached maximum delay as a function of iterations when each router transmits four packets and receives eight packets

Table 4: Reached maximum delay when each router transmits eight packets and receives 16.

Length of the taboo list	MaxDelay	Relative improvement
10	3024	1.31%
50	2719	11.26%
100	2621	14.46%

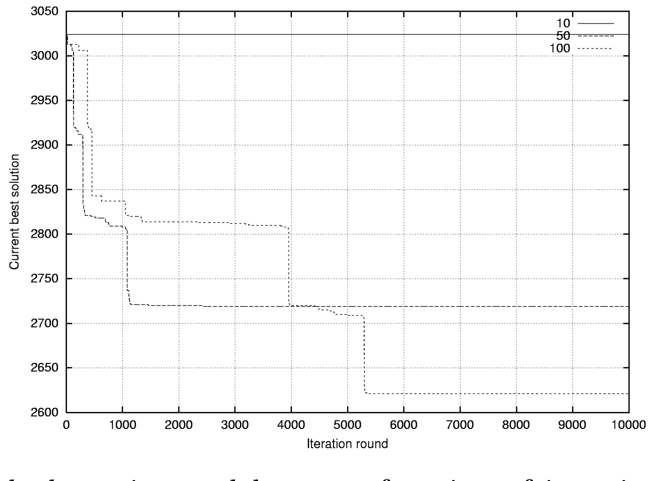

Figure 11: The reached maximum delay as a function of iterations when each router transmits eight packets and receives 16 packets

taboo list. During the first iterations the taboo list length of 50 seems to be working well in our test cases. Nevertheless, a better solution is found in the long run when increasing the length of the taboo list to 100. Hence, our empirical results support the conclusions given in Chapter 3.3; especially, we see that the smaller the length of the taboo list is the more likely the algorithm terminates to a local optimum only.

These observations suggest that perhaps the length of the taboo list should be varied during the iterative solution process. However, this possibility was not tested here.

5.2 A Realistic Large Scale Network

This case is an instance of an imaginary but realistic network and the routers of the data network were distributed with the following rule: 40% of the routers were linked directly to the gateway and 40% of the routers were linked through two links to the gateway. The rest of the routers (20%) were linked through three links. The network consisted of 41 nodes in total.

The first example reported here is an instance where each router received four packets and transmitted two. Hundred iterations were computed and the taboo list length was 10, 30 and 50. Basically, this number of iterations is too small being a compromise between comptation time and quality of results. Running this test instance took not more than 30 minutes on a Compaq Tru64 UNIX platform.

Figure 12 presents the maximum delay of the best solution found so far as the function of number of iterations, and the table 5 presents the numerical values of the maximum delays of the best solutions the algorithm had found by its termination. It can be seen that the length of the taboo list really matters in this case, too. Also, we saw that our methodology is suitable for solving this kind of scheduling problems. Nevertheless, in large scale problems the lenght of the taboo list need to be carefully reconsidered and

Table 5: The reached maximum delay when each router transmits one packet and receives two packets.

Taboolist	MaxDelay	Relative improvement
10	361	23.35%
30	353	25.05%
50	343	27.18%
100	343	27.18%

then the solution process need to be continued long enough.

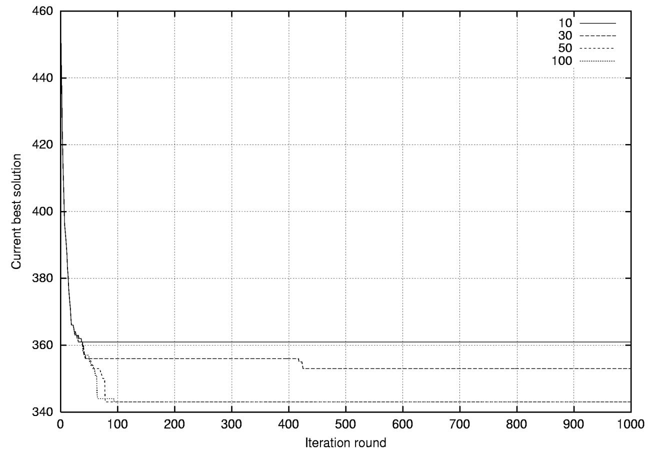

Figure 12: The reached maximum delay as a function of iterations when each router transmits a packet and receives two packets

Similar results are presented in Figure 13 and table 6 when the number of transmitted packets was three and the number of received ones was six.

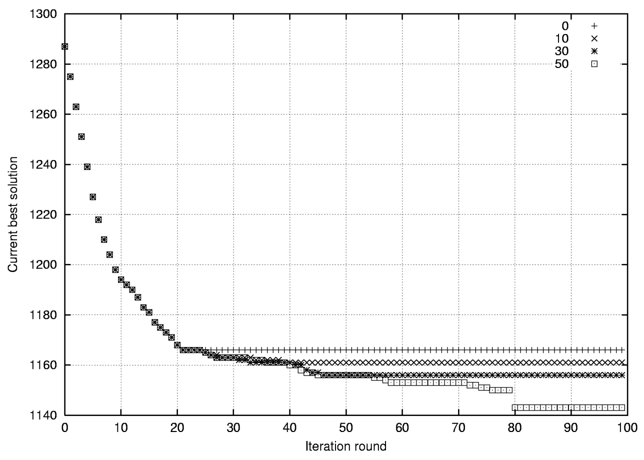

Figure 13: The reached maximum delay as a function of iterations when each router transmits three packets and receives six packets

Table 6: The reached maximum delay when each router transmits three and receives six packets.

