Metaheuristics for the multicast tree scheduling problem

Bachelor Thesis

Lukas F. Lang
e0504043@student.tuwien.ac.at

Accomplished at the
Database and Artificial Intelligence Group,
Institute for Information Systems,
Vienna University of Technology.

Supervised by Priv.-Doz. Dr. Nysret Musliu
(Vienna University of Technology) and
Priv.-Doz. Dr. Sandford Bessler
(Telecommunications Research Center Vienna).

April 26, 2010

Abstract

Since online streaming content and Video-On-Demand systems have reached widespread distribution over the internet and the requested content is known to originate from a long-tail Zipf distribution, multicasting has become a broadly adopted technique to deal with repeatedly requested content objects. Recently, rather new *push-VoD* techniques have been tested where the content is pushed to the customer's local device in hours of low network utilization, e.g. at night. The Repeated Content Download Problem (RCDP) is to find a schedule for such a system, which satisfies all requests with a minimal number of time periods needed so that the requested content titles are transmitted with a minimal number of repetitions. In this work, we propose the application of metaheuristics such as Tabu Search, Variable Neighborhood Search and Iterated Local Search to the problem and show that even for large instances, acceptable runtimes can be achieved.

Keywords: Video-on-Demand, multicast, scheduling, Tabu Search, Variable Neighborhood Search, Iterated Local Search.

Acknowledgments

I dedicate this work to my parents who supported me in any respect and enabled my academic studies. I am deeply grateful.

My sincerest appreciation goes to Dr. Sandford Bessler and Dr. Nysret Musliu for the inspiration I found in working with them. My thesis would have not been possible without their effort.

I want to thank my colleagues from university and my friends for the endless revealing discussions, the entertaining moments and their caring.

Contents

1 Introduction **5**

2 Problem definition and objectives **7**
2.1 Formalization of the RCDP 7
2.2 Summary of symbols 10
2.3 NP-Completeness 10

3 IP formulation **12**

4 Local search techniques **12**
4.1 Hill Climbing 14
4.2 Tabu Search 15
4.3 Iterated Local Search 15
4.4 Variable Neighborhood Search 17

5 Metaheuristic formulation of the RCDP **18**
5.1 Search space 18
5.2 Evaluation function 19
5.3 Neighborhood structures 20
5.3.1 Merge neighborhood 20
5.3.2 Min-conflict neighborhood 21
5.3.3 Shift neighborhood 23
5.3.4 Compress neighborhood 25

6 Implementation **25**

7 Experimental analysis **26**
7.1 Statistical methods 28
7.2 Instances 28
7.3 Parameter determination 29
7.3.1 Tabu length 30
7.3.2 Perturbation size 30
7.4 Results 31

8 Conclusion and future work **33**

List of Tables

1 Summary of symbols of the IP formulation. 13
2 Summary of variables measured. 27
3 Test instances used for the RCDP. 30
4 Summary of metaheuristics, parameters and values. 31
5 Comparison of various tabu lengths used for Tabu Search. . . 39
6 Perturbation size of ILS using Hill Climbing as embedded local search. 39
7 Evaluation function. 40
8 Running times. 40
9 Splitting rate and top level utilization. 41
10 Neighborhood 41

List of Figures

1 A tree-like content distribution network (CDN). 6
2 Example of a multicast tree. 9
3 A possible move in the merge neighborhood. 21
4 A possible move in the min-conflict neighborhood. 22
5 A possible move in the shift neighborhood. 23
6 A swap move. 24

List of Algorithms

1 Pseudocode of Hill climbing (minimization problem). 14
2 Pseudocode of Tabu Search. 16
3 Pseudocode of selectAdmissible(·). 16
4 Pseudocode of Iterated Local Search. 17
5 Pseudocode of Variable Neighborhood Descent. 18

1 Introduction

Over the last years, real-time entertainment services such as streamed or buffered video and audio have emerged and today contribute to a significant part of the worldwide internet traffic. According to Sandvine Inc.'s report on global broadband usage [21], traffic on real-time entertainment has more than doubled from 12,6% in 2008 to 26,6% in 2009. *Peer to Peer (P2P)* traffic has decreased from 31,6% to 20,4% in favor of *Video-on-Demand (VoD)* services such as YouTube[1].

In practice, uncoordinated streaming and progressive download technologies have achieved widespread distribution due to a lack of sophisticated multicast streaming technologies, but simultaneously suffer from major defects. It is symptomatic for such *True Video-on-Demand* systems, to operate at low performance/cost efficiency and to poorly scale with an increasing number of requests and traffic generated [16].

A common approach to cope with these issues is to batch several requests and broadcast the same content simultaneously to multiple sinks. This technique is called *multicasting* and brings two main benefits. On the one hand, multicasting reduces congestion of network links by transmitting a stream to all receivers only once and on the other hand, the content provider issues the time of broadcasting, hence network resources can be scheduled more effectively.

To determine the *Quality of Service (QoS)* of such a system from the user's perspective, performance measures such as latency, defection rate or fairness can be specified. Ma and Shin [16] give a structured and extensive overview of multicast Video-on-Demand systems, requirements, architectures and discuss QoS measures.

Recently, several attempts have been made to develop and establish *Push Video-on-Demand* technologies in the field of IPTV, where a content is requested by a user and is then distributed to the user's personal video recorder or hard disk in times of lower network utilization, e.g over night. A tradeoff that comes in mind with this approach is the user's waiting time, which is the time between a user issues a request and the content was downloaded to the local device. Therefore, an upper bound must be guaranteed to the user, e.g. a 24h interval. The obvious advantage of such a system is the capability to bundle all requests for a certain period of time and a content

[1] http://www.youtube.com

and simultaneously stream multiple contents at full network capacity.

The underlying network, called *Content Distribution Network (CDN)*, typically consists of several sinks, intermediate nodes and content sources. Typically, intermediate nodes are servers, routers and switches. The most basic instance of a CDN is a tree shaped network with a single source (the content provider) distributing content to the leafs, i.e. the customers (see Figure 1).

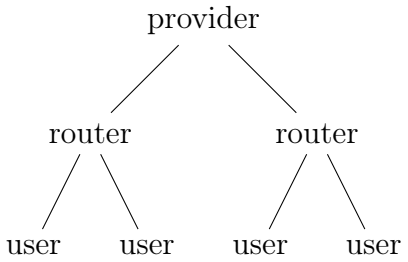

Figure 1: A tree-like content distribution network (CDN).

The *Repeated Content Download Problem (RCDP)* is to find an assignment for a number of multicast streams in a tree-like content distribution network, which satisfies all requests within as few periods as possible and does not exceed bandwidth limits of the underlying network [3].

In the last two decades, many researchers have worked on similar problems or problems related to multicasting. Aggarwal et al. studied the effectiveness of multicast streams in [1]. Furthermore, Wu et al. compare a greedy algorithm for multicast packing to network coding in [25]. In [4], Chen, Günlük and Yener discuss the packing of group multicast trees, that is multicast trees with multiple sources, which is also referred as *many-to-many multicasting*. In 1999, Wang et al. [24] show the application of a brunch-and-bound algorithm to construct multicast trees.

Since the problem is strongly connected to the *Steiner tree packing problem*, which is to find a maximum number of Steiner trees covering a set of terminal vertices, previous work on Steiner tree packing has strong influences in multicast packing. In terms of the VoD it is to find as many edge-disjoint Steiner trees in a network multi-graph as possible, rooted in the content provider and covering all users. In [13], Jain et al. discuss an approximation algorithm for a similar problem. Furthermore, Randaccio and Atzori proposed a genetic algorithm for another version of the problem in [22].

In this thesis, we will tie in with Bessler's formulation of the problem in [3], as it provides a basic level of abstraction, focuses on the scheduling aspects of the problem and brings major simplifications w.r.t. the network topology. The aim of this thesis is to show the application of metaheuristic techniques such as Tabu Search, Variable Neighborhood Descent and Iterated Local Search to the problem, provide experimental results and come up with an interpretation.

The remainder of this thesis is structured as follows: In Section 2, we will develop a detailed formulation of the problem, suitable for the application of metaheuristics. Section 3 briefly discusses a possible IP formulation given by Bessler in [3]. In Section 4 and Section 5, we examine the applied metaheuristics in general and come up with a metaheuristic formulation of the RCDP. Section 6 gives details on our implementation and finally, in Section 7, the experimental results will be analyzed.

2 Problem definition and objectives

Given a content distribution network, a number of users connected to it and a Video-on-Demand system providing a selection of content objects for purchase, it is generally known that the popularity of the content objects follows a long-tail Zipf distribution [2,5,6]. Since the capacity of the installed network is fixed and should be operated at a certain point of utilization and the download rate for single titles even varies during the time of the day, it is in the interest of the provider to use intelligent scheduling mechanisms to deal with the upcoming requests. In this section, we will develop a consistent formulation of the Repeated Content Download Problem before we proceed with the proposed metaheuristics.

