Towards a Theory of Collaborative Multimedia *

Kasım S. Candan, V.S. Subrahmanian
Department of Computer Science
University of Maryland
College Park, Maryland 20742.
{candan, vs}@cs.umd.edu

P. Venkat Rangan
Department of Computer Science
University of California at San Diego
La Jolla, CA 92093.
venkat@cs.ucsd.edu.

Abstract

We develop a theory of media objects, and present optimal algorithms for collaborative object synthesis for constructing multimedia documents by composing together a given set of media objects. We then extend the algorithms to incorporate quality constraints (such as image size) as well as distribution across multiple nodes. The theoretical model is validated by an experimental implementation that supports the theoretical results.

1 Introduction

Collaborative multimedia systems consist of collaborators constructing and manipulating various kinds of media objects, such as video-clips, pictures, text files, or perhaps some complex entity constructed out of these simpler entities. In this paper, we develop a theory of media objects, and present optimal algorithms for collaborative object synthesis.

2 Media Objects

We classify all media-objects into three types:

1. **Static:** Intuitively, a static media object is an object that does not change when it is presented. Examples of static media-objects include `.gif` files and ordinary text files that do not change when presented to the user (though of course they may change as a result of editing by a human).

2. **Quasi-Static:** A quasi-static media-object is one which may be broken up into a contiguous sequence of chunks of information, each of which is presented to the user sequentially, one after another. However, it is upto the individual viewing the quasi-static object to determine how long to spend in browsing one such chunk of information. A good example of a quasi-static media-object is a postscript file. A user browsing a postscript document through a postscript viewer such as `ghostview` may move from one page of the postscript file to another at his discretion/leisure.

3. **Temporal:** A temporal media-object is one which may be broken up into a contiguous sequence of chunks of information, each of which is presented to the user sequentially, one after another. In temporal objects,

*This research was supported by the Army Research Office under grant DAAL-03-92-G-0225, by the Air Force Office of Scientific Research under grant F49620-93-1-0065, by ARPA/Rome Labs contract Nr. F30602-93-C-0241 (Order Nr. A716), and by an NSF Young Investigator award IRI-93-57756.

we asssume that the display time of each "chunk" in the afore-mentioned sequence of contiguous chunks is the same. An example of a temporal media-object is audio, where an analog audio stream may be broken up into 5 millisecond frames for sampling/analysis purposes.

4. **Quasi-Temporal:** A quasi-temporal media object is similar to a temporal media-object; the only difference is that the display time of different chunks in the afore-mentioned sequence of contiguous chunks may vary. An example of quasi-temporal media-objects could be video where different frames may be of different lengths – this is particularly useful in annotating the video (by hand or otherwise).

Definition 2.1 A *media-object* o is a 5-tuple consisting of:

1. A *data type* $\texttt{ds}(o)$ – e.g. the data structure specification may be a *.ps* file or a *.gif* file or something else that is completely application-specific.
2. A *name*, $\texttt{name}(o)$, of the form `<string>.<type>` where `<type>` is the data type in the preceding item. For example, if the data type is `ps`, then the name of such an object may be `zzz.ps`.
3. An *object-type*, $\texttt{ot}(o)$, which is either *temporal, quasi-temporal* , *static* or *quasi-static.*
4. An *object-characteristic*, $\texttt{oc}(o)$, that has the following form:
   (a) If a given media-object is of type *temporal*, then the object characteristic is a pair $(n, \Delta t)$ specifying that the media-object consists of n "chunks" of data, each having duration Δt.
   (b) If the given media-object is of type *quasi-temporal*, then the object characteristic is a pair (n, ψ_o) where ψ_o is a function from $\{1, \ldots, n\} \rightarrow \mathcal{N}$. Intuitively, this means that the media-object o has n "chunks" of data, and that chunk i lasts for time $\psi_o(i)$ time units.
   (c) If the given media-object is of type *quasi-static*, then the object characteristic is a pair $(n, \perp)$ denoting that the object has n "chunks" of information where the time taken by each chunk is unpredictable and is determined by the user.
   (d) If the given media-object is of type *static*, then the object characteristic is of type $(\perp, \perp)$ specifying that the object characteristic is not predictable – neither the number of chunks, nor the time taken in viewing the chunks is predictable and are user-dependent.
5. A component $\textsf{size}(o)$ specifying the size requirements of o.

The reader will observe that the above definition is very robust. For example, we may have two different objects

$$\begin{aligned} o_1 &= (\texttt{ds}(o_1), \texttt{name}(\texttt{o}_1), \texttt{ot}(o_1), \texttt{oc}(o_1), \textsf{size}(o_1)) \\ o_2 &= (\texttt{ds}(o_1), \texttt{name}(\texttt{o}_1), \texttt{ot}(o_1), \texttt{oc}(o_1), \textsf{size}(o_2)) \end{aligned}$$

In this example, objects o_1 and o_2 are *identical* except for their size attribute. Such an example may occur, for instance, if a utility such as `xv` is used to re-size an image.

Example 2.1 (Text Object) One of the simplest types of media-objects is a textual object. For example, a file of the form `a.txt` is a textual object having the following properties:

1. Name: `a.txt`
2. Data Structure Specification: `.txt`

3. Object Type: static

4. Object-Characteristic: $(\perp, \perp)$,

5. Size: 500 (Kbytes) $\square$

Example 2.2 (GIF Object) A slightly more complex media-object is a pictorial object. For example, a file of the form `p.gif` is a pictorial media-object having the following properties:

1. Name: `p.gif`

2. Data Structure Specification: `.gif`

3. Object Type: static

4. Object-Characteristic: $(\perp, \perp)$

5. Size: 580 (Kbytes) $\square$

Note that a picture file using a different format (e.g. .tiff) would be defined in a way similar to the above format.

Example 2.3 (Audio Object) Consider, on the other hand, an audio media-object `b.avi` containing 5000 frames, each of length 2 milliseconds. This media-object is characterized by the following properties:

1. Name: `b.avi`

2. Data Structure Specification: `.avi`

3. Object Type: temporal

4. Object-Characteristic: $(5000, 2)$

5. Size: 1 Mbyte $\square$

Example 2.4 (Video Object) Consider a slightly more complex situation where we have a video media-object containing 10000 frames, the first 5000 of which are of 2 millisecond duration, the next 2000 of which are of 1 millisecond duration, and the last 3000 of which are of 3 millisecond duration. This media-object is characterized by the following properties:

1. Name: `c.avi`

2. Data Structure Specification: `.avi`

3. Object Type: quasi-temporal

4. Object-Characteristic: $(5000, \psi)$ where ψ is the function: $\psi(n) = 2$ if $1 \leq n \leq 5000$; 1 if $5001 \leq n \leq 7000$ and 3 if $7000 \leq n \leq 10000$.

5. Size: 3 MBytes. $\square$

Certain kinds of objects could be declared in many ways depending upon their intended use. For example, consider a 25 page postscript document (the same comments apply to many other types of documents). This could be declared as a static object (which indicates that the collaborative multimedia system we define will not attempt to automatically have its pages scroll through) or it could be viewed as a temporal object (where each page is displayed for Δt time units), or it could be viewed as a quasi-static object where the user scrolls through it at his/her leisure. It is entirely possible that some postscript documents in a collaborative environment are described as temporal objects, while others are defined to be of temporal or quasi-temporal types.

3 Collaborative Multimedia Systems

Having defined the concept of a media-object in Section 2, we are now in a position to start work on defining a collaborative multimedia system. Intuitively, such a formal definition should take into account, the following aspects of any collaborative endeavor:

1. **Collaborators:** First and foremost, we consider a single collaborative effort where there are k collaborators. Each of these collaborators may be located at different locations on the network.
2. **Host Capabilities:** The site/machine hosting a given collaborator may have a set of capabilities. Such capabilities correspond to the system functionalities available at that host node.
3. **Distributed Media Objects:** We assume that the purpose of the collaboration is to develop a multimedia-document (a concept to be defined below) that composes together a given set of media-objects. For example, a multimedia-document may be composed of a sequence of video clip v_1 followed by video clip v_2 followed by a presentation slide (e.g. `.dvi` file) followed by an audio file. At any given point in time, a multimedia document may consist of various media-objects, located at different sites on the network. These different media-objects may be linked together by various constraints expressing spatial/temporal layout constraints.

In this section, we will provide a formal definition of collaborative multimedia systems.

3.1 Simple Collaborative Multimedia Systems

This section presents the "basic" notion of a collaborative multimedia system. When studying collaboration systems, it is important for the members of the collaboration to be aware of each others capabilities. For the purposes of multimedia collaborations, we will study three types of capabilities:

Definition 3.1 A *display capability* is a function that maps media-objects to $\{\texttt{true}, \texttt{false}\}$.

For example, the Unix utility `ghostview` may be thought of as a display capability that maps all objects of the form `X.ps` to `true` (indicating that it can display them) and all other objects to `false` indicating that it cannot display them.

Definition 3.2 An *edit capability* is a function that maps media-objects to $\{\texttt{true}, \texttt{false}\}$.

For example, if we have a special image editor called `ed_tiff` to edit `.tiff` files, then this is an edit-capability that assigns `true` to all files of the form `X.tiff` and assigns `false` to all other objects.