Tabulista	MaxDelay	Relative improvement
0	1166	10.24%
10	1161	10.62%
30	1156	11.01%
50	1143	12.01%

6 Discussion

Our scheduling problem is $\mathcal{NP}$-complete, and even one of the most difficult among them [14]. Hence, solving it requires much CPU time even if an exact optimum is not being searched. Our solution methodology was based on the assumption that an exact optimum is not needed and therefore we heavily focused on applying the taboo search heuristics. This approach appeared to be successful for our purposes because it offers a relatively sound framework for solving large scale scheduling problems.

A common problem in the large scale optimization of the $\mathcal{NP}$-complete problems is that not even a good estimate for the exact optimum is not typically known. This makes the thoughtful analysis of the applied methodology difficult. Nevertheless, the presented methodology offers a comparative benchmark for the scheduling heuristics applied at the hardware level. Additionally, our framework offers a basis for improving existing solutions generated by any other heuristics.

The formulation of the taboo search is strongly dependent of the underlying problem being solved. The core of the taboo search involves definition of the neighborhood and choice of the taboo list length. Likewise, the generation of the initial solution affects strongly the performance of the algorithm. There is a variety of different possibilities to vary the taboo search in certain types of scheduling problems discussed in literature. Nevertheless, systematic variation of our solution heuristics is not within scope of this work but is an interesting area for future research. Especially, references [3], [7], [8], [12], [16], [18] and [19] and references in them provide a variety of ideas on how to develop our taboo search method. Basically, the simplest modifications include testing whether enlargement of the neighborhood by allowing reversions of multiple disjunctive arcs is reasonable or not. Also, a deeper understanding of the implications of the taboo list length is needed and it would be interesting to test varying it during the iterative solution process.

More than 95% of CPU time needed to run the algorithm is used when calculating the critical paths and the related maximum delays. Hence, development of an efficient approximation method to search the length of the critical path is needed. We suggest the interested reader to grasp with [3] for some simple ideas.

Another interesting pattern in this problem is that by assuming constant periodic nonbursty traffic in the network the optimal schedule will be periodic too. Periodic solutions are described in literature, see, e.g., [9]. Nevertheless, the presented models are based on searching an exact optimum and seem to be even more difficult to solve than the LP-formulation given in Chapter 2.3. Hence, we do not have addressed this aspect seriously so far but this could be an interesting area for future research as well.

References

[1] Edward Anderson, Celia Glass, and Chris Potts. Machine scheduling. In *Aarts, E. and J.K. Lenstra (eds.). Local Search in Combinatorial Optimization.* John Wiley & Sons, 1997.

[2] Nirwan Ansari and Edwin Hou. *Computational Intelligence for Optimization.* Kluver, 1997.

[3] Egon Balas and Alkis Vazacopoulos. Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 44(2):262–275, February 1998.

[4] Mokhtar Bazaraa, Hanif Sherali, and C. Shetty. *Nonlinear Programming, Theory and Algorithms.* John Wiley and Sons, 1993.

[5] Dimitri Bertsekas. *Network Optimization: Continuous and Discrete Models.* Athena Scientific, 1998.

[6] Dimitri Bertsimas and John Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1997.

[7] Peter Brucker, Andreas Drexl, Rolf Möhring, Klaus Neumann, and Erwin Pesh. Resource-constrained project scheduling: Notation classification, models and methods. *European Journal of Operations Research*, 112(1):3–41, January 1999.

[8] In-Chan Choi and Osman Korkmaz. Job shop scheduling with sequence-dependent setup times. In *Proceedings of INFORMS Conference on Information Systems & Technology*, Washington D.C., USA, May 1996.

[9] Philippe Chrétienne, Edward Coffmann, Jan Lenstra, and Zhen Liu. *Scheduling Theory and its Applications.* John Wiley & Sons, 1997.

[10] Cormen, Leiserson, and Rivest. *Introduction to algorithms.* MIT Press, 1990.

[11] Salah Elmaghraby. *Activity Networks Project Planning and Control by Network Models*. John Wiley & Sons, 1977.

[12] Roland Heilmann. Resource-constrained project scheduling: a heuristic for the multi-mode case. *OR-Spektrum*, 23(3):335–357, 2001.

[13] Hindsberger and Vidal. Tabu search - a guided tour. *Control and Cybernetics*, 29(3), 2000.

[14] Ananat Jain, Balasubramanian Rangaswamy, and Sheik Meeran. New and "stronger" job-shop neighbourhoods: A focus on the method of nowicki and smutnicki (1996). *Journal of Heuristics*, 6(4):457–480, September 2000.

[15] Alf Kimms. *Mathematical Programming and Financial Objectives for Scheduling Projects*. Kluver, 2001.

[16] Yazid Mati, Rezg Nidhal, and Xiaolan Xieaolan. A taboo search approach for deadlock-free scheduling of automated manufacturing systems. *Journal of Intelligent Manufacturing*, 12(5):535–552, November 2001.

[17] David Musser and Atul Saini. *STL tutorial & reference guide: C++ programming with standard template library*. Addison-Wesley, 1996.

[18] Eugeniusz Nowicki and Czeslaw Smutnicki. A fast taboo search algorithm for the job shop scheduling. *Management Science*, 42(6):797–813, June 1996.

[19] Cintia Scrich, Vinícius Armentano, and Manuel Laguna. Tardiness minimization in a flexible job shop: a tabu search approach. Journal of Intelligent Manufacturing, To Appear.

[20] Bjarne Stroutrup. *The C++ Programming Language*. Addison Wesley, 1997.

[21] Vesa Timonen. Static fairness criteria in telecommunications. Special assignment, Networking Laboratory, Helsinki University of Technology, 2002.

[22] van Laarhoven, Aarts, and Lenstra. Job shop scheduling by simulated annealing. *Operatons Research*, 40(1):113–125, 1992.