2.1 Formalization of the RCDP

A basic Video-on-Demand system consists of a content distribution network with users located at the leafs, issuing requests for a selection of offered titles. Let U be the set of users participating and let Q bet the set of requests issued. Every request $q \in Q$ is assigned to a user $q(u) \in U$ and a content title $t(q) \in T$ in the set of requested titles T. Furthermore, $d_t \in \mathbb{R}, d_t > 0$ defines the required link capacity for the download of title t.

Definition 1. A *Content Distribution Network (CDN)* is a 4-tuple $G = (V, E, \zeta, s)$, with (V, E) being a tree of order $o \in \mathbb{N}$ (i.e. the depth of the tree) where V is the set of vertices, E is the set of edges, $\zeta : E \to \mathbb{R}^+$ a function which assigns a capacity c_e to every edge $e \in E$ and s is the source vertex of all streams.

Furthermore, let us specify the set of users U more precisely as $U \subset V \wedge \forall u \in U \Leftrightarrow deg(u) = 1$, i.e. all users are located at the leafs of the tree. Given content distribution network G, there exists exactly one path $\mathcal{P} = \{v_0, e_0, v_1, e_1, \ldots, e_{n-1}, v_n\}$, with $v_i \neq v_j \Leftrightarrow i \neq j$ in (V, E) from every vertex v_i to every vertex v_j. In a CDN, a *unicast stream* is a path $\mathcal{P}$ between the source $v_0 = s$ and a user $v_n \in U$. The length of every such path $\mathcal{P}$ is constrained by $|\mathcal{P}| = o$ and the maximum bandwidth of a path can be calculated by

$$c_{\mathcal{P}} = \min_{e \in e(\mathcal{P})} c_e,$$

where $e(\mathcal{P}) \subset E$ denotes the edges along $\mathcal{P}$. Consequentially, a stream along a path can only broadcast titles $t \in T$ where $d_t \leq c_{\mathcal{P}}$ applies.

Definition 2. A *multicast tree* $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{P}_i$ is the union of n unicast streams broadcasting the same title $t(\mathcal{P}_i) = t(\mathcal{P}_j), \forall i, j$ such that every edge $e \in e(\mathcal{P}_i)$ and every vertex $v \in v(\mathcal{P}_i)$ is contained only once in $\mathcal{T}$.

This statement reveals the outstanding advantage of a multicast application compared to single unicast streams, which is the sharing of common edges in a distribution network. Figure 2 shows an example of such a tree $\mathcal{T} = \mathcal{P}_1 \bigcup \mathcal{P}_2$ consisting of two streams $\mathcal{P}_1 = \{s, e_4, v_4, e_0, v_0\}$ and $\mathcal{P}_2 = \{s, e_4, v_4, e_1, v_1\}$ broadcasting the same title. Due to the Definition 2, edge e_4 will be stressed only once with the demand d_t of the broadcasted title. We will later refer to a link's capacity utilization as the *congestion* of a link.

Furthermore, let S be the set of available periods and let the binary variable $z^s_{\mathcal{T}}$ denote the assignment of a tree $\mathcal{T} \in \mathcal{M}$ to the period s where $\mathcal{M}$ denotes the set of all trees in the current assignment. The *congestion* λ^s_e on an edge e in period $s \in S$ can now be calculated as

$$\lambda^s_e = \sum_{\mathcal{T} \in \mathcal{M}} r^e_{\mathcal{T}} d_{t(\mathcal{T})} z^s_{\mathcal{T}}, \tag{1}$$

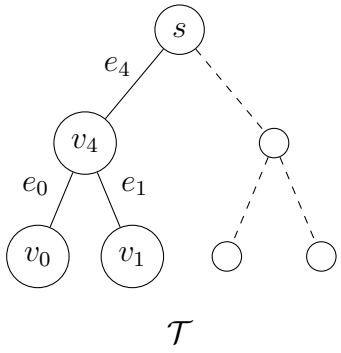

Figure 2: Example of a multicast tree.

where $r^e_{\mathcal{T}} = 1 \Leftrightarrow e \in e(\mathcal{T})$ indicates that edge e is part of $\mathcal{T}$ and t is the title broadcasted by $\mathcal{T}$.

The *Repeated Content Download Problem (RCDP)* is to find an admissible assignment, which is an arrangement of multicast trees $\mathcal{M}$, such that every request $q \in Q$ is part of a tree $\mathcal{T}$, which is assigned to one period and the accumulated demand on no edge in any period exceeds the edge's capacity and the number of periods $|S|$ needed to fulfill all requests is minimal. The objective of the RCDP is to

$$\min \sum_{s \in S, \mathcal{T} \in \mathcal{M}} z^s_{\mathcal{T}}. \tag{2}$$

In order to reduce the number of repetitions a content has to be broadcasted and also to keep the assignment valid, further constraints must be defined. Firstly, for every assignment the statement

$$\lambda^s_e \leq c_e, \forall e \in E, \forall s \in S \tag{3}$$

must hold to ensure that no bandwidth capacity on any edge in any period is violated (further referred to as *capacity constraint*). Moreover, let $x^q_{\mathcal{T}} = 1 \Leftrightarrow t(q) = t(\mathcal{T})$ denote that request q is satisfied by tree $\mathcal{T}$ then

$$\sum_{\mathcal{T} \in \mathcal{M}} x^q_{\mathcal{T}} = 1, \forall q \in Q \tag{4}$$

constraints that every request has to be part of a tree and therefore is satisfied within some period. In further consequence, it has to be ensured that every

tree is assigned to a period, which is expressed by

$$\sum_{s \in S} z_{\mathcal{T}}^{s} = 1, \forall \mathcal{T} \in \mathcal{M}. \tag{5}$$

2.2 Summary of symbols

Symbol	Description
E	Set of edges.
$\mathcal{M}$	Set of trees.
Q	Set of requests.
S	Set of periods.
T	Set of titles.
U	Set of users.
V	Set of vertices.
$e(X)$	Edges of X, where X can be a tree $\mathcal{T}$ or a path $\mathcal{P}$.
$t(X)$	Title of X, where X can be a request q, a path $\mathcal{P}$ or a tree $\mathcal{T}$.
$v(X)$	Vertices of X, where X can be a tree $\mathcal{T}$ or a path $\mathcal{P}$.
λ_e^s	Congestion on link e in period s.
$x_{\mathcal{T}}^q$	Binary variable. 1 if request q is satisfied by $\mathcal{T}$.
$z_{\mathcal{T}}^s$	Binary variable. 1 if tree $\mathcal{T}$ is scheduled for period s.

2.3 NP-Completeness

In the last section, we gave an abstract definition of the problem. It can be shown that the RCDP is $\mathcal{NP}$-complete and no polynomial time algorithm is known to solve the problem, unless $\mathcal{P} = \mathcal{NP}$. Therefore, a polynomial time reduction of the BIN-PACKING problem to the decision variant of the RCDP will be given by us.

Definition 3. The one dimensional BIN-PACKING problem is:

Input: A set of n items $\{a_1, a_2, \ldots, a_n\}$ to be packed into $b \in \mathbb{N}$ or less bins of size $c \in \mathbb{R}$.

Question: Is there an assignment $f : \{1, \ldots, n\} \to \{1, \ldots, b\}$ such that $\sum_{f(i)=k} a_i \leq c, \forall k = \{1, \ldots, b\}$?

Definition 4. The decision version of the RCDP is:

Input: An instance of the RCDP and a $k \in \mathbb{N}$.
Question: Is there a minimal assignment with $|S| \leq k$?

Claim 1. The BIN-PACKING problem is NP-complete.

Proof. See Garey and Johnson [7, 14] for a details. □

Claim 2. The decision variant of the RCDP is NP-complete.

Proof. To justify this claim, we first have to show the membership of the RCDP in $\mathcal{NP}$ and in further consequence that it is $\mathcal{NP}$-hard. To show the membership, it is sufficient to guess a random assignment, which can be evaluated in polynomial time. To show the hardness, we assume that given an instance of the BIN-PACKING problem, an instance of the RCDP decision problem can be constructed by the following procedure in polynomial time.

Create an instance of a RCDP with $T = \{1, \ldots, n\}$, $|S| = b$ and a CDN with $|V| = 2$. Due to the *handshaking lemma* and the connectivity of a CDN, $|E| = 1$ must hold. Thus, edge $(s, V \setminus s)$ connects the source and a single sink. Furthermore, assign capacity $c_e = c, \forall e \in E$ and let $d_i = a_i, \forall i = \{1, \ldots, n\}$ denote that the demand of available titles equal the weights of the items. Subsequently, create a request q_i concerning title i for every item $a_i, \forall i = \{1, \ldots, n\}$. Therefore, $|Q| = n$ and $|T| = n$. Consequentially, the number of multicast trees is therefore restricted to

$$|\mathcal{M}| = |T| = n.$$

Equations (4) and (5) imply that every single request becomes a tree and Equation (4) can be discarded. Constraint (3) and (5) ensure that a CDN's capacity is not exceeded in any period and every tree is assigned to some period. Therefore, a valid assignment x for an instance of the BIN-PACKING problem is a valid assignment for an instance of the decision variant of the RCDP with $b = k$ and vice versa.