Definition 3.3 A *conversion capability* is a function that takes as input, a media-object o, and returns as output, a media-object o'.

For example, the standard Unix utility, `dvips` may be viewed as a conversion capability that converts `dvi` files to `postscript` files.

We now define the concept of a *simple* collaborative multimedia system.

Definition 3.4 A *simple collaborative multimedia system* (s-COMS for short) consists of:

- an un-directed, weighted graph $G = (V, E, \wp)$ where V is the set of nodes in the graph (representing sites where a member of the collaboration team is located), E refers to the connectivity of the graph, and $\wp : E \rightarrow \mathbf{R}^+$ specifies the cost of sending a byte of information along an edge in the graph.

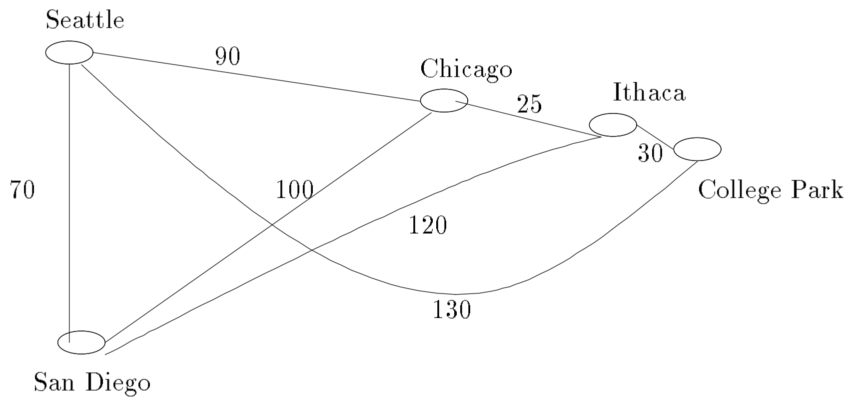

Figure 1: Network for COMS of Motivating Example.

- a set Obj of *media-objects*,
- a function $loc : Obj \rightarrow V$ specifying where the media-objects are located.
- A set $\mathcal{HC}$ specifying a set of host capabilities – these may not be an exhaustive list of capabilities of participating nodes, but just a list of those capabilities that are of interest for the particular application being developed.
- a function $\mathcal{CAP} : V \rightarrow 2^{\mathcal{HC}}$ specifying what capabilities are available at a given node. (As usual, if X is any set, 2^X denotes the power set of X).

Example 3.1 (Motivating Example) Figure 1 shows a diagrammatic representation of a collaborative multimedia system that involves 5 participating entities. The entities are hosts in Seattle, San Diego, Chicago, Ithaca, and College Park. The numbers marked along the edges in the graph shows the weights associated with the edges. A COMS involving the graph of Figure 1 may be described by specifying the values of the different components described in definition 3.4.

1. The undirected weighted graph $G = (V, E, \wp)$ is shown in Figure 1.
2. The set Obj of media-objects involved is shown in the table below:

o	$\mathsf{ds}(o)$	Type	Characteristic	$\mathsf{size}(o)$
f1.gif	.gif	static	$(\perp, \perp)$	250
f2.txt	.txt	static	$(\perp, \perp)$	45
f3.avi	.avi	temporal	$(5000, 10)$	975
f4.avi	.avi	quasi-temporal	$(4000, \psi_{f4})$ where $\psi_{f4}(i) = \begin{cases} 5 & \text{if } 1 \leq i \leq 3000 \\ 8 & \text{if } 3001 \leq i \leq 4000 \end{cases}$	950
f5.tiff	.tiff	static	$(\perp, \perp)$	300
f6.vid	.vid	temporal	$(1000, 4)$	800
f7.vid	.vid	temporal	$(2000, 15)$	1200
f8.tiff	.tiff	static	$(\perp, \perp)$	900
f9.gif	.gif	static	$(\perp, \perp)$	450
f10.ps	.ps	quasi-static	$(25, \perp)$	100
f11.ps	.ps	static	$(\perp, \perp)$	140
f12.dvi	.dvi	static	$(14, \perp)$	100
f13.avi	.avi	quasi-temporal	$(2000, \psi_{f13})$ where $\psi_{f13}(i) = \begin{cases} 2 & \text{if } 1 \leq i \leq 1000 \\ 4 & \text{if } 1001 \leq i \leq 2000 \end{cases}$	1500

3. The function *loc* specifying the locations of objects is shown in the table below:

Object	Location
f1.gif	Seattle
f2.txt	College Park
f3.avi	Ithaca
f4.avi	San Diego
f5.tiff	Chicago
f6.vid	San Diego
f7.vid	San Diego
f8.tiff	Chicago
f9.gif	Seattle
f10.ps	College Park
f11.ps	College Park
f12.dvi	Ithaca
f13.avi	Chicago

4. The set $\mathcal{HC}$ of host capabilities consists of the following:

Capability Name	Input	Output
`tiff2bmp`	File.tiff	File.bmp
`tiff2gif`	File.tiff	File.gif
`gif2tiff`	File.gif	File.tiff
`dvips`	File.dvi	File.ps
`capture_image`	(File.vid,integer)	File.gif
`audio2text`	File.avi	File.txt
`ghostview`	File.ps	display(file.ps)
`xtex`	File.dvi	display(file.dvi)
`xv1`	File.tiff	display(File.tiff)
`xv2`	File.gif	display(File.gif)
`showvid`	File.vid	display(File.vid)
`showavi`	File.avi	display(File.avi)
`ed_tiff`	File.tiff	File.tiff

5. The mapping $\mathcal{CAP}$ is defined as shown below:

V	$\mathcal{CAP}(V)$
Seattle	$\{$`xv1`, `dvips`, `xtex`, `gif2tiff`, `tiff2bmp`$\}$
San Diego	$\{$`capture_image`, `audio2text`, `showavi`$\}$
Chicago	$\{$`capture_image`, `gif2tiff`, `dvips`, `showavi`$\}$
Ithaca	$\{$`xv1`, `xv2`, `gif2tiff`, `tiff2gif`, `ghostview`, `xtex`$\}$
College Park	$\{$`ed_tiff`, `tiff2gif`, `audio2text`, `ghostview`$\}$

□

3.2 Collaboration in an s-COMS: An Informal Description

Collaboration in an s-COMS is accomplished through *read-write* requests made by the different collaborators involved. For example, consider the example s-COMS given in Section 3.1. The individual located in College Park may wish to work on the image file `f1.gif` located in Seattle. For the time being, we assume that all partners participating in an s-COMS have full read/write access to all objects in Obj. What happens when the College Park collaborator wishes to work on file `f1.gif` ?

Checkout: First of all, the College Park collaborator must send a request to a server in Seattle requesting the desired files. The server in Seattle must try to send the documents to the College Park collaborator, *keeping in mind the fact that the person in College Park must be able to edit the* `.gif` *file.* However, we know that the College Park collaborator cannot edit `.gif` files as he has no `.gif` editor. Therefore either the server in Seattle or the client in College Park must find a "loop" in the graph/network such that:

- The loop is of the form $\ell_1, \ldots, \ell_i, \ldots, \ell_n$ where $\ell_1 = \texttt{Seattle} = \ell_n$ and $\ell_i = \texttt{College Park}$.
- There is some j, $1 \leq j \leq i$ such that $\mathcal{HC}(\ell_j)$ achieves the goal of converting the `.gif` file to an image file that can be edited using the `a.tiff` editor that the College Park site has. The conversion does not need to be done on a single node on the path. Instead, each node can help to the process of conversion by performing subconversions.

- Once the College Park collaborator has completed his/her work on the `.tiff` file, he must return the file to Seattle in `.gif` format – the format the Seattle site expects the file to be returned in. This means that there must be some k, $i \leq k \leq n$ such that $\mathcal{HC}(\ell_k)$ achieves the goal of converting the `.tiff` file to a `.gif` file.
- Finally, the *total network cost* should be minimized, i.e. the cost of sending the file across the different links on the network must be minimized.

In general, it is preferable if the client specifies the loop to be used – as the server may be servicing multiple clients, passing this responsibility onto the server may lead to an unacceptable load on the server. Furthermore, a server may decline to compute such a path, and hence, the client should be responsible for specifying the path by which the desired object is routed.

If we examine the example in Section 3.1, then the desired loop is:

$$\texttt{Seattle} \rightarrow \texttt{CollegePark} \rightarrow \texttt{Ithaca} \rightarrow \texttt{CollegePark} \rightarrow \texttt{Seattle}.$$

This means that when Seattle is initially requested for the `.gif` file by College Park, it sends it directly to College Park who passes in to Ithaca which in turn converts it (using `gif2tiff`) to `tiff` format and passes it back to College Park who edits the tiff file using `ed_tiff` and then converts it to `.gif` format at College Park itself using `tiff2gif` and sends the result back directly to Seattle. If we assume that the cost is independent of the size of the objects, then the total cost of this operation is

$$(130 + 30 + 30 + 130) \times 250 = 80000.$$

The reader may easily verify that this total cost is the cheapest possible total cost possible, even though other loops may be used to satisfy the same objective.