Suppose that we find an algorithm which computes a solution for an instance of the BIN-PACKING problem which we have converted into an instance of the RCDP by the given procedure, a polynomial-time algorithm for the decision variant of the RCDP would yield a polynomial-time algorithm for the one dimensional BIN-PACKING problem, which is known not to exist (unless $\mathcal{P} = \mathcal{NP}$).

□

3 IP formulation

In Section 2.1, we gave a very figurative definition of the problem. Above all, we used the concept of *trees* to think of the problem. Bessler gives an integer program (IP) formulation for the RCDP which uses trees rather implicitly [3]. In this work, we will briefly discuss the formulation to provide another point of view to the reader. We have adapted the notation used by Bessler to our notation to provide better comparability.

The major difference to the formulation given in Section 2.1 is that trees are treated implicitly. A tree arises if a title is scheduled for broadcasting in a certain period. Table 1 contains all symbols used in the formulation. The IP is given as follows:

$$\min \sum_{t \in T, t \in S} z_t^s \tag{6}$$

$$\text{subject to } \sum_{s \in S} x_q^s = 1, \forall q \in Q, \tag{7}$$

$$\sum_{t \in T} r_t^e d_{t(q)} z_{t(q)}^s \leq c_e, \forall e \in E, s \in S, \tag{8}$$

$$z_t^s \leq \sum_{q \in Q, t(q)=t} x_q^s, \forall t \in T, s \in S, \tag{9}$$

$$z_t^s \geq \frac{1}{S \cdot T} \sum_{q \in Q, t(q)=t} x_q^s, \forall t \in T, s \in S. \tag{10}$$

4 Local search techniques

In the last sections, we introduced the RCDP and developed a mathematical formulation. The goal is find a globally optimal assignment w.r.t. the objective function (2) and numerous constraints (3), (4) and (5) within an admissible number of computational steps. In other terms, we are looking for an efficient algorithm, i.e. an algorithm running in polynomial time w.r.t. the input size [7]. As we will see in Section 5.1, the exhaustive search space of the RCDP is intractable for exact search algorithms. Moreover, in Section 2.3 we have shown that the RCDP is $\mathcal{NP}$-hard. To the best of our knowledge, there exists neither an exact algorithm for the RCDP nor an approximation

Symbol	Description
E	Set of edges.
Q	Set of requests.
S	Set of periods.
T	Set of titles.
U	Set of users.
$u(q)$	User $u \in U$ that issues request $q \in Q$.
$t(q)$	Title $t \in T$ of request $q \in Q$.
$d(t)$	Download rate for title $t \in T$.
x_q^s	1 if request $q \in Q$ is scheduled for period $s \in S$.
z_t^s	1 if title $t \in T$ is scheduled for period $s \in S$.
r_t^e	1 if title t is routed over edge $e \in E$.
c_e	Capacity of edge $e \in E$.

Table 1: Summary of symbols of the IP formulation.

algorithm. However, we will apply metaheuristics to the problem and show that even for larger input sizes acceptable running times can be achieved.

The minimization variant of a general optimization problem can be written as

$$\min f(x)$$

so that $x \in X$, where $f : X \to \mathbb{R}$ is an evaluation function and X is the set of admissible solutions w.r.t. a number of constraints.

Furthermore, a general *neighborhood function* is defined as

$$\mathcal{N} : X \to \mathcal{P}(X)$$

where $\mathcal{P}(X) = \{U \mid U \subseteq X\}$ is the power-set of X [10]. Consequentially, $\mathcal{N}_m(x)$ denotes the set of neighbors of x which can be reached by a move m and is further referred as to a *neighborhood structure*.

A solution candidate x^* is called a *global optimum* or simply *optimum*, if no $x \in X$ exists with an objective lower than x^*:

$$f(x^*) \leq f(x), \forall x \in X.$$

A solution x_L^* is called a *local optimum* with respect to a neighborhood $\mathcal{N}$, if there exists no $x \in \mathcal{N}(x_L^*)$ having an objective lower than x_L^*:

$$f(x_L^*) \leq f(x), \forall x \in \mathcal{N}(x_L^*).$$

Several local search techniques, which are search procedures starting from an often randomly generated initial solution moving towards better solutions, have been suggested to solve combinatorial optimization problems. To prevent the search from getting stuck in local optima, various approaches such as *Tabu Search*, where certain neighbors are marked as tabu for a number of iterations to explore other areas of the solution space, or *Variable Neighborhood Search*, which uses multiple neighborhood structures to explore the search space, are used. Each of these approaches comes with advantages and disadvantages which we will discuss in the following.

4.1 Hill Climbing

A very simple approach for optimization problems is *Hill Climbing* [9, 20]. Starting from a random initial assignment x_s, the algorithm generates a number of neighbors $n \in \mathcal{N}$, evaluates them and iteratively moves towards better candidates until a global optimum is found or a termination condition is reached. Algorithm 1 illustrates the basic idea.

Algorithm 1: Pseudocode of Hill climbing (minimization problem).

```
Input: Initial solution x_s
x ← x_s
repeat
    x' ← select(N(x))
    if f(x') < f(x) then
        x ← x'
until termination condition
return x
```

The function $select(\cdot)$ takes a set of neighbors as input and depending on the implementation returns either the best or the first better candidate found with respect to f. As a termination condition, a maximum number of iterations or a fixed time duration can be set. Unfortunately, Hill Climbing as simple as it is, is not enough sophisticated to escape local optima efficiently. Therefore, it is often used as embedded procedure within other metaheuristics such as Iterated Local Search or Variable Neighborhood Search.

4.2 Tabu Search

Tabu Search was first proposed by Fred Glover [10] and targets at exploring a wide range of neighbors by "locking" recently accepted neighbors or moves in order to escape local optima [17]. Thus, the algorithm is prevented from reverting recently made changes. Our implementation uses a *recency based memory*, which blocks recently touched multicast trees for a fixed number of iterations.

Let $M : X \rightarrow \mathbb{N}$ be a function that denotes the iteration, a neighbor $x_{c-1} \in X$ recently has been accepted by the heuristic. A candidate $x_c \in X$ explored in the neighborhood $\mathcal{N}(x_{c-1})$ will be accepted if it is the best so far within the current local tour, i.e.

$$\forall y \in \mathcal{N}(x_{c-1}) : f(x_c) \leq f(y)$$

and if x_c is not tabu, denoted by

$$t + M(x_c) \leq c, \tag{11}$$

where c is the current iteration and t is the number of iterations, a move is tabu. However, it might happen that surpassing moves might be blocked. Therefore, the neighbor will also be accepted if an *aspiration criteria* α applies. As an aspiration criteria, the degree of improvement w.r.t. the most recently accepted neighbor x_{c-1} is used so that a candidate x_c gets accepted if

$$\alpha \leq 1 - \frac{f(x_c)}{f(x_{c-1)}}$$

holds, even if it is tabu. Thus strong improvements will be approved.

Algorithm 2 illustrates the basic procedure of Tabu Search. The tabu mechanism is simplified to set operations where $x \in T \Leftrightarrow t + M(x) > c$ defines that a move $x \in X$ is tabu and $x \notin T \Leftrightarrow t + M(x) \leq c$ permits a move. If a neighbor gets accepted, $T \bigcup x \Leftrightarrow M(x) := c$ adds it to the set of tabu moves for the next $c + t$ iterations.

Moreover, the inner loop in Algorithm 2 corresponds to a local tour started from the best solution found so far. Each local tour is guided by the tabu mechanism as shown in Algorithm 3.

4.3 Iterated Local Search

In the last section, we have seen with Tabu Search a guided method to explore the search space. As problems become harder and instance sizes

Algorithm 2: Pseudocode of Tabu Search.

```
Input: Initial solution x_s
x ← x_s
x_l ← x
T = ∅
repeat
    repeat
        x' ← selectAdmissible(x, N(x))
        if f(x') < f(x_l) then
            x_l ← x'
            T = T ⋃ x_l
    until max iterations reached
    if f(x_l) < f(x) then
        x ← x_l
until termination condition
return x
```

Algorithm 3: Pseudocode of selectAdmissible($\cdot$).

```
Input: Local optimum x, set of neighbors N
foreach n ∈ N do
    if n ∉ T ∧ f(n) < f(x) ∨ applies(α, x, n) then
        x ← n
return x
```

increase, the number of local optima and plateaus increases exponentially [20]. Hill Climbing and Tabu Search very often can not find a path to escape. Therefore, *Iterated Local Search (ILS)* was developed by Lourenço, Martin and Stützle [15]. Starting from an initial solution x_s, a local search procedure is done leading to a local optimum x^*. In further consequence, a number of (random) perturbations were applied. The structure and the number of perturbations may also depend on the search history, possibly leading to better exploration. Again, a local search is performed and a local minimum $x^{*'}$ obtained. Assuming that the "jump" and the subsequent search lead to an overall improvement, the candidate is selected by some criterion, which

again could depend on the search history. The power of ILS lies in both it's simpleness in the implementation and the execution as the perturbations can be generated and applied very fast. Algorithm 4 demonstrates the basic ILS.

Algorithm 4: Pseudocode of Iterated Local Search.

Input: Initial solution x_s
$x^* \leftarrow search(x_s)$
repeat
  $x' \leftarrow perturbate(x^*, history)$
  $x^{*\prime} \leftarrow search(x')$
  $x^* \leftarrow select(x^{*\prime}, x^*, history)$
until *termination condition*
return x^*

4.4 Variable Neighborhood Search

In the last sections, we introduced Tabu Search and Iterated Local Search. Both aim at escaping local optima by effectively exploring a great number of neighbors. We have not yet considered the fact that exploring the complete neighborhood $\mathcal{N}(x)$ of a solution $x \in X$ may not lead to a global optimum x^* in reasonable time. *Variable Neighborhood Search (VNS)* was introduced by Mladenović and Hansen in [11, 12, 19] and exploits the fact that a local optimum x_L^* w.r.t. a neighborhood structure is not necessarily optimal to another neighborhood, but a global optimum x^* is optimal w.r.t. all neighborhoods.

Let us denote by $\mathcal{N}_k(x)$ the k-th neighborhood of a solution $x \in X$. Depending on the implementation, the metaheuristic explores the k_{max} neighborhood structures either systematically or randomly. In our approach, we implemented the *Variable Neighborhood Descent (VND)* variant [8, 19] as shown in Algorithm 5. The algorithm starts from the smallest neighborhood and iteratively increases k until k_{max} is reached. However, the ordering of the exploration $\mathcal{N}_1 \subset \mathcal{N}_2 \subset \cdots \subset \mathcal{N}_{max}$ is crucial and will be chosen in a way so that smaller and computationally less complex neighborhoods will be searched first. In Section 5.3, we will introduce various neighborhoods which are used in our VND implementation.

Algorithm 5: Pseudocode of Variable Neighborhood Descent.

```
Input: Initial solution x_s
x* ← x_s
k ← 0
repeat
    repeat
        x' ← select(N_k(x))
        x*' ← search(x')
        repeat
            x_L' ← select(N_k(x*'))
            x*_L' ← search(x'_L)
            if x*_L' < x*' then
                x*' ← x*_L'
                k ← 0
                break
        until max iterations reached
    until k > k_max
    k ← k + 1
until termination condition
return x*
```

5 Metaheuristic formulation of the RCDP

In the last section, the basic algorithms of our approach have been discussed. In this section, we will present a metaheuristic formulation for the RCDP including a short estimation of the search space, the selection of a suitable evaluation function and the definition of capable neighborhood structures.

5.1 Search space

The naive search space, which is the number of states to expand in the worst case scenario in a search tree using an exact algorithm to decide whether an instance of the RCDP has a valid solution or not, is exponential to the size of the input. Solution vector coding will be used to justify this claim.

Claim 3. The exhaustive search space of the RCDP is in $\mathcal{O}(|S|^{|Q|})$.

Proof. Suppose, the row vector $\vec{v} = \{1, \ldots, s\}^q$ denotes a state in the search tree and v_i is the i-th element of $\vec{v}$, then v_i states that request $q_i \in Q$ is is assigned to period v_i. As every request $q \in Q$ can be satisfied in any period $s \in S$, the number of possible states is $|S|^{|Q|}$. □

Assuming that all multicast trees are maximal with respect to the number of requests satisfied and can be assigned to every period without being split, the search space is still $\mathcal{O}(|S|^{|\mathcal{T}|})$.

5.2 Evaluation function

In order to evaluate a possible solution candidate $x \in X$ which was explored in the neighborhood, we define the polynomial $f : X \to \mathbb{R}$ as evaluation function to express the fitness of the candidate. As the heuristic implementation also allows non-feasible assignments w.r.t. the constraints defined in Section 2.1, the evaluation function is designed to consist of three parts: The first part aims at reducing the overall congestion, as defined in Equation (1). The congestion λ_e^s on link $e \in E$ in period $s \in S$ is defined as

$$\lambda_e^s = \sum_{\mathcal{T} \in \mathcal{M}} r^e d_{t(\mathcal{T})} z_{\mathcal{T}}^s$$

and the overall exceeding can be written as a function $e : X \to \mathbb{R}$ as

$$e(x) = \sum_{e \in E, s \in S} max\{0, \lambda_e^s - c_e\}, \tag{12}$$

which should be minimized.

Moreover, we want to reduce the number of periods $|S|$ needed to broadcast all requests. The third part of the objective, which is the number of times a title is repeatedly distributed, should also be kept as small as possible. We can simply express this as the number of trees $|\mathcal{M}|$ needed to satisfy all requests. After adding a multiplicative constant to each part to allow a fine-grained control of the optimization procedure, the objective function f can now be written as the sum of the the three weighted parts as

$$\min f(x) = \alpha e(x) + \beta |S| + \gamma \sum_{\mathcal{T} \in \mathcal{M}, s \in S} z_{\mathcal{T}}^s. \tag{13}$$

As one can see, the objective consists of conflicting criteria so that by reducing the number of periods, the splitting of large trees in favor of avoiding

congested links pays off. Therefore, a title will be broadcasted more often and also the network utilization increases as less edges can be shared by multicast trees. It is highly dependable on an operator's network and cost structure how to set the weights properly. Two major cases can be distinguished: A low number of repetitions reduces the average utilization of the network and increase the user's waiting time for a title. Approaching a high operating level in terms of working load will automatically increase the number of multicast trees to a certain point. Assuming that $|S|$ is fixed and the metaheuristic yields a solution with $e(x) = 0$, then f is equivalent to the IP objective defined in Section 3.

Based on the fact that every request has to be fulfilled within some period, a lower bound for the evaluation function can be calculated. Assuming that there exists an optimal assignment, i.e. the number of repetitions is minimal (i.e. $|\mathcal{M}| - |T| = 0$) and all trees can be scheduled to one period without bandwidth exceeds, the lower bound of an instance of the RCDP is

$$LB(x) = \beta + \gamma|T|.$$

In the next section, we will introduce appropriate neighborhood structures to take care of the different characteristics of the objective.

5.3 Neighborhood structures

Since we now have defined the evaluation function, we proceed with the introduction of neighborhood structures. A neighborhood structure is a function yielding solution candidates for a given solution. Three major aspects are the efficient generation of neighbors, the size of a neighborhood structure and the effectiveness w.r.t. the evaluation function. In the last section, we have seen search procedures drawing neighbors either from single neighborhood structures or from the complete neighborhood, which we have implemented as the compound of all structures.

5.3.1 Merge neighborhood

As the name already reveals, the *merge neighborhood* consolidates two or more multicast trees broadcasting the same title from either the same period s or from multiple periods $s_1, s_2, \ldots, s_n \in S$. Both aim at improving the objective by either decreasing the number of repetitions of the respective title or by reducing the congestion through shared edges. Imagine an assignment

that violates the capacity constraint (3) on some upper edge e in period s. By merging two trees $\mathcal{T}_1$ and $\mathcal{T}_2$, the congestion λ_e^s on edge e decreases if both trees were already scheduled for the same period. In case of merging trees from different periods, the congestion of e stays the same if the new larger tree is scheduled for one of the two periods. If the new tree is assigned to another period, the congestion of e decreases in both source periods. In any case the number of trees in the assignment is shortened. Figure 3 shows the merge of two trees.

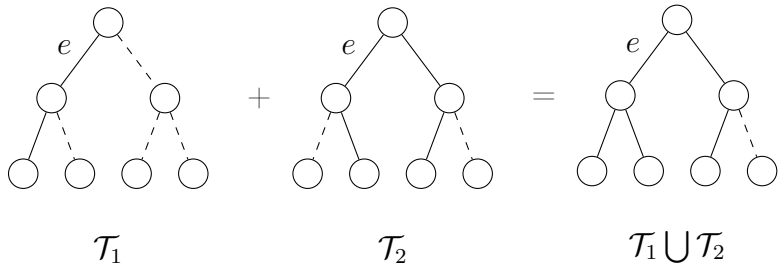

Figure 3: A possible move in the merge neighborhood.

Let us denote by $|\mathcal{M}|$ the number of trees in an assignment and let $|T|$ be the number of titles requested by the users. The number of repetitions is $|\mathcal{M}| - |T|$. In Section 5.2, we considered the number of repetitions as a measure of fitness. Therefore, three major cases for the size of the neighborhood can be distinguished: Either all repetitions arise from different titles, only one title is broadcasted multiple times, or anything in between. The former gives an upper bound for the size, namely $\mathcal{O}(|\mathcal{M}| - |T|)$ neighbors.

Unfortunately, merges of large trees are likely to produce further capacity exceeds which will be taken care of by neighborhood structures we will introduce in the next sections.

5.3.2 Min-conflict neighborhood

In order to address a very contributive part of the evaluation function, namely the number of exceeding edges and the accumulated overflow, a new neighborhood structure is introduced. We call this the *min-conflict neighborhood* as it tries to repair non-feasible assignments [18].