3.3 Collaboration in an s-COMS: A Formal Description

Consider an s-COMS $\Gamma = (G, Obj, loc, \mathcal{HC}, \mathcal{CAP})$ where $G = (V, E)$. Suppose $N \in V$ is a node in the network and o is an object that we wish to create from existing objects.

Definition 3.5 A capability c *may be applied* to object o just in case o is of the input-type of capability c. If c is a conversion capability, then the *result of applying capability* c to object o is $c(o')$. This is denoted by

$$o \xrightarrow{c} o'.$$

3.3.1 Synthesizing Objects within a Node

We will first consider the case when objects are constructed within a single node, using only the objects and capabilities contained within that node. It will turn out that once we know how to synthesize objects within a given node, then we may use this solution to solve the problem of synthesizing objects in a network of nodes.

Definition 3.6 An object o_m may be *synthesized entirely within node* N just in case there is a sequence of objects $o_1, \ldots, o_m$ such that:

1. $N \in loc(o_1)$, i.e. object o_1 is available to node N and
2. For all $1 < i \leq m$, there is a conversion capability $c_{i-1} \in \mathcal{HC}(N)$ such that

   $$o_{i-1} \xrightarrow{c_{i-1}} o_i$$

   holds.

We will call

$$o_1 \xrightarrow{c_1} o_2 \xrightarrow{c_2} o_3 \xrightarrow{c_3} \cdots o_{m-1} \xrightarrow{c_{m-1}} o_m$$

a *synthesis sequence* for object o_m within node N.

Definition 3.7 An object o may be *displayed (resp. edited) entirely within node* N if there is a synthesis sequence for object o within node N and there is a display (resp. edit capability) capability, $\mathcal{D} \in \mathcal{HC}(N)$ such that $\mathcal{D}(o) = \texttt{true}$.

We now present an algorithm that solves the following problem:

> Object Synthesis Problem. Given a node N, and an object o, determine if o can be synthesized entirely within node N. If yes, return a synthesis sequence for object o within node N.

Algorithm 1 (Object Synthesis Algorithm (OSA))

Input:

- A COMS $\Gamma = (G, Obj, loc, \mathcal{HC}, \mathcal{CAP})$ where $G = (V, E)$
- A node N in V and
- An object o to be synthesized entirely within node N.

Tried: A global set variable initialized to $\emptyset$.

1. Let $Name(o)$ be of the form `str.type`.
2. Let $X = \{(c, Ob.type1) \mid c$ is a conversion capability in $\mathcal{HC}(N)$ and $Name(c(Ob.type1)) = \texttt{str.type}\}$. (* Note that Ob may not necessarily be such that $N \in loc(Ob)$ *).
3. **if** $\{(c, Ob.type1) \mid (c, Ob.type1) \in X \;\&\; (c, Ob.type1) \notin Tried\} = \emptyset$ **then return** failure and **halt.**
4. Otherwise (* i.e. X still has something in it that can be processed *)
5. **if** there exists a $(c, Ob) \in X$ such that $(c, Ob) \notin Tried$ and such that $N \in loc(Ob)$ **then return**

   $$SOL = Ob.type1 \xrightarrow{c} \texttt{str.type}$$

   and **halt.**
6. Otherwise, non-deterministically select a pair $(c, Ob) \in X$.
7. Set X to $X - \{(c, Ob)\}$ and set $Tried$ to $Tried \cup \{(c, Ob)\}$.
8. **If** the call $\mathsf{OSA}(\Gamma, N, Ob)$ succeeds and returns $SOL1$, **then**

   $$SOL = SOL1 \xrightarrow{c} \texttt{str.type}.$$

   Return SOL. **Halt** with success.
9. Otherwise, goto Step 3.

To see how the above algorithm works, let us return to the motivating example COMS discussed in Section 3.1.

Example 3.2 Suppose we wish to synthesize the object named `f1.bmp` in Seattle. In other words, for whatever reason, the site at Seattle wishes to obtain a bitmapped version of `f1.gif`. In this case, we call the OSA algorithm with the example COMS, the node `Seattle` and the object `f1.bmp` that we wish to synthesize.

- In step 2 of the OSA algorithm, we set

$$X = \{(\texttt{tiff2bmp}, \texttt{f1.tiff})\}.$$

  This indicates that:

- One way to synthesize `f1.bmp` is by first synthesizing a file `f1.tiff` and then by applying the operator `tiff2bmp` to convert it into a bitmapped file.
- The test in Step 3 succeeds, but that in Step 5 fails, and control passes to Step 6. The only triple that can be selected is `(tiff2bmp,f1.tiff)`, so it is selected and in Step 7, we reset$X$ to $\emptyset$ and $Tried$ is set to $\{(\texttt{tiff2bmp}, \texttt{f1.tiff})\}$.
- In step 8, we call the OSA algorithm recursively with a request to synthesize `f1.tiff`.
- This latter request succeeds immediately via step 5, leading to $SOL1 = \texttt{f1.gif} \stackrel{gif2tiff}{\longrightarrow} \texttt{f1.tiff}$.
- In step 8 of the original call, we now return

$$SOL = \texttt{f1.gif} \stackrel{gif2tiff}{\longrightarrow} \texttt{f1.tiff} \stackrel{tiff2bmp}{\longrightarrow} \texttt{f1.bmp}.$$

- The algorithm now halts. □

Example 3.3 A slightly more egregious example shows what can happen when a node contains "reversible" actions. For instance, suppose Γ is a COMS containing just one node having the object $o1.fmt1$ and having two conversion functions:

1. c_1 that converts $X.fmt1$ to $X.fmt2$;
2. c_2 that converts $X.fmt2$ to $X.fmt1$;

Suppose we wish to create an object $o2.fmt2$. The OSA algorithm terminates because the global variable $Tried$ in it is initially empty; after executing step 3, $X = \{(c_1, o2.fmt1)\}$; however, after step 7, $X = \emptyset$ and $Tried = \{(c_1, o2.fmt1)\}$; in step 8, we recursively call the OSA algorithm with arguments $(\Gamma, N, o2.fmt1)$. In step 3 of the recursive call, $X = \{(c_2, o2.fmt2)\}$; in Step 7, $X = \emptyset$ and $Tried = \{(c_1, o2.fmt1), (c_2, o2.fmt2)\}$. In Step 8, we recursively call the OSA algorithm with arguments $(\Gamma, N, o2.fmt2)$. In Step 3 of the recursive call, we return with failure, cuasing the initial invocation of the OSA algorithm to terminate with failure. □

The following result tells us that the OSA algorithm is sound and complete, and terminates in linear-time (proofs have been omitted for brevity).

Theorem 3.1 (Soundness and Completeness of the OSA Algorithm) Suppose $\Gamma = (G, Obj, loc, \mathcal{HC}, \mathcal{CAP})$ is a COMS, $G = (V, E)$, N is a node in V, and o is an object we wish to synthesize.

1. If SS is a synthesis sequence for object o within node N, then there exists a way of selecting pairs in Step 6 of the OSA algorithm such that the OSA algorithm terminates with success and returns $SOL = SS$.

2. If the OSA algorithm terminates with failure, then there is no synthesis sequence for object o within node N.

3. If the OSA algorithm terminates with success and returns SOL, then SOL is a synthesis sequence for object o within node N.

4. The OSA algorithm is guaranteed to terminate in time $O(\mathsf{card}(\mathcal{HC}(N)) \times \mathsf{card}(Obj_N))$ where $Obj_N = \{o \mid N \in loc(o)\}$.

3.3.2 Optimally Synthesizing Objects within a Node

The OSA algorithm assumes that any way of synthesizing an object is acceptable. However, in practice, different ways of synthesizing objects may lead to very different results. For example, a synthesis sequence SS_1 to synthesize an object may involve invoking various expensive conversion methods. In contrast, a different synthesis sequence SS_2 may achieve the desired synthesis in a "much cheaper" way. In this subsection, we will define the "cost" of a synthesis sequence and then develop a technique called the OptOSA technique that is always guaranteed to optimally synthesize objects within a given node.

Let us suppose that each conversion capability c_i has an associated *cost rate*, $\mathsf{cost\_rate}(c_i)$, and an associated *size ratio*, $\mathsf{size\_ratio}(c_i)$. Intuitively, if the cost rate of c_i is 24, then this means that the cost of converting an object o of size $\mathsf{size}(o)$ bytes is $24 \times \mathsf{size}(o)$. Similarly, if the the size ratio of c_i is 1.6, then this means that the size of the object $c_i(o)$ is 1.6 times the size of object o. Therefore, if

$$SS = o_1 \xrightarrow{c_1} o_2 \xrightarrow{c_2} o_3 \xrightarrow{c_3} \cdots o_{m-1} \xrightarrow{c_{m-1}} o_m$$

is a synthesis sequence for object o within node N, then the *total cost* of synthesizing o entirely within node N is given by:

$$\mathtt{TotCost(SS)} = \sum_{i=1}^{m-1} \mathsf{cost\_rate}(c_i) \times \mathsf{size}(o_i).$$

As $\mathsf{size}(o_i) = \mathsf{size}(o_{i-1}) \times \mathsf{size\_ratio}(c_{i-1})$ for $i > 1$, it follows that

$$\mathsf{size}(o_i) = \mathsf{size}(o_1) \times \prod_{j=1}^{i-1} \mathsf{size\_ratio}(c_j) \text{ when } i > 1$$

Thus,

$$\mathtt{TotCost(SS)} = \sum_{i=1}^{m-1} \left(\mathsf{cost\_rate}(c_i) \times \left(\mathsf{size}(o_1) \times \prod_{j=1}^{i-1} \mathsf{size\_ratio}(c_j) \right) \right).$$

When attempting to synthesize an object entirely within a given node, we would like to find a synthesis sequence SS that has the least possible total cost `TotCost(SS)`. Obviously, we would like to do this without explicitly constructing all possible synthesis sequences for object o within node N.

> OPTIMAL OBJECT SYNTHESIS PROBLEM. Given a node N, and an object o, find the *minimal cost* (if one exists) synthesis sequence for object o within node N.

Before developing an algorithm to efficiently compute optimal ways of synthesizing objects, we present a couple of examples to illustrate the basic ideas.

Example 3.4 Suppose we consider a very simple COMS Γ containing a node N that has five conversion functions $c_1, \ldots, c_5$ described below.

Capability Name	Input	Output	Cost Rate
c_1	X.f1	X.f2	10
c_2	X.f2	X.f3	3
c_3	X.f2	X.f5	20
c_4	X.f3	X.f4	4
c_5	X.f4	X.f5	7

Suppose node N has one object named $o.f1$ of size 50 Kbytes and suppose we wish to synthesize the object $o.f5$. Furthermore suppose all of transformations $c_1, \ldots, c_5$ are size-invariant, i.e. the sizes of the objects do not change when these conversions are applied to them. In other words, $\mathsf{size\_ratio}(c_i) = 1$ for $i = 1, \ldots, 5$. There are two synthesis sequences that accomplish the desired goal. They are:

$$SS_1 : \qquad o.f1 \xrightarrow{c_1} o.f2 \xrightarrow{c_2} o.f3 \xrightarrow{c_4} o.f4 \xrightarrow{c_5} o.f5$$

$$SS_2 : \qquad a.f1 \xrightarrow{c_1} a.f2 \xrightarrow{c_3} a.f5.$$

Even though SS_1 is a longer sequence (in terms of the number of conversions that must be performed), it turns out, in this example, that the total cost of SS_1 is less than the total cost of SS_2.

$$\mathtt{TotCost}(SS_1) = (10 \times 50) + (3 \times 50) + (4 \times 50) + (7 \times 50) = 1200.$$

$$\mathtt{TotCost}(SS_2) = (10 \times 50) + (20 \times 50) = 1500.$$

□

Example 3.5 Suppose we consider another very simple COMS Γ containing a node N that has four conversion functions $c_1, \ldots, c_4$ described below. Node N has one object named $o.f1$ of size s Kbytes.

Capability Name	Input	Output	Cost Rate	Size Ratio
c_1	X.f1	X.f2	5	2
c_2	X.f1	X.f3	9	3
c_3	X.f2	X.f4	8	1
c_4	X.f3	X.f4	3	1

Suppose we are interested in synthesizing the object $o.f4$. There are two synthesis sequences that accomplish the desired goal. They are:

$$SS_1 : \qquad o.f1 \xrightarrow{c_1} o.f2 \xrightarrow{c_3} o.f4$$

and

$$SS_2 : \qquad o.f1 \xrightarrow{c_2} o.f3 \xrightarrow{c_4} o.f4.$$

$$\mathtt{TotCost}(SS_1) = 5s + 16s = 21s.$$

$$\mathtt{TotCost}(SS_2) = 9s + 9s = 18s.$$

Therefore, SS_2 is a cheaper way of synthesizing object $o.f4$. □

We are now ready to present the OptOSA technique for finding optimal ways of synthesizing objects. The OptOSA technique for finding optimal ways of synthesizing objects attempts to first construct an object *bottom-up* by iteratively computing a function T defined below.

Definition 3.8 (Operator T)

Input:

- A COMS $\Gamma = (G, Obj, loc, \mathcal{HC}, \mathcal{CAP})$ where $G = (V, E)$
- A node N in V and
- An object o of the form $X.type$ that we wish to synthesize in an optimal manner.

Output: An optimal synthesis sequence for object o entirely within node N.

1. $T^0(X.type) = \{(X.t, \mathsf{size}(X.t), 0, X.t) \mid N \in loc(X.t)\}$.
2. $T^{i+1}(X.type) = T^i(X.type) \cup \{(X.t, S, C, Seq) \mid$ there is a quadruple $(X.t', S1, C1, Seq1) \in T^i(X.type)$ and there exists a $c \in \mathcal{HC}(N)$ such that $c(X.t') = X.t$ and $C = C1 + \mathsf{cost\_rate}(c) \times S1$ and $S = \mathsf{size\_ratio}(c) \times S1$ and $Seq = Seq1 \xrightarrow{c} X.t\}$.

To see how the above definition works, consider the following example:

Example 3.6 Suppose we wish to synthesize the object `a.tiff` within a node that has the following capabilities:

$$\texttt{dvi2ps}, \texttt{ps2tiff}, \texttt{dvi2tiff}, \texttt{dvi2bmp}, \texttt{bmp2tiff}.$$

As is common, `x2y` indicates a conversion capability that converts files of type `x` to one of type `y`. Furthermore, suppose that the file `a.dvi` of size (1000 bytes) is available within node N. In addition, $\mathsf{size\_ratio}(c_i)$ and $\mathsf{cost\_rate}(c_i)$ are given by:
size_ratio(`dvi2ps`) $= 3$; size_ratio(`ps2tiff`) $= 1.4$; size_ratio(`dvi2tiff`) $= 10$; size_ratio(`dvi2bmp`) $= 14$; size_ratio(`bmp2tiff`) $= 1.2$;
cost_rate(`dvi2ps`) $= 20$; cost_rate(`ps2tiff`) $= 12$; cost_rate(`dvi2tiff`) $= 16$; cost_rate(`dvi2bmp`) $= 2$; cost_rate(`bmp2tiff`) $= 9$;
There are three synthesis sequences that can be used to synthesize object `a.tiff`. These sequences are:

$$\mathsf{SS}_1 : \mathsf{a.dvi} \xrightarrow{\texttt{dvi2ps}} \mathsf{a.ps} \xrightarrow{\texttt{ps2tiff}} \texttt{a.tiff}.$$

$$\mathsf{SS}_2 : \mathsf{a.dvi} \xrightarrow{\texttt{dvi2tiff}} \texttt{a.tiff}.$$

$$\mathsf{SS}_3 : \mathsf{a.dvi} \xrightarrow{\texttt{dvi2bmp}} \mathsf{a.bmp} \xrightarrow{\texttt{bmp2tiff}} \texttt{a.tiff}.$$

The operator T works in the following way:

- $T^0(\texttt{a.tiff}) = \{(\texttt{a.dvi}, 1000, 0, \texttt{a.dvi})\}$.
- $T^1(\texttt{a.tiff}) = T^0(\texttt{a.tiff}) \cup \{$

$$(\texttt{a.ps}, 3000, 20000, \texttt{a.dvi} \xrightarrow{\texttt{dvi2ps}} \texttt{a.ps}),$$
$$(\texttt{a.tiff}, 10000, 16000, \texttt{a.dvi} \xrightarrow{\texttt{dvi2tiff}} \texttt{a.tiff})^{\mathsf{SS}_1},$$
$$(\texttt{a.bmp}, 14000, 2000, \texttt{a.dvi} \xrightarrow{\texttt{dvi2bmp}} \texttt{a.bmp})\}.$$

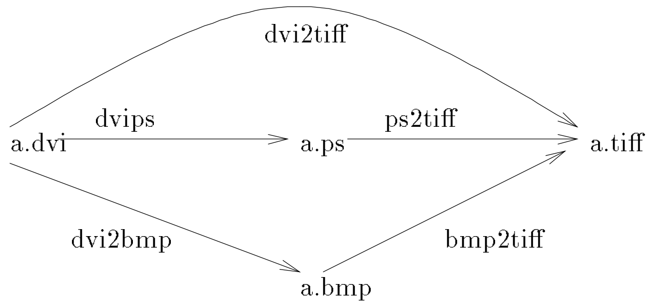

Figure 2: Synthesis Sequences for Example

- $T^2(\texttt{a.tiff}) = T^1(\texttt{a.tiff}) \cup \{$

$$(\texttt{a.tiff}, 4200, 56000, \texttt{a.dvi} \overset{\texttt{dvi2ps}}{\longrightarrow} \texttt{a.ps} \overset{\texttt{ps2tiff}}{\longrightarrow} \texttt{a.tiff})^{SS_2}.$$
$$(\texttt{a.tiff}, 16800, 128000, \texttt{a.dvi} \overset{\texttt{dvi2bmp}}{\longrightarrow} \texttt{a.bmp} \overset{\texttt{bmp2tiff}}{\longrightarrow} \texttt{a.tiff})^{SS_3}.$$

It is easy to see that $T^2(\texttt{a.tiff}) = T^3(\texttt{a.tiff})$ and hence, $T^2(\texttt{a.tiff})$ is a fixpoint of the operator T. Each of the marked quadruples in this fixpoint encodes one of the sequences SS_1, SS_2, SS_3 together with the cost of that synthesis sequence. The reader will easily observe that the cheapest cost sequence is SS_1 whose total cost is only 16000. Figure 2 shows a diagrammatic rendering of the sequences involved.