The neighborhood yields candidates by first identifying conflicting edges and in further consequence splitting trees which participate to this conflict. First, we define $F_s \subseteq E$ as the set of edges so that for every edge in F_s,

the sum of demands of multicast trees stressing that edge in period $s \in S$, exceeds the edge's capacity. By that definition, the capacity constraint (3) is violated in period s and F_s can be defined as:

$$\forall e \in F_s : \sum_{\mathcal{T} \in \mathcal{M}} r_{\mathcal{T}}^{e} z_{\mathcal{T}}^{s} d_{t(\mathcal{T})} > c_e. \tag{14}$$

Subsequently, from all multicast trees assigned to this period, neighbors were generated by subtracting vertices which have a path containing an exceeding edge. We refer to this operation as *subset move*, because it splits a tree into two disjoint trees in such way that the new tree is subject to reassignment to another period. However, let us define a subtree $\mathcal{U}$ of a multicast tree $\mathcal{T}$ as

$$\mathcal{U} \subset \mathcal{T} \Leftrightarrow \forall \mathcal{P} \in \mathcal{U} \Rightarrow \mathcal{P} \in \mathcal{T}$$

such that both $\mathcal{T}$ and $\mathcal{U}$ are broadcasting the same content title. A more descriptive way to think of subtrees would be graph connectivity. Assuming that we remove a conflicting edge from a tree scheduled in the respective period, a new component would be created. The resulting component corresponds to the (maximum) subtree, which again has to be connected to the source to yield valid paths to the leafs. The subtree $\mathcal{U}$ is now scheduled for another period s.

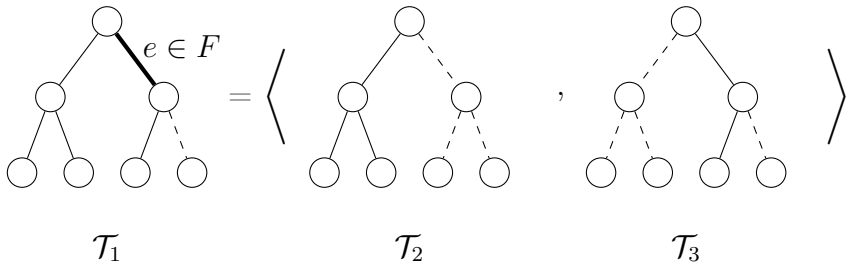

Figure 4: A possible move in the min-conflict neighborhood.

Figure 4 illustrates a possible outcome of the min-conflict neighborhood structure. Edge e is indicted as conflicting and therefore $\mathcal{T}_1$ will be split into two disjoint trees $\mathcal{T}_2$ and $\mathcal{T}_3$. From the example shown, we can conclude that there exists at least one other tree issuing demand on edge e, which is also considered as a possible candidate for the subset operation. In case there was no other tree, this instance would never yield an admissible solution, as the

capacity constraint (3) could never be satisfied, no matter what operation is applied or what period $\mathcal{T}_1$ and its subtrees are assigned to.

Furthermore, this neighborhood structure yields numerous candidates including shifts and swaps of conflicting trees to other periods, as further described in Section 5.3.3.

As the exact estimation of the size of the neighborhood is non-trivial and depending on the size of $|F_s|$, which raises very fast with an increasing number of $|Q|$, our implementation selects a $f \in F_s$ randomly and generates only a few neighbors, i.e. subtrees, for this conflict.

5.3.3 Shift neighborhood

The two previously described neighborhoods aim at constructing larger trees and at reducing the number of exceeds. No neighborhood has yet been defined to allow the reassignment of a complete tree $\mathcal{T}$ from period s_i to another period s_j. Such shift operations lead to re-ordering of tree assignments and could be crucial to escape from local optima. The *shift neighborhood* $\mathcal{N}_{shift}(x)$ yields neighbors of a candidate $x \in X$ such that a tree $\mathcal{T}_i \in x$ being assigned to period s_i in x is scheduled for period s_j in $y \in \mathcal{N}_{shift}(x)$ and $s_i \neq s_j$. Let us denote by k the number of randomly selected and reassigned trees, then a k-shift neighbor $y \in \mathcal{N}_{shift}$ results by the subsequent application of k shifts. In further consequence, we extend the shift neighborhood to yield k-swap neighbors. Figure 5 and Figure 6 respectively show a 1-shift and a 1-swap move.

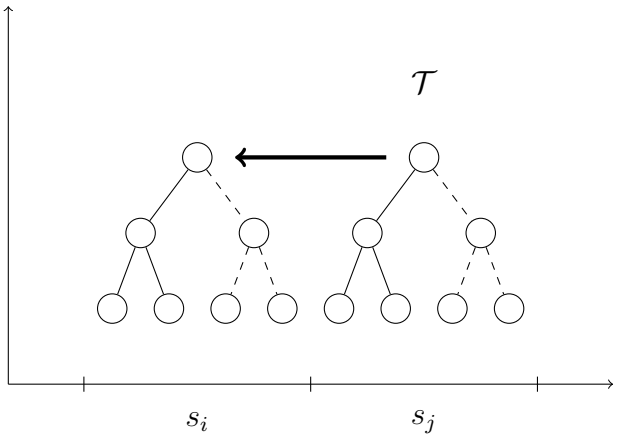

Figure 5: A possible move in the shift neighborhood.

Let m be the number of trees and s be the number of periods in an assignment x. Each tree $\mathcal{T} \in \mathcal{M}$ can now be scheduled for $s-1$ periods, as the neighborhood relation should be irreflexive, i.e. $x \notin \mathcal{N}(x)$. Assuming that a tree can be selected only once for a shift within a neighbor of x, the size of the k-shift neighborhood is $m(s-1)(m-1)(s-1)(m-2)(s-1)\dots(m-k+1)(s-1)$. Therefore the size of a k-shift neighborhood is $\mathcal{O}(m^k s^k)$.

Moreover, we want to determine an upper bound for the size of the k-swap neighborhood. Let us assume that for each $i = \{1, \dots, k\}$ two trees were selected. Again, no tree must be selected more than once and can not be swapped with itself. For the sake of simplicity of our estimation of the neighborhood size, we also allow swaps to take place within the same period (since we don't know the exact number of trees being scheduled for the same period).

Theorem 1. The number of possible swaps in a k-swap neighborhood is $|\mathcal{N}_{swap}| = \frac{m!}{2^k(m-2k)!}$ (see Appendix for a proof).

Furthermore, we know that in a m-tree assignment, k is bound by $1 \leq k \leq \lfloor \frac{m}{2} \rfloor$. Consequentially the size of the k-swap neighborhood is $\mathcal{O}(m^{2k})$. Therefore, we limit k by 1. Another reason for the limitation is that the same result could be achieved by applying multiple subsequent swap and shift moves.

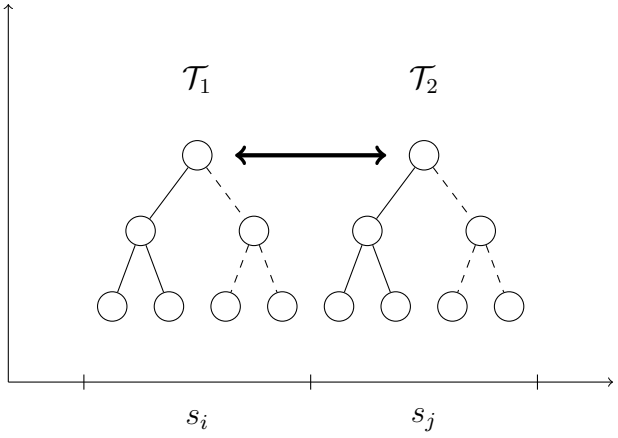

Figure 6: A swap move.

5.3.4 Compress neighborhood

None of the neighborhood structures described above has the strong ability to reduce the number of periods used in an assignment effectively because all of them only touch a small number of trees in each operation and furthermore, the application of several subsequent moves which could lead to a an assignment having one period less is very unlikely. A possibility to reduce the number of periods used $|S|$ is to reschedule all trees of a certain period $s \in S$ to the remaining periods $S \setminus \{s\}$. Therefore, let us introduce the *compress neighborhood*, which is a relation $\mathcal{N}_{compress} : X \to \mathcal{P}(X)$ with $x \in X, y \in \mathcal{N}_{compress}(x)$ such that $|S_y| = |S_x| - 1$. In other words, the number of periods used has decreased by one. In an assignment $y \in \mathcal{N}_{compress}(x)$, all trees which were scheduled for the last period s_n in x are now randomly assigned to some period $s_1, s_2, \ldots s_{n-1}$ in y with the same probability. The size of $\mathcal{N}_{compress}$ is $\mathcal{O}(1)$. Nevertheless, a move in this neighborhood is likely to involve a great number of trees and to result in a large number of computations.

The effectiveness of the neighborhood clearly depends on the coefficient β of the evaluation function (see Section 5.2). In case a large β was chosen, the metaheuristic tends to select a neighbor from this neighborhood in favor of decreasing $|S|$ and in return possibly accepts a high number of exceeding links in the remaining periods $s_1, s_2, \ldots, s_{n-1}$. Furthermore, the compress neighborhood is only used for instances without a fixed number of periods.

6 Implementation

In the course of this work, implementations of the metaheuristics described in Sections 4 and 5, have been established using the Java[2] programming language. We have chosen Java because of our prior knowledge and the great tool support.

In order to support comparability to other parties, we have implemented a *SolutionChecker*, which takes a solution and calculates the evaluation function according to Equation (13). As an input, the checker takes three simple CSV (comma separated values) files:

- An instance of the RCDP containing $|Q|$ tuples of format (t, u) denoting that user $u \in U$ has requested title $t \in T$.

[2]The Java Language Specifiction: http://java.sun.com/docs/books/jls/

- A file containing tuples of format (t, d_t), with $t \in T$ and $d_t \in \mathbb{R}, d_t > 0$, determining the demand of a title.
- A solution file containing tuples of format $(t, s, \{u_1, \ldots, u_n\})$, where $t \in T$, $s \in S$ and $u_1, \ldots, u_n \in U$, indicating that a title is broadcasted to the specified users in given period.

All algorithms were implemented as depicted in Section 4. Unfortunately, the overall calculation of f is computational intensive, therefore we added a so called Δ-function, which is a function $\Delta : X \times X \to \mathbb{R}$ giving the improvement or degradation of f after applying a move to $x \in X$. At the expense of a higher memory demand, the application of all moves can be calculated efficiently in constant time. However, the most expensive part is still the neighborhood generation as it involves a lot of graph operations such as subgraph calculation, reachability and nearest common ancestor determination.

7 Experimental analysis

In the last sections, we discussed a metaheuristic approach to the RCDP and presented details on our implementation. In this section, we will analyze our methods and important parameters w.r.t. a number of questions, mostly related to solution quality and runtimes. Due to the lack of real data from a large VoD operator, as used in [6], our analysis limits to a number of 5 randomly generated instances which we consider as characteristic for the problem. However, we will study the following questions:

A How do our metaheuristics perform in general?

B Are there any differences in performance w.r.t. solution quality/runtime?

C What is the largest (feasible) instance any of the heuristics is able to solve?

D How many neighbors does each of the heuristics compare/accept?

E Do the heuristics address different parts of the evaluation function?

F How does the utilization of a CDN increase with an increasing number of requests?

To answer these questions, let us first identify the parameters needed to continue. For general comparison, the objective f and the average runtime t were measured. Moreover, the number of neighbors compared (i.e. generated) n_c and the number of accepted neighbors n_a were recorded. Additionally, we traced certain parts of the evaluation function namely the number of trees $|\mathcal{M}|$ and the overall exceeding $e(x)$ of the best solution $x \in X$ found (see Equation (12)). In order to answer the last question, the top-level utilization u, which is the average network utilization on upper edges, is recorded. Table 2 shows a summary of all measured variables. All experiments were computed using an Apple MacBook Pro, 2,66GHz Intel Core 2 Duo, 2x2GB 1066MHz DDR3 SDRAM.

Symbol	Description
e	Exceeding in best solution.
f	Value of the evaluation function of the best solution found.
$\mathcal{M}$	Number of trees in best solution.
n_a	Number of accepted neighbors.
n_g	Number of neighbors generated.
r	Number of title repetitions in the best solution.
t	Run time until best solution was found.
u	Top-level utilization [%] in best solution.

Table 2: Summary of variables measured.

The remainder of this section is structured as follows: First, we will define the statistical methods used for comparison. In Section 7.2, the problem instances and their generation will be discussed. In Section 7.3, we will describe the parameters used for the experiments. Finally, we will discuss the obtained results.

7.1 Statistical methods

In order to learn about the performance of our implementations, we need methods, which allow us on the one hand to decide which parameters to use with our algorithms and on the other hand, to draw significant conclusions from the obtained results (such as: Heuristic A performs better than heuristic B for instances larger than x.). Furthermore, we want to guarantee results which are representative for our metaheuristics and reproducible for others.

Therefore, we consider a run of an algorithm on an instance as input as the multi-dimensional random variable from a probability space $(\Omega, \mathcal{F}, P)$ [23] as the vector

$$X = (X_1, \ldots, X_n)^\intercal : \Omega \to \mathbb{R}^n,$$

where X_1 to X_n denote the single random variables measured such as runtime (t), solution quality (f) or neighbors compared (n_c). A number of 30 runs for each heuristic and instance were performed to gain a minimum of information about the distribution of X. Furthermore, the runtime and the objective were obtained from the best solution found. A run was stopped if there was no improvement for a number of 30 iterations[3]. A general time limit was not set. In further consequence, for each dimension of X (see Section 7 and Table 2), we calculated the mean $\overline{x}$, the median $\tilde{x}$ and the standard deviation σ_x for the drawn samples.

Moreover, we use statistical significance tests to support or reject our hypotheses. All statistical operations and tests were performed using the software R[4].

7.2 Instances

As already stated at the beginning of the section, we will use randomly generated instances for our experiments. In [2, 5, 6], they have found that the popularity of titles requested by customers in large VoD and streaming systems follow a Zipf distribution, i.e. the number of downloads correlates to the popularity of a title. Assuming a digital library containing a number of $n = 13000$ downloadable titles $\{t_1, t_2, \ldots, t_n\}$, we determined the index i of

[3]We only count the iterations of the enclosing metaheuristic. Embedded search procedures were not taken into account (as often the number of iterations is fixed and the strength of finding global optima lies in the enclosing metaheuristic).

[4]The R Project for Statistical Computing, http://www.r-project.org/

the requested title t_i for every request by

$$i = \left\lfloor \frac{1}{\mathcal{U}(0{,}001, 0{,}2)^{1{,}372}} \right\rfloor,$$

where $\mathcal{U}$ is the (continuous) uniform distribution. The resulting distribution cuts off very popular objects, as the data simply could be cached by network infrastructure which is near to the users. Furthermore, popular content may be subject to user-centric or data-centric channel-multicast techniques as described in [16]. However, we assigned each request a title and a randomly chosen user with the restriction that no customer requests the same title multiple times. Note that the set of titles T, defined in Section 2.1, contains only the subset of all titles $T \subseteq \{t_1, t_2, \ldots, t_n\}$ such that every $t \in T$ is requested by some user.

In further consequence, we assumed the CDN to be a tree-like, fixed network specified by two vectors, the branching factors B of each tree-level and a vector defining the capacities C assigned to each layer. Additionally, $|B| = |C|$ is required. For example, the vectors $B = \{10, 20, 30\}$ and $C = \{200, 100, 10\}$ denote a CDN with 10 links on the top layer, $10 \cdot 20 = 200$ on the 2nd and $10 \cdot 20 \cdot 30 = 6000$ on the bottom layer. Therefore, the number of users $|U|$ is also 6000. Before we continue, let us give a more formal way how to construct a valid CDN from B and C.

Given the two vectors B and C and a tree root s, a valid CDN G of oder $o := |B|$, i.e. the depth of the tree, can be generated. Let $V = \{1, \ldots, n\} \cup \{s\}$ be the set of nodes of the tree and let $E = \{1, \ldots, n\}$ be the set of edges, where $n = \sum_{l=1}^{|B|} \prod_{i=1}^{l} b_i$. The set of users $U \subseteq V$ is the set of nodes located at the bottom, i.e. $U = \{1, \ldots, \prod_{b \in B} b\}$. The capacity c_e of an edge $e \in E$ can be calculated as follows: Let $n(k) = \prod_{i=1}^{k} b_i$ with $1 \leq k \leq |B|$ be the number of edges on layer k then the maximum $l \leq |B|$ gives us the level of e if and only if, $e \leq \sum_{l=1}^{|B|} n(|B| - l - 1)$. Hence, the capacity of e is $c_e := C(l)$, where $C(l)$ is the i-th entry of C.

Additionally, we randomly generated the demand d_t of all titles $t \in T$ using $\mathcal{U}(1.5, 3)$ as probability distribution. The properties of the generated instances can be found in Table 3.

7.3 Parameter determination

The performance of the presented metaheuristics heavily depends on the values of their parameters. In this section, we discuss the most important

♯	$\|Q\|$	$\|U\|$	Branching factors	Capacities
1	2000	1000	$\{10, 10, 10\}$	$\{100, 20, 10\}$
2	4000	2000	$\{10, 10, 20\}$	$\{100, 20, 10\}$
3	8000	4000	$\{20, 20, 10\}$	$\{100, 20, 10\}$
4	16000	8000	$\{10, 20, 40\}$	$\{100, 20, 10\}$
5	32000	16000	$\{20, 20, 40\}$	$\{100, 20, 10\}$

Table 3: Test instances used for the RCDP.

parameters and values used in our experiments. First, we determine the best value for the tabu length used in Tabu Search (TS), then the perturbation size of Iterated Local Search (ILS) will be investigated. A summary of all configurations can be found in Table 4.

7.3.1 Tabu length

In Section 4.2, Tabu Search and its parameters, e.g. the tabu length t, were introduced. The tabu length is the number of iterations a tree should be blocked if it was part of a recently accepted move. To determine the optimal length of the recency based memory, a randomly generated instance of the 8000-4000[5] class was used. The maximum number of periods $|S|$ was set to 8 to obtain a problem accordingly hard. For each tabu length t, a number of 30 runs were performed. As our prior interest is in short running times, a tabu length of $t = 12$ will be used for further experiments. The full results regarding t can be found in the Appendix.

7.3.2 Perturbation size

Since the performance of the Iterated Local Search depends on the size of the perturbations [15], several experiments were performed to determine the optimal value of p. Again, 30 runs on the same random 8000-4000 instance were performed and an optimal value for $p = 2$ obtained. See the Appendix for detailed results.

[5] $B = \{10, 10, 40\}, C = \{100, 20, 10\}$

	Name	Description	Value
TS	α	Aspiration criterion.	$f(x) < f(x_L^*)$
	c	Acceptance criterion.	$f(x) < f(y), y \in \mathcal{N}(x)$
	l	Max. local tour iterations.	30
	t	Tabu memory length.	12
		Select Function	Best
ILS	c	Acceptance criterion.	$f(x) < f(x_L^*)$
	l	Max. local iterations.	10
	p	Perturbation size.	2
VND	c	Acceptance criterion.	$f(x) < f(x_L^*)$
		Max. iter. per neighborhood.	2
	l	Max. local tour iterations.	2
		Max. local iterations.	2
		Neighborhood structures.	$\mathcal{N}_{merge} \subset \mathcal{N}_{min-conflict} \subset \mathcal{N}_{compress} \subset \mathcal{N}_{shift}$
		Select Function	Best
f	α	Weight of capacity exceed.	50
	β	Weight of periods needed.	2
	γ	Weight of content repetitions.	10

Table 4: Summary of metaheuristics, parameters and values.

7.4 Results

We have performed a number of experiments with the configuration described in the last section. In further consequence, we will give interpretations for the obtained data and try to answer the questions raised in Section 7. Before drawing conclusions about the performance of the metaheuristics, we need to analyze the obtained data in a statistical manner.

First, we want to test whether the runtime of each algorithm, which is the average time the best solution was found in a run given an instance of the RCDP, is normally distributed. Therefore, we formulate the following hypothesis:

H_0 : t is normally distributed.

H_1 : t is **not** normally distributed.

Performing a number of *Shapiro-Wilk* tests at a significance level of $p = 0.05$ on the results of the largest instance (i.e. instance 5) gives us the following matrix which tells us whether the samples of the runs of an algorithm are compatible with H_0 or not:

	TS	VND	ILS
f	H_0	H_0	H_1
$\mathcal{M}$	H_0	H_0	H_0
n_a	H_0	H_0	H_0
n_g	H_1	H_1	H_0
t	H_0	H_1	H_0
u	H_0	H_0	H_0

This gives us important information on which statistical tests we can apply and which not.

Let us first start by analyzing the runtime t of the three algorithms. A *Welch* test ($p = 0.05$) for the null hypothesis $H_0 : \mu_{t_{TS}} = \mu_{t_{ILS}}$ and $H_1 : \mu_{t_{TS}} \neq \mu_{t_{ILS}}$ shows no significance, hence we can conclude that the samples obtained from TS and ILS arise from the same population with a probability of 0.95. Since we know that t_{VND} is not normally distributed, we perform a *Wilcoxon-Mann-Whitney* test ($p = 0.05$) with $H_0 : \mu_{t_{VND}} = \mu_{t_{TS}}$ and $H_1 : \mu_{t_{VND}} \neq \mu_{t_{TS}}$. As the test is significant, we must reject H_0 and assume H_1. From that we conclude that VND in general converges faster against the best solution than TS and ILS do (see Appendix, Table 8). Note that no assumption about the quality of the solution can be made yet.

In a second step, we want to learn about the quality of the solutions generated by the algorithms. Therefore, we perform a t-test with $H_0 : \mu_{f_{TS}} = \mu_{f_{VND}}$ and $H_1 : \mu_{f_{TS}} < \mu_{f_{VND}}$. As the test is significant, we are safe to state that Tabu Search on average delivers better solutions than Variable Neighborhood Descent. Additionally, we perform another test from which we conclude that VND performs better than ILS. At this point, question B can be answered: Regarding f, $\mu_{f_{TS}} < \mu_{f_{VND}} < \mu_{f_{ILS}}$ holds and regarding t, $\mu_{t_{VND}} < \mu_{t_{TS}} = \mu_{t_{ILS}}$ holds.

Concerning question C, which is the largest instance our algorithms are capable to solve, we can report that we tried to scale the CDN and the number of requests to a maximum. The maximum heap size memory was set to one gigabyte. At a number of $|Q| = 256000$ and $|U| = 128000$ we stopped

since we did not hit the memory limit and the metaheuristics were still able to solve the given instance[6] in $t_{TS} \approx 360s, t_{VND} \approx 135s, t_{ILS} \approx 223s$.

The next question to be answered is to determine the characteristics of the neighborhood exploration (D). Therefore, we analyze two measures: The number of generated/compared neighbors n_g and the number of accepted neighbors n_a[7]. Table 10 indicates that TS generates a large number of neighbors compared to the other heuristics but accepts only a few (good) neighbors. We assume that the combination of the chosen tabu length and a high number of local tour iterations force TS to explore a great number of neighbors leading to a better solution quality. Furthermore, we can see that ILS accepts a surprisingly great number of neighbors even though the solution quality of ILS is moderate. We conclude that ILS leads to a lot of small, but local improvements since the running time is worse compared to TS and VLS. About the characteristics of VND we can state that the iterative exploration of neighborhood structures results in a reasonable number of generated neighbors in favor of fast convergence even though in average TS surpasses VND in terms of solution quality. Moreover, there is a small anomaly concerning the explored neighborhood of instance 2 which is due to the varied capacities and branching factors of the instance (see Table 3).

Let us now focus on the last questions (E and F). As in our experiments, the number of periods $|S|$ was fixed and we only considered feasible solutions, i.e. solutions having $e(x) = 0$, we look at the number of repetitions $R = |\mathcal{M}| - |T|$ of titles (see Table 9). As expected, Tabu Search again is able to surpass the other metaheuristics. Surprisingly, the average top level utilization TLU is not substantially higher with VND and ILS even though one would think that there exists a correlation between TLU and R (As a repetition forces the complete content to be broadcasted over the network again.).

8 Conclusion and future work

In the course of this thesis, we discussed the Repeated Contend Download Problem (RCDP), which is a problem occurring in the context of content distribution and multicast tree scheduling. We gave a formal definition of the problem similar to Bessler's formulation in [3] but targeted at the application

[6] $C = \{300, 150, 10\}, B = \{20, 80, 80\}, |S| = 8$

[7] Note that neither n_g nor n_a count unique neighbors.

of metaheuristics. Moreover, we showed that the RCDP is $\mathcal{NP}$-complete and no polynomial-time algorithm yielding an optimal solution can exist.

A major part of our work deals with the local search procedures, such as Tabu Search, Variable Neighborhood Descent and Iterated Local Search, which we all applied to the problem. However, we gave a metaheuristic formulation including an evaluation function and various neighborhood structures.

The last part of our work contained an experimental analysis of the three applied metaheuristics. First, we formulated our experiments as questions, secondly, the procedure and the estimation of the parameters was explained. Then the measured variables were defined to answer the questions. Finally, we performed several statistical tests on the obtained data and gave interpretations for the results.

We found that our implementation of Tabu Search, Variable Neighborhood Descent and Iterated Local Search are capable of solving even large instances of the RCDP showing different characteristics. For instance, Tabu Search yields the best solutions at the expense of runtime. Variable Neighborhood Descent outperforms Tabu Search and Iterated Local Search in terms of runtime, but performs worse that Tabu Search w.r.t. solution quality.

In the future, experiments using data from large VoD systems should be used to study the detailed characteristics of the presented algorithms and to adjust the algorithm's parameters. Furthermore, a *rolling schedule*, which is the iterative re-scheduling after the last period was broadcasted and new requests were added, could be implemented. However, we think that there is still potential for all of the applied heuristics.

References

[1] Vaneet Aggarwal, Robert Caldebank, Vijay Gopalakrishnan, Rittwik Jana, K. K. Ramakrishnan, and Fang Yu. The effectiveness of intelligent scheduling for multicast video-on-demand. In *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*, pages 421–430, New York, NY, USA, 2009. ACM.

[2] Jussara M. Almeida, Jeffrey Krueger, Derek L. Eager, and Mary K. Vernon. Analysis of educational media server workloads. In *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 21–30, New York, NY, USA, 2001. ACM.

[3] S. Bessler. Optimized content distribution in a push-vod scenario. pages 161 –166, april 2008.

[4] Shiwen Chen, Oktay Günlük, and Bülent Yener. The multicast packing problem. *IEEE/ACM Trans. Netw.*, 8(3):311–318, 2000.