The reader will notice that each and every possible synthesis sequence for the object `a.tiff` is present in the fixpoint $T^2(\texttt{a.tiff})$ in the above example, and furthermore, that this fixpoint enumerates each and every "path" between `a.dvi` and `a.tiff` in Figure 2. The OptOSA algorithm will in many cases, never explore many of these paths by optimizing the computation of the fixpoint of $T^2(\texttt{a.tiff})$ so as to eliminate paths that are not likely to lead to a low cost. However, before developing the OptOSA algorithm, we present some elementary properties of the T operator.

Lemma 3.1 (Properties of T Operator) Consider a COMS Γ, a node N in Γ, and an object o of the form $X.type$ that we wish to synthesize within node N. For all j, if $T^j(X.type)$ contains a quadruple of the form $(o', \mathsf{size}(o'), C', SS')$, then there is a synthesis sequence SS' of object o' entirely within node N. $\Box$

The main problem with the above lemma is that it computes *all* synthesis sequences for the object o. However, in practice, we would like to compute an optimal (i.e. least cost) synthesis sequence without computing all such sequences. We are now ready to define the OptOSA technique which efficiently computes optimal synthesis sequences.

Algorithm 2 (Optimal Object Synthesis Algorithm (OptOSA))

Input:

- A COMS $\Gamma = (G, Obj, loc, \mathcal{HC}, \mathcal{CAP})$ where $G = (V, E)$
- A node N in V and
- An object $o = a.type$ to be synthesized in an optimal manner.

Output: An optimal synthesis sequence for object o entirely within node N.

1. Our algorithm uses a special node structure that has the following fields: name, size, cost, seq, overallcost, next – the first four of these fields refer to the four components of the quadruples manipulated in Definition 3.8. overallcost refers to the total cost of the sequence associated with a node. In other words, if a node has SS in its seq field, then the overallcost field contains the value TotCost defined earlier. The next field is a pointer to another node of the same type.

2. SOL := NIL.

3. Compute $T^0(a.type)$.

4. For each quadruple of the form $(a.t, \mathsf{size}(a.t), 0, a.t)$ in $T^0(a.type)$, create a node V having

$$V.\mathsf{name} = a.t; V.\mathsf{size} = \mathsf{size}(a.t);$$

$$V.\mathsf{cost} = 0; V.\mathsf{seq} = a.t; V.\mathsf{overallcost} = 0.$$

   Let X be a linked list consisting of all these nodes.

   (* We will always assume that X is sorted in ascending order according to the overallcost field. As initially all these nodes have their overallcost field set to 0, this means that when this step is first executed, any ordering will suffice. *)

5. **if** $X = NULL$ **then halt** with failure.

6. **else**

   (a) i. $Cur := head(X)$; $X := tail(X)$;

   ii. **if** Cur.name $= a.type$ then **return** Cur.seq and **halt.**

   iii. $Cand := \{(c, o') \mid c \in \mathcal{HC}(N)$ and $c(Cur.\mathsf{name}) = o'$ for and $N \in loc(o')\}$;

   iv. **For each** $(c, o') \in Cand$ create a node $V_{c,o'}$ with fields:

   $V_{c,o'}.\mathsf{name} = o'$;
   $V_{c,o'}.\mathsf{size} = \mathsf{size}(o') = \mathsf{size\_ratio}(c) \times \mathsf{size}(Cur.\mathsf{name})$;
   $V_{c,o'}.\mathsf{cost} = \mathsf{size}(Cur.name) \times \mathsf{cost\_rate}(c)$;
   $V_{c,o'}.\mathsf{seq} = Cur.\mathsf{seq} \xrightarrow{c} o'$;
   $V_{c,o'}.\mathsf{overallcost} = Cur.\mathsf{overallcost} + \mathsf{cost\_rate}(c) \times \mathsf{size}(Cur.\mathsf{name})$.

   if there is not a node M in $V_{c,o'}$.seq such that $V_{c,o'}$.name $= M$.name and $V_{c,o'}.\mathsf{size} \geq M.size$ **then** insert $V_{c,o'}$ into the list X, keeping X in sorted order w.r.t. the totcost field.

   v. Goto Step 5.

To illustrate why the OptOSA algorithm works much more efficiently than the OSA algorithm, let us return to Example 3.6.

Example 3.7 The reader may already have noticed that T^1(a.tiff) contains a synthesis sequence which is captured by the quadruple:

$$(\texttt{a.tiff}, 10000, 16000, \texttt{a.dvi} \xrightarrow{\texttt{dvi2tiff}} \texttt{a.tiff}).$$

This quadruple says that the object a.tiff may be synthesized by using the synthesis sequence

$$(\texttt{a.dvi} \stackrel{\texttt{dvi2tiff}}{\longrightarrow} \texttt{a.tiff})$$

at a total cost of 16,000. Notice that in $T^1(\texttt{a.tiff})$, we also have the quadruple

$$(\texttt{a.ps}, 3000, 20000, \texttt{a.dvi} \stackrel{\texttt{dvips}}{\longrightarrow} \texttt{a.ps})$$

which only goes *part of the way* towards computing a synthesis sequence for a.tiff, but which has already incurred a higher cost, viz. 20,000. Furthermore, in the definition of the T operator, this quadruple leads to the generation of the new quadruple

$$(\texttt{a.tiff}, 4200, 56000, \texttt{a.dvi} \stackrel{\texttt{dvips}}{\longrightarrow} \texttt{a.ps} \stackrel{\texttt{ps2tiff}}{\longrightarrow} \texttt{a.tiff})$$

in $T^2(\texttt{a.tiff})$ even though there is already a better solution. The OptOSA algorithm would eliminate such redundant possibilities. To see how, let us now apply the OptOSA algorithm to the case of Example 3.6. Here is what happens:

- Initially, $T^0(\texttt{a.tiff}) = \{(\texttt{a.dvi}, 1000, 0, \texttt{a.dvi})\}$. X points to a list of one node containign the quadruple ((a.dvi,1000,0,a.dvi).
- Step 6(a) of the OptOSA algorithm now leads to the following quadruples stored in the order listed below (the order is significant).

$$(\texttt{a.bmp}, 14000, 2000, \texttt{a.dvi} \stackrel{\texttt{dvi2bmpf}}{\longrightarrow} \texttt{a.bmp}).$$
$$(\texttt{a.tiff}, 10000, 16000, \texttt{a.dvi} \stackrel{\texttt{dvi2tiff}}{\longrightarrow} \texttt{a.tiff}),$$
$$(\texttt{a.ps}, 3000, 20000, \texttt{a.dvi} \stackrel{\texttt{dvips}}{\longrightarrow} \texttt{a.ps}).$$

- The first element in this list is the quadruple

$$(\texttt{a.bmp}, 14000, 2000, \texttt{a.dvi} \stackrel{\texttt{dvi2bmp}}{\longrightarrow} \texttt{a.bmp}).$$

  Note that at this stage, it is still possible that there is a cheap synthesis sequence which first converts a.dvi to a.tiff via an intermediate conversion to a.bmp. That is why the solution of cost 16000 in the above list of quadruples is not returned immediately.
- At the next stage, X now points to the ordered list:

$$(\texttt{a.tiff}, 10000, 16000, \texttt{a.dvi} \stackrel{\texttt{dvi2tiff}}{\longrightarrow} \texttt{a.tiff}),$$
$$(\texttt{a.ps}, 3000, 20000, \texttt{a.dvi} \stackrel{\texttt{dvips}}{\longrightarrow} \texttt{a.ps}).$$
$$(\texttt{a.tiff}, 16800, 128000, \texttt{a.dvi} \stackrel{\texttt{dvi2bmp}}{\longrightarrow} \texttt{a.bmp} \stackrel{\texttt{bmp2tiff}}{\longrightarrow} \texttt{a.tiff}).$$

- In the next stage, the first element of this list, viz. the quadruple

$$(\texttt{a.tiff}, 10000, 16000, \texttt{a.dvi} \stackrel{\texttt{dvi2tiff}}{\longrightarrow} \texttt{a.tiff})$$

  is returned as the optimal solution.

The reader will notice that many quadruples listed in Example 3.6 never occur in this computation, because they have been discarded by the OptOSA algorithm, thus leading to highly improved performance (Section 4 reports on experimental results).

Theorem 3.2 (Soundness and Completeness of the OptOSA Algorithm) Suppose $\Gamma = (G, Obj, loc, \mathcal{HC}, \mathcal{CAP})$ is a COMS, $G = (V, E)$, N is a node in V, and o is an object we wish to synthesize. Then:

1. If the OptOSA algorithm returns a synthesis sequence, then that synthesis sequence is an optimal synthesis sequence for object o within node N.
2. The OptOSA algorithm returns with failure iff there is no synthesis sequence for object o in node N.
3. If $\{SS_1, \ldots, SS_n\}$, $n \geq 1$, is the set of all optimal synthesis sequences for N, then the OptOSA algorithm is guaranteed to return some SS_i.

Theorem 3.2 is extremely important for a number of reasons. First and foremost, the theorem says that the OptOSA algorithm correctly finds the *optimal* synthesis sequence for any object that we wish to construct. Furthermore, it says that the OptOSA algorithm always finds the optimal solution first, without finding any other solutions at all. In other words, the search strategy used by OptOSA in looking for an optimal synthesis sequence is perfect – the first synthesis sequence it comes up with is guaranteed to be the best one – hence, there is no need to evaluate multiple alternative sequences and pick the best one.

3.3.3 Optimal Object Synthesis with Quality Constraints

One of the key problems that has not been discussed in the preceding sections is the issue of *quality*. Many transformations may cause the size of a media-object to decrease, yet these transformations may not preserve the quality of the object. For example, utilities like `xv` in the Unix platform allow us to re-size images (like those in `tiff` and `gif` files). In such cases, we need to know whether the transformations affect the quality of the image – for example, when we reduce an image to 50% of its original size, this is accompanied by a concomitant loss of quality. Often users not only wish to synthesize objects, but they wish to synthesize objects with a certain quality measure.

In this section, we will study the problem of synthesizing objects *in the presence of quality constraints.* Even more important is the fact that in many cases, a user may wish to simultaneously achieve two objectives:

(Objective 1) minimize$_{SS}$overallcost(SS) – i.e. minimize the cost of synthesizing the object, and

(Objective 2) maximize *quality*.

However, it is entirely possible that these two goals may conflict with one another, and we will study ways to deal with such conflicts.

Consider an s-COMS $\Gamma = (G, Obj, loc, \mathcal{HC}, \mathcal{CAP})$ where $G = (V, E)$. A *quality-assessment* is a function $\mathtt{QA} : Obj \to \mathbf{R}^+$ where $\mathbf{R}^+$ is the set of non-negative real numbers. Intuitively, $\mathtt{QA}(o_1) = 300$ says that the quality of object o_1 is deemed to be 300 w.r.t. some scale. Thus, if $\mathtt{QA}(o_2)$ were 200, then o_1 would be a "better" quality object than o_1.

A *quality function* is a map $\mathtt{QF} : \mathcal{HC} \to [0.0, 1.0]$. Intuitively, $\mathtt{QF}(\mathtt{tiff2gif}) = 0.9$ says that when a tiff-image is converted to gif-format, the resulting gif image is only 90% as good as the original.

Suppose SS is a synthesis sequence for an object o_m, and suppose SS has the form

$$o_1 \xrightarrow{c_1} o_2 \xrightarrow{c_2} o_3 \xrightarrow{c_3} \cdots o_{m-1} \xrightarrow{c_{m-1}} o_m$$

The *final quality* of o_m w.r.t. SS is defined by

$$\texttt{FinQual}(o_m, SS) \quad = \quad \texttt{QA}(o_1) \times \prod_{j=1}^{m-1} \texttt{QF}(c_j).$$

Given any object o to be synthesized, let us define $\texttt{BestQual}(o) = \{SS \mid SS$ is a synthesis sequence for synthesizing object o such that $\texttt{FinQual}(o, SS) = q$ and there is no synthesis sequence SS' for o such that $\texttt{FinQual}(o, SS') < q\}$. Thus, $\texttt{BestQual}(o)$ is the set of all synthesis sequences for synthesizing object o that yield the "highest" possible quality.

The first problem that an end-user may wish to pose is the following:

> Maximal Quality at Minimal Cost Object Synthesis Problem (`BestQualLstCost`.) In this problem, the user wishes to first synthesize object o at the *maximal* quality-level possible and *subsequently* minimize the total cost. In other words, quality is the primary concern, while *cost* is to be minimized only after the optimal quality is achieved.
>
> Formally, this problem can be specified as follows: Let $\texttt{BestQualLstCost}(o) = \{SS \mid SS \in \texttt{BestQual}(o)$ and there is no synthesis sequence $SS' \in \texttt{BestQual}(o)$ such that $\texttt{TotCost}(\texttt{SS}') < \texttt{TotCost}(\texttt{SS})\}$.

For an algorithm to correctly solve this problem, given any object o to be synthesized w.r.t. a COMS Γ, the algorithm must return a synthesis sequence in $\texttt{BestQual}(o)$. We show below how both the OSA and the OptOSA algorithms may be modified to compute synthesis sequences that yield maximal quality objects at the minimal possible cost.

We first replace the operator T by a new operator, TQ. Instead of operating on quadruples as T did, TQ operates on 5-tuples obtained by augmenting the quadruples T worked on by a fifth "quality" argument.

Definition 3.9 (Operator TQ)

Input: Same as operator T (cf. Definition 3.8).

Output: A member of $\texttt{BestQual}(o)$.

1. $TQ^0(X.type) = \{(X.t, size(X.t), 0, X.t, \texttt{QA}(X.t)) \mid (X.t, size(X.t), 0, X.t) \in T^0(X.type)\}$.
2. $TQ^{i+1}(X.type) = TQ^i(X.type) \cup (X.t, S, C, Seq, Q) |$ there is a 5-tuple $(X.t', S1, C1, Seq1, Q1) \in TQ^i(X.type)$ where there exists a $c \in \mathcal{HC}(N)$ such that $c(X.t') = X.t$ and $C = C1 + \textsf{cost\_rate}(c) \times S1$ and $S = \textsf{size\_ratio}(c) \times S1$ and $Seq = Seq1 \xrightarrow{c} X.t$ and $Q = Q1 \times \texttt{QF}(c)\}$.

Note that the operator TQ is exactly like the operator T except that it deals with 5-tuples instead of quadruples – the fifth component being a quality component.

Algorithm 3 QmaxLcost Algorithm for Computing Maximal Quality, Least Cost Synthesis Sequences We use the same algorithm as the OptOSA algorithm, with the following modifications.

1. In Step 1 of the OptOSA algorithm, we assume that nodes have one extra field, denoted qual.
2. Step 3 of the OptOSA algorithm: replace computation of $T^0(a.type)$ by $TQ^0(a.type)$.

3. In Step 4 of the OptOSA algorithm, we make one additional Assignment: V.qual $:= -1$. (The reason V.qual is set to *minus* 1 is that we will eventually minimize the *the minus* of quality which is the same as maximizing quality.)

4. Furthermore, in Step 4, we assume that X is sorted in ascending order on two keys: the primary key is the qual field, the secondary key is the overallcost field. In particular, note that this means that node V_1 precedes node V_2 only if either:
   - V_1.qual $<$ $V2$.qual or
   - V_1.qual $=$ $V2$.qual and V_1.overallcost $\leq$ V_2.overallcost.

5. In Step 6(a)(iv), add an extra assignment statement:

$$V_{c,o'}.\mathsf{qual} = Cur.\mathsf{qual} \times \mathtt{QF}(c).$$

6. In Step 6(a)(iv), when inserting $V_{c,o'}$ into the list X, ensure that X is kept in sorted order w.r.t. the primary key, qual, and subsequently w.r.t. the secondary key overallcost as outlined in item 4 above.

Theorem 3.3 (Soundness and Completeness of the QmaxLcost Algorithm) Suppose $\Gamma = (G, Obj, loc, \mathcal{HC}, \mathcal{CAP})$ is a COMS, $G = (V, E)$, N is a node in V, and o is an object we wish to synthesize. Then:

1. If the **QmaxLcost** algorithm returns a synthesis sequence, then that synthesis sequence is in $\mathtt{BestQual}(o)$, i.e. this is an optimal synthesis sequence for object o within node N.

2. If $\mathtt{BestQual}(o) = \{SS_1, \ldots, SS_n\}$, $n \geq 1$, then the **QmaxLcost** algorithm is guaranteed to return some SS_i.

3. The **QmaxLcost** algorithm returns with failure iff there is no synthesis sequence for object o in node N.

4. The first solution found by the **QmaxLcost** algorithm is guaranteed to be an optimal one.

One problem with the **QmaxLcost** algorithm is that it may turn out that the cost of synthesizing a high-quality object may be too much. In such cases, a user may wish to indicate a *trade-off* between cost and quality. To do so, the user may place *weights* on cost and quality. For instance, the assignment of weights 5 and 1 to quality and cost, respectively, indicates that the user feels that quality is 5 times more important than cost.

In general, suppose w_c and w_q are positive integers denoting the weights assigned by a given user to cost and quality, respectively. Then we may define the *badness* of a synthesis sequence $SS = o_1 \xrightarrow{c_1} o_2 \xrightarrow{c_2} o_3 \xrightarrow{c_3} \cdots o_{m-1} \xrightarrow{c_{m-1}} o_m$ of an object o_m as follows:

$$\mathsf{BAD}(SS) = w_c \times \mathtt{TotCost}(SS) - w_q(\mathtt{FinQual}(SS).$$

Intuitively, we would like to minimize badness (i.e. minimize cost, maximize quality).

> Minimizing bad-ness Problem. The user may be interested in finding a synthesis sequence SS for object o such that there is no other synthesis sequence SS' for o such that $\mathsf{BAD}(SS') < \mathsf{BAD}(SS)$. This is call the problem of finding the minimally bad synthesis sequence for o.