[5] Maureen Chesire, Alec Wolman, Geoffrey M. Voelker, and Henry M. Levy. Measurement and analysis of a streaming-media workload. In *USITS'01: Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems*, pages 1–1, Berkeley, CA, USA, 2001. USENIX Association.

[6] Roberto García, Xabiel G. Paneda, Victor Garcia, David Melendi, and Manuel Vilas. Statistical characterization of a real video on demand service: User behaviour and streaming-media workload analysis. *Simulation Modelling Practice and Theory*, 15(6):672 – 689, 2007.

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman & Co Ltd, January 1979.

[8] Michel Gendreau and Jean-Yves Potvin. Metaheuristics in combinatorial optimization. *Annals OR*, 140(1):189–213, 2005.

[9] Fred Glover. Tabu search - part i. *INFORMS Journal on Computing*, 1(3):190–206, 1989.

[10] Fred Glover and Fred Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[11] Pierre Hansen and Nenad Mladenovic. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449 – 467, 2001.

[12] Pierre Hansen and Nenad Mladenović. Variable neighborhood search. In *Handbook of Metaheuristics*, pages 145–184. Springer, 2003.

[13] Kamal Jain, Mohammad Mahdian, and Mohammad R. Salavatipour. Packing steiner trees. In *in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA*, pages 266–274, 2003.

[14] Ming-Yang Kao, editor. *Encyclopedia of Algorithms*. Springer, 2008.

[15] H.R. Lourenço, O. Martin, and T. Stützle. A beginner's introduction to iterated local search. In *Proceedings of the Fourth Metaheuristics International Conference*, volume 1, pages 1–6, 2001.

[16] Huadong Ma and Kang G. Shin. Multicast video-on-demand services. *ACM Computer Communication Review*, 32:2002, 2002.

[17] Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, December 2004.

[18] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: A heuristic repair method for constraint-satisfaction and scheduling problems. *ARTIFICIAL INTELLIGENCE*, 58(1):161–205, 1992.

[19] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097 – 1100, 1997.

[20] S. J. Russell and Norvig. *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall, 2003.

[21] Sandvine. 2009 global broadband phenomena. Technical report, Sandvine Inc., 2009.

[22] L. Sanna Randaccio and L. Atzori. Group multicast routing problem: A genetic algorithms based approach. *Comput. Netw.*, 51(14):3989–4004, 2007.

[23] Reinhard Viertl. *Einführung in die Stochastik (Third Edition)*. Springer, 2003.

[24] Chu-Fu Wang, Bo-Rong Lai, and Rong-Hong Jan. Optimum multicast of multimedia streams. *Comput. Oper. Res.*, 26(5):461–480, 1999.

[25] Yunnan Wu, Philip A. Chou, and Kamal Jain. A comparison of network coding and tree packing. In *IN PROC. 2004 IEEE INTERNATIONAL SYMPOSIUM ON INFORMATION THEORY (ISIT 2004*, page 143, 2004.

Appendix

Proof of Theorem 1

Claim 4. We want to show by induction that

$$\frac{m(m-1)}{2}\frac{(m-2)(m-3)}{2}\cdots\frac{(m-2(k-1))(m-2(k-1)-1)}{2}=\frac{m!}{2^k(m-2k)!}.$$

holds for every $1 \leq k \leq \lfloor \frac{m}{2} \rfloor$.

Proof. **Induction basis:** $k = 1$

$$\begin{aligned}
\frac{m(m-1)}{2} &= \frac{m!}{2^1(m-2)!} \\
m(m-1) &= \frac{m!}{(m-2)!} \\
m(m-1) &= \frac{m(m-1)(m-2)\dots(m-(m-1))}{(m-2)!} \\
m(m-1) &= m(m-1)
\end{aligned}$$

Induction step: $k+1$

$$\begin{aligned}
\frac{m(m-1)}{2}\cdots\frac{(m-2(k+1-1))(m-2(k+1-1)-1)}{2} &= \frac{m!}{2^{k+1}(m-2(k+1))!} \\
\frac{m(m-1)}{2}\cdots\frac{(m-2k)(m-2k-1)}{2} &= \frac{m!}{2^{k+1}(m-2(k+1))!} \\
m(m-1)\dots(m-2k-1) &= \frac{m!}{(m-2k-2)!} \\
m(m-1)\dots(m-2k-1) &= \frac{m(m-1)\dots(m-2k-1)(m-2k-2)!}{(m-2k-2)!} \\
m(m-1)\dots(m-2k-1) &= m(m-1)\dots(m-2k-1)
\end{aligned}$$

□

Tabu length

t	$\bar{Z}$	$\tilde{Z}$	σ_Z	$\bar{t}$ [ms]	$\tilde{t}$ [ms]	σ_t [ms]
0	7858.33	7861	18.88	6431.63	4418	3841
2	7856.67	**7851**	25.45	8084.07	5111	5745.12
4	7863	7866	20.37	7875.13	6564	4735.63
6	7866.33	7866	**16.5**	6913.43	4952	5631.14
8	7867.33	7866	22.55	7637.43	5533	4643.46
10	7864.67	7866	22.24	6662.8	5066	**3462.56**
12	7867.67	7871	22.14	**6207**	**4340**	4178.01
14	**7854.67**	7856	19.43	9369.03	8394	5854.97
16	7865.67	7861	21.73	8228.27	7827	3918.11

Table 5: Comparison of various tabu lengths used for Tabu Search.

Perturbation size

p	$\bar{Z}$	$\tilde{Z}$	σ_Z	$\bar{t}$ [ms]	$\tilde{t}$ [ms]	σ_t [ms]
1	8183	8171	63.81	**1277**	0	**1816.01**
2	**8164**	**8166**	59.69	1555	0	2581.63
3	8167	**8166**	**50.89**	1742	0	2049.42
4	8201	8181	88.35	1884	0	3204.23

Table 6: Perturbation size of ILS using Hill Climbing as embedded local search.

Inst.	$\|S\|$	TS			VND			ILS		
		$\bar{Z}$	$\tilde{Z}$	σ_Z	$\bar{Z}$	$\tilde{Z}$	σ_Z	$\bar{Z}$	$\tilde{Z}$	σ_Z
1	8	3336	3336	0	3336	3336	0	3336	3336	0
2	8	5157	5156	3.05	5159	5156	5.96	5158.33	5156	5.04
3	8	7816	7816	0	7816.33	7816	1.83	7816	7816	0
4	8	11761.33	11761	54.06	11955.33	11951	73.53	12095.67	12096	109.59
5	8	18006.73	18016	134.4	18460.33	18456	98.63	18737.43	18711	296. 93

Table 7: Evaluation function.

Inst.	$\|S\|$	TS			VND			ILS		
		$\bar{t}$ [ms]	$\tilde{t}$ [ms]	σ_t [ms]	$\bar{t}$ [ms]	$\tilde{t}$ [ms]	σ_t [ms]	$\bar{t}$ [ms]	$\tilde{t}$ [ms]	σ_t [ms]
1	8	1.97	0	7.49	3.83	0	11.77	1.7	0	4.76
2	8	200.77	156	191.93	106.57	98	45	74.1	72	33.23
3	8	528.7	397	655.91	114.8	140	60.15	34.33	32	27.89
4	8	8409.27	7309	3716.07	5448	4463	2417.78	9639.43	9476.5	2063.79
5	8	35411.03	34549.5	9427.56	21609.47	20263	5294.83	64303.07	65503.5	9556.63

Table 8: Running times.

Inst.	$\|S\|$	$\|T\|$	TS			VND			ILS		
			$\overline{\|\mathcal{M}\|}$	$\overline{R}$	$\overline{TLU}$ [%]	$\overline{\|\mathcal{M}\|}$	$\overline{R}$	$\overline{TLU}$ [%]	$\overline{\|\mathcal{M}\|}$	$\overline{R}$	$\overline{TLU}$ [%]
1	8	332	332	0	23.01	332	0	23.01	332	0	23.01
2	8	514	514.1	0.1	38.56	514.3	0.3	38.57	514.23	0.23	38.57
3	8	780	780	0	37.42	780.03	0.03	37.42	780	0	37.42
4	8	1141	1174.53	33.53	85.6	1193.93	52.93	86.05	1207.97	66.97	86.32
5	8	1690	1789.87	99.87	87.1	1844.43	154.43	87.66	1866.63	176.63	87.91

Table 9: Splitting rate and top level utilization.

Inst.	$\|S\|$	TS		VND		ILS	
		$\overline{n_g}$	$\overline{n_a}$	$\overline{n_g}$	$\overline{n_a}$	$\overline{n_g}$	$\overline{n_a}$
1	8	302.07	0.07	32.2	0.03	61.4	0.2
2	8	521.87	5.23	77.07	5.5	106.33	6.77
3	8	1297.8	1.23	105.53	0.73	172.4	1.77
4	8	2432.33	134.53	1296.8	153.07	1629.7	208.2
5	8	6366.77	309.37	3121.9	348.27	5013.93	626.97

Table 10: Neighborhood