This problem can be easily solved by a small modification of the **QmaxLcost** algorithm. The node structures and all parts of the algorithm remain unchanged except for one item: *Instead of the array X being sorted according to two keys, we keep it sorted in ascending order w.r.t. the value of the expression:*

$$w_c \times V.\mathsf{overallcost} - w_q \times V.\mathsf{qual}..$$

With this minor change, the **QmaxLcost** *algorithm works and correctly computes minimally bad synthesis sequences for any object.*

3.4 Collaboration in Distributed-COMS

In this section, we show how the framework for synthesizing objects within a single node may be easily extended to synthesize objects across a network. We have already shown how an object o may be synthesized entirely within a given node N. Suppose now that node N wishes to synthesize object o, but instead of doing so entirely within node N, it may access data and/or conversion capabilities located at other nodes. This may be modeled as follows.

Definition 3.10 Suppose $\Gamma = (G, Obj, loc, \mathcal{HC}, \mathcal{CAP})$ is a COMS where $G = (V, E)$. A *send* operation is of the form $sends(Sender, Object, Recipient)$ where $(Sender, Recipient) \in E$ — the statement $sends(Sender, Object, Recipient)$ indicates that $Sender$ is sending the specified object to the recipient.

Definition 3.11 (Distributed Synthesis Sequence) Suppose $\Gamma = (G, Obj, loc, \mathcal{HC}, \mathcal{CAP})$ is a COMS where $G = (V, E)$. An object o_m may be *synthesized by node* N just in case there is a sequence $(N_1, c_1, o_1), \ldots, (N_{m-1}, C_{m-1}, o_{m-1}), (N_m, \star, o_m)$ such that:

1. $N_m = N$ and
2. $N_1 \in loc(o_1)$ and
3. For all $1 < i \leq m$:
   (a) $N_i \in loc(o_i)$ **or**
   (b) there exists a $j < i$ c_j is a conversion capability in $\mathcal{HC}(N_j)$ such that $c_j(o_j) = o_i$ and $N_i = N_j$ **or**
   (c) there exists a $j < i$ such that $(N_j, N_i) \in E$ and $c_i = sends(N_j, o, N_i)$.

We will call

$$(N_1, o_1) \xrightarrow{c_1} (N_2, o_2) \xrightarrow{c_2} (N_3, o_3) \xrightarrow{c_3} \cdots (N_{m-1}, o_{m-1}) \xrightarrow{c_{m-1}} (N_m, o_m)$$

a *distributed synthesis sequence* for object o_m w.r.t. node N.

The main idea behind distributed synthesis sequences is that they allow a node to perform an arbitrary sequence of operations within the node on one or more objects and then send the results to another node that may then do the same. This process can be continued till the desired object is synthesized. The operator TC described below captures the above process.

Definition 3.12 (Operator TC)

Input:

- A COMS $\Gamma = (G, Obj, loc, \mathcal{HC}, \mathcal{CAP})$ where $G = (V, E)$
- A node N in V and
- An object o of the form $X.type$ that we wish to synthesize in an optimal manner.

Output: An optimal synthesis sequence for object o.

1. $TC^0(X.type) = \{(X.t, \mathsf{size}(X.t), 0, (X.t, N), N) \mid N \in loc(X.t)\}$.
2. $TC^{i+1}(X.type) = TC^i(X.type) \cup \{(X.t, S, C, Seq, N1) |$ there is a 5-tuple $(X.t', S1, C1, Seq1, N2) \in TC^i(X.type)$ and **either**

(a) $N1 = N2$ and there exists a $c \in \mathcal{HC}(N1)$ such that $c(X.t') = X.t$ and $C = C1 + \mathsf{cost\_rate}(c) \times S1$ and $S = \mathsf{size\_ratio}(c) \times S1$ and $Seq = Seq1 \xrightarrow{c} (X.t, N_1)$ **or**

(b) $(N1, N2) \in E$ and $X.t = X.t'$ and $C = C1 + \wp(N1, N2) \times S1$ and $Seq = Seq1 \xrightarrow{sends(N2,X.t',N1)} (X.t, N_2)$ }.

Intuitively, if the 5-tuple $(X.t, \mathsf{size}(X.t), S, C, Seq, N1)$ appears in TC^i for some i, then this means that node $N1$ can synthesize object $X.t$ of size $\mathsf{size}(X.t)$ using the distributed synthesis sequence Seq and incur a cost of *at most* C in the process of doing so. Note that it is entirely possible that TC^i may contain two or more tuples that are identical in all attributes execept for the cost and distributed synthesis sequence attributes – these will correspond to two or more ways in which object $X.t$ can be synthesized at node N.

Lemma 3.2 (Properties of TC Operator) Consider a COMS Γ, a node N in Γ, and an object o of the form $X.type$ that we wish to synthesize w.r.t. node N. For all j, if $TC^j(X.type)$ contains a 5-tuple of the form $(o', \mathsf{size}(o'), C', SS', N)$, then there is a distributed synthesis sequence SS' of object o' w.r.t. node N. $\square$

It is easy to see that we can easily modify both the OSA and the OptOSA algorithms to compute optimal distributed synthesis sequences.

3.4.1 The Distributed OSA Algorithm (DOSA)

In this section, we present an algorithm that extends the OSA algorithm to handle the construction of synthesis sequences across multiple nodes in a COMS.

Algorithm 4 (Distibuted Object Synthesis Algorithm (DOSA))

Input:

- A COMS $\Gamma = (G, Obj, loc, \mathcal{HC}, \mathcal{CAP})$ where $G = (V, E)$
- A node N in V and
- An object o to be synthesized in node N.

Tried: A global set variable initialized to $\emptyset$.

1. Let $Name(o)$ be of the form `str.type`.
2. Let $X_1 = \{(c, Ob.type1, N_1) \mid c$ is a conversion capability in $\mathcal{HC}(N_1)$ and $N_1 = N$ $Name(c(Ob.type1)) =$ `str.type`$\}$. (* Note that Ob may not necessarily be such that $N_1 \in loc(Ob)$ *).
3. Let $X_2 = \{(sends(N_1, Ob.type1, N_2), Ob.type1, N_1) \mid (N1, N2) \in E$ and $Name(Ob.type1) =$ `str.type`$\}$ and $N_2 = N$.
4. Let $X = X_1 \bigcup X_2$.
5. **if** $\{x \mid x \in X \,\&\, x \notin Tried\} = \emptyset$ **then return** failure and **halt.**
6. Otherwise (* i.e. X still has something in it that can be processed *)
7. **if** there exists a $(c, Ob, N_1) \in X$ such that $(c, Ob, N_1) \notin Tried$ and such that $N_1 \in loc(Ob)$ **then return**

$$SOL = (Ob, N_1) \xrightarrow{c} (\texttt{str.type}, N_1)$$

and **halt.**

8. **Otherwise if** there exists a $(sends(N_1, Ob, N_2), Ob, N_1) \in X$ such that $(sends(N_1, Ob, N_2), Ob, N_1) \notin Tried$ and such that $N_1 \in loc(Ob)$ **then return**

$$SOL = (Ob, N_1) \stackrel{sends(N_1,Ob,N_2)}{\longrightarrow} (\texttt{str.type}, N_2),$$

   and **halt.**

9. Otherwise, non-deterministically select a pair $(oper, Ob, N_1) \in X$.
10. Set X to $X - \{(oper, Ob, N_1)\}$ and set $Tried$ to $Tried \cup \{(oper, Ob, N_1)\}$.
11. **If** the call $\mathsf{OSA}(\Gamma, N_1, Ob)$ succeeds and returns $SOL1$, **then**
    (a) **if** $oper = c$ **then**

$$SOL = SOL1 \stackrel{c}{\longrightarrow} (\texttt{str.type}, N_1).$$

    Return SOL. **Halt** with success.

    (b) **if** $oper = sends(N_1, Ob, N_2)$ **then**

$$SOL = SOL1 \stackrel{sends(N_1,Ob,N_2)}{\longrightarrow} (\texttt{str.type}, N_2).$$

    Return SOL. **Halt** with success.

12. Otherwise, goto Step 5.

3.4.2 The Distributed OptOSA Algorithm (DOptOSA)

In this section, we present an algorithm that extends the DOptOSA algorithm to handle the construction of optimal synthesis sequences across multiple nodes in a COMS.

Algorithm 5 (Distributed Optimal Object Synthesis Algorithm (DOptOSA))

Input:

- A COMS $\Gamma = (G, Obj, loc, \mathcal{HC}, \mathcal{CAP})$ where $G = (V, E)$
- A node N in V and
- An object $o = a.type$ to be synthesized in an optimal manner.

Output: An optimal synthesis sequence for object o in node N.

1. Our algorithm uses a special node structure that has the following fields: name, size, cost, seq, overallcost, loc, next.
2. SOL := NIL.
3. Compute $TC^0(a.type)$.
4. For each five-tuple of the form $(a.t, \mathsf{size}(a.t), 0, a.t, N_1)$ in $T^0(a.type)$, create a node V having

$$V.\texttt{name} = a.t; V.\mathsf{size} = \mathsf{size}(a.t);$$

$$V.\mathsf{cost} = 0; V.\mathsf{seq} = a.t; V.\mathsf{overallcost} = 0; V.\mathsf{loc} = N_1.$$

   Let X be a linked list consisting of all these nodes.

   (* We will always assume that X is sorted in ascending order according to the overallcost field. As initially all these nodes have their overallcost field set to 0, this means that when this step is first executed, any ordering will suffice. *)

5. **if** $X = NULL$ **then halt** with failure.
6. **else**

(a) i. $Cur := head(X)$; $X := tail(X)$;

ii. **if** Cur.name $= a.type$ then **return** Cur.seq and **halt.**

iii. $Cand_1 := \{(c, o', N) \mid c \in \mathcal{HC}(N)$ and $c(Cur.\mathsf{name}) = o'$ and $N \in loc(o')$ and $N = Cur.\mathsf{loc}\}$;

iv. **For each** $(c, o', N_1) \in Cand$ create a node V_{c,o',N_1} with fields:

V_{c,o',N_1}.name $= o'$;
V_{c,o',N_1}.size $=$ size(o') $=$ size_ratio$(c) \times$ size$(Cur$.name$)$;
V_{c,o',N_1}.cost $=$ size$(Cur.name) \times$ cost_rate(c);
V_{c,o',N_1}.seq $= Cur$.seq $\xrightarrow{c} (o', N_1)$;
V_{c,o',N_1}.loc $= N_1$;
V_{c,o',N_1}.overallcost $= Cur$.overallcost $+$ cost_rate$(c) \times$ size$(Cur$.name$)$.

if there is not a node M in V_{c,o',N_1}.seq such that V_{c,o',N_1}.name $= M$.name and V_{c,o',N_1}.size $\geq M.size$ **then** insert V_{c,o',N_1} into the list X, keeping X in sorted order w.r.t. the `totcost` field.

v. $Cand_2 := \{(sends(N_1, o', N_2), o', N_1) | Cur.\mathsf{name} = o'$ and $(N1, N2) \in E$ and $N_1 \in loc(o')$ and $N_1 = Cur.\mathsf{loc}\}$;

vi. **For each** $(sends(N_1, o', N_2), o', N_1) \in Cand_2$ create a node V_{o',N_2} with fields:

V_{o',N_2}.name $= o'$;
V_{o',N_2}.size $=$ size(o');
V_{o',N_2}.cost $=$ size$(Cur.name) \times \wp(N1, N2)$;
V_{o',N_2}.seq $= Cur$.seq $\xrightarrow{sends(N_1,o',N_2)} (o', N_2)$;
V_{o',N_2}.loc $= N_2$;
V_{o',N_2}.overallcost $= Cur$.overallcost $+$ size$(Cur.name) \times \wp(N1, N2)$;

if there is not a node M in V_{o',N_2}.seq such that V_{o',N_2}.name $= M$.name and V_{o',N_2}.size $\geq M.size$ **then** insert V_{o',N_2} into the list X, keeping X in sorted order w.r.t. the `totcost` field.

vii. Goto Step 5.

4 Implementation and Experiments

We have implemented the OSA, OptOSA, DOSA and DOptOSA algorithms on a SUN workstation running Unix. The OSA and OptOSA algorithms included about 1300 lines of C-code. We will now describe the experiments we conducted. Each point shown in the graphs reflecting experimental results was obtained by averaging the results of about 20 runs.

We ran experiments to compare the cost of a synthesis sequence computed by the OSA algorithm against the cost of a synthesis sequence computed by the OptOSA algorithm. In this experiment, we varied the number of objects at a given node from 50 to 600 at intervals of 50. The types of these objects were varied from between 5 and 30 at steps of 10.

Figure 3 shows the graph describing the costs of synthesis sequences computed by OSA as opposed to the costs of synthesis sequences computed by OptOSA. As can be easily seen, OptOSA performs significantly better than OSA – it consistently yields better results than OSA. Both OSA and OptOSA produce better and better synthesis sequences as the number of objects is increased. Both algorithms appear to reach a "steady state" after 250 objects (in the case of OptOSA) and 500 in the case of OSA. In this steady state, OptOSA computes synthesis sequences that are about 40% as expensive as those computed by OSA. Before the steady state, OptOSA computes synthesis

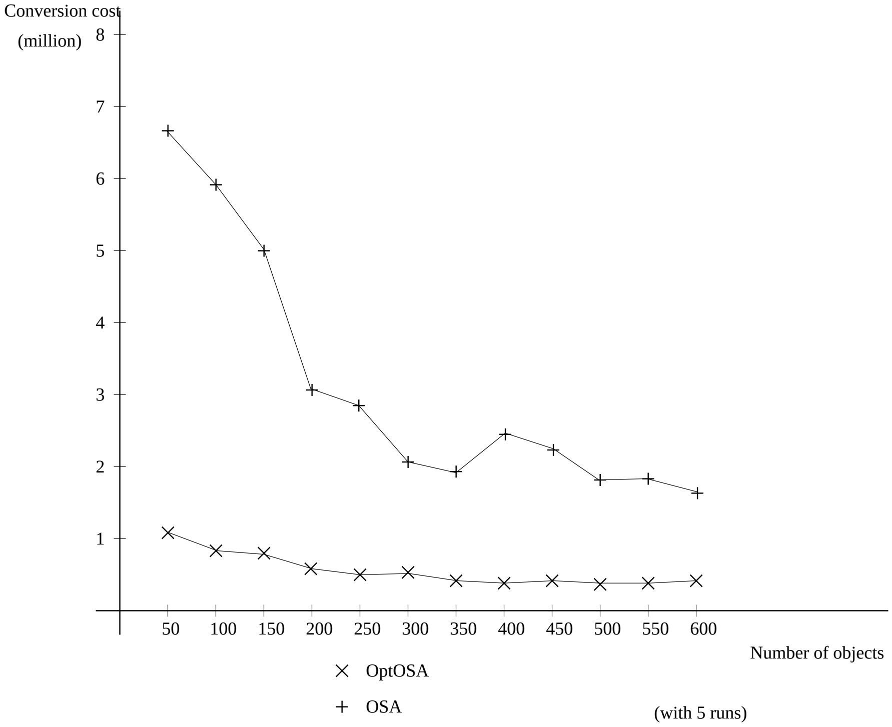

Figure 3: OSA vs. OptOSA: Cost of Synthesis Sequences Generated

sequences that are well below 40% as expensive as those computed by OSA. The complete version of this paper reports on numerous other experiments that we have conducted - due to lack of space, we are unable to report on them here.

5 Related Work

Multimedia collaboration has been a topic of wide interest. In addition to significant amount of work in the areas of CSCW, recent work in this area includes the hypermedia conversation recording system of Imai et al. [1], multipoint conferencing system of Gong [2], and Argo [3]. Synchronization has been studied by Manohar [4], and application sharing by Wolf et al [5]. Ahuja's group at AT&T [6] also has had significant contributions in collaborative services. Little [7] has presented document management system for shared data, and Wray et al. [8] have built an experimental collaborative environment called Medusa. Rajan, Vin et al. [9] started some work on formalizing the notion of multimedia collaboration.

6 Conclusions

We have classified media objects into four broad categories: static, quasi-static, temporal, and quasi-temporal, and developed a theory of media objects in which each media object is represented as a 5-tuple. We have then developed a formal definition of a collaborative multimedia system, consisting of collaborators and distributed media objects. We have presented optimal algorithms for collaborative object synthesis: i.e., for constructing multimedia documents by composing together a given set of media objects. These algorithms are then extended to incorporate quality constraints (such as image size) as well as distribution across multiple nodes. We have implemented these algorithms, and evaluated their performance.

7 References

[1] T. Imai, K. Yamaguchi, T. Muranaga, "Hypermedia Conversation Recording to Preserve Informal Artifacts in Realtime", ACM Multimedia 94, Pages 417-424
[2] F. Gong, "Multipoint Audio and Video Control for Packet-Based Multimedia Conferencing", ACM Multimedia 94, Pages 425-432
[3] H. Gajewska, "Argo: A System for Distributed Collaboration", ACM Multimedia 94, Pages 433-440
[4] N.R. Manohar and A. Prakash, "Dealing with synchronization and timing variability in the playback of interactive session recordings", ACM Multimedia 95, Pages 45-56
[5] K.H. Wolf, K. Froitzheim and P. Schulthess, "Multimedia Application Sharing in a Heterogeneous Environment", ACM Multimedia 95, Pages 57-64
[6] A. Ginsberg and S. Ahuja, "Automating envisionment of virtual meeting room histories", ACM Multimedia 95, Pages 65-76
[7] T.M. Wittenburg and T.D.C. Little, "An Adaptive Document Management System for Shared Multimedia Data", ACM Multimedia 95, Pages 245-254
[8] S. Wray, T. Glauert, and A. Hopper, "The Medusa Applications Environment", ACM Multimedia 95, Pages 265-274
[9] S. Rajan, P.V. Rangan, and H.M. Vin, "A Formal Basis for Structured Multimedia Collaborations", ACM Multimedia 95, Pages 194